THE ART OF COMPUTER PROGRAMMING

PRE-FASCICLE 2B

A DRAFT OF SECTION 7.2.1.2: GENERATING ALL PERMUTATIONS

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.

See also `http://www-cs-faculty.stanford.edu/~knuth/mmixware.html` for downloadable software to simulate the `MMIX` computer.

Zeroth printing (revision 12), 10 December 2004

PREFACE

I thought it worth a Dayes labour,
to write something on this Art or Science,
that the Rules thereof might not be lost and obscured.
— RICHARD DUCKWORTH, *Tintinnalogia* (1668)

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 7.2.1.2 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 — which, in turn, begins with Section 7.2.1.1, "Generating all n-tuples." (Readers of the present booklet should have already looked at Section 7.2.1.1, a draft of which is available as Pre-Fascicle 2A.) That sets the stage for the main contents of this booklet, Section 7.2.1.2: "Generating all permutations." Then will come 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 permutation generation turns out to be surprisingly rich, with ties to Sections 1.2.9, 1.3.3, 2.2.3, 2.3.4.2, 3.4.2, 4.1, 5.1.1, 5.1.2, 5.1.4, 5.2.1, 5.2.2, 5.3.1, and 6.1 of the first three volumes. There also is material related to the `MMIX` computer, defined in Section 1.3.1$'$ of Fascicle 1. 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, 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 things presented are 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 71 and 109; I've also implicitly posed additional unsolved questions in the answers to exercises 28, 58, 63, 67, 100, 106, and 112. Are those problems still open? Please let me know if you know of a solution to any of these intriguing questions. 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 6, 7, 20, 25, 41, 55, 60, 65, 66, 67, 69, 70, 76, 89, 99, 104, and/or 106.

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:–)

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.
31 December 2001

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

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

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

— D. H. LEHMER (1957)

7.2.1.2. Generating all permutations.

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

Algorithm L (*Lexicographic permutation generation*). Given a sequence of n elements $a_1a_2\ldots a_n$, initially sorted so that

$$a_1 \le a_2 \le \cdots \le a_n, \qquad (1)$$

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

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

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

L1. [Visit.] Visit the permutation $a_1a_2\ldots a_n$.

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

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

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

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

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

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

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

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

Adjacent interchanges. We saw in Section 7.2.1.1 that Gray codes are advantageous for generating n-tuples, and similar considerations apply when we want to generate permutations. The simplest possible change to a permutation is to interchange adjacent elements, and we know from Chapter 5 that any permutation can be sorted into order if we make a suitable sequence of such interchanges. (For example, Algorithm 5.2.2B works in this way.) Hence we can go backward and obtain any desired permutation, by starting with all elements in order and then exchanging appropriate pairs of adjacent elements.

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

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

$$\begin{tikzcd}[arrows=dash] & & 2112 \arrow[dr] & & \\ 1122 \arrow[r] & 1212 \arrow[ur] \arrow[dr] & & 2121 \arrow[r] & 2211; \\ & & 1221 \arrow[ur] & & \end{tikzcd} \qquad (2)$$

this graph has no Hamiltonian path.

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

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

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

(3)

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

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

0000	0010	0020	0120	0110	0100
0001	0011	0021	0121	0111	0101
0002	0012	0022	0122	0112	0102
0003	0013	0023	0123	0113	0103

(4)

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

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

Algorithm P (*Plain changes*). Given a sequence $a_1a_2\ldots a_n$ of n distinct elements, this algorithm generates all of their permutations by repeatedly interchanging adjacent pairs. It uses an auxiliary array $c_1c_2\ldots c_n$, which represents inversions as described above, running through all sequences of integers such that

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

Another array $o_1o_2\ldots o_n$ governs the directions by which the entries c_j change.

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

P2. [Visit.] Visit the permutation $a_1a_2\ldots a_n$.

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

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

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

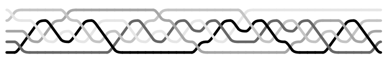
P6. [Increase s.] Terminate if $j = 1$; otherwise set $s \leftarrow s + 1$.

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

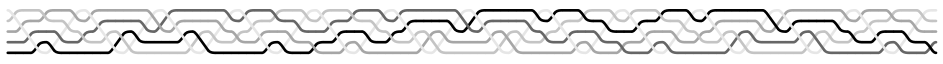
This procedure, which clearly works for all $n \ge 1$, originated in 17th-century England, when bell ringers began the delightful custom of ringing a set of bells in all possible permutations. They called Algorithm P the method of *plain changes*. Figure 18(a) illustrates the "Cambridge Forty-Eight,"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

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

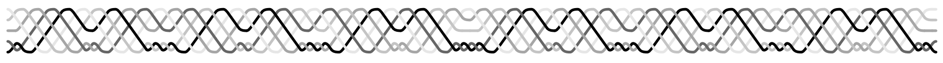
— DUCKWORTH and STEDMAN, *Tintinnalogia* (1668)



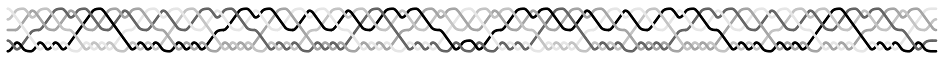
(a) The Cambridge Forty-Eight.



(b) Plain Changes.



(c) Grandsire Doubles.



(d) Stedman Doubles.

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\begin{array}{r} \texttt{SEND} \\ +\,\underline{\texttt{MORE}} \\ \texttt{MONEY} \end{array} \qquad (6)$$

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

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

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

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

Equivalently, we want to solve puzzles such as

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$176478 + 176478 + 17640 \;=\; 2576 + 368020$$

and

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

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

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

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

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

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

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

$$\alpha = \binom{012345}{250143}$$

or in the more compact cycle form

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

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

$$\alpha\beta = \binom{012345}{250143}\binom{012345}{543210} = \binom{012345}{250143}\binom{250143}{305412} = \binom{012345}{305412}$$

or in the cycle form

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

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

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

but $\beta\alpha$. In general, the operation of replacing a permutation $\alpha = a_0a_1\ldots a_{n-1}$ by some rearrangement $a_{0\beta}a_{1\beta}\ldots a_{(n-1)\beta}$ takes $k \mapsto a_{k\beta} = k\beta\alpha$. Permuting the *positions* by β corresponds to *premultiplication* by β, changing α to $\beta\alpha$; permuting the *values* by β corresponds to *postmultiplication* by β, changing α to $\alpha\beta$. Thus, for example, a permutation generator that interchanges $a_1 \leftrightarrow a_2$ is premultiplying the current permutation by (1 2), postmultiplying it by $(a_1\ a_2)$.

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

$$\boxed{\begin{array}{cccc} \texttt{0} & \texttt{1} & \texttt{3} & \texttt{2} \\ \texttt{4} & \texttt{5} & \texttt{7} & \texttt{6} \\ \texttt{c} & \texttt{d} & \texttt{f} & \texttt{e} \\ \texttt{8} & \texttt{9} & \texttt{b} & \texttt{a} \end{array}} \qquad (12)$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Then steps G3 and G4 maintain the property that

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

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

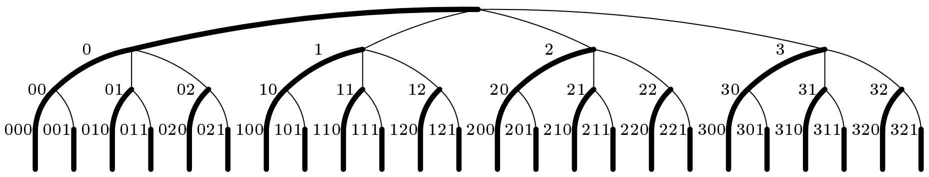


Fig. 19. Algorithm G implicitly traverses this tree when $n = 4$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Another significant special case of Algorithm G was introduced by R. J. Ord-Smith [*CACM* **10** (1967), 452; **12** (1969), 638; see also *Comp. J.* **14** (1971), 136–139], whose algorithm is obtained by setting

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

Now it is clear from (15) that

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

and once again we have

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

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

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

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

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

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

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

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

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

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

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

Step G1 should also set $k \leftarrow n-1$. Notice that the new steps are careful to preserve condition (17). The algorithm has become more complicated, because we need to know the permutations $\tau(k,j)$ and $\sigma(k,k)$ in addition to the permutations $\tau(k,j)\omega(k-1)^-$ that appear in G4. But the additional complications are often worth the effort, because the resulting program might run significantly faster.

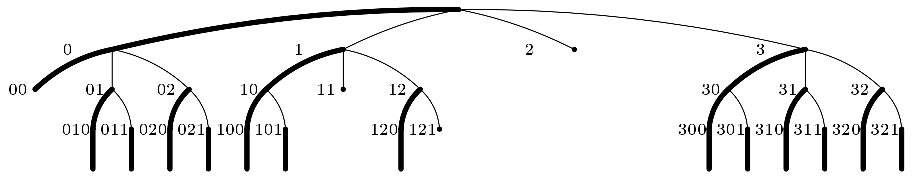


Fig. 20. Unwanted branches can be pruned from the tree of Fig. 19, if Algorithm G is suitably extended.

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

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

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

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

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

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

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

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

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

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

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

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

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

Algorithm X (*Lexicographic permutations with restricted prefixes*). This algorithm generates all permutations $a_1a_2\ldots a_n$ of $\{1,2,\ldots,n\}$ that pass a given sequence of tests

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

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

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

then we have

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

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

```
X1. [Initialize.] Set $l_k \leftarrow k+1$ for $0 \le k < n$, and $l_n \leftarrow 0$. Then set $k \leftarrow 1$.
X2. [Enter level $k$.] Set $p \leftarrow 0$, $q \leftarrow l_0$.
X3. [Test $a_1\ldots a_k$.] Set $a_k \leftarrow q$. If $t_k(a_1,\ldots,a_k)$ is false, go to X5. Otherwise,
    if $k = n$, visit $a_1\ldots a_n$ and go to X6.
X4. [Increase $k$.] Set $u_k \leftarrow p$, $l_p \leftarrow l_q$, $k \leftarrow k+1$, and return to X2.
X5. [Increase $a_k$.] Set $p \leftarrow q$, $q \leftarrow l_p$. If $q \ne 0$ return to X3.
X6. [Decrease $k$.] Set $k \leftarrow k-1$, and terminate if $k = 0$. Otherwise set $p \leftarrow u_k$,
    $q \leftarrow a_k$, $l_p \leftarrow q$, and go to X5. ▌
```

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

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

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

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

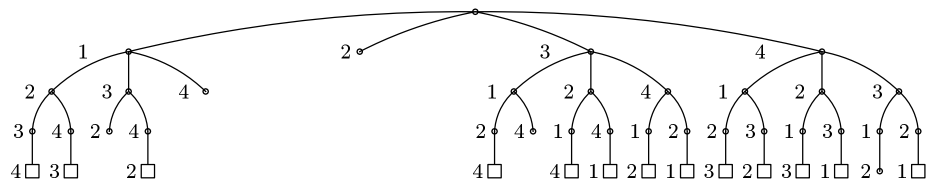


Fig. 21. The tree implicitly traversed by Algorithm X when $n=4$, if all permutations are visited except those beginning with 132, 14, 2, 314, or 4312.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Another natural idea is to construct a Sims table for which step H4 always makes a single transposition of two elements, by analogy with the construction of (27) that achieves maximum efficiency in step G4. But such a mission now turns out to be impossible: We cannot achieve it even when $n = 4$. For if we start with the identity permutation $a_0a_1a_2a_3 = 0123$, the transitions that take us from control table $c_0c_1c_2c_3 = 0000$ to 0001 to 0002 to 0003 must move the 3; so, if they are transpositions, they must be $(3\,a)$, $(a\,b)$, and $(b\,c)$ for some permutation abc of $\{0,1,2\}$. The permutation corresponding to $c_0c_1c_2c_3 = 0003$ is now $\sigma(3,3)^- = (b\,c)(a\,b)(3\,a) = (3\,a\,b\,c)$; and the next permutation, which corresponds to $c_0c_1c_2c_3 = 0010$, will be $\sigma(2,1)^-$, which must fix the element 3. The only suitable transposition is $(3\,c)$, hence $\sigma(2,1)^-$ must be $(3\,c)(3\,a\,b\,c) = (a\,b\,c)$. Similarly we find that $\sigma(2,2)^-$ must be $(a\,c\,b)$, and the permutation corresponding to $c_0c_1c_2c_3 = 0023$ will be $(3\,a\,b\,c)(a\,c\,b) = (3\,c)$. Step H4 is now supposed to convert this to the permutation $\sigma(1,1)^-$, which corresponds to the control table 0100 that follows 0023. But the only transposition that will convert $(3\,c)$ into a permutation that fixes 2 and 3 is $(3\,c)$; and the resulting permutation also fixes 1, so it cannot be $\sigma(1,1)^-$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
E1. [Initialize.] Set $b_j \leftarrow j$ and $c_{j+1} \leftarrow 0$ for $0 \le j < n$.
E2. [Visit.] Visit the permutation $a_0 \ldots a_{n-1}$.
E3. [Find $k$.] Set $k \leftarrow 1$. Then if $c_k = k$, set $c_k \leftarrow 0$, $k \leftarrow k+1$, and repeat until
    $c_k < k$. Terminate if $k = n$, otherwise set $c_k \leftarrow c_k + 1$.
E4. [Swap.] Interchange $a_0 \leftrightarrow a_{b_k}$.
E5. [Flip.] Set $j \leftarrow 1$, $k \leftarrow k-1$. If $j < k$, interchange $b_j \leftrightarrow b_k$, set $j \leftarrow j+1$,
    $k \leftarrow k-1$, and repeat until $j \ge k$. Return to E2. ▌
```

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

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

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

$$E_\infty = 121213212123121213212124313132131312\ldots. \qquad (39)$$

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

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

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

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

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

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

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

Proof. See exercise 73. ▮

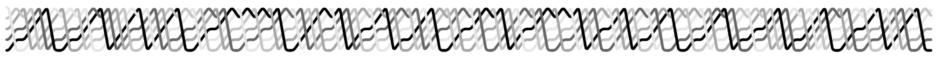
In particular, when $\alpha=\sigma$ and $\beta=\tau$ as above, we have $g=n!$, $a=n$, $b=2$, and $c=n-1$, because $\sigma\tau^- = (2\ \ldots\ n)$. Therefore we conclude that no Hamiltonian cycle is possible when $n\ge 4$ is even. However, a Hamiltonian *path* is easy to construct when $n = 4$, because we can join up the 12-cycles

$$\begin{aligned}&1234 \to 2341 \to 3412 \to 4312 \to 3124 \to 1243 \to 2431\\ &\qquad \to 4231 \to 2314 \to 3142 \to 1423 \to 4123 \to 1234,\\ &2134 \to 1342 \to 3421 \to 4321 \to 3214 \to 2143 \to 1432\\ &\qquad \to 4132 \to 1324 \to 3241 \to 2413 \to 4213 \to 2134,\end{aligned} \tag{41}$$

by starting at 2341 and jumping from 1234 to 2134, ending at 4213.

Ruskey, Jiang, and Weston [*Discrete Applied Math.* **57** (1995), 75–83] undertook an exhaustive search in the σ–τ graph for $n = 5$ and discovered that it has five essentially distinct Hamiltonian cycles, one of which (the "most beautiful") is illustrated in Fig. 22(a). They also found a Hamiltonian path for $n = 6$; this was a difficult feat, because it is the outcome of a 720-stage binary decision tree. Unfortunately the solution they discovered has no apparent logical structure. A somewhat less complex path is described in exercise 70, but even that path cannot be called simple. Therefore a σ–τ approach will probably not be of practical interest for larger values of n unless a new construction is discovered. R. C. Compton and S. G. Williamson [*Linear and Multilinear Algebra* **35** (1993), 237–293] have proved that Hamiltonian cycles exist for all n if the three generators σ, σ^-, and τ are allowed instead of just σ and τ; their cycles have the interesting property that every nth transformation is τ, and the intervening $n-1$ transformations are either all σ or all σ^-. But their method is too complicated to explain in a short space.

Exercise 69 describes a general permutation algorithm that is reasonably simple and needs only three generators, each of order 2. Figure 22(b) illustrates the case $n = 5$ of this method, which was motivated by examples of bell-ringing.


(a) Using only transitions (1 2 3 4 5) and (1 2).

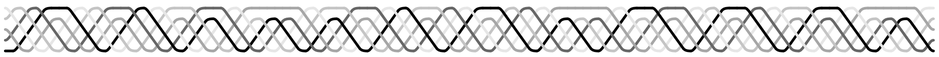

(b) Using only transitions (1 2)(3 4), (2 3)(4 5), and (3 4).

Fig. 22. Hamiltonian cycles for 5! permutations.

Faster, faster. What is the fastest way to generate permutations? This question has often been raised in computer publications, because people who examine $n!$ possibilities want to keep the running time as small as possible. But the answers have generally been contradictory, because there are many different ways to formulate the question. Let's try to understand the related issues by studying how permutations might be generated most rapidly on the `MMIX` computer.

Suppose first that our goal is to produce permutations in an array of n consecutive memory words (octabytes). The fastest way to do this, of all those we've seen in this section, is to streamline Heap's method (27), as suggested by R. Sedgewick [*Computing Surveys* **9** (1977), 157–160].

The key idea is to optimize the code for the most common cases of steps G2 and G3, namely the cases in which all activity occurs at the beginning of the array. If registers u, v, and w contain the contents of the first three words, and if the next six permutations to be generated involve permuting those words in all six possible ways, we can clearly do the job as follows:

```
PUSHJ 0,Visit
STO v,A0;  STO u,A1;  PUSHJ 0,Visit
STO w,A0;  STO v,A2;  PUSHJ 0,Visit
STO u,A0;  STO w,A1;  PUSHJ 0,Visit          (42)
STO v,A0;  STO u,A2;  PUSHJ 0,Visit
STO w,A0;  STO v,A1;  PUSHJ 0,Visit
```

(Here `A0` is the address of octabyte a_0, etc.) A complete permutation program, which takes care of getting the right things into u, v, and w, appears in exercise 77, but the other instructions are less important because they need to be performed only $\frac{1}{6}$ of the time. The total cost per permutation, not counting the $4v$ needed for `PUSHJ` and `POP` on each call to `Visit`, comes to approximately $2.77\mu + 5.69v$ with this approach. If we use four registers u, v, w, x, and if we expand (42) to 24 calls on `Visit`, the running time per permutation drops to about $2.19\mu + 3.07v$. And with r registers and $r!$ `Visit`s, exercise 78 shows that the cost is $(2 + O(1/r!))(\mu + v)$, which is very nearly the cost of two `STO` instructions.

The latter is, of course, the minimum possible time for any method that generates all permutations in a sequential array. . . . Or is it? We have assumed that the visiting routine wants to see permutations in consecutive locations, but perhaps that routine is able to read the permutations from different starting points. Then we can arrange to keep a_{n-1} fixed and to keep two copies of the other elements in its vicinity:

$$a_0 a_1 \ldots a_{n-2} a_{n-1} a_0 a_1 \ldots a_{n-2}. \tag{43}$$

If we now let $a_0 a_1 \ldots a_{n-2}$ run through $(n-1)!$ permutations, always changing both copies simultaneously by doing two `STO` commands instead of one, we can let every call to `Visit` look at the n permutations

$$a_0 a_1 \ldots a_{n-1}, \quad a_1 \ldots a_{n-1} a_0, \quad \ldots, \quad a_{n-1} a_0 \ldots a_{n-2}, \tag{44}$$

which all appear consecutively. The cost per permutation is now reduced to the cost of three simple instructions like `ADD`, `CMP`, `PBNZ`, plus $O(1/n)$. [See Varol and Rotem, *Comp. J.* **24** (1981), 173–176.]

Furthermore, we might not want to waste time storing permutations into memory at all. Suppose, for example, that our goal is to generate all permutations of $\{0, 1, \ldots, n-1\}$. The value of n will probably be at most 16, because $16! = 20{,}922{,}789{,}888{,}000$ and $17! = 355{,}687{,}428{,}096{,}000$. Therefore an entire permutation will fit in the 16 nybbles of an octabyte, and we can keep it in a single register. This will be advantageous only if the visiting routine doesn't need to unpack the individual nybbles; but let's suppose that it doesn't. How fast can we generate permutations in the nybbles of a 64-bit register?

One idea, suggested by a technique due to A. J. Goldstein [*U. S. Patent 3383661* (14 May 1968)], is to precompute the table $(t[1],\ldots,t[5039])$ of plain-change transitions for seven elements, using Algorithm T. These numbers $t[k]$ lie between 1 and 6, so we can pack 20 of them into a 64-bit word. It is convenient to put the number $\sum_{k=1}^{20} 2^{3k-1}t[20j+k]$ into word j of an auxiliary table, for $0 \le j < 252$, with $t[5040]=1$; for example, the table begins with the codeword

$$00|001|010|011|100|101|110|100|110|101|100|011|010|001|110|001|010|011|100|101|110|00.$$

The following program reads such codes efficiently:

```
Perm  ⟨Set register a to the first permutation⟩
0H    LDA   p,T       p ← address of first codeword.
      JMP   3F
1H    ⟨Visit the permutation in register a⟩
      ⟨Swap the nybbles of a that lie t bits from the right⟩
      SRU   c,c,3     c ← c ≫ 3.
2H    AND   t,c,#1c   t ← c ∧ (11100)₂.                        (45)
      PBNZ  t,1B      Branch if t ≠ 0.
      ADD   p,p,8
3H    LDO   c,p,0     c ← next codeword.
      PBNZ  c,2B      (The final codeword is followed by 0.)
      ⟨If not done, advance the leading n − 7 nybbles and return to 0B⟩
```

Exercise 79 shows how to ⟨Swap the nybbles ... ⟩ with seven instructions, using bit manipulation operations that are found on most computers. Therefore the cost per permutation is just a bit more than $10v$. (The instructions that fetch new codewords cost only $(\mu+5v)/20$; and the instructions that advance the leading $n-7$ nybbles are even more negligible since their cost is divided by 5040.) Notice that there is now no need for PUSHJ and POP as there was with (42); we ignored those instructions before, but they did cost $4v$.

We can, however, do even better by adapting Langdon's cyclic-shift method, Algorithm C. Suppose we start with the lexicographically largest permutation and operate as follows:

```
      GREG @
0H    OCTA #fedcba9876543210&(1<<(4*N)-1)
Perm  LDOU a,0B                              Set a ← #...3210.
      JMP  2F
1H    SRU  a,a,4*(16-N)                      a ← ⌊a/16^(16−n)⌋.
      OR   a,a,t                             a ← a ∨ t.              (46)
2H    ⟨Visit the permutation in register a⟩
      SRU  t,a,4*(N-1)                       t ← ⌊a/16^(n−1)⌋.
      SLU  a,a,4*(17-N)                      a ← 16^(17−n) a mod 16^16.
      PBNZ t,1B                              To 1B if t ≠ 0.
      ⟨Continue with Langdon's method⟩
```

The running time per permutation is now only $5v + O(1/n)$, again without the need for PUSHJ and POP. See exercise 81 for an interesting way to extend (46) to a complete program, obtaining a remarkably short and fast routine.

Fast permutation generators are amusing, but in practice we can usually save more time by streamlining the visiting routine than by speeding up the generator.

Topological sorting. Instead of working with all $n!$ permutations of $\{1, \ldots, n\}$, we often want to look only at permutations that obey certain restrictions. For example, we might be interested only in permutations for which 1 precedes 3, 2 precedes 3, and 2 precedes 4; there are five such permutations of $\{1, 2, 3, 4\}$, namely

$$1234,\ 1243,\ 2134,\ 2143,\ 2413. \tag{47}$$

The problem of *topological sorting*, which we studied in Section 2.2.3 as a first example of nontrivial data structures, is the general problem of finding a permutation that satisfies m such conditions $x_1 \prec y_1, \ldots, x_m \prec y_m$, where $x \prec y$ means that x should precede y in the permutation. This problem arises frequently in practice, so it has several different names; for example, it is often called the *linear embedding* problem, because we want to arrange objects in a line while preserving certain order relationships. It is also the problem of extending a partial ordering to a total ordering (see exercise 2.2.3–14).

Our goal in Section 2.2.3 was to find a *single* permutation that satisfies all the relations. But now we want rather to find *all* such permutations, all topological sorts. Indeed, we will assume in the present section that the elements x and y on which the relations are defined are integers between 1 and n, and that we have $x < y$ whenever $x \prec y$. Consequently the permutation $12\ldots n$ will always be topologically correct. (If this simplifying assumption is not met, we can preprocess the data by using Algorithm 2.2.3T to rename the objects appropriately.)

Many important classes of permutations are special cases of this topological ordering problem. For example, the permutations of $\{1, \ldots, 8\}$ such that

$$1 \prec 2, \quad 2 \prec 3, \quad 3 \prec 4, \quad 6 \prec 7, \quad 7 \prec 8$$

are equivalent to permutations of the multiset $\{1, 1, 1, 1, 2, 3, 3, 3\}$, because we can map $\{1, 2, 3, 4\} \mapsto 1$, $5 \mapsto 2$, and $\{6, 7, 8\} \mapsto 3$. We know how to generate permutations of a multiset using Algorithm L, but now we will learn another way.

Notice that x precedes y in a permutation $a_1 \ldots a_n$ if and only if $a'_x < a'_y$ in the inverse permutation $a'_1 \ldots a'_n$. Therefore the algorithm we are about to study will also find all permutations $a'_1 \ldots a'_n$ such that $a'_j < a'_k$ whenever $j \prec k$. For example, we learned in Section 5.1.4 that a Young tableau is an arrangement of $\{1, \ldots, n\}$ in rows and columns so that each row is increasing from left to right and each column is increasing from top to bottom. The problem of generating all 3×3 Young tableaux is therefore equivalent to generating all $a'_1 \ldots a'_9$ such that

$$\begin{aligned} &a'_1 < a'_2 < a'_3, \quad a'_4 < a'_5 < a'_6, \quad a'_7 < a'_8 < a'_9,\\ &a'_1 < a'_4 < a'_7, \quad a'_2 < a'_5 < a'_8, \quad a'_3 < a'_6 < a'_9, \end{aligned} \tag{48}$$

and this is a special kind of topological sorting.

We might also want to find all *matchings* of $2n$ elements, namely all ways to partition $\{1,\ldots,2n\}$ into n pairs. There are $(2n-1)(2n-3)\ldots(1) = (2n)!/(2^n n!)$ ways to do this, and they correspond to permutations that satisfy

$$a'_1 < a'_2, \quad a'_3 < a'_4, \quad \ldots, \quad a'_{2n-1} < a'_{2n}, \qquad a'_1 < a'_3 < \cdots < a'_{2n-1}. \tag{49}$$

An elegant algorithm for exhaustive topological sorting was discovered by Y. L. Varol and D. Rotem [*Comp. J.* **24** (1981), 83–84], who realized that a method analogous to plain changes (Algorithm P) can be used. Suppose we have found a way to arrange $\{1,\ldots,n-1\}$ topologically, so that $a_1\ldots a_{n-1}$ satisfies all the conditions that do not involve n. Then we can easily write down all the allowable ways to insert the final element n without changing the relative order of $a_1\ldots a_{n-1}$: We simply start with $a_1\ldots a_{n-1}n$, then shift n left one step at a time, until it cannot move further. Applying this idea recursively yields the following straightforward procedure.

Algorithm V (*All topological sorts*). Given a relation $\prec$ on $\{1,\ldots,n\}$ with the property that $x \prec y$ implies $x < y$, this algorithm generates all permutations $a_1\ldots a_n$ and their inverses $a'_1\ldots a'_n$ with the property that $a'_j < a'_k$ whenever $j \prec k$. We assume for convenience that $a_0 = 0$ and that $0 \prec k$ for $1 \le k \le n$.

V1. [Initialize.] Set $a_j \leftarrow j$ and $a'_j \leftarrow j$ for $0 \le j \le n$.

V2. [Visit.] Visit the permutation $a_1\ldots a_n$ and its inverse $a'_1\ldots a'_n$. Then set $k \leftarrow n$.

V3. [Can k move left?] Set $j \leftarrow a'_k$ and $l \leftarrow a_{j-1}$. If $l \prec k$, go to V5.

V4. [Yes, move it.] Set $a_{j-1} \leftarrow k$, $a_j \leftarrow l$, $a'_k \leftarrow j-1$, and $a'_l \leftarrow j$. Go to V2.

V5. [No, put k back.] While $j < k$, set $l \leftarrow a_{j+1}$, $a_j \leftarrow l$, $a'_l \leftarrow j$, and $j \leftarrow j+1$. Then set $a_k \leftarrow a'_k \leftarrow k$. Decrease k by 1 and return to V3 if $k > 0$. ▮

For example, Theorem 5.1.4H tells us that there are exactly 42 Young tableaux of size 3×3. If we apply Algorithm V to the relations (48) and write the inverse permutation in array form

$$\boxed{\begin{array}{c} a'_1 a'_2 a'_3 \\ a'_4 a'_5 a'_6 \\ a'_7 a'_8 a'_9 \end{array}}\,, \tag{50}$$

we get the following 42 results:

$$\boxed{\begin{array}{c}123\\456\\789\end{array}}\ \boxed{\begin{array}{c}123\\457\\689\end{array}}\ \boxed{\begin{array}{c}123\\458\\679\end{array}}\ \boxed{\begin{array}{c}123\\467\\589\end{array}}\ \boxed{\begin{array}{c}123\\468\\579\end{array}}\ \boxed{\begin{array}{c}124\\356\\789\end{array}}\ \boxed{\begin{array}{c}124\\357\\689\end{array}}\ \boxed{\begin{array}{c}124\\358\\679\end{array}}\ \boxed{\begin{array}{c}124\\367\\589\end{array}}\ \boxed{\begin{array}{c}124\\368\\579\end{array}}\ \boxed{\begin{array}{c}125\\367\\489\end{array}}\ \boxed{\begin{array}{c}125\\368\\479\end{array}}\ \boxed{\begin{array}{c}125\\346\\789\end{array}}\ \boxed{\begin{array}{c}125\\347\\689\end{array}}$$

$$\boxed{\begin{array}{c}125\\348\\679\end{array}}\ \boxed{\begin{array}{c}126\\347\\589\end{array}}\ \boxed{\begin{array}{c}126\\348\\579\end{array}}\ \boxed{\begin{array}{c}127\\348\\569\end{array}}\ \boxed{\begin{array}{c}126\\357\\489\end{array}}\ \boxed{\begin{array}{c}126\\358\\479\end{array}}\ \boxed{\begin{array}{c}127\\358\\469\end{array}}\ \boxed{\begin{array}{c}134\\256\\789\end{array}}\ \boxed{\begin{array}{c}134\\257\\689\end{array}}\ \boxed{\begin{array}{c}134\\258\\679\end{array}}\ \boxed{\begin{array}{c}134\\267\\589\end{array}}\ \boxed{\begin{array}{c}134\\268\\579\end{array}}\ \boxed{\begin{array}{c}135\\267\\489\end{array}}\ \boxed{\begin{array}{c}135\\268\\479\end{array}}$$

$$\boxed{\begin{array}{c}145\\267\\389\end{array}}\ \boxed{\begin{array}{c}145\\268\\379\end{array}}\ \boxed{\begin{array}{c}135\\246\\789\end{array}}\ \boxed{\begin{array}{c}135\\247\\689\end{array}}\ \boxed{\begin{array}{c}135\\248\\679\end{array}}\ \boxed{\begin{array}{c}136\\247\\589\end{array}}\ \boxed{\begin{array}{c}136\\248\\579\end{array}}\ \boxed{\begin{array}{c}137\\248\\569\end{array}}\ \boxed{\begin{array}{c}136\\257\\489\end{array}}\ \boxed{\begin{array}{c}136\\258\\479\end{array}}\ \boxed{\begin{array}{c}137\\258\\469\end{array}}\ \boxed{\begin{array}{c}146\\257\\389\end{array}}\ \boxed{\begin{array}{c}146\\258\\379\end{array}}\ \boxed{\begin{array}{c}147\\258\\369\end{array}}$$

Let t_r be the number of topological sorts for which the final $n-r$ elements are in their initial position $a_j = j$ for $r < j \le n$. Equivalently, t_r is the number of topological sorts $a_1 \ldots a_r$ of $\{1, \ldots, r\}$, when we ignore the relations involving elements greater than r. Then the recursive mechanism underlying Algorithm V shows that step V2 is performed N times and step V3 is performed M times, where

$$M = t_n + \cdots + t_1 \qquad \text{and} \qquad N = t_n. \tag{51}$$

Also, step V4 and the loop operations of V5 are performed $N-1$ times; the rest of step V5 is done $M-N+1$ times. Therefore the total running time of the algorithm is a linear combination of M, N, and n.

If the element labels are chosen poorly, M might be much larger than N. For example, if the constraints input to Algorithm V are

$$2 \prec 3, \quad 3 \prec 4, \quad \ldots, \quad n-1 \prec n, \tag{52}$$

then $t_j = j$ for $1 \le j \le n$ and we have $M = \frac{1}{2}(n^2+n)$, $N = n$. But those constraints are also equivalent to

$$1 \prec 2, \quad 2 \prec 3, \quad \ldots, \quad n-2 \prec n-1, \tag{53}$$

under renaming of the elements; then M is reduced to $2n-1 = 2N-1$.

Exercise 89 shows that a simple preprocessing step will find element labels so that a slight modification of Algorithm V is able to generate all topological sorts in $O(N+n)$ steps. Thus topological sorting can always be done efficiently.

Think twice before you permute. We have seen several attractive algorithms for permutation generation in this section, but many algorithms are known by which permutations that are optimum for particular purposes can be found *without* running through all possibilities. For example, Theorem 6.1S showed that we can find the best way to arrange records on a sequential storage simply by sorting them with respect to a certain cost criterion, and this process takes only $O(n \log n)$ steps. In Section 7.5.2 we will study the *assignment problem*, which asks how to permute the columns of a square matrix so that the sum of the diagonal elements is maximized. That problem can be solved in at most $O(n^3)$ operations, so it would be foolish to use a method of order $n!$ unless n is extremely small. Even in cases like the traveling salesrep problem, when no efficient algorithm is known, we can usually find a much better approach than to examine every possible solution. Permutation generation is best used when there is good reason to look at each permutation individually.

EXERCISES

▸ **1.** [*20*] Explain how to make Algorithm L run faster, by streamlining its operations when the value of j is near n.

2. [*20*] Rewrite Algorithm L so that it produces all permutations of $a_1 \ldots a_n$ in reverse colex order. (In other words, the values of the reflections $a_n \ldots a_1$ should be lexicographically decreasing, as in (11). This form of the algorithm is often simpler and faster than the original, because fewer calculations depend on the value of n.)

▶ **3.** [*M21*] The *rank* of a combinatorial arrangement X with respect to a generation algorithm is the number of other arrangements that the algorithm visits prior to X. Explain how to compute the rank of a given permutation $a_1 \ldots a_n$ with respect to Algorithm L, if $\{a_1, \ldots, a_n\} = \{1, \ldots, n\}$. What is the rank of 314592687?

4. [*M23*] Generalizing exercise 3, explain how to compute the rank of $a_1 \ldots a_n$ with respect to Algorithm L when $\{a_1, \ldots, a_n\}$ is the multiset $\{n_1 \cdot x_1, \ldots, n_t \cdot x_t\}$; here $n_1 + \cdots + n_t = n$ and $x_1 < \cdots < x_t$. (The total number of permutations is, of course, the multinomial coefficient

$$\binom{n}{n_1, \ldots, n_t} = \frac{n!}{n_1! \ldots n_t!};$$

see Eq. 5.1.2–(3).) What is the rank of 314159265?

5. [*HM25*] Compute the mean and variance of the number of comparisons made by Algorithm L in (a) step L2, (b) step L3, when the elements $\{a_1, \ldots, a_n\}$ are distinct.

6. [*HM34*] Derive generating functions for the mean number of comparisons made by Algorithm L in (a) step L2, (b) step L3, when $\{a_1, \ldots, a_n\}$ is a general multiset as in exercise 4. Also give the results in closed form when $\{a_1, \ldots, a_n\}$ is the binary multiset $\{s \cdot 0, (n-s) \cdot 1\}$.

7. [*HM35*] What is the limit as $t \to \infty$ of the average number of comparisons made per permutation in step L2 when Algorithm L is being applied to the multiset (a) $\{2 \cdot 1,\ 2 \cdot 2,\ \ldots,\ 2 \cdot t\}$? (b) $\{1 \cdot 1,\ 2 \cdot 2,\ \ldots,\ t \cdot t\}$? (c) $\{2 \cdot 1,\ 4 \cdot 2,\ \ldots,\ 2^t \cdot t\}$?

▶ **8.** [*21*] The *variations* of a multiset are the permutations of all its submultisets. For example, the variations of $\{1, 2, 2, 3\}$ are

ϵ, 1, 12, 122, 1223, 123, 1232, 13, 132, 1322,
2, 21, 212, 2123, 213, 2132, 22, 221, 2213, 223, 2231, 23, 231, 2312, 232, 2321,
3, 31, 312, 3122, 32, 321, 3212, 322, 3221.

Show that simple changes to Algorithm L will generate all variations of a given multiset $\{a_1, a_2, \ldots, a_n\}$.

9. [*22*] Continuing the previous exercise, design an algorithm to generate all r-variations of a given multiset $\{a_1, a_2, \ldots, a_n\}$, also called its r-permutations, namely all permutations of its r-element submultisets. (For example, the solution to an alphametic with r distinct letters is an r-variation of $\{0, 1, \ldots, 9\}$.)

10. [*20*] What are the values of $a_1a_2 \ldots a_n$, $c_1c_2 \ldots c_n$, and $o_1o_2 \ldots o_n$ at the end of Algorithm P, if $a_1a_2 \ldots a_n = 12 \ldots n$ at the beginning?

11. [*M22*] How many times is each step of Algorithm P performed? (Assume that $n \geq 2$.)

▶ **12.** [*M23*] What is the 1000000th permutation visited by (a) Algorithm L, (b) Algorithm P, (c) Algorithm C, if $\{a_1, \ldots, a_n\} = \{0, \ldots, 9\}$? *Hint:* In mixed-radix notation we have $1000000 = \left[\begin{smallmatrix} 2, & 6, & 6, & 2, & 5, & 1, & 2, & 2, & 0, & 0 \\ 10, & 9, & 8, & 7, & 6, & 5, & 4, & 3, & 2, & 1 \end{smallmatrix} \right] = \left[\begin{smallmatrix} 0, & 0, & 1, & 2, & 3, & 0, & 2, & 7, & 1, & 0 \\ 1, & 2, & 3, & 4, & 5, & 6, & 7, & 8, & 9, & 10 \end{smallmatrix} \right]$.

13. [*M21*] (Martin Gardner, 1974.) True or false: If $a_1a_2 \ldots a_n$ is initially $12 \ldots n$, Algorithm P begins by visiting all $n!/2$ permutations in which 1 precedes 2; then the next permutation is $n \ldots 21$.

14. [*M22*] True or false: If $a_1a_2 \ldots a_n$ is initially $x_1x_2 \ldots x_n$ in Algorithm P, we always have $a_{j-c_j+s} = x_j$ at the beginning of step P5.

15. [*M23*] (Selmer Johnson, 1963.) Show that the offset variable s never exceeds 2 in Algorithm P.

16. [*21*] Explain how to make Algorithm P run faster, by streamlining its operations when the value of j is near n. (This problem is analogous to exercise 1.)

▸ **17.** [*20*] Extend Algorithm P so that the *inverse permutation* $a'_1 \ldots a'_n$ is available for processing when $a_1 \ldots a_n$ is visited in step P2. (The inverse satisfies $a'_k = j$ if and only if $a_j = k$.)

18. [*21*] (*Rosary permutations.*) Devise an efficient way to generate $(n-1)!/2$ permutations that represent all possible undirected cycles on the vertices $\{1, \ldots, n\}$; that is, no cyclic shift of $a_1 \ldots a_n$ or $a_n \ldots a_1$ will be generated if $a_1 \ldots a_n$ is generated. The permutations (1234, 1324, 3124) could, for example, be used when $n = 4$.

19. [*25*] Construct an algorithm that generates all permutations of n distinct elements *looplessly* in the spirit of Algorithm 7.2.1.1L.

▸ **20.** [*20*] The n-cube has $2^n n!$ symmetries, one for each way to permute and/or complement the coordinates. Such a symmetry is conveniently represented as a *signed permutation*, namely a permutation with optional signs attached to the elements. For example, $23\bar{1}$ is a signed permutation that transforms the vertices of the 3-cube by changing $x_1x_2x_3$ to $x_2x_3\bar{x}_1$, so that $000 \mapsto 001$, $001 \mapsto 011$, $\ldots$, $111 \mapsto 110$. Design a simple algorithm that generates all signed permutations of $\{1, 2, \ldots, n\}$, where each step either interchanges two adjacent elements or negates the first element.

21. [*M21*] (E. P. McCravy, 1971.) How many solutions does the alphametic (6) have in radix b?

22. [*M15*] True or false: If an alphametic has a solution in radix b, it has a solution in radix $b + 1$.

23. [*M20*] True or false: A pure alphametic cannot have two identical signatures $s_j = s_k \neq 0$ when $j \neq k$.

24. [*25*] Solve the following alphametics by hand or by computer:

a) SEND + A + TAD + MORE = MONEY.

b) ZEROES + ONES = BINARY. (Peter MacDonald, 1977)

c) DCLIX + DLXVI = MCCXXV. (Willy Enggren, 1972)

d) COUPLE + COUPLE = QUARTET. (Michael R. W. Buckley, 1977)

e) FISH + N + CHIPS = SUPPER. (Bob Vinnicombe, 1978)

f) SATURN + URANUS + NEPTUNE + PLUTO = PLANETS. (Willy Enggren, 1968)

g) EARTH + AIR + FIRE + WATER = NATURE. (Herman Nijon, 1977)

h) AN + ACCELERATING + INFERENTIAL + ENGINEERING + TALE + ELITE + GRANT + FEE + ET + CETERA = ARTIFICIAL + INTELLIGENCE.

i) HARDY + NESTS = NASTY + HERDS.

▸ **25.** [*M21*] Devise a fast way to compute $\min(a \cdot s)$ and $\max(a \cdot s)$ over all valid permutations $a_1 \ldots a_{10}$ of $\{0, \ldots, 9\}$, given the signature vector $s = (s_1, \ldots, s_{10})$ and the first-letter set F of an alphametic problem. (Such a procedure makes it possible to rule out many cases quickly when a large family of alphametics is being considered, as in several of the exercises that follow, because a solution can exist only when $\min(a \cdot s) \leq 0 \leq \max(a \cdot s)$.)

26. [*25*] What is the unique alphametic solution to

$$\texttt{NIIHAU} \pm \texttt{KAUAI} \pm \texttt{OAHU} \pm \texttt{MOLOKAI} \pm \texttt{LANAI} \pm \texttt{MAUI} \pm \texttt{HAWAII} = 0?$$

27. [*30*] Construct pure additive alphametics in which all words have five letters.

28. [*M25*] A *partition* of the integer n is an expression of the form $n = n_1 + \cdots + n_t$ with $n_1 \geq \cdots \geq n_t > 0$. Such a partition is called *doubly true* if $\alpha(n) = \alpha(n_1) + \cdots + \alpha(n_t)$ is also a pure alphametic, where $\alpha(n)$ is the "name" of n in some language. Doubly true partitions were introduced by Alan Wayne in *AMM* **54** (1947), 38, 412–414, where he suggested solving $\texttt{TWENTY} = \texttt{SEVEN} + \texttt{SEVEN} + \texttt{SIX}$ and a few others.

a) Find all partitions that are doubly true in English when $1 \leq n \leq 20$.

b) Wayne also gave the example $\texttt{EIGHTY} = \texttt{FIFTY} + \texttt{TWENTY} + \texttt{NINE} + \texttt{ONE}$. Find all doubly true partitions for $1 \leq n \leq 100$ in which the parts are *distinct*, using the names $\texttt{ONE}$, $\texttt{TWO}$, ..., $\texttt{NINETYNINE}$, $\texttt{ONEHUNDRED}$.

▶ **29.** [*M25*] Continuing the previous exercise, find all equations of the form $n_1 + \cdots + n_t = n'_1 + \cdots + n'_{t'}$ that are both mathematically and alphametically true in English, when $\{n_1, \ldots, n_t, n'_1, \ldots, n'_{t'}\}$ are distinct positive integers less than 20. For example,

$$\texttt{TWELVE} + \texttt{NINE} + \texttt{TWO} = \texttt{ELEVEN} + \texttt{SEVEN} + \texttt{FIVE};$$

the alphametics should all be pure.

30. [*25*] Solve these multiplicative alphametics by hand or by computer:

a) $\texttt{TWO} \times \texttt{TWO} = \texttt{SQUARE}$. (H. E. Dudeney, 1929)

b) $\texttt{HIP} \times \texttt{HIP} = \texttt{HURRAY}$. (Willy Enggren, 1970)

c) $\texttt{PI} \times \texttt{R} \times \texttt{R} = \texttt{AREA}$. (Brian Barwell, 1981)

d) $\texttt{NORTH}/\texttt{SOUTH} = \texttt{EAST}/\texttt{WEST}$. (Nob Yoshigahara, 1995)

e) $\texttt{NAUGHT} \times \texttt{NAUGHT} = \texttt{ZERO} \times \texttt{ZERO} \times \texttt{ZERO}$. (Alan Wayne, 2003)

31. [*M22*] (Nob Yoshigahara.) What is the unique solution to $\texttt{A}/\texttt{BC} + \texttt{D}/\texttt{EF} + \texttt{G}/\texttt{HI} = 1$, when $\{\texttt{A}, \ldots, \texttt{I}\} = \{1, \ldots, 9\}$?

32. [*M25*] (H. E. Dudeney, 1901.) Find all ways to represent 100 by inserting a plus sign and a slash into a permutation of the digits $\{1, \ldots, 9\}$. For example, $100 = 91 + 5742/638$. The plus sign should precede the slash.

33. [*25*] Continuing the previous exercise, find all positive integers less than 150 that (a) cannot be represented in such a fashion; (b) have a unique representation.

34. [*M26*] Make the equation $\texttt{EVEN} + \texttt{ODD} + \texttt{PRIME} = x$ doubly true when (a) x is a perfect 5th power; (b) x is a perfect 7th power.

▶ **35.** [*M20*] The automorphisms of a 4-cube have many different Sims tables, only one of which is shown in (14). How many different Sims tables are possible for that group, when the vertices are numbered as in (12)?

36. [*M23*] Find a Sims table for the group of all automorphisms of the 4×4 tic-tac-toe board

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

namely the permutations that take lines into lines, where a "line" is a set of four elements that belong to a row, column, or diagonal.

▶ **37.** [*HM22*] How many Sims tables can be used with Algorithms G or H? Estimate the logarithm of this number as $n \to \infty$.

38. [*HM21*] Prove that the average number of transpositions per permutation when using Ord-Smith's algorithm (26) is approximately $\sinh 1 \approx 1.175$.

39. [*16*] Write down the 24 permutations generated for $n = 4$ by (a) Ord-Smith's method (26); (b) Heap's method (27).

40. [*M23*] Show that Heap's method (27) corresponds to a valid Sims table.

▶ **41.** [*M33*] Design an algorithm that generates all r-variations of $\{0, 1, \ldots, n-1\}$ by interchanging just two elements when going from one variation to the next. (See exercise 9.) *Hint:* Generalize Heap's method (27), obtaining the results in positions $a_{n-r} \ldots a_{n-1}$ of an array $a_0 \ldots a_{n-1}$. For example, one solution when $n = 5$ and $r = 2$ uses the final two elements of the respective permutations 01234, 31204, 30214, 30124, 40123, 20143, 24103, 24013, 34012, 14032, 13042, 13402, 23401, 03421, 02431, 02341, 12340, 42310, 41320, 41230.

42. [*M20*] Construct a Sims table for all permutations in which every $\sigma(k,j)$ and every $\tau(k,j)$ for $1 \le j \le k$ is a cycle of length ≤ 3.

43. [*M24*] Construct a Sims table for all permutations in which every $\sigma(k,k)$, $\omega(k)$, and $\tau(k,j)\,\omega(k-1)^-$ for $1 \le j \le k$ is a cycle of length ≤ 3.

44. [*20*] When blocks of unwanted permutations are being skipped by the extended Algorithm G, is the Sims table of Ord-Smith's method (23) superior to the Sims table of the reverse colex method (18)?

45. [*20*] (a) What are the indices $u_1 \ldots u_9$ when Algorithm X visits the permutation 314592687? (b) What permutation is visited when $u_1 \ldots u_9 = 314157700$?

46. [*20*] True or false: When Algorithm X visits $a_1 \ldots a_n$, we have $u_k > u_{k+1}$ if and only if $a_k > a_{k+1}$, for $1 \le k < n$.

▶ **47.** [*M21*] Express the number of times that each step of Algorithm X is performed in terms of the numbers N_0, N_1, ..., N_n, where N_k is the number of prefixes $a_1 \ldots a_k$ that satisfy $t_j(a_1, \ldots, a_j)$ for $1 \le j \le k$.

▶ **48.** [*M25*] Compare the running times of Algorithm X and Algorithm L, in the case when the tests $t_1(a_1)$, $t_2(a_1, a_2)$, ..., $t_n(a_1, a_2, \ldots, a_n)$ always are true.

▶ **49.** [*28*] The text's suggested method for solving additive alphametics with Algorithm X essentially chooses digits from right to left; in other words, it assigns tentative values to the least significant digits before considering digits that correspond to higher powers of 10.

Explore an alternative approach that chooses digits from left to right. For example, such a method will deduce immediately that $\mathtt{M} = 1$ when $\mathtt{SEND} + \mathtt{MORE} = \mathtt{MONEY}$. *Hint:* See exercise 25.

50. [*M15*] Explain why the dual formula (32) follows from (13).

51. [*M16*] True or false: If the sets $S_k = \{\sigma(k,0), \ldots, \sigma(k,k)\}$ form a Sims table for the group of all permutations, so also do the sets $S_k^- = \{\sigma(k,0)^-, \ldots, \sigma(k,k)^-\}$.

▶ **52.** [*M22*] What permutations $\tau(k,j)$ and $\omega(k)$ arise when Algorithm H is used with the Sims table (36)? Compare the resulting generator with Algorithm P.

▶ **53.** [*M26*] (F. M. Ives.) Construct a Sims table for which Algorithm H will generate all permutations by making only $n! + O\bigl((n-2)!\bigr)$ transpositions.

54. [*20*] Would Algorithm C work properly if step C3 did a right-cyclic shift, setting $a_1 \ldots a_{k-1}a_k \leftarrow a_k a_1 \ldots a_{k-1}$, instead of a left-cyclic shift?

55. [*M27*] Consider the *factorial ruler function*

$$\rho_!(m) = \max\{k \mid m \bmod k! = 0\}.$$

Let σ_k and τ_k be permutations of the nonnegative integers such that $\sigma_j\tau_k = \tau_k\sigma_j$ whenever $j \le k$. Let α_0 and β_0 be the identity permutation, and for $m > 0$ define

$$\alpha_m = \beta_{m-1}^{-}\tau_{\rho_!(m)}\beta_{m-1}\alpha_{m-1}, \qquad \beta_m = \sigma_{\rho_!(m)}\beta_{m-1}.$$

For example, if σ_k is the flip operation $(1\ k{-}1)(2\ k{-}2)\ldots = (0\ k)\phi(k)$ and if $\tau_k = (0\ k)$, and if Algorithm E is started with $a_j = j$ for $0 \le j < n$, then α_m and β_m are the contents of $a_0 \ldots a_{n-1}$ and $b_0 \ldots b_{n-1}$ after step E5 has been performed m times.

a) Prove that $\beta_{(n+1)!}\alpha_{(n+1)!} = \sigma_{n+1}\sigma_n^{-}\tau_{n+1}\tau_n^{-}(\beta_{n!}\alpha_{n!})^{n+1}$.

b) Use the result of (a) to establish the validity of Algorithm E.

56. [*M22*] Prove that Algorithm E remains valid if step E5 is replaced by

E5′. [Transpose pairs.] If $k > 2$, interchange $b_{j+1} \leftrightarrow b_j$ for $j = k-2,\ k-4,\ \ldots,$ (2 or 1). Return to E2. ▌

57. [*HM22*] What is the average number of interchanges made in step E5?

58. [*M21*] True or false: If Algorithm E begins with $a_0 \ldots a_{n-1} = x_1 \ldots x_n$ then the final permutation visited begins with $a_0 = x_n$.

59. [*M20*] Some authors define the arcs of a Cayley graph as running from π to $\pi\alpha_j$ instead of from π to $\alpha_j\pi$. Are the two definitions essentially different?

▸ **60.** [*21*] A *Gray cycle for permutations* is a cycle $(\pi_0, \pi_1, \ldots, \pi_{n!-1})$ that includes every permutation of $\{1, 2, \ldots, n\}$ and has the property that π_k differs from $\pi_{(k+1) \bmod n!}$ by an adjacent transposition. It can also be described as a Hamiltonian cycle on the Cayley graph for the group of all permutations on $\{1, 2, \ldots, n\}$, with the $n{-}1$ generators $((1\ 2), (2\ 3), \ldots, (n{-}1\ n))$. The *delta sequence* of such a Gray cycle is the sequence of integers $\delta_0\delta_1 \ldots \delta_{n!-1}$ such that

$$\pi_{(k+1) \bmod n!} = (\delta_k\ \ \delta_k{+}1)\,\pi_k.$$

(See 7.2.1.1–(24), which describes the analogous situation for binary n-tuples.) For example, Fig. 23 illustrates the Gray cycle defined by plain changes when $n = 4$; its delta sequence is $(32131231)^3$.

a) Find all Gray cycles for permutations of $\{1, 2, 3, 4\}$.

b) Two Gray cycles are considered to be equivalent if their delta sequences can be obtained from each other by cyclic shifting $(\delta_k \ldots \delta_{n!-1}\delta_0 \ldots \delta_{k-1})$ and/or reversal $(\delta_{n!-1} \ldots \delta_1\delta_0)$ and/or complementation $((n{-}\delta_0)(n{-}\delta_1) \ldots (n{-}\delta_{n!-1}))$. Which of the Gray cycles in (a) are equivalent?

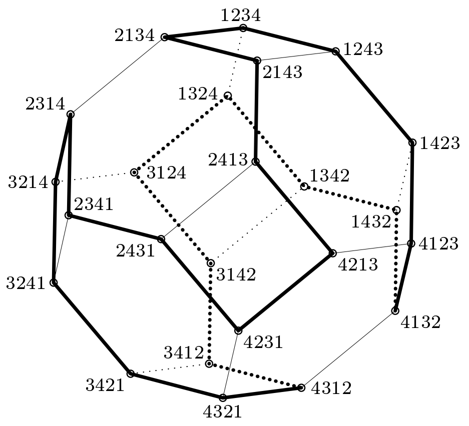


Fig. 23. Algorithm P traces out this Hamiltonian cycle on the truncated octahedron of Fig. 5–1.

61. [*21*] Continuing the previous exercise, a *Gray code for permutations* is like a Gray cycle except that the final permutation $\pi_{n!-1}$ is not required to be adjacent to the initial permutation π_0. Study the set of all Gray codes for $n = 4$ that start with 1234.

▶ **62.** [*M23*] What permutations can be reached as the final element of a Gray code that starts at $12\ldots n$?

63. [*M25*] Estimate the total number of Gray cycles for permutations of $\{1, 2, 3, 4, 5\}$.

64. [*23*] A "doubly Gray" code for permutations is a Gray cycle with the additional property that $\delta_{k+1} = \delta_k \pm 1$ for all k. Compton and Williamson have proved that such codes exist for all $n \geq 3$. How many doubly Gray codes exist for $n = 5$?

65. [*M25*] For which integers N is there a Gray path through the N lexicographically smallest permutations of $\{1, \ldots, n\}$? (Exercise 7.2.1.1–26 solves the analogous problem for binary n-tuples.)

66. [*22*] Ehrlich's swap method suggests another type of Gray cycle for permutations, in which the $n - 1$ generators are the star transpositions $(1\ 2)$, $(1\ 3)$, $\ldots$, $(1\ n)$. For example, Fig. 24 shows the relevant graph when $n = 4$. Analyze the Hamiltonian cycles of this graph.

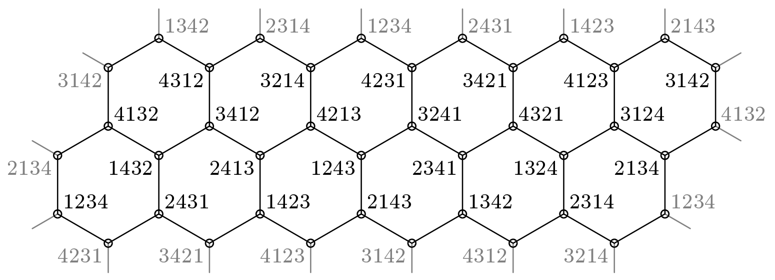


Fig. 24. The Cayley graph for permutations of $\{1, 2, 3, 4\}$, generated by the star transpositions $(1\,2)$, $(1\,3)$, and $(1\,4)$, drawn as a twisted torus.

67. [*26*] Continuing the previous exercise, find a first-element-swap Gray cycle for $n = 5$ in which each star transposition $(1\ j)$ occurs 30 times, for $2 \leq j \leq 5$.

68. [*M30*] (Kompel'makher and Liskovets, 1975.) Let G be the Cayley graph for all permutations of $\{1, \ldots, n\}$, with generators $(\alpha_1, \ldots, \alpha_k)$ where each α_j is a transposition $(u_j\ v_j)$; also let A be the graph with vertices $\{1, \ldots, n\}$ and edges u_j — v_j for $1 \leq j \leq k$. Prove that G has a Hamiltonian cycle if and only if A is connected. (Figure 23 is the special case when A is a path; Figure 24 is the special case when A is a "star.")

▶ **69.** [*28*] If $n \geq 4$, the following algorithm generates all permutations $A_1A_2A_3\ldots A_n$ of $\{1, 2, 3, \ldots, n\}$ using only three transformations,

$$\rho = (1\,2)(3\,4)(5\,6)\ldots, \qquad \sigma = (2\,3)(4\,5)(6\,7)\ldots, \qquad \tau = (3\,4)(5\,6)(7\,8)\ldots,$$

never applying ρ and τ next to each other. Explain why it works.

Z1. [Initialize.] Set $A_j \leftarrow j$ for $1 \leq j \leq n$. Also set $a_j \leftarrow 2j$ for $j \leq n/2$ and $a_{n-j} \leftarrow 2j + 1$ for $j < n/2$. Then invoke Algorithm P, but with parameter $n - 1$ instead of n. We will treat that algorithm as a coroutine, which should return control to us whenever it "visits" $a_1 \ldots a_{n-1}$ in step P2. We will also share its variables (except n).

Z2. [Set x and y.] Invoke Algorithm P again, obtaining a new permutation $a_1 \ldots a_{n-1}$ and a new value of j. If $j = 2$, interchange $a_{1+s} \leftrightarrow a_{2+s}$ (thereby undoing the effect of step P5) and repeat this step; in such a case we are at the halfway point of Algorithm P. If $j = 1$ (so that Algorithm P has terminated), set $x \leftarrow y \leftarrow 0$ and go to Z3. Otherwise set

$$x \leftarrow a_{j-c_j+s+[o_j=-1]}, \qquad y \leftarrow a_{j-c_j+s-[o_j=+1]};$$

these are the two elements most recently interchanged in step P5.

Z3. [Visit.] Visit the permutation $A_1 \ldots A_n$. Then go to Z5 if $A_1 = x$ and $A_2 = y$.

Z4. [Apply ρ, then σ.] Interchange $A_1 \leftrightarrow A_2$, $A_3 \leftrightarrow A_4$, $A_5 \leftrightarrow A_6$, ... Visit $A_1 \ldots A_n$. Then interchange $A_2 \leftrightarrow A_3$, $A_4 \leftrightarrow A_5$, $A_6 \leftrightarrow A_7$, ... Terminate if $A_1 \ldots A_n = 1 \ldots n$, otherwise return to Z3.

Z5. [Apply τ, then σ.] Interchange $A_3 \leftrightarrow A_4$, $A_5 \leftrightarrow A_6$, $A_7 \leftrightarrow A_8$, ... Visit $A_1 \ldots A_n$. Then interchange $A_2 \leftrightarrow A_3$, $A_4 \leftrightarrow A_5$, $A_6 \leftrightarrow A_7$, ..., and return to Z2. ▮

Hint: Show first that the algorithm works if modified so that $A_j \leftarrow n+1-j$ and $a_j \leftarrow j$ in step Z1, and if the "flip" permutations

$$\rho' = (1\ n)(2\ n{-}1)\ldots, \qquad \sigma' = (2\ n)(3\ n{-}1)\ldots, \qquad \tau' = (2\ n{-}1)(3\ n{-}2)\ldots$$

are used instead of ρ, σ, τ in steps Z4 and Z5. In this modification, step Z3 should go to Z5 if $A_1 = x$ and $A_n = y$.

▸ **70.** [*M33*] The two 12-cycles (41) can be regarded as σ–τ cycles for the twelve permutations of $\{1,1,3,4\}$:

$$1134 \to 1341 \to 3411 \to 4311 \to 3114 \to 1143 \to 1431$$
$$\to 4131 \to 1314 \to 3141 \to 1413 \to 4113 \to 1134.$$

Replacing $\{1,1\}$ by $\{1,2\}$ yields disjoint cycles, and we obtained a Hamiltonian path by jumping from one to the other. Can a σ–τ path for all permutations of 6 elements be formed in a similar way, based on a 360-cycle for the permutations of $\{1,1,3,4,5,6\}$?

71. [*48*] Does the Cayley graph with generators $\sigma = (1\,2\,\ldots\,n)$ and $\tau = (1\,2)$ have a Hamiltonian cycle whenever $n \ge 3$ is odd?

72. [*M21*] Given a Cayley graph with generators $(\alpha_1, \ldots, \alpha_k)$, assume that each α_j takes $x \mapsto y$. (For example, both σ and τ in exercise 71 take $1 \mapsto 2$.) Prove that any Hamiltonian path starting at $12\ldots n$ in G must end at a permutation that takes $y \mapsto x$.

▸ **73.** [*M30*] Let α, β, and σ be permutations of a set X, where $X = A \cup B$. Assume that $x\sigma = x\alpha$ when $x \in A$ and $x\sigma = x\beta$ when $x \in B$, and that the order of $\alpha\beta^-$ is odd.

a) Prove that all three permutations α, β, σ have the same sign; that is, they are all even or all odd. *Hint:* A permutation has odd order if and only if its cycles all have odd length.

b) Derive Theorem R from part (a).

74. [*M30*] (R. A. Rankin.) Assuming that $\alpha\beta = \beta\alpha$ in Theorem R, prove that a Hamiltonian cycle exists if and only if there is a number k such that $0 \le k \le g/c$ and $t + k \perp c$, where $\beta^{g/c} = \gamma^t$, $\gamma = \alpha\beta^-$. *Hint:* Represent elements of the group in the form $\beta^j\gamma^k$.

75. [*M25*] The directed torus $C_m \times C_n$ has mn vertices (x, y) for $0 \le x < m$, $0 \le y < n$, and arcs $(x, y) \to (x, y)\alpha = ((x+1) \bmod m, y)$, $(x, y) \to (x, y)\beta = (x, (y+1) \bmod n)$. Prove that, if $m > 1$ and $n > 1$, the number of Hamiltonian cycles of this digraph is

$$\sum_{k=1}^{d-1} \binom{d}{k} [\gcd((d-k)m, kn) = d], \qquad d = \gcd(m, n).$$

76. [*M31*] The cells numbered 0, 1, ..., 63 in Fig. 25 illustrate a *northeasterly knight's tour* on an 8×8 torus: If k appears in cell (x_k, y_k), then $(x_{k+1}, y_{k+1}) = (x_k+2, y_k+1)$ or (x_k+1, y_k+2), modulo 8, and $(x_{64}, y_{64}) = (x_0, y_0)$. How many such tours are possible on an $m \times n$ torus, when $m, n \ge 3$?

29	24	19	14	49	44	39	34
58	53	48	43	38	9	4	63
23	18	13	8	3	62	33	28
52	47	42	37	32	27	22	57
17	12	7	2	61	56	51	46
6	41	36	31	26	21	16	11
35	30	1	60	55	50	45	40
0	59	54	25	20	15	10	5

Fig. 25. A northeasterly knight's tour.

▶ **77.** [*22*] Complete the `MMIX` program whose inner loop appears in (42), using Heap's method (27).

78. [*M23*] Analyze the running time of the program in exercise 77, generalizing it so that the inner loop does $r!$ visits (with $a_0 \ldots a_{r-1}$ in global registers).

79. [*20*] What seven `MMIX` instructions will ⟨Swap the nybbles ... ⟩ as (45) requires? For example, if register `t` contains the value 4 and register `a` contains the nybbles #`12345678`, register `a` should change to #`12345687`.

80. [*21*] Solve the previous exercise with only five `MMIX` instructions. *Hint:* Use `MXOR`.

▶ **81.** [*22*] Complete the `MMIX` program (46) by specifying how to ⟨Continue with Langdon's method⟩.

82. [*M21*] Analyze the running time of the program in exercise 81.

83. [*22*] Use the σ–τ path of exercise 70 to design an `MMIX` routine analogous to (42) that generates all permutations of #`123456` in register `a`.

84. [*20*] Suggest a good way to generate all $n!$ permutations of $\{1, \ldots, n\}$ on p processors that are running in parallel.

▶ **85.** [*25*] Assume that n is small enough that $n!$ fits in a computer word. What's a good way to convert a given permutation $\alpha = a_1 \ldots a_n$ of $\{1, \ldots, n\}$ into an integer $k = r(\alpha)$ in the range $0 \le k < n!$? Both functions $k = r(\alpha)$ and $\alpha = r^{[-1]}(k)$ should be computable in only $O(n)$ steps.

86. [*20*] A partial order relation is supposed to be transitive; that is, $x \prec y$ and $y \prec z$ should imply $x \prec z$. But Algorithm V does not require its input relation to satisfy this condition.

Show that if $x \prec y$ and $y \prec z$, Algorithm V will produce identical results whether or not $x \prec z$.

87. [*20*] (F. Ruskey.) Consider the inversion tables $c_1 \ldots c_n$ of the permutations visited by Algorithm V. What noteworthy property do they have? (Compare with the inversion tables (4) in Algorithm P.)

88. [*21*] Show that Algorithm V can be used to generate all ways to partition the digits $\{0,1,\ldots,9\}$ into two 3-element sets and two 2-element sets.

▶ **89.** [*M30*] Consider the numbers t_0, t_1, ..., t_n in (51). Clearly $t_0 = t_1 = 1$.

a) Say that index j is "trivial" if $t_j = t_{j-1}$. For example, 9 is trivial with respect to the Young tableau relations (48). Explain how to modify Algorithm V so that the variable k takes on only nontrivial values.

b) Analyze the running time of the modified algorithm. What formulas replace (51)?

c) Say that the interval $[j\,..\,k]$ is not a chain if we do not have $l \prec l+1$ for $j \le l < k$. Prove that in such a case $t_k \ge 2t_{j-1}$.

d) Every inverse topological sort $a'_1 \ldots a'_n$ defines a labeling that corresponds to relations $a'_{j_1} \prec a'_{k_1}$, ..., $a'_{j_m} \prec a'_{k_m}$, which are equivalent to the original relations $j_1 \prec k_1$, ..., $j_m \prec k_m$. Explain how to find a labeling such that $[j\,..\,k]$ is not a chain when j and k are consecutive nontrivial indices.

e) Prove that with such a labeling, $M < 4N$ in the formulas of part (b).

90. [*M21*] Algorithm V can be used to produce all permutations that are h-ordered for all h in a given set, namely all $a'_1 \ldots a'_n$ such that $a'_j < a'_{j+h}$ for $1 \le j \le n-h$ (see Section 5.2.1). Analyze the running time of Algorithm V when it generates all permutations that are both 2-ordered and 3-ordered.

91. [*HM21*] Analyze the running time of Algorithm V when it is used with the relations (49) to find matchings.

92. [*M18*] How many permutations is Algorithm V likely to visit, in a "random" case? Let P_n be the number of partial orderings on $\{1,\ldots,n\}$, namely the number of relations that are reflexive, antisymmetric, and transitive. Let Q_n be the number of such relations with the additional property that $j < k$ whenever $j \prec k$. Express the expected number of ways to sort n elements topologically, averaged over all partial orderings, in terms of P_n and Q_n.

93. [*35*] Prove that all topological sorts can be generated in such a way that only one or two adjacent transpositions are made at each step. (The example $1 \prec 2$, $3 \prec 4$ shows that a single transposition per step cannot always be achieved, even if we allow nonadjacent swaps, because only two of the six relevant permutations are odd.)

▶ **94.** [*25*] Show that in the case of matchings, using the relations in (49), all topological sorts can be generated with just one transposition per step.

95. [*21*] Discuss how to generate all *up-down permutations* of $\{1,\ldots,n\}$, namely those $a_1 \ldots a_n$ such that $a_1 < a_2 > a_3 < a_4 > \cdots$.

96. [*21*] Discuss how to generate all *cyclic permutations* of $\{1,\ldots,n\}$, namely those $a_1 \ldots a_n$ whose cycle representation consists of a single n-cycle.

97. [*21*] Discuss how to generate all *derangements* of $\{1,\ldots,n\}$, namely those $a_1 \ldots a_n$ such that $a_1 \ne 1$, $a_2 \ne 2$, $a_3 \ne 3$, ...

98. [*HM23*] Analyze the asymptotic running time of the method in the previous exercise.

99. [*M30*] Given $n \ge 3$, show that all derangements of $\{1,\ldots,n\}$ can be generated by making at most two transpositions between visits.

100. [*21*] Discuss how to generate all *indecomposable* permutations of $\{1,\ldots,n\}$, namely those $a_1 \ldots a_n$ such that $\{a_1,\ldots,a_j\} \ne \{1,\ldots,j\}$ for $1 \le j < n$.

101. [*21*] Discuss how to generate all *involutions* of $\{1,\ldots,n\}$, namely those permutations $a_1 \ldots a_n$ with $a_{a_1} \ldots a_{a_n} = 1 \ldots n$.

102. [*M30*] Show that all involutions of $\{1,\ldots,n\}$ can be generated by making at most two transpositions between visits.

103. [*M32*] Show that all even permutations of $\{1,\ldots,n\}$ can be generated by successive *rotations of three consecutive elements.*

▸ **104.** [*M22*] A permutation $a_1\ldots a_n$ of $\{1,\ldots,n\}$ is *well-balanced* if

$$\sum_{k=1}^{n} k a_k = \sum_{k=1}^{n} (n+1-k) a_k .$$

For example, 3142 is well-balanced when $n=4$.

a) Prove that no permutation is well-balanced when $n \bmod 4 = 2$.

b) Prove that if $a_1\ldots a_n$ is well-balanced, so are its reversal $a_n\ldots a_1$, its complement $(n{+}1{-}a_1)\ldots(n{+}1{-}a_n)$, and its inverse $a'_1\ldots a'_n$.

c) Determine the number of well-balanced permutations for small values of n.

▸ **105.** [*26*] A *weak order* is a relation $\preceq$ that is transitive ($x \preceq y$ and $y \preceq z$ implies $x \preceq z$) and complete ($x \preceq y$ or $y \preceq x$ always holds). We can write $x \equiv y$ if $x \preceq y$ and $y \preceq x$; $x \prec y$ if $x \preceq y$ and $y \not\preceq x$. There are thirteen weak orders on three elements $\{1,2,3\}$, namely

$$1\equiv2\equiv3,\quad 1\equiv2\prec3,\quad 1\prec2\equiv3,\quad 1\prec2\prec3,\quad 1\equiv3\prec2,\quad 1\prec3\prec2,$$
$$2\prec1\equiv3,\quad 2\prec1\prec3,\quad 2\equiv3\prec1,\quad 2\prec3\prec1,\quad 3\prec1\equiv2,\quad 3\prec1\prec2,\quad 3\prec2\prec1.$$

a) Explain how to generate all weak orders of $\{1,\ldots,n\}$ systematically, as sequences of digits separated by the symbols $\equiv$ or $\prec$.

b) A weak order can also be represented as a sequence $a_1\ldots a_n$ where $a_j=k$ if j is preceded by k $\prec$ signs. For example, the thirteen weak orders on $\{1,2,3\}$ are respectively 000, 001, 011, 012, 010, 021, 101, 102, 100, 201, 110, 120, 210 in this form. Find a simple way to generate all such sequences of length n.

106. [*M40*] Can exercise 105(b) be solved with a Gray-like code?

▸ **107.** [*30*] (John H. Conway, 1973.) To play the solitaire game of "topswops," start by shuffling a pack of n cards labeled $\{1,\ldots,n\}$ and place them face up in a pile. Then if the top card is $k>1$, deal out the top k cards and put them back on top of the pile, thereby changing the permutation from $a_1\ldots a_n$ to $a_k\ldots a_1a_{k+1}\ldots a_n$. Continue until the top card is 1. For example, the 7-step sequence

$$31452 \to 41352 \to 53142 \to 24135 \to 42135 \to 31245 \to 21345 \to 12345$$

might occur when $n=5$. What is the longest sequence possible when $n=13$?

108. [*M27*] If the longest n-card game of topswops has length $f(n)$, prove that $f(n) \le F_{n+1}-1$.

109. [*M47*] Find good upper and lower bounds on the topswops function $f(n)$.

▸ **110.** [*25*] Find all permutations $a_0\ldots a_9$ of $\{0,\ldots,9\}$ such that

$$\begin{aligned}\{a_0,a_2,a_3,a_7\}&=\{2,5,7,8\},\\ \{a_1,a_4,a_5\}&=\{0,3,6\},\\ \{a_1,a_3,a_7,a_8\}&=\{3,4,5,7\},\\ \{a_0,a_3,a_4\}&=\{0,7,8\}.\end{aligned}$$

Also suggest an algorithm for solving large problems of this type.

▸ **111.** [*M25*] Several permutation-oriented analogs of de Bruijn cycles have been proposed. The simplest and nicest of these is the notion of a *universal cycle of permutations*, introduced by B. W. Jackson in *Discrete Math.* **117** (1993), 141–150, namely a cycle of $n!$ digits such that each permutation of $\{1, \ldots, n\}$ occurs exactly once as a block of $n - 1$ consecutive digits (with its redundant final element suppressed). For example, (121323) is a universal cycle of permutations for $n = 3$, and it is essentially the only such cycle.

Find a universal cycle of permutations for $n = 4$, and prove that such cycles exist for all $n \geq 2$.

▸ **112.** [*HM43*] Exactly how many universal cycles exist, for permutations of ≤ 9 objects?

SECTION 7.2.1.2

1. [J. P. N. Phillips, *Comp. J.* **10** (1967), 311.] Assuming that $n \geq 3$, we can replace steps L2–L4 by:

```
L2′. [Easiest case?] Set $y \leftarrow a_{n-1}$ and $z \leftarrow a_n$. If $y < z$, set $a_{n-1} \leftarrow z$, $a_n \leftarrow y$, and return to L1.
L2.1′. [Next easiest case?] Set $x \leftarrow a_{n-2}$. If $x \geq y$, go on to step L2.2′. Otherwise set $(a_{n-2}, a_{n-1}, a_n) \leftarrow (z, x, y)$ if $x < z$, $(y, z, x)$ if $x \geq z$. Return to L1.
L2.2′. [Find $j$.] Set $j \leftarrow n - 3$ and $y \leftarrow a_j$. If $y \geq x$, set $j \leftarrow j - 1$, $x \leftarrow y$, $y \leftarrow a_j$, and repeat until $y < x$. Terminate if $j = 0$.
L3′. [Easy increase?] If $y < z$, set $a_j \leftarrow z$, $a_{j+1} \leftarrow y$, $a_n \leftarrow x$, and go to L4.1′.
L3.1′. [Increase $a_j$.] Set $l \leftarrow n-1$; if $y \geq a_l$, repeatedly decrease $l$ by 1 until $y < a_l$. Then set $a_j \leftarrow a_l$ and $a_l \leftarrow y$.
L4′. [Begin to reverse.] Set $a_n \leftarrow a_{j+1}$ and $a_{j+1} \leftarrow z$.
L4.1′. [Reverse $a_{j+1} \ldots a_{n-1}$.] Set $k \leftarrow j + 2$, $l \leftarrow n - 1$. Then, if $k < l$, interchange $a_k \leftrightarrow a_l$, set $k \leftarrow k + 1$, $l \leftarrow l - 1$, and repeat until $k \geq l$. Return to L1. ▌
```

The program might run still faster if a_t is stored in memory location `A`$[n - t]$ for $0 \leq t \leq n$, or if reverse colex order is used as in the following exercise.

2. Again we assume that $a_1 \leq a_2 \leq \cdots \leq a_n$ initially; the permutations generated from $\{1, 2, 2, 3\}$ will, however, be 1223, 2123, 2213, …, 2321, 3221. Let a_{n+1} be an auxiliary element, *larger* than a_n.

```
L1. [Visit.] Visit the permutation $a_1 a_2 \ldots a_n$.
L2. [Find $j$.] Set $j \leftarrow 2$. If $a_{j-1} \geq a_j$, increase $j$ by 1 until $a_{j-1} < a_j$. Terminate if $j > n$.
L3. [Decrease $a_j$.] Set $l \leftarrow 1$. If $a_l \geq a_j$, increase $l$ until $a_l < a_j$. Then swap $a_l \leftrightarrow a_j$.
L4. [Reverse $a_1 \ldots a_{j-1}$.] Set $k \leftarrow 1$ and $l \leftarrow j - 1$. Then, if $k < l$, swap $a_k \leftrightarrow a_l$, set $k \leftarrow k + 1$, $l \leftarrow l - 1$, and repeat until $k \geq l$. Return to L1. ▌
```

3. Let $C_1 \ldots C_n = c_{a_1} \ldots c_{a_n}$ be the inversion table, as in exercise 5.1.1–7. Then rank$(a_1 \ldots a_n)$ is the mixed-radix number $\left[{C_1,\ \ldots,\ C_{n-1},\ C_n \atop n,\ \ldots,\ 2,\ 1}\right]$. [See H. A. Rothe, *Sammlung combinatorisch-analytischer Abhandlungen* **2** (1800), 263–264; and see also the pioneering work of Nārāyaṇa cited in Section 7.2.1.7.] For example, 314592687 has rank $\left[{2,\ 0,\ 1,\ 1,\ 4,\ 0,\ 0,\ 1,\ 0 \atop 9,\ 8,\ 7,\ 6,\ 5,\ 4,\ 3,\ 2,\ 1}\right] = 2 \cdot 8! + 6! + 5! + 4 \cdot 4! + 1! = 81577$; this is the factorial number system featured in Eq. 4.1–(10).

4. Use the recurrence $\mathrm{rank}(a_1 \ldots a_n) = \frac{1}{n} \sum_{j=1}^{t} n_j [x_j < a_1] \binom{n}{n_1, \ldots, n_t} + \mathrm{rank}(a_2 \ldots a_n)$. For example, rank(314159265) is

$$\tfrac{3}{9}\tbinom{9}{2,1,1,1,2,1,1} + 0 + \tfrac{2}{7}\tbinom{7}{1,1,1,2,1,1} + 0 + \tfrac{1}{5}\tbinom{5}{1,2,1,1} + \tfrac{3}{4}\tbinom{4}{1,1,1,1} + 0 + \tfrac{1}{2}\tbinom{2}{1,1} = 30991.$$

5. (a) Step L2 is performed $n!$ times. The probability that exactly k comparisons are made is $q_k - q_{k+1}$, where q_t is the probability that $a_{n-t+1} > \cdots > a_n$, namely $[t \leq n]/t!$. Therefore the mean is $\sum k(q_k - q_{k+1}) = q_1 + \cdots + q_n = \lfloor n!\, e \rfloor / n! - 1 \approx e - 1 \approx 1.718$, and the variance is

$$\sum k^2 (q_k - q_{k+1}) - \mathrm{mean}^2 = q_1 + 3q_2 + \cdots + (2n-1)q_n - (q_1 + \cdots + q_n)^2 \approx e(3 - e) \approx 0.766.$$

[For higher moments, see R. Kemp, *Acta Informatica* **35** (1998), 17–89, Theorem 4.]

Incidentally, the average number of interchange operations in step L4 is therefore $\sum\lfloor k/2\rfloor(q_k - q_{k+1}) = q_2 + q_4 + \cdots \approx \cosh 1 - 1 = (e + e^{-1} - 2)/2 \approx 0.543$, a result due to R. J. Ord-Smith [*Comp. J.* **13** (1970), 152–155].

(b) Step L3 is performed only $n! - 1$ times, but we will assume for convenience that it occurs once more (with 0 comparisons). Then the probability that exactly k comparisons are made is $\sum_{j=k+1}^{n} 1/j!$ for $1 \le k < n$ and $1/n!$ for $k = 0$. Hence the mean is $\frac{1}{2}\sum_{j=0}^{n-2} 1/j! \approx e/2 \approx 1.359$; exercise 1 reduces this number by $\frac{2}{3}$. The variance is $\frac{1}{3}\sum_{j=0}^{n-3} 1/j! + \frac{1}{2}\sum_{j=0}^{n-2} 1/j! - \text{mean}^2 \approx \frac{5}{6}e - \frac{1}{4}e^2 \approx 0.418$.

6. (a) Let $e_n(z) = \sum_{k=0}^{n} z^k/k!$; then the number of different prefixes $a_1 \ldots a_j$ is $j!\,[z^j]\,e_{n_1}(z)\ldots e_{n_t}(z)$. This is $N = \binom{n}{n_1,\ldots,n_t}$ times the probability q_{n-j} that at least $n-j$ comparisons are made in step L2. Therefore the mean is $\frac{1}{N}w(e_{n_1}(z)\ldots e_{n_t}(z))-1$, where $w(\sum x_k z^k/k!) = \sum x_k$. In the binary case the mean is $M/\binom{n}{s} - 1$, where $M = \sum_{l=0}^{s}\sum_{k=l}^{n-s+l}\binom{k}{l} = \sum_{l=0}^{s}\binom{n-s+l+1}{l+1} = \binom{n+2}{s+1} - 1 = \binom{n}{s}\bigl(2 + \frac{s}{n-s+1} + \frac{n-s}{s+1}\bigr) - 1$.

(b) If $\{a_1,\ldots,a_j\} = \{n_1'\cdot x_1,\ldots,n_t'\cdot x_t\}$, the prefix $a_1\ldots a_j$ contributes altogether $\sum_{1\le k<l\le t}(n_k - n_k')[n_l < n_l']$ to the total number of comparisons made in step L3. Thus the mean is $\frac{1}{N}\sum_{1\le k<l\le t} w(f_{kl}(z))$, where

$$\begin{aligned} f_{kl}(z) &= \Biggl(\prod_{\substack{1\le m\le t\\ m\ne k,\, m\ne l}} e_{n_m}(z)\Biggr)\Biggl(\sum_{r=0}^{n_k}(n_k - r)\frac{z^r}{r!}\Biggr)e_{n_l-1}(z) \\ &= e_{n_1}(z)\ldots e_{n_t}(z)\bigl(n_k - z\,r_k(z)\bigr)r_l(z), \qquad \text{where } r_k(z) = \frac{e_{n_k-1}(z)}{e_{n_k}(z)}. \end{aligned}$$

In the two-valued case this formula reduces to $\frac{1}{N}w\bigl((se_s(z) - ze_{s-1}(z))e_{n-s-1}(z)\bigr) = \frac{s}{N}\bigl(\binom{n+1}{s+1} - 1\bigr) - \frac{1}{N}\bigl(\binom{n+1}{s+1}\bigl(s - \frac{s+1}{n-s+1}\bigr) + 1\bigr) = \frac{1}{N}\bigl(-s - 1 + \binom{n+1}{s}\bigr) = \frac{n+1}{n-s+1} - \frac{s+1}{N}$.

7. In the notation of the previous answer, the quantity $\frac{1}{N}w(e_{n_1}(z)\ldots e_{n_t}(z)) - 1$ is

$$\frac{n_1 + \cdots + n_t}{n} + \frac{(n_1n_2 + n_1n_3 + \cdots + n_{t-1}n_t) + n_1(n_1-1) + \cdots + n_t(n_t-1)}{n(n-1)} + \cdots - 1.$$

One can show using Eq. 1.2.9–(38) that the limit is $-1 + \exp\sum_{k\ge1} r_k/k$, where $r_k = \lim_{t\to\infty}(n_1^k + \cdots + n_t^k)/(n_1 + \cdots + n_t)^k$. In cases (a) and (b) we have $r_k = [k=1]$, so the limit is $e - 1 \approx 1.71828$. In case (c) we have $r_k = 1/(2^k - 1)$, so the limit is $-1 + \exp\sum_{k\ge1} 1/(k(2^k-1)) \approx 2.46275$.

8. Assume that j is initially zero, and change step L1 to

L1′. [Visit.] Visit the variation $a_1\ldots a_j$. If $j < n$, set $j \leftarrow j + 1$ and repeat this step. ▌

This algorithm is due to L. J. Fischer and K. C. Krause, *Lehrbuch der Combinationslehre und der Arithmetik* (Dresden: 1812), 55–57.

Incidentally, the total number of variations is $w(e_{n_1}(z)\ldots e_{n_t}(z))$ in the notation of answer 6. This counting problem was first treated by James Bernoulli in *Ars Conjectandi* (1713), Part 2, Chapter 9.

9. **R1.** [Visit.] Visit the variation $a_1\ldots a_r$. (At this point $a_{r+1} \le \cdots \le a_n$.)

R2. [Easy case?] If $a_r < a_n$, interchange $a_r \leftrightarrow a_j$ where j is the smallest subscript such that $j > r$ and $a_j > a_r$, and return to R1.

R3. [Reverse.] Set $(a_{r+1},\ldots,a_n) \leftarrow (a_n,\ldots,a_{r+1})$ as in step L4.

R4. [Find j.] Set $j \leftarrow r-1$. If $a_j \ge a_{j+1}$, decrease j by 1 repeatedly until $a_j < a_{j+1}$. Terminate if $j = 0$.

R5. [Increase a_j.] Set $l \leftarrow n$. If $a_j \ge a_l$, decrease l by 1 repeatedly until $a_j < a_l$. Then interchange $a_j \leftrightarrow a_l$.

R6. [Reverse again.] Set $(a_{j+1}, \ldots, a_n) \leftarrow (a_n, \ldots, a_{j+1})$ as in step L4, and return to R1. ▮

The number of outputs is $r!\,[z^r]\,e_{n_1}(z)\ldots e_{n_t}(z)$; this is, of course, $n^{\underline{r}}$ when the elements are distinct.

10. $a_1a_2\ldots a_n = 213\ldots n$, $c_1c_2\ldots c_n = 010\ldots 0$, $o_1o_2\ldots o_n = 1(-1)1\ldots 1$, if $n \ge 2$.

11. Step (P1, ..., P7) is performed $(1, n!, n!, n!+x_n, n!, (x_n+3)/2, x_n)$ times, where $x_n = \sum_{k=1}^{n-1} k!$, because P7 is performed $(j-1)!$ times when $2 \le j \le n$.

12. We want the permutation of rank 999999. The answers are (a) 2783915460, by exercise 3; (b) 8750426319, because the reflected mixed-radix number corresponding to $\left[{0,\ 0,\ 1,\ 2,\ 3,\ 0,\ 2,\ 7,\ 0,\ 9 \atop 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10}\right]$ is $\left[{0,\ 0,\ 1,\ 3-2,\ 3,\ 5-0,\ 2,\ 7,\ 8-0,\ 9-9 \atop 1,\ 2,\ 3,\ \ 4,\ \ 5,\ \ 6,\ \ 7,\ 8,\ \ 9,\ \ 10}\right]$ by 7.2.1.1–(50); (c) the product $(0\,1\ \ldots\ 9)^9(0\,1\ \ldots\ 8)^0(0\,1\ \ldots\ 7)^7(0\,1\ \ldots\ 6)^2\ldots(0\,1\,2)^1$, namely 9703156248.

13. The first statement is true for all $n \ge 2$. But when 2 crosses 1, namely when c_2 changes from 0 to 1, we have $c_3 = 2$, $c_4 = 3$, $c_5 = \cdots = c_n = 0$, and the next permutation when $n \ge 5$ is $432156\ldots n$. [See *Time Travel* (1988), page 74.]

14. True at the beginning of steps P4, P5, and P6, because exactly $j-1-c_j+s$ elements lie to the left of x_j, namely $j-1-c_j$ from $\{x_1, \ldots, x_{j-1}\}$ and s from $\{x_{j+1}, \ldots, x_n\}$. (In a sense, this formula is the main point of Algorithm P.)

15. If $\left[{b_{n-1},\ \ldots,\ b_0 \atop 1,\ \ldots,\ n}\right]$ corresponds to the reflected Gray code $\left[{c_1,\ \ldots,\ c_n \atop 1,\ \ldots,\ n}\right]$, we get to step P6 if and only if $b_k = k-1$ for $j \le k \le n$ and B_{n-j+1} is even, by 7.2.1.1–(50). But $b_{n-k} = k-1$ for $j \le k \le n$ implies that B_{n-k} is odd for $j < k \le m$. Therefore $s = [c_{j+1} = j] + [c_{j+2} = j+1] = [o_{j+1} < 0] + [o_{j+2} < 0]$ in step P5. [See *Math. Comp.* **17** (1963), 282–285.]

16. **P1′.** [Initialize.] Set $c_j \leftarrow j$ and $o_j \leftarrow -1$ for $1 \le j < n$; also set $z \leftarrow a_n$.

P2′. [Visit.] Visit $a_1\ldots a_n$. Then go to P3.5′ if $a_1 = z$.

P3′. [Hunt down.] For $j \leftarrow n-1, n-2, \ldots, 1$ (in this order), set $a_{j+1} \leftarrow a_j$, $a_j \leftarrow z$, and visit $a_1\ldots a_n$. Then set $j \leftarrow n-1$, $s \leftarrow 1$, and go to P4′.

P3.5′. [Hunt up.] For $j \leftarrow 1, 2, \ldots, n-1$ (in this order), set $a_j \leftarrow a_{j+1}$, $a_{j+1} \leftarrow z$, and visit $a_1\ldots a_n$. Then set $j \leftarrow n-1$, $s \leftarrow 0$.

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

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

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

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

17. Initially $a_j \leftarrow a'_j \leftarrow j$ for $1 \le j \le n$. Step P5 should now set $t \leftarrow j - c_j + s$, $u \leftarrow j - q + s$, $v \leftarrow a_u$, $a_t \leftarrow v$, $a'_v \leftarrow t$, $a_u \leftarrow j$, $a'_j \leftarrow u$, $c_j \leftarrow q$. (See exercise 14.)

But with the inverse required and available we can actually simplify the algorithm significantly, avoiding the offset variable s and letting the control table $c_1\ldots c_n$ count only downwards, as noted by G. Ehrlich [*JACM* **20** (1973), 505–506]:

Q1. [Initialize.] Set $a_j \leftarrow a'_j \leftarrow j$, $c_j \leftarrow j-1$, and $d_j \leftarrow -1$ for $1 \le j \le n$. Also set $c_0 = -1$.

Q2. [Visit.] Visit the permutation $a_1 \ldots a_n$ and its inverse $a'_1 \ldots a'_n$.

Q3. [Find k.] Set $k \leftarrow n$. Then if $c_k = 0$, set $c_k \leftarrow k-1$, $o_k \leftarrow -o_k$, $k \leftarrow k-1$, and repeat until $c_k \neq 0$. Terminate if $k = 0$.

Q4. [Change.] Set $c_k \leftarrow c_k - 1$, $j \leftarrow a'_k$, and $i = j + o_k$. Then set $t \leftarrow a_i$, $a_i \leftarrow k$, $a_j \leftarrow t$, $a'_t \leftarrow j$, $a'_k \leftarrow i$, and return to Q2. ▮

18. Set $a_n \leftarrow n$, and use $(n-1)!/2$ iterations of Algorithm P to generate all permutations of $\{1, \ldots, n-1\}$ such that 1 precedes 2. [M. K. Roy, *CACM* **16** (1973), 312–313; see also exercise 13.]

19. For example, we can use the idea of Algorithm P, with the n-tuples $c_1 \ldots c_n$ changing as in Algorithm 7.2.1.1H with respect to the radices $(1, 2, \ldots, n)$. That algorithm maintains the directions correctly, although it numbers subscripts differently. The offset s needed by Algorithm P can be computed as in the answer to exercise 15, or the inverse permutation can be maintained as in exercise 17. [See G. Ehrlich, *CACM* **16** (1973), 690–691.] Other algorithms, like that of Heap, can also be implemented looplessly.

(*Note:* In most applications of permutation generation we are interested in minimizing the *total* running time, not the maximum time between successive visits; from this standpoint looplessness is usually undesirable, except on a parallel computer. Yet there's something intellectually satisfying about the fact that a loopless algorithm exists, whether practical or not.)

20. For example, when $n = 3$ we can begin 123, 132, 312, $\bar{3}12$, $1\bar{3}2$, $12\bar{3}$, $21\bar{3}$, ..., 213, $\bar{2}13$, ... If the delta sequence for n is $(\delta_1\delta_2 \ldots \delta_{2^n n!})$, the corresponding sequence for $n+1$ is $(\Delta_n\delta_1\Delta_n\delta_2 \ldots \Delta_n\delta_{2^n n!})$, where Δ_n is the sequence of $2n-1$ operations n $n{-}1$ ... 1 $-$ 1 ... $n{-}1$ n; here $\delta_k = j$ means $a_j \leftrightarrow a_{j+1}$ and $\delta_k = -$ means $a_1 \leftarrow -a_1$.

(Signed permutations appear in another guise in exercises 5.1.4–43 and 44. The set of all signed permutations is called the octahedral group.)

21. Clearly $\mathtt{M} = 1$, hence $\mathtt{O}$ must be 0 and $\mathtt{S}$ must be $b-1$. Then $\mathtt{N} = \mathtt{E}+1$, $\mathtt{R} = b-2$, and $\mathtt{D}+\mathtt{E} = b + \mathtt{Y}$. This leaves exactly $\max(0, b-7-k)$ choices for $\mathtt{E}$ when $\mathtt{Y} = k \geq 2$, hence a total of $\sum_{k=2}^{b-7}(b-7-k) = \binom{b-8}{2}$ solutions when $b \geq 8$. [*Math. Mag.* **45** (1972), 48–49. Incidentally, D. Eppstein has proved that the task of solving alphametics with a given radix is NP-complete; see *SIGACT News* **18**, 3 (1987), 38–40.]

22. $(\mathtt{XY})_b + (\mathtt{XX})_b = (\mathtt{XYX})_b$ is solvable only when $b = 2$.

23. Almost true, because the number of solutions will be even, *unless* $[j \in F] \neq [k \in F]$. (Consider the ternary alphametic $\mathtt{X} + (\mathtt{XX})_3 + (\mathtt{YY})_3 + (\mathtt{XZ})_3 = (\mathtt{XYX})_3$.)

24. (a) $9283 + 7 + 473 + 1062 = 10825$. (b) $698392 + 3192 = 701584$. (c) $63952 + 69275 = 133227$. (d) $653924 + 653924 = 1307848$. (e) $5718 + 3 + 98741 = 104462$. (f) $127503 + 502351 + 3947539 + 46578 = 4623971$. (g) $67432 + 704 + 8046 + 97364 = 173546$. (h) $59 + 577404251698 + 69342491650 + 49869442698 + 1504 + 40614 + 82591 + 344 + 41 + 741425 = 5216367650 + 691400684974$. [All solutions are unique. References for (b)–(g): *J. Recreational Math.* **10** (1977), 115; **5** (1972), 296; **10** (1977), 41; **10** (1978), 274; **12** (1979), 133–134; **9** (1977), 207.]

(i) In this case there are $\frac{8}{10}10! = 2903040$ solutions, because *every* permutation of $\{0, 1, \ldots, 9\}$ works except those that assign $\mathtt{H}$ or $\mathtt{N}$ to 0. (A well-written general additive alphametic solver will be careful to reduce the amount of output in such cases.)

25. We may assume that $s_1 \leq \cdots \leq s_{10}$. Let i be the least index $\notin F$, and set $a_i \leftarrow 0$; then set the remaining elements a_j in order of increasing j. A proof like that of Theorem 6.1S shows that this procedure maximizes $a \cdot s$. A similar procedure yields the minimum, because $\min(a \cdot s) = -\max(a \cdot (-s))$.

26. $400739 + 63930 - 2379 - 1252630 + 53430 - 1390 + 738300$.

27. Readers can probably improve upon the following examples: `BLOOD` + `SWEAT` + `TEARS` = `LATER`; `EARTH` + `WATER` + `WRATH` = `HELLO` + `WORLD`; `AWAIT` + `ROBOT` + `ERROR` = `SOBER` + `WORDS`; `CHILD` + `THEME` + `PEACE` + `ETHIC` = `IDEAL` + `ALPHA` + `METIC`. (This exercise was inspired by `WHERE` + `SEDGE` + `GRASS` + `GROWS` = `MARSH` [A. W. Johnson, Jr., *J. Recr. Math.* **15** (1982), 51], which would be marvelously pure except that `D` and `O` have the same signature.)

28. (a) $11 = 3+3+2+2+1$, $20 = 11+3+3+3$, $20 = 11+3+3+2+1$, $20 = 11+3+3+1+1+1$, $20 = 8+8+2+1+1$, $20 = 7+7+6$, $20 = 7+7+2+2+2$, $20 = 7+7+2+1+1+1+1$, $20 = 7+5+5+2+1$, $20 = 7+5+2+2+2+1+1$, $20 = 7+5+2+2+1+1+1+1$, $20 = 7+3+3+2+2+1+1+1$, $20 = 7+3+3+1+1+1+1+1+1+1$, $20 = 5+3+3+3+3+3$. [These fourteen solutions were first computed by Roy Childs in 1999. The next doubly partitionable values of n are 30 (in 20 ways), then 40 (in 94 ways), 41 (in 67), 42 (in 57), 50 (in 190 ways, including $50 = 2+2+\cdots+2$), etc.]

(b) $51 = 20+15+14+2$, $51 = 15+14+10+9+3$, $61 = 19+16+11+9+6$, $65 = 17+16+15+9+7+1$, $66 = 20+19+16+6+5$, $69 = 18+17+16+10+8$, $70 = 30+20+10+7+3$, $70 = 20+16+12+9+7+6$, $70 = 20+15+12+11+7+5$, $80 = 50+20+9+1$, $90 = 50+12+11+9+5+2+1$, $91 = 45+19+11+10+5+1$. [The two 51s are due to Steven Kahan; see his book *Have Some Sums To Solve* (Farmingdale, New York: Baywood, 1978), 36–37, 84, 112. Amazing examples with seventeen distinct terms in Italian and fifty-eight distinct terms in Roman numerals have been found by Giulio Cesare, *J. Recr. Math.* **30** (1999), 63.]

Notes: The beautiful example `THREE` = `TWO`+`ONE`+`ZERO` [Richard L. Breisch, *Recreational Math. Magazine* **12** (December 1962), 24] is unfortunately ruled out by our conventions. The total number of doubly true partitions into distinct parts is probably finite, in English, although nomenclature for arbitrarily large integers is not standard. Is there an example bigger than `NINETYNINENONILLIONNINETYNINESEXTILLIONSIXTYONE` = `NINETYNINENONILLIONNINETYNINESEXTILLIONNINETEEN`+`SIXTEEN`+`ELEVEN`+`NINE`+`SIX` (suggested by G. González-Morris)?

29. $10+7+1 = 9+6+3$, $11+10 = 8+7+6$, $12+7+6+5 = 11+10+9$, $\ldots$, $19+10+3 = 14+13+4+1$ (31 examples in all).

30. (a) $567^2 = 321489$, $807^2 = 651249$, or $854^2 = 729316$. (b) $958^2 = 917764$. (c) $96 \times 7^2 = 4704$. (d) $51304/61904 = 7260/8760$. (e) $328509^2 = 4761^3$. [*Strand* **78** (1929), 91, 208; *J. Recr. Math* **3** (1970), 43; **13** (1981), 212; **27** (1995), 137; **31** (2003), 133. The solutions to (b), (c), (d), and (e) are unique. With a right-to-left approach based on Algorithm X, the answers are found in (14, 13, 11, 3423, 42) kilomems, respectively. Nob also noticed that `NORTH`/`SOUTH` = `WEST`/`EAST` has the unique solution $67104/27504 = 9320/3820$.]

31. $5/34 + 7/68 + 9/12$(!). One can verify uniqueness with Algorithm X using the side condition `A` < `D` < `G`, in about 265 Kμ. [*Quark Visual Science Magazine*, No. 136 (Tokyo: Kodansha, October 1993).] Curiously, a very similar puzzle also has a unique solution: $1/(3\times 6) + 5/(8\times 9) + 7/(2\times 4) = 1$; see Scot Morris, *Omni* **17**,4 (January 1995), 97.

32. There are eleven ways, of which the most surprising is $3 + 69258/714$. [See *The Weekly Dispatch* (9 and 23 June 1901); *Amusements in Mathematics* (1917), 158–159.]

33. (a) 1, 2, 3, 4, 15, 18, 118, 146. (b) 6, 9, 16, 20, 27, 126, 127, 129, 136, 145. [*The Weekly Dispatch* (11 and 30 November, 1902); *Amusements in Math.* (1917), 159.]

In this case one suitable strategy is to find all variations where $a_k \ldots a_{l-1}/a_l \ldots a_9$ is an integer, then to record solutions for all permutations of $a_1 \ldots a_{k-1}$. There are exactly 164959 integers with a unique solution, the largest being 9876533. There are solutions for all years in the 21st century except 2091. The most solutions (125) occur when $n = 6443$; the longest stretch of representable n's is $5109 < n < 7060$. Dudeney was able to get the correct answers by hand for small n by "casting out nines."

34. (a) $x = 10^5$, $7378{+}155{+}92467 = 7178{+}355{+}92467 = 1016{+}733{+}98251 = 100000$. (b) $x = 4^7$, $3036 + 455 + 12893 = 16384$ is unique. The fastest way to resolve this problem is probably to start with a list of the 2529 primes that consist of five distinct digits (namely 10243, 10247, ..., 98731) and to permute the five remaining digits.

Incidentally, the unrestricted alphametic `EVEN` + `ODD` = `PRIME` has ten solutions; both `ODD` and `PRIME` are prime in just one of them. [See M. Arisawa, *J. Recr. Math.* **8** (1975), 153.]

35. In general, if $s_k = |S_k|$ for $1 \le k < n$, there are $s_1 \ldots s_{k-1}$ ways to choose each of the nonidentity elements of S_k. Hence the answer is $\prod_{k=1}^{n-1}(\prod_{j=1}^{k-1} s_j^{s_k-1})$, which in this case is $2^2 \cdot 6^3 \cdot 24^{15} = 436196692474023836123136$.

(But if the vertices are renumbered, the s_k values may change. For example, if vertices $(\mathtt{0},\mathtt{3},\mathtt{5})$ of $(\mathtt{12})$ are interchanged with $(\mathtt{e},\mathtt{d},\mathtt{c})$, we have $s_{14} = 1$, $s_{13} = 6$, $s_{12} = 4$, $s_{11} = 1$, and $4^5 \cdot 24^{15}$ Sims tables.)

36. Since each of $\{\mathtt{0},\mathtt{3},\mathtt{5},\mathtt{6},\mathtt{9},\mathtt{a},\mathtt{c},\mathtt{f}\}$ lies on three lines, but every other element lies on only two, it is clear that we may let $S_{\mathtt{f}} = \{(),\sigma,\sigma^2,\sigma^3,\alpha,\alpha\sigma,\alpha\sigma^2,\alpha\sigma^3\}$, where $\sigma = (\mathtt{03fc})(\mathtt{17e4})(\mathtt{2bd4})(\mathtt{56a9})$ is a 90° rotation and $\alpha = (\mathtt{05})(\mathtt{14})(\mathtt{27})(\mathtt{36})(\mathtt{8d})(\mathtt{9c})(\mathtt{af})(\mathtt{be})$ is an inside-out twist. Also $S_{\mathtt{e}} = \{(),\beta,\gamma,\beta\gamma\}$, where $\beta = (\mathtt{14})(\mathtt{28})(\mathtt{3c})(\mathtt{69})(\mathtt{be})$ is a transposition and $\gamma = (\mathtt{12})(\mathtt{48})(\mathtt{5a})(\mathtt{69})(\mathtt{7b})(\mathtt{de})$ is another twist; $S_{\mathtt{d}} = \cdots = S_{\mathtt{1}} = \{()\}$. (There are $4^7 - 1$ alternative answers.)

37. The set S_k can be chosen in $k!^{k-1}$ ways (see exercise 35), and its nonidentity elements can be assigned to $\sigma(k,1)$, ..., $\sigma(k,k)$ in $k!$ further ways. So the answer is $A_n = \prod_{k=1}^{n-1} k!^k = n!^{\binom{n}{2}}/\prod_{k=1}^{n} k^{\binom{k}{2}}$. For example, $A_{10} \approx 6.256 \times 10^{148}$. We have

$$\sum_{k=1}^{n-1}\binom{k}{2}\ln k = \frac{1}{2}\int_1^n x(x-1)\ln x\,dx + O(n^2\log n) = \frac{1}{6}n^3\ln n + O(n^3)$$

by Euler's summation formula; thus $\ln A_n = \frac{1}{3}n^3\ln n + O(n^3)$.

38. The probability that $\phi(k)$ is needed in step G4 is $1/k! - 1/(k+1)!$, for $1 \le k < n$; the probability is $1/n!$ that we don't get to step G4 at all. Since $\phi(k)$ does $\lceil k/2\rceil$ transpositions, the average is $\sum_{k=1}^{n-1}(1/k! - 1/(k+1)!)\lceil k/2\rceil = \sum_{k=1}^{n-1}(\lceil k/2\rceil - \lceil (k-1)/2\rceil)/k! - \lceil (n-1)/2\rceil/n! = \sum_{k\,\mathrm{odd}} 1/k! + O(1/(n-1)!)$.

39. (a) 0123, 1023, 2013, 0213, 1203, 2103, 3012, 0312, 1302, 3102, 0132, 1032, 2301, 3201, 0231, 2031, 3021, 0321, 1230, 2130, 3120, 1320, 2310, 3210; (b) 0123, 1023, 2013, 0213, 1203, 2103, 3102, 1302, 0312, 3012, 1032, 0132, 0231, 2031, 3021, 0321, 2301, 3201, 3210, 2310, 1320, 3120, 2130, 1230.

40. By induction we find $\sigma(1,1) = (0\ 1)$, $\sigma(2,2) = (0\ 1\ 2)$,

$$\sigma(k,k) = \begin{cases} (0\ k)(k{-}1\ k{-}2\ \ldots\ 1), & \text{if } k \ge 3 \text{ is odd,} \\ (0\ k{-}1\ k{-}2\ 1\ \ldots\ k{-}3\ k), & \text{if } k \ge 4 \text{ is even;} \end{cases}$$

also $\omega(k) = (0\ k)$ when k is even, $\omega(k) = (0\ k{-}2\ \ldots\ 1\ k{-}1\ k)$ when $k \ge 3$ is odd. Thus when $k \ge 3$ is odd, $\sigma(k,1) = (k\ k{-}1\ 0)$ and $\sigma(k,j)$ takes $k \mapsto j-1$ for $1 < j < k$; when $k \ge 4$ is even, $\sigma(k,j) = (0\ k\ k{-}3\ \ldots\ 1\ k{-}2\ k{-}1)^j$ for $1 \le j \le k$.

Notes: The first scheme that causes Algorithm G to generate all permutations by single transpositions was devised by Mark Wells [*Math. Comp.* **15** (1961), 192–195], but it was considerably more complicated. W. Lipski, Jr., studied such schemes in general and found a variety of additional methods [*Computing* **23** (1979), 357–365].

41. We may assume that $r < n$. Algorithm G will generate r-variations for any Sims table if we simply change '$k \leftarrow 1$' to '$k \leftarrow n-r$' in step G3, provided that we redefine $\omega(k)$ to be $\sigma(n-r, n-r) \ldots \sigma(k,k)$ instead of using (16).

If $n-r$ is odd, the method of (27) is still valid, although the formulas in answer 40 need to be revised when $k < n-r+2$. The new formulas are $\sigma(k,j) = (k\ j{-}1\ \ldots\ 1\ 0)$ and $\omega(k) = (k\ \ldots\ 1\ 0)$ when $k = n-r$; $\sigma(k,j) = (k\ \ldots\ 1\ 0)^j$ when $k = n-r+1$.

If $n-r$ is even, we can use (27) with even and odd reversed, if $r \le 3$. But when $r \ge 4$ a more complex scheme is needed, because a fixed transposition like $(k\ 0)$ can be used for odd k only if $\omega(k-1)$ is a k-cycle, which means that $\omega(k-1)$ must be an even permutation; but $\omega(k)$ is odd for $k \ge n-r+2$.

The following scheme works when $n-r$ is even: Let $\tau(k,j)\omega(k-1)^- = (k\ k{-}j)$ for $1 \le j \le k = n-r$, and use (27) when $k > n-r$. Then, when $k = n-r+1$, we have $\omega(k-1) = (0\ 1\ \ldots\ k{-}1)$, hence $\sigma(k,j)$ takes $k \mapsto (2j-1) \bmod k$ for $1 \le j \le k$, and $\sigma(k,k) = (k\ k{-}1\ k{-}3\ \ldots\ 0\ k{-}2\ \ldots\ 1)$, $\omega(k) = (k\ \ldots\ 1\ 0)$, $\sigma(k{+}1,j) = (k{+}1\ \ldots\ 0)^j$.

42. If $\sigma(k,j) = (k\ j{-}1)$ we have $\tau(k,1) = (k\ 0)$ and $\tau(k,j) = (k\ j{-}1)(k\ j{-}2) = (k\ j{-}1\ j{-}2)$ for $2 \le j \le k$.

43. Of course $\omega(1) = \sigma(1,1) = \tau(1,1) = (0\ 1)$. The following construction makes $\omega(k) = (k{-}2\ k{-}1\ k)$ for all $k \ge 2$: Let $\alpha(k,j) = \tau(k,j)\,\omega(k-1)^-$, where $\alpha(2,1) = (2\ 0)$, $\alpha(2,2) = (2\ 0\ 1)$, $\alpha(3,1) = \alpha(3,3) = (3\ 1)$, $\alpha(3,2) = (3\ 1\ 0)$; this makes $\sigma(2,2) = (0\ 2)$, $\sigma(3,3) = (0\ 3\ 1)$. Then for $k \ge 4$, let

$$\begin{array}{rcccccc} & & k \bmod 3 = 0 & & k \bmod 3 = 1 & & k \bmod 3 = 2 \\ \alpha(k,k{-}2) & = & (k\ k{-}2\ 0) & \text{or} & (k\ k{-}3\ 0) & \text{or} & (k\ k{-}1\ 0), \\ \alpha(k,k{-}1) & = & (k\ k{-}2\ k{-}3) & \text{or} & (k\ k{-}3) & \text{or} & (k\ k{-}1\ k{-}3), \\ \alpha(k,k) & = & (k\ k{-}2) & \text{or} & (k\ k{-}3\ k{-}2) & \text{or} & (k\ k{-}2); \end{array}$$

this makes $\sigma(k,k) = (k{-}3\ k\ k{-}2)$ as required.

44. No, because $\tau(k,j)$ is a $(k+1)$-cycle, not a transposition. (See (19) and (24).)

45. (a) 202280070, since $u_k = \max\bigl(\{0,1,\ldots,a_k-1\} \setminus \{a_1,\ldots,a_{k-1}\}\bigr)$. (Actually u_n is never set by the algorithm, but we can assume that it is zero.) (b) 425368917.

46. True (assuming that $u_n = 0$). If either $u_k > u_{k+1}$ or $a_k > a_{k+1}$ we must have $a_k > u_k \ge a_{k+1} > u_{k+1}$.

47. Steps (X1, X2, …, X6) are performed respectively $(1, A, B, A-1, B-N_n, A)$ times, where $A = N_0 + \cdots + N_{n-1}$ and $B = nN_0 + (n-1)N_1 + \cdots + 1N_{n-1}$.

48. Steps (X2, X3, X4, X5, X6) are performed respectively $A_n + (1, n!, 0, 0, 1)$ times, where $A_n = \sum_{k=1}^{n-1} n^{\underline{k}} = n!\sum_{k=1}^{n-1} 1/k! \approx n!\,(e-1)$. Assuming that they cost respectively $(1,1,3,1,3)$ mems, for operations involving a_j, l_j, or u_j, the total cost is about $9e - 8 \approx 16.46$ mems per permutation.

Algorithm L uses approximately $(e, 2+e/2, 2e+2e^{-1}-4)$ mems per permutation in steps (L2, L3, L4), for a total of $3.5e + 2e^{-1} - 2 \approx 8.25$ (see exercise 5).

Algorithm X could be tuned up for this case by streamlining the code when k is near n. But so can Algorithm L, as shown in exercise 1.

49. Order the signatures so that $|s_0| \geq \cdots \geq |s_9|$; also prepare tables $w_0 \ldots w_9$, $x_0 \ldots x_9$, $y_0 \ldots y_9$, so that the signatures $\{s_k, \ldots, s_9\}$ are $w_{x_k} \leq \cdots \leq w_{y_k}$. For example, when $\texttt{SEND} + \texttt{MORE} = \texttt{MONEY}$ we have $(s_0, \ldots, s_9) = (-9000, 1000, -900, 91, -90, 10, 1, -1, 0, 0)$ for the respective letters $(\texttt{M}, \texttt{S}, \texttt{O}, \texttt{E}, \texttt{N}, \texttt{R}, \texttt{D}, \texttt{Y}, \texttt{A}, \texttt{B})$; also $(w_0, \ldots, w_9) = (-9000, -900, -90, -1, 0, 0, 1, 10, 91, 1000)$, and $x_0 \ldots x_9 = 0112233344$, $y_0 \ldots y_9 = 9988776554$. Yet another table $f_0 \ldots f_9$ has $f_j = 1$ if the digit corresponding to w_j cannot be zero; in this case $f_0 \ldots f_9 = 1000000001$. These tables make it easy to compute the largest and smallest values of

$$s_k a_k + \cdots + s_9 a_9$$

over all choices $a_k \ldots a_9$ of the remaining digits, using the method of exercise 25, since the links l_j tell us those digits in increasing order.

This method requires a rather expensive computation at each node of the search tree, but it often succeeds in keeping that tree small. For example, it solves the first eight alphametics of exercise 24 with costs of only 7, 13, 7, 9, 5, 343, 44, and 89 kilomems; this is a substantial improvement in cases (a), (b), (e), and (h), although case (f) comes out significantly worse. Another bad case is the '$\texttt{CHILD}$' example of answer 27, where left-to-right needs 2947 kilomems compared to 588 for the right-to-left approach. Left-to-right does, however, fare better on $\texttt{BLOOD} + \texttt{SWEAT} + \texttt{TEARS}$ (73 versus 360) and $\texttt{HELLO} + \texttt{WORLD}$ (340 versus 410).

50. If α is in a permutation group, so are all its powers α^2, α^3, $\ldots$, including $\alpha^{m-1} = \alpha^-$, where m is the order of α (the least common multiple of its cycle lengths). And (32) is equivalent to $\alpha^- = \sigma_1 \sigma_2 \ldots \sigma_{n-1}$.

51. False. For example, $\sigma(k,i)^-$ and $\sigma(k,j)^-$ might both take $k \mapsto 0$.

52. $\tau(k,j) = (k{-}j\;\; k{-}j{+}1)$ is an adjacent interchange, and

$$\omega(k) = (n{-}1\ \ldots\ 0)(n{-}2\ \ldots\ 0)\ldots(k\ \ldots\ 0) = \phi(n-1)\phi(k-1)$$

is a k-flip followed by an n-flip. The permutation corresponding to control table $c_0 \ldots c_{n-1}$ in Algorithm H has c_j elements to the right of j that are less than j, for $0 \leq j < n$; so it is the same as the permutation corresponding to $c_1 \ldots c_n$ in Algorithm P, except that subscripts are shifted by 1.

The only essential difference between Algorithm P and this version of Algorithm H is that Algorithm P uses a reflected Gray code to run through all possibilities of its control table, while Algorithm H runs through those mixed-radix numbers in ascending (lexicographic) order.

Indeed, Gray code can be used with any Sims table, by modifying either Algorithm G or Algorithm H. Then all transitions are by $\tau(k,j)$ or by $\tau(k,j)^-$, and the permutations $\omega(k)$ are irrelevant.

53. The text's proof that $n! - 1$ transpositions cannot be achieved for $n = 4$ also shows that we can reduce the problem from n to $n-2$ at the cost of a single transposition $(n{-}1\;\; n{-}2)$, which was called '$(3\,c)$' in the notation of that proof.

Thus we can generate all permutations by making the following transformation in step H4: If $k = n-1$ or $k = n-2$, transpose $a_{j \bmod n} \leftrightarrow a_{(j-1) \bmod n}$, where $j = c_{n-1} - 1$. If $k = n-3$ or $k = n-4$, transpose $a_{n-1} \leftrightarrow a_{n-2}$ and also $a_{j \bmod (n-2)} \leftrightarrow a_{(j-1) \bmod (n-2)}$, where $j = c_{n-3} - 1$. And in general if $k = n - 2t - 1$ or $k = n - 2t - 2$, transpose $a_{n-2i+1} \leftrightarrow a_{n-2i}$ for $1 \le i \le t$ and also $a_{j \bmod (n-2t)} \leftrightarrow a_{(j-1) \bmod (n-2t)}$, where $j = c_{n-2t-1} - 1$. [See *CACM* **19** (1976), 68–72.]

The corresponding Sims table permutations can be written down as follows, although they don't appear explicitly in the algorithm itself:

$$\sigma(k,j)^- = \begin{cases} (0\ 1\ \ldots\ j{-}1\ k), & \text{if } n-k \text{ is odd;} \\ (0\ 1\ \ldots\ k)^j, & \text{if } n-k \text{ is even.} \end{cases}$$

The value of $a_{j \bmod (n-2t)}$ will be $n-2t-1$ after the interchange. For efficiency we can also use the fact that k usually equals $n-1$. The total number of transpositions is $\sum_{t=0}^{\lfloor n/2 \rfloor}(n-2t)! - \lfloor n/2 \rfloor - 1$.

54. Yes; the transformation can be any k-cycle on positions $\{1,\ldots,k\}$.

55. (a) Since $\rho_!(m) = \rho_!(m \bmod n!)$ when $n > \rho_!(m)$, we have $\rho_!(n!+m) = \rho_!(m)$ for $0 < m < n\cdot n! = (n+1)! - n!$. Therefore $\beta_{n!+m} = \sigma_{\rho_!(n!+m)} \ldots \sigma_{\rho_!(n!+1)}\beta_{n!} = \sigma_{\rho_!(m)} \ldots \sigma_{\rho_!(1)}\beta_{n!} = \beta_m\beta_{n!}$ for $0 \le m < n\cdot n!$, and we have in particular

$$\beta_{(n+1)!} = \sigma_{n+1}\beta_{(n+1)!-1} = \sigma_{n+1}\beta_{n!-1}\beta_{n!}^n = \sigma_{n+1}\sigma_n^-\beta_{n!}^{n+1}.$$

Similarly $\alpha_{n!+m} = \beta_{n!}^-\alpha_m\beta_{n!}\alpha_{n!}$ for $0 \le m < n\cdot n!$.

Since $\beta_{n!}$ commutes with τ_n and τ_{n+1} we find $\alpha_{n!} = \tau_n\alpha_{n!-1}$, and

$$\begin{aligned}\alpha_{(n+1)!} = \tau_{n+1}\alpha_{(n+1)!-1} &= \tau_{n+1}\beta_{n!}^-\alpha_{(n+1)!-1-n}\beta_{n!}\alpha_{n!} = \cdots \\ &= \tau_{n+1}\beta_{n!}^{-n}\alpha_{n!-1}(\beta_{n!}\alpha_{n!})^n \\ &= \beta_{n!}^{-n-1}\tau_{n+1}\tau_n^-(\beta_{n!}\alpha_{n!})^{n+1} \\ &= \beta_{(n+1)!}^-\sigma_{n+1}\sigma_n^-\tau_{n+1}\tau_n^-(\beta_{n!}\alpha_{n!})^{n+1}.\end{aligned}$$

(b) In this case $\sigma_{n+1}\sigma_n^- = (n\ n{-}1\ \ldots\ 1)$ and $\tau_{n+1}\tau_n^- = (n{+}1\ n\ 0)$, and we have $\beta_{(n+1)!}\alpha_{(n+1)!} = (n{+}1\ n\ \ldots\ 0)$ by induction. Therefore $\alpha_{jn!+m} = \beta_{n!}^{-j}\alpha_m(n\ \ldots\ 0)^j$ for $0 \le j \le n$ and $0 \le m < n!$. All permutations of $\{0,\ldots,n\}$ are achieved because $\beta_{n!}^{-j}\alpha_m$ fixes n and $(n\ \ldots\ 0)^j$ takes $n \mapsto n-j$.

56. If we set $\sigma_k = (k{-}1\ k{-}2)(k{-}3\ k{-}4)\ldots$ in the previous exercise, we find by induction that $\beta_{n!}\alpha_{n!}$ is the $(n+1)$-cycle $(0\ n\ n{-}1\ n{-}3\ \ldots\ (2 \text{ or } 1)\ (1 \text{ or } 2)\ \ldots\ n{-}4\ n{-}2)$.

57. Arguing as in answer 5, we obtain $\sum_{k=2}^{n-1}[k \,\text{odd}]/k! - (\lfloor n/2 \rfloor - 1)/n! = \sinh 1 - 1 - O(1/(n-1)!)$.

58. True. By the formulas of exercise 55 we have $\alpha_{n!-1} = (0\ n)\beta_{n!}^-(n\ \ldots\ 0)$, and this takes $0 \mapsto n-1$ because $\beta_{n!}$ fixes n. (Consequently Algorithm E will define a Hamiltonian *cycle* on the graph of exercise 66 if and only if $\beta_{n!} = (n{-}1\ \ldots\ 2\ 1)$, and this holds if and only if the length of every cycle of $\beta_{(n-1)!}$ is a divisor of n. The latter is true for $n = 2$, 3, 4, 6, 12, 20, and 40, but for no other $n \le 250{,}000$.)

59. The Cayley graph with generators $(\alpha_1,\ldots,\alpha_k)$ in the text's definition is isomorphic to the Cayley graph with generators $(\alpha_1^-,\ldots,\alpha_k^-)$ in the alternative definition, since $\pi \to \alpha_j\pi$ in the former if and only if $\pi^- \to \pi^-\alpha_j^-$ in the latter.

60. There are 88 delta sequences, which reduce to four classes: $P = (32131231)^3$ (plain changes, represented by 8 different delta sequences); $Q = (32121232)^3$ (a doubly Gray variant of plain changes, with 8 representatives); $R = (121232321232)^2$ (a doubly Gray code with 24 representatives); $S = 2\alpha 3\alpha^R$, $\alpha = 12321312121$ (48 representatives). Classes P and Q are cyclic shifts of their complements; classes P, Q, and S are shifts of their reversals; class R is a shifted reversal of its complement. [See A. L. Leigh Silver, *Math. Gazette* **48** (1964), 1–16.]

61. There are respectively $(26, 36, 20, 26, 28, 40, 40, 20, 26, 28, 28, 26)$ such paths ending at $(1243, 1324, 1432, 2134, 2341, 2413, 3142, 3214, 3421, 4123, 4231, 4312)$.

62. There are only two paths when $n = 3$, ending respectively at 132 and 213. But when $n \geq 4$ there are Gray codes leading from $12\ldots n$ to any odd permutation $a_1a_2\ldots a_n$. Exercise 61 establishes this when $n = 4$, and we can prove it by induction for $n > 4$ as follows.

Let $A(j)$ be the set of all permutations that begin with j, and let $A(j,k)$ be those that begin with jk. If $(\alpha_0, \alpha_1, \ldots, \alpha_n)$ are any odd permutations such that $\alpha_j \in A(x_j, x_{j+1})$, then $(1\,2)\alpha_j$ is an even permutation in $A(x_{j+1}, x_j)$. Consequently, if $x_1x_2\ldots x_n$ is a permutation of $\{1, 2, \ldots, n\}$, there is at least one Hamiltonian path of the form

$$(1\,2)\alpha_0 - \cdots - \alpha_1 - (1\,2)\alpha_1 - \cdots - \alpha_2 - \cdots - (1\,2)\alpha_{n-1} - \cdots - \alpha_n;$$

the subpath from $(1\,2)\alpha_{j-1}$ to α_j includes all elements of $A(x_j)$.

This construction solves the problem in at least $(n-2)!^n/2^{n-1}$ distinct ways when $a_1 \neq 1$, because we can take $\alpha_0 = 2\,1\ldots n$ and $\alpha_n = a_1a_2\ldots a_n$; there are $(n-2)!$ ways to choose $x_2\ldots x_{n-1}$, and $(n-2)!/2$ ways to choose each of $\alpha_1, \ldots, \alpha_{n-1}$.

Finally, if $a_1 = 1$, take any path $12\ldots n - \cdots - a_1a_2\ldots a_n$ that runs through all of $A(1)$, and choose any step $\alpha - \alpha'$ with $\alpha \in A(1,j)$ and $\alpha' \in A(1,j')$ for some $j \neq j'$. Replace that step by

$$\alpha - (1\,2)\alpha_1 - \cdots - \alpha_2 - \cdots - (1\,2)\alpha_{n-1} - \cdots - \alpha_n - \alpha',$$

using a construction like the Hamiltonian path above but now with $\alpha_1 = \alpha$, $\alpha_n = (1\,2)\alpha'$, $x_1 = 1$, $x_2 = j$, $x_n = j'$, and $x_{n+1} = 1$. (In this case the permutations α_1, $\ldots$, α_n might all be even.)

63. Monte Carlo estimates using the techniques of Section 7.2.3 suggest that the total number of equivalence classes will be roughly 1.2×10^{21}; most of those classes will contain 480 Gray cycles.

64. Exactly 2,005,200 delta sequences have the doubly Gray property; they belong to 4206 equivalence classes under cyclic shift, reversal, and/or complementation. Nine classes, such as the code $2\alpha2\alpha^R$ where

$$\alpha = 1234323432123212123232123212123434321212343212343212123232 1,$$

are shifts of their reversal; 48 classes are composed of repeated 60-cycles. One of the most interesting of the latter type is $\alpha\alpha$ where

$$\alpha = \beta2\beta4\beta4\beta4\beta4, \qquad \beta = 32121232123.$$

65. Such a path exists for any given $N \leq n!$: Let the Nth permutation be $\alpha = a_1\ldots a_n$, and let $j = a_1$. Also let Π_k be the set of all permutations $\beta = b_1\ldots b_n$ for which $b_1 = k$ and $\beta \leq \alpha$. By induction on N there is a Gray path P_1 for Π_j. We can then construct Gray paths P_k for $\Pi_j \cup \Pi_1 \cup \cdots \cup \Pi_{k-1}$ for $2 \leq k \leq j$, successively combining P_{k-1} with a Gray cycle for Π_{k-1}. (See the "absorption" construction of answer 62. In fact, P_j will be a Gray *cycle* when N is a multiple of 6.)

66. Defining the delta sequence by the rule $\pi_{(k+1) \bmod n!} = (1\ \delta_k)\pi_k$, we find exactly 36 such sequences, all of which are cyclic shifts of a pattern like $(xyzyzyxzyzyz)^2$. (The next case, $n = 5$, probably has about 10^{18} solutions that are inequivalent with respect to cyclic shifting, reversal, and permutation of coordinates, thus about 6×10^{21} different delta sequences.) Incidentally, Igor Pak has shown that the Cayley graph generated by star transpositions is an $(n-2)$-dimensional torus in general.

67. If we let π be equivalent to $\pi(12345)$, we get a reduced graph on 24 vertices that has 40768 Hamiltonian cycles, 240 of which lead to delta sequences of the form α^5 in which α uses each transposition 6 times (for example, $\alpha = 354232534234532454352452$). The total number of solutions to this problem is probably about 10^{16}.

68. If A isn't connected, neither is G. If A is connected, we can assume that it is a free tree. Moreover, in this case we can prove a generalization of the result in exercise 62: For $n \ge 4$ there is a Hamiltonian path in G from the identity permutation to any odd permutation. For we can assume without loss of generality that A contains the edge $1 — 2$ where 1 is a leaf of the tree, and a proof like that of exercise 62 applies.

[This elegant construction is due to M. Tchuente, *Ars Combinatoria* **14** (1982), 115–122. Extensive generalizations have been discussed by Ruskey and Savage in *SIAM J. Discrete Math.* **6** (1993), 152–166. See also the original Russian publication in *Kibernetika* **11**, 3 (1975), 17–25; English translation, *Cybernetics* **11** (1975), 362–366.]

69. Following the hint, the modified algorithm behaves like this when $n = 5$:

$$\begin{array}{ccccccccccccccccccccccc}
12\underbrace{34} & & 1\underbrace{24}3 & & \underbrace{14}23 & & 41\underbrace{23} & & \underbrace{41}32 & & 1\underbrace{43}2 & & 13\underbrace{42} & & \underbrace{13}24 & & 31\underbrace{24} & & 3\underbrace{14}2 & & \underbrace{34}12 & & 4312 \\
\downarrow & & \uparrow & & \downarrow & & \uparrow & & \downarrow & & \uparrow & & \downarrow & & \uparrow & & \downarrow & & \uparrow & & \downarrow & & \uparrow \\
54321 & & 24351 & & 24153 & & 54123 & & 14523 & & 14325 & & 24315 & & 24513 & & 54213 & & 14253 & & 14352 & & 54312 \\
12345 & & 15342 & & 35142 & & 32145 & & 32541 & & 52341 & & 51342 & & 31542 & & 31245 & & 35241 & & 25341 & & 21345 \\
15342 & & 12435 & & 32415 & & 35412 & \leftarrow & 31452 & & 51432 & & 52431 & & 32451 & \leftarrow & 35421 & & 31425 & & 21435 & & 25431 \\
23451 & & 53421 & & 51423 & & 21453 & \rightarrow & 25413 & & 23415 & & 13425 & & 15423 & \rightarrow & 12453 & & 52413 & & 53412 & & 13452 \\
21543 & & 51243 & & 53241 & & 23541 & & 23145 & & 25143 & & 15243 & & 13245 & & 13542 & & 53142 & & 52143 & & 12543 \\
34512 & & 34215 & & 14235 & & 14532 & & 54132 & & 34152 & & 34251 & & 54231 & & 24531 & & 24135 & & 34125 & & 34521 \\
32154 & \rightarrow & 35124 & & 15324 & \rightarrow & 12354 & & 52314 & & 32514 & \leftarrow & 31524 & & 51324 & & 21354 & \rightarrow & 25314 & & 35214 & \rightarrow & 31254 \\
45123 & \leftarrow & 42153 & & 42351 & \leftarrow & 45321 & & 41325 & & 41523 & \rightarrow & 42513 & & 42315 & & 45312 & \leftarrow & 41352 & & 41253 & \leftarrow & 45213 \\
43215 & & 43512 & \leftarrow & 41532 & & 41235 & & 45231 & \rightarrow & 43251 & & 43152 & \rightarrow & 45132 & & 42135 & & 42531 & \leftarrow & 43521 & & 43125 \\
51234 & & 21534 & \rightarrow & 23514 & & 53214 & & 13254 & \leftarrow & 15234 & & 25134 & \leftarrow & 23154 & & 53124 & & 13524 & \rightarrow & 12534 & & 52134 \\
\downarrow & & \uparrow & & \downarrow & & \uparrow & & \downarrow & & \uparrow & & \downarrow & & \uparrow & & \downarrow & & \uparrow & & \downarrow & & \uparrow
\end{array}$$

Here the columns represent sets of permutations that are cyclically rotated and/or reflected in all $2n$ ways; therefore each column contains exactly one "rosary permutation" (exercise 18). We can use Algorithm P to run through the rosary permutations systematically, knowing that the pair xy will occur before yx in its column, at which time τ' instead of ρ' will move us to the right or to the left. Step Z2 omits the interchange $a_1 \leftrightarrow a_2$, thereby causing the permutations $a_1 \ldots a_{n-1}$ to repeat themselves going backwards. (We implicitly use the fact that $t[k] = t[n! - k]$ in the output of Algorithm T.)

Now if we replace $1 \ldots n$ by $24 \ldots 31$ and change $A_1 \ldots A_n$ to $A_1 A_n A_2 A_{n-1} \ldots$, we get the unmodified algorithm whose results are shown in Fig. 22(b).

This method was inspired by a (nonconstructive) theorem of E. S. Rapoport, *Scripta Math.* **24** (1959), 51–58. It illustrates a more general fact observed by Carla Savage in 1989, namely that the Cayley graph for *any* group generated by three involutions ρ, σ, τ has a Hamiltonian cycle when $\rho\tau = \tau\rho$ [see I. Pak and R. Radoičić, "Hamiltonian paths in Cayley graphs," to appear].

70. No; the longest cycle in that digraph has length 358. But there do exist pairs of disjoint 180-cycles from which a Hamiltonian path of length 720 can be derived. For example, consider the cycles $\alpha\sigma\beta\sigma$ and $\gamma\sigma\sigma$ where

$$\alpha = \tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^5\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^1\tau\sigma^1;$$

$$\beta = \sigma^3\tau\sigma^5\tau\sigma^2\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^5\tau\sigma^1\tau\sigma^3\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^1\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^4;$$

$$\gamma = \sigma\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^5\tau\sigma^1\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^1\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^3\tau\sigma^2$$
$$\tau\sigma^5\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^5\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^5\tau\sigma^1\tau\sigma^3\tau\sigma^3\tau\sigma^5\tau\sigma^5\tau\sigma^1\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^2.$$

If we start with 134526 and follow $\alpha\sigma\beta\tau$ we reach 163452; then follow $\gamma\sigma\tau$ and reach 126345; then follow $\sigma\gamma\tau$ and reach 152634; then follow $\beta\sigma\alpha$, ending at 415263.

71. Brendan McKay and Frank Ruskey have found such cycles by computer when $n = 7$, 9, and 11, but no nice structure was apparent.

72. Any Hamiltonian path includes $(n-1)!$ vertices that take $y \mapsto x$, each of which (if not the last) is followed by a vertex that takes $x \mapsto x$. So one must be last; otherwise $(n-1)! + 1$ vertices would take $x \mapsto x$.

73. (a) Assume first that β is the identity permutation (). Then every cycle of α that contains an element of A lies entirely within A. Hence the cycles of σ are obtained by omitting all cycles of α that contain no element of A. All remaining cycles have odd length, so σ is an even permutation.

If β is not the identity, we apply this argument to $\alpha' = \alpha\beta^-$, $\beta' = ()$, and $\sigma' = \sigma\beta^-$, concluding that σ' is an even permutation; thus σ and β have the same sign.

Similarly, σ and α have the same sign, because $\beta\alpha^- = (\alpha\beta^-)^-$ has the same order as $\alpha\beta^-$.

(b) Let X be the vertices of the Cayley graph in Theorem R, and let α be the permutation of X that takes a vertex π into $\alpha\pi$; this permutation has g/a cycles of length a. Define the permutation β similarly. Then $\alpha\beta^-$ has g/c cycles of length c. If c is odd, any Hamiltonian cycle in the graph defines a cycle σ that contains all the vertices and satisfies the hypotheses of (a). Therefore α and β have an odd number of cycles, because the sign of a permutation on n elements with r cycles is $(-1)^{n-r}$ (see exercise 5.2.2–2).

[This proof, which shows that X cannot be the union of any odd number of cycles, was presented by Rankin in *Proc. Cambridge Phil. Soc.* **62** (1966), 15–16.]

74. The representation $\beta^j\gamma^k$ is unique if we require $0 \le j < g/c$ and $0 \le k < c$. For if we had $\beta^j = \gamma^k$ for some j with $0 < j < g/c$, the group would have at most jc elements. It follows that $\beta^{g/c} = \gamma^t$ for some t.

Let σ be a Hamiltonian cycle, as in the previous answer. If $x\sigma = x\alpha$ then $x\gamma\sigma$ must be $x\gamma\alpha$, because $x\gamma\beta = \alpha$. And if $x\sigma = x\beta$ then $x\gamma\sigma$ cannot be $x\gamma\alpha$, because that would imply $x\gamma^c\sigma = x\gamma^c\alpha$. Thus the elements $x\gamma^k$ all have equivalent behavior with respect to their successors in σ.

Notice that if $j \ge 0$ there is a $k \le j$ such that $x\sigma^j = x\alpha^k\beta^{j-k} = x\beta^j\gamma^k$. Therefore $x\sigma^{g/c} = x\gamma^{t+k}$ is equivalent to x, and the same behavior will repeat. We return to x for the first time in g steps if and only if $t + k$ is relatively prime to c.

75. Apply the previous exercise with $g = mn$, $a = m$, $b = n$, $c = mn/d$. The number t satisfies $t \equiv 0$ (modulo m), $t + d \equiv 0$ (modulo n); and it follows that $k + t \perp c$ if and only if $(d-k)m/d \perp kn/d$.

Notes: The modular Gray code of exercise 7.2.1.1–78 is a Hamiltonian path from $(0,0)$ to $(m-1, (-m) \bmod n)$, so it is a Hamiltonian cycle if and only if m is a multiple of n. It is natural to conjecture (falsely) that at least one Hamiltonian cycle exists whenever $d > 1$. But P. Erdős and W. T. Trotter have observed [*J. Graph Theory* **2**

(1978), 137–142] that if p and $2p+1$ are odd prime numbers, no suitable k exists when $m = p(2p+1)(3p+1)$ and $n = (3p+1)\prod_{q=1}^{3p} q^{[q\text{ is prime}][q\neq p][q\neq 2p+1]}$.

See J. A. Gallian, *Mathematical Intelligencer* **13**, 3 (Summer 1991), 40–43, for interesting facts about other kinds of cycles in $C_m \times C_n$.

76. We may assume that the tour begins in the lower left corner. There are no solutions when m and n are both divisible by 3, because 2/3 of the cells are unreachable in that case. Otherwise, letting $d = \gcd(m,n)$ and arguing as in the previous exercise but with $(x,y)\alpha = ((x+2) \bmod m, (y+1) \bmod n)$ and $(x,y)\beta = ((x+1) \bmod m, (y+2) \bmod n)$, we find the answer

$$\sum_{k=1}^{d-1} \binom{d}{k} \bigl[\gcd\bigl((2d-k)m, (k+d)n\bigr) = d \text{ or } \bigl(mn \perp 3 \text{ and } \gcd\bigl((2d-k)m, (k+d)n\bigr) = 3d\bigr)\bigr].$$

77.
```
* Permutation generator \'a la Heap
N    IS     10                   The value of n (3 or more, not large)
t    IS     $255
j    IS     $0                   8j
k    IS     $1                   8k
ak   IS     $2
aj   IS     $3

     LOC    Data_Segment
a    GREG   @                    Base address for a_0 ... a_{n-1}
A0   IS     @
A1   IS     @+8
A2   IS     @+16
     LOC    @+8*N                Space for a_0 ... a_{n-1}
c    GREG   @-8*3                Location of 8c_0
     LOC    @-8*3+8*N            8c_3 ... 8c_{n-1}, initially zero
     OCTA   -1                   8c_n = -1, a convenient sentinel
u    GREG   0                    Contents of a_0, except in inner loop
v    GREG   0                    Contents of a_1, except in inner loop
w    GREG   0                    Contents of a_2, except in inner loop

     LOC    #100
1H   STCO   0,c,k        B - A   c_k <- 0.
     INCL   k,8          B - A   k <- k + 1.
0H   LDO    j,c,k          B     j <- c_k.
     CMP    t,j,k          B
     BZ     t,1B           B     Loop if c_k = k.
     BN     j,Done         A     Terminate if c_k < 0 (k = n).
     LDO    ak,a,k       A - 1   Fetch a_k.
     ADD    t,j,8        A - 1
     STO    t,c,k        A - 1   c_k <- j + 1.
     AND    t,k,#8       A - 1
     CSZ    j,t,0        A - 1   Set j <- 0 if k is even.
     LDO    aj,a,j       A - 1   Fetch a_j.
     STO    ak,a,j       A - 1   Replace it by a_k.
     CSZ    u,j,ak       A - 1   Set u <- a_k if j = 0.
     SUB    j,j,8        A - 1   j <- j - 1.
     CSZ    v,j,ak       A - 1   Set v <- a_k if j = 0.
```

```
37        SUB    j,j,8        A−1   j ← j − 1.
38        CSZ    w,j,ak       A−1   Set w ← a_k if j = 0.
39        STO    aj,a,k       A−1   Replace a_k by what was a_j.
40  Inner PUSHJ  0,Visit       A
          ...                       (See (42))
55        PUSHJ  0,Visit       A
56        SET    t,u           A    Swap u ↔ w.
57        SET    u,w           A
58        SET    w,t           A
59        SET    k,8*3         A    k ← 3.
60        JMP    0B            A
61  Main  LDO    u,A0          1
62        LDO    v,A1          1
63        LDO    w,A2          1
64        JMP    Inner         1      ▮
```

78. Lines 31–38 become $2r-1$ instructions, lines 61–63 become r, and lines 56–58 become $3+(r-2)[r \text{ even}]$ instructions (see $\omega(r-1)$ in answer 40). The total running time is therefore $\bigl((2r!+2)A+2B+r-5\bigr)\mu+\bigl((2r!+2r+7+(r-2)[r\text{ even}])A+7B-r-4\bigr)v$, where $A=n!/r!$ and $B=n!(1/r!+\cdots+1/n!)$.

79. `SLU u,[#f],t; SLU t,a,4; XOR t,t,a; AND t,t,u; SRU u,t,4; OR t,t,u; XOR a,a,t`; here, as in the answer to exercise 1.3.1′–34, the notation '`[#f]`' denotes a register that contains the constant value `#f`.

80. `SLU u,a,t; MXOR u,[#8844221188442211],u; AND u,u,[#ff000000]; SRU u,u,t; XOR a,a,u`. This cheats, since it transforms `#12345678` to `#13245678` when $t=4$, but (45) still works.

Even faster and trickier would be a routine analogous to (42): Consider

```
PUSHJ 0,Visit; MXOR a,a,c1; PUSHJ 0,Visit; ... MXOR a,a,c5; PUSHJ 0,Visit
```

where `c1`, ..., `c5` are constants that would cause `#12345678` to become successively `#12783456`, `#12567834`, `#12563478`, `#12785634`, `#12347856`. Other instructions, executed only 1/6 or 1/24 as often, can take care of shuffling nybbles within and between bytes. Very clever, but it doesn't beat (46) in view of the `PUSHJ`/`POP` overhead.

81.
```
t  IS   $255 ;k IS $0 ;kk IS $1 ;c IS $2 ;d IS $3
   SET  k,1       k ← 1.
3H SRU  d,a,60    d ← leftmost nybble.
   SLU  a,a,4     a ← 16a mod 16^16.
   CMP  c,d,k
   SLU  kk,k,2
   SLU  d,d,kk
   OR   t,t,d     t ← t + 16^k d.
   PBNZ c,1B      Return to main loop if d ≠ k.
   INCL k,1       k ← k + 1.
   PBNZ a,3B      Return to second loop if k < n.  ▮
```

82. $\mu+\bigl(5n!+11A-(n-1)!+6\bigr)v=\bigl((5+10/n)v+O(n^{-2})\bigr)n!$, plus the visiting time, where $A=\sum_{k=1}^{n-1}k!$ is the number of times the loop at `3H` is used.

83. With suitable initialization and a 13-octabyte table, only about a dozen `MMIX` instructions are needed:

```
magic  GREG #8844221188442211
0H     ⟨Visit register a⟩
       PBN  c,Sigma
Tau    MXOR t,magic,a; ANDNL t,#ffff; JMP 1F
Sigma  SRU  t,a,20; SLU a,a,4; ANDNML a,#f00
1H     XOR  a,a,t; SLU c,c,1
2H     PBNZ c,0B; INCL p,8
3H     LDOU c,p,0; PBNZ c,0B                  ▮
```

84. Assuming that the processors all have essentially the same speed, we can let the kth processor generate all permutations of rank r for $(k-1)n!/p \le r < kn!/p$, using any method based on control tables $c_1 \ldots c_n$. The starting and ending control tables are easily computed by converting their ranks to mixed-radix notation (exercise 12).

85. We can use a technique like that of Algorithm 3.4.2P: To compute $k = r(\alpha)$, first set $a'_{a_j} \leftarrow j$ for $1 \le j \le n$ (the inverse permutation). Then set $k \leftarrow 0$, and for $j = n$, $n-1$, …, 2 (in this order) set $t \leftarrow a'_j$, $k \leftarrow kj + t - 1$, $a_t \leftarrow a_j$, $a'_{a_j} \leftarrow t$. To compute $r^{[-1]}(k)$, start with $a_1 \leftarrow 1$. Then for $j = 2$, …, $n-1$, n (in this order) set $t \leftarrow (k \bmod j)+1$, $a_j \leftarrow a_t$, $a_t \leftarrow j$, $k \leftarrow \lfloor k/j \rfloor$. [See S. Pleszczyński, *Inf. Proc. Letters* **3** (1975), 180–183; W. Myrvold and F. Ruskey, *Inf. Proc. Letters* **79** (2001), 281–284.]

Another method is preferable if we want to rank and unrank only the $n^{\underline{m}}$ *variations* $a_1 \ldots a_m$ of $\{1, \ldots, n\}$: To compute $k = r(a_1 \ldots a_m)$, start with $b_1 \ldots b_n \leftarrow b'_1 \ldots b'_n \leftarrow 1 \ldots n$; then for $j = 1$, …, m (in this order) set $t \leftarrow b'_{a_j}$, $b_t \leftarrow b_{n+1-j}$, and $b'_{b_t} \leftarrow t$; finally set $k \leftarrow 0$ and for $j = m, \ldots, 1$ (in this order) set $k \leftarrow k \times (n+1-j) + b'_{a_j} - 1$. To compute $r^{[-1]}(k)$, start with $b_1 \ldots b_n \leftarrow 1 \ldots n$; then for $j = 1, \ldots, m$ (in this order) set $t \leftarrow \bigl(k \bmod (n+1-j)\bigr) + 1$, $a_j \leftarrow b_t$, $b_t \leftarrow b_{n+1-j}$, $k \leftarrow \lfloor k/(n+1-j) \rfloor$. (See exercise 3.4.2–15 for cases with large n and small m.)

86. If $x \prec y$ and $y \prec z$, the algorithm will never move y to the left of x, nor z to the left of y, so it will never test x versus z.

87. They appear in lexicographic order; Algorithm P used a reflected Gray order.

88. Generate inverse permutations with $a'_0 < a'_1 < a'_2$, $a'_3 < a'_4 < a'_5$, $a'_6 < a'_7$, $a'_8 < a'_9$, $a'_0 < a'_3$, $a'_6 < a'_8$.

89. (a) Let $d_k = \max\{j \mid 0 \le j \le k$ and j is nontrivial$\}$, where 0 is considered nontrivial. This table is easily precomputed, because j is trivial if and only if it must follow $\{1, \ldots, j-1\}$. Set $k \leftarrow d_n$ in step V2 and $k \leftarrow d_{k-1}$ in step V5. (Assume $d_n > 0$.)

(b) Now $M = \sum_{j=1}^{n} t_j [j$ is nontrivial$]$.

(c) There are at least two topological sorts $a_j \ldots a_k$ of the set $\{j, \ldots, k\}$, and either of them can be placed after any topological sort $a_1 \ldots a_{j-1}$ of $\{1, \ldots, j-1\}$.

(d) Algorithm 2.2.3T repeatedly outputs minimal elements (elements with no predecessors), removing them from the relation graph. We use it in reverse, repeatedly removing and giving the highest labels to *maximal* elements (elements with no successors). If only one maximal element exists, it is trivial. If k and l are both maximal, they both are output before any element x with $x \prec k$ or $x \prec l$, because steps T5 and T7 keep maximal elements in a queue (not a stack). Thus if k is nontrivial and output first, element l might become trivial, but the next nontrivial element j will not be output before l; and k is unrelated to l.

(e) Let the nontrivial t's be $s_1 < s_2 < \cdots < s_r = N$. Then we have $s_j \ge 2s_{j-2}$, by (c). Consequently $M = s_2 + \cdots + s_r \le s_r(1+\frac{1}{2}+\frac{1}{4}+\cdots) + s_{r-1}(1+\frac{1}{2}+\frac{1}{4}+\cdots) < 4s_r$.

(A sharper estimate is in fact true, as observed by M. Peczarski: Let $s_0 = 1$, let the nontrivial indices be $0 = k_1 < k_2 < \cdots < k_r$, and let $k'_j = \max\{k \mid 1 \le k < k_j,\ k \not\prec k_j\}$ for $j \ge 1$. Then $k'_j \ge k_{j-1}$. There are s_j topological sorts of $\{1, \ldots, k_{j+1}\}$ that end with k_{j+1}; and there are at least s_{j-1} that end with k'_{j+1}, since each of the s_{j-1} topological sorts of $\{1, \ldots, k_j - 1\}$ can be extended. Hence

$$s_{j+1} \ \ge\ s_j + s_{j-1} \qquad \text{for } 1 \le j < r.$$

Now let $y_0 = 0$, $y_1 = F_2 + \cdots + F_r$, and $y_j = y_{j-2} + y_{j-1} - F_{r+1}$ for $1 < j < r$. Then

$$F_{r+1}(s_1 + \cdots + s_r) + \sum_{j+1}^{r-1} y_j\,(s_{r+1-j} - s_{r-j} - s_{r-1-j}) \ =\ (F_2 + \cdots + F_{r+1})s_r,$$

and each $y_j = F_{r+1} - 2F_j + (-1)^j F_{r+1-j}$ is nonnegative. Hence $s_1 + \cdots + s_r \le \bigl((F_2 + \cdots + F_{r+1})/F_{r+1}\bigr)s_r \approx 2.6s_r$. The following exercise shows that this bound is best possible.)

90. The number N of such permutations is F_{n+1} by exercise 5.2.1–25. Therefore $M = F_{n+1} + \cdots + F_2 = F_{n+3} - 2 \approx \phi^2 N$. Notice incidentally that all such permutations satisfy $a_1 \ldots a_n = a'_1 \ldots a'_n$. They can be arranged in a Gray path (exercise 7.2.1.1–89).

91. Since $t_j = (j-1)(j-3)\ldots(2 \text{ or } 1)$, we find $M = \bigl(1 + 2/\sqrt{\pi n} + O(1/n)\bigr)N$.

Note: The inversion tables $c_1 \ldots c_{2n}$ for permutations satisfying (49) are characterized by the conditions $c_1 = 0$, $0 \le c_{2k} \le c_{2k-1}$, $0 \le c_{2k+1} \le c_{2k-1} + 1$.

92. The total number of pairs (R, S), where R is a partial ordering and S is a linear ordering that includes R, is equal to P_n times the expected number of topological sorts; it is also Q_n times $n!$. So the answer is $n!\,Q_n/P_n$.

We will discuss the computation of P_n and Q_n in Section 7.2.3. For $1 \le n \le 12$ the expectation turns out to be approximately

$$(1,\ 1.33,\ 2.21,\ 4.38,\ 10.1,\ 26.7,\ 79.3,\ 262,\ 950,\ 3760,\ 16200,\ 74800).$$

Asymptotic values as $n \to \infty$ have been deduced by Brightwell, Prömel, and Steger [*J. Combinatorial Theory* **A73** (1996), 193–206], but the limiting behavior is quite different from what happens when n is in a practical range. The values of Q_n were first determined for $n \le 5$ by S. P. Avann [*Æquationes Math.* **8** (1972), 95–102].

93. The basic idea is to introduce dummy elements $n+1$ and $n+2$ with $j \prec n+1$ and $j \prec n+2$ for $1 \le j \le n$, and to find all topological sorts of such an extended relation via adjacent interchanges; then take every *second* permutation, suppressing the dummy elements. An algorithm similar to Algorithm V can be used, but with a recursion that reduces n to $n-2$ by inserting $n-1$ and n among $a_1 \ldots a_{n-2}$ in all possible ways, assuming that $n-1 \not\prec n$, occasionally swapping $n+1$ with $n+2$. [See G. Pruesse and F. Ruskey, *SICOMP* **23** (1994), 373–386. A loopless implementation has been described by Canfield and Williamson, *Order* **12** (1995), 57–75.]

94. The case $n = 3$ illustrates the general idea of a pattern that begins with $1 \ldots (2n)$ and ends with $1(2n)2(2n{-}1)\ldots n(n{+}1)$: 123456, 123546, 123645, 132645, 132546, 132456, 142356, 142536, 142635, 152634, 152436, 152346, 162345, 162435, 162534.

Matchings can also be regarded as involutions of $\{1, \ldots, 2n\}$ that have n cycles. With that representation this pattern involves two transpositions per step.

Notice that the C inversion tables of the permutations just listed are respectively 000000, 000100, 000200, 010200, 010100, 010000, 020000, 020100, 020200, 030200, 030100, 030000, 040000, 040100, 040200. In general, $C_1 = C_3 = \cdots = C_{2n-1} = 0$

and the n-tuples $(C_2, C_4, \ldots, C_{2n})$ run through a reflected Gray code on the radices $(2n-1, 2n-3, \ldots, 1)$. Thus the generation process can easily be made loopless if desired. [See Timothy Walsh, *J. Combinatorial Math. and Combinatorial Computing* **36** (2001), 95–118, Section 1.]

Note: Algorithms to generate all matchings go back to J. F. Pfaff [*Abhandlungen Akad. Wissenschaften* (Berlin: 1814–1815), 124–125], who described two such procedures: His first method was lexicographic, which also corresponds to lexicographic order of the C inversion tables; his second method corresponds to *colex* order of those tables. Even and odd permutations alternate in both cases.

95. Generate inverse permutations with $a'_1 < a'_n > a'_2 < a'_{n-1} > \cdots$, using Algorithm V. (See exercise 5.1.4–23 for the number of solutions.)

96. For example, we can start with $a_1 \ldots a_{n-1}a_n = 2 \ldots n1$ and $b_1b_2 \ldots b_nb_{n+1} = 12 \ldots n1$, and use Algorithm P to generate the $(n-1)!$ permutations $b_2 \ldots b_n$ of $\{2, \ldots, n\}$. Just after that algorithm swaps $b_i \leftrightarrow b_{i+1}$, we set $a_{b_{i-1}} \leftarrow b_i$, $a_{b_i} \leftarrow b_{i+1}$, $a_{b_{i+1}} \leftarrow b_{i+2}$, and visit $a_1 \ldots a_n$.

97. Use Algorithm X, with $t_k(a_1, \ldots, a_k) =$ '$a_k \neq k$'.

98. Using the notation of exercise 47, we have $N_k = \sum \binom{k}{j}(-1)^j(n-j)^{\underline{k-j}}$ by the method of inclusion and exclusion (exercise 1.3.3–26). If $k = O(\log n)$ then $N_{n-k} = (n!\,e^{-1}/k!)(1 + O(\log n)^2/n)$; hence $A/n! \approx (e-1)/e$ and $B/n! \approx 1$. The number of memory references, under the assumptions of answer 48, is therefore $\approx A + B + 3A + B - N_n + 3A \approx n!\,(9 - \frac{8}{e}) \approx 6.06n!$, about 16.5 per derangement. [See S. G. Akl, *BIT* **20** (1980), 2–7, for a similar method.]

99. Suppose L_n generates $D_n \cup D_{n-1}$, beginning with $(1\ 2\ \ldots\ n)$, then $(2\ 1\ \ldots\ n)$, and ending with $(1\ \ldots\ n{-}1)$; for example, $L_3 = (1\,2\,3)$, $(2\,1\,3)$, $(1\,2)$. Then we can generate D_{n+1} as $K_{nn}, \ldots, K_{n2}, K_{n1}$, where $K_{nk} = (1\ 2\ \ldots\ n)^{-k}(n\ n{+}1)L_n(1\ 2\ \ldots\ n)^k$; for example, D_4 is

$$(1\,2\,3\,4),\ (2\,1\,3\,4),\ (1\,2)(3\,4),\ (3\,1\,2\,4),\ (1\,3\,2\,4),\ (3\,1)(2\,4),\ (2\,3\,1\,4),\ (3\,2\,1\,4),\ (2\,3)(1\,4).$$

Notice that K_{nk} begins with the cycle $(k{+}1\ \ldots\ n\ 1\ \ldots\ k\ n{+}1)$ and ends with $(k{+}1\ \ldots\ n\ 1\ \ldots\ k{-}1)(k\ n{+}1)$; so premultiplication by $(k{-}1\ k)$ takes us from K_{nk} to $K_{n(k-1)}$. Also, premultiplication by $(1\ n)$ will return from the last element of D_{n+1} to the first. Premultiplication by $(1\ 2\ n{+}1)$ takes us from the last element of D_{n+1} to $(2\ 1\ 3\ \ldots\ n)$, from which we can return to $(1\ 2\ \ldots\ n)$ by following the cycle for D_n backwards, thereby completing the list L_{n+1} as desired.

100. Use Algorithm X, with $t_k(a_1, \ldots, a_k) =$ '$p > 0$ or $l[q] \neq k+1$'.

Notes: The number of indecomposable permutations is $[z^n]\,(1 - 1/\sum_{k=0}^{\infty} k!\,z^k)$; see L. Comtet, *Comptes Rendus Acad. Sci.* **A275** (Paris, 1972), 569–572. It appears likely that the indecomposable permutations can be generated by adjacent transpositions; for example, when $n = 4$ they are 3142, 3412, 3421, 3241, 2341, 2431, 4231, 4321, 4312, 4132, 4123, 4213, 2413.

101. Here is a lexicographic involution generator analogous to Algorithm X.

Y1. [Initialize.] Set $a_k \leftarrow k$ and $l_{k-1} \leftarrow k$ for $1 \leq k \leq n$. Then set $l_n \leftarrow 0$, $k \leftarrow 1$.

Y2. [Enter level k.] If $k > n$, visit $a_1 \ldots a_n$ and go to Y3. Otherwise set $p \leftarrow l_0$, $u_k \leftarrow p$, $l_0 \leftarrow l_p$, $k \leftarrow k+1$, and repeat this step. (We have decided to let $a_p = p$.)

Y3. [Decrease k.] Set $k \leftarrow k-1$, and terminate if $k=0$. Otherwise set $q \leftarrow u_k$ and $p \leftarrow a_q$. If $p=q$, set $l_0 \leftarrow q$, $q \leftarrow 0$, $r \leftarrow l_p$, and $k \leftarrow k+1$ (preparing to make $a_p > p$). Otherwise set $l_{u_{k-1}} \leftarrow q$, $r \leftarrow l_q$ (preparing to make $a_p > q$).

Y4. [Increase a_p.] If $r=0$ go to Y5. Otherwise set $l_q \leftarrow l_r$, $u_{k-1} \leftarrow q$, $u_k \leftarrow r$, $a_p \leftarrow r$, $a_q \leftarrow q$, $a_r \leftarrow p$, $k \leftarrow k+1$, and go to Y2.

Y5. [Restore a_p.] Set $l_0 \leftarrow p$, $a_p \leftarrow p$, $a_q \leftarrow q$, $k \leftarrow k-1$, and return to Y3. ▌

Let $t_{n+1} = t_n + nt_{n-1}$, $a_{n+1} = 1 + a_n + na_{n-1}$, $t_0 = t_1 = 1$, $a_0 = 0$, $a_1 = 1$. (See Eq. 5.1.4–(40).) Step Y2 is performed t_n times with $k > n$ and a_n times with $k \le n$. Step Y3 is performed a_n times with $p = q$ and $a_n + t_n$ times altogether. Step Y4 is performed $t_n - 1$ times; step Y5, a_n times. The total number of mems for all t_n outputs is therefore approximately $11a_n + 12t_n$, where $a_n < 1.25331414t_n$. (Optimizations are clearly possible if speed is essential.)

102. We construct a list L_n that begins with () and ends with $(n{-}1\ n)$, starting with $L_3 = ()$, $(1\ 2)$, $(1\ 3)$, $(2\ 3)$. If n is odd, L_{n+1} is L_n, K_{n1}^R, K_{n2}, …, K_{nn}^R, where $K_{nk} = (k\ \ldots\ n)^- L_{n-1}(k\ \ldots n)(k\ n{+}1)$. For example,

$$L_4 \;=\; (),\,(1\ 2),\,(1\ 3),\,(2\ 3),\,(2\ 3)(1\ 4),\,(1\ 4),\,(2\ 4),\,(1\ 3)(2\ 4),\,(1\ 2)(3\ 4),\,(3\ 4).$$

If n is even, L_{n+1} is L_n, $K_{n(n-1)}$, $K_{n(n-2)}^R$, …, K_{n1}, $(1\ n{-}2)L_{n-1}^R(1\ n{-}2)(n\ n{+}1)$.

For further developments, see the article by Walsh cited in answer 94.

103. The following elegant solution by Carla Savage needs only $n-2$ different operations ρ_j, for $1 < j < n$, where ρ_j replaces $a_{j-1}a_ja_{j+1}$ by $a_{j+1}a_{j-1}a_j$ when j is even, $a_ja_{j+1}a_{j-1}$ when j is odd. We may assume that $n \ge 4$; let $A_4 = (\rho_3\rho_2\rho_2\rho_3)^3$. In general A_n will begin and end with ρ_{n-1}, and it will contain $2n-2$ occurrences of ρ_{n-1} altogether. To get A_{n+1}, replace the kth ρ_{n-1} of A_n by $\rho_n A'_n \rho_n$, where $k = 1, 2, 4, \ldots, 2n-2$ if n is even and $k = 1, 3, \ldots, 2n-3, 2n-2$ if n is odd, and where A'_n is A_n with its first or last element deleted. Then, if we begin with $a_1 \ldots a_n = 1 \ldots n$, the operations ρ_{n-1} of A_n will cause position a_n to run through the successive values $n \to p_1 \to n \to p_2 \to \cdots \to p_{n-1} \to n$, where $p_1 \ldots p_{n-1} = (n{-}1 - [n\,\mathrm{even}]) \ldots 4213 \ldots (n{-}1 - [n\,\mathrm{odd}])$; the final permutation will again be $1 \ldots n$.

104. (a) A well-balanced permutation has $\sum_{k=1}^n ka_k = n(n+1)^2/4$, an integer.

(b) Replace k by a_k when summing over k.

(c) A fairly fast way to count, when n is not too large, can be based on the streamlined plain-change algorithm of exercise 16, because the quantity $\sum ka_k$ changes in a simple way with each adjacent interchange, and because $n-1$ of every n steps are "hunts" that can be done rapidly. We can save half the work by considering only permutations in which 1 precedes 2. The values for $1 \le n \le 15$ are 0, 0, 0, 2, 6, 0, 184, 936, 6688, 0, 420480, 4298664, 44405142, 0, 6732621476.

105. (a) For each permutation $a_1 \ldots a_n$, insert $\prec$ between a_j and a_{j+1} if $a_j > a_{j+1}$; insert either $\equiv$ or $\prec$ between them if $a_j < a_{j+1}$. (A permutation with k "ascents" therefore yields 2^k weak orders. Weak orders are sometimes called "preferential arrangements; exercise 5.3.1–4 shows that there are approximately $n!/(2(\ln 2)^{n+1})$ of them. A Gray code for weak orders, in which each step changes $\prec \leftrightarrow \equiv$ and/or $a_j \leftrightarrow a_{j+1}$, can be obtained by combining Algorithm P with Gray binary code at the ascents.

(b) Start with $a_1 \ldots a_na_{n+1} = 0 \ldots 00$ and $a_0 = -1$. Perform Algorithm L until it stops with $j = 0$. Find k such that $a_1 > \cdots > a_k = a_{k+1}$, and terminate if $k = n$. Otherwise set $a_l \leftarrow a_{k+1} + 1$ for $1 \le l \le k$ and go to step L4. [See M. Mor and A. S. Fraenkel, *Discrete Math.* **48** (1984), 101–112. Weak ordering sequences are characterized by the property that, if k appears and $k > 0$, then $k-1$ also appears.]

106. All weak ordering sequences can be obtained by a sequence of elementary operations $a_i \leftrightarrow a_j$ or $a_i \leftarrow a_j$. (Perhaps one could actually restrict the transformations further, allowing only $a_j \leftrightarrow a_{j+1}$ or $a_j \leftarrow a_{j+1}$ for $1 \le j < n$.)

107. Every step increases the quantity $\sum_{k=1}^{n} 2^k [a_k = k]$, as noted by H. S. Wilf, so the game must terminate. At least three approaches to the solution are plausible: one bad, one good, and one better.

The bad one is to play the game on all 13! shuffles and to record the longest. This method does produce the correct answer; but 13! is 6,227,020,800, and the average game lasts ≈ 8.728 steps.

The good one [A. Pepperdine, *Math. Gazette* **73** (1989), 131–133] is to play backwards, starting with the final position $1{*}\ldots{*}$ where $*$ denotes a card that is face down; we will turn a card up only when its value becomes relevant. To move backward from a given position $a_1 \ldots a_n$, consider all $k > 1$ such that either $a_k = k$ or $a_k = *$ and k has not yet turned up. Thus the next-to-last positions are $21{*}\ldots{*}$, $3{*}1{*}\ldots{*}$, $\ldots$, $n{*}\ldots{*}1$. Some positions (like $6{*}{*}213$ for $n = 6$) have no predecessors, even though we haven't turned all the cards up. It is easy to explore the tree of potential backwards games systematically, and one can in fact show that the number of nodes with t $*$'s is exactly $(n-1)!/t!$. Hence the total number of nodes considered is exactly $\lfloor (n-1)!\,e \rfloor$. When $n = 13$ this is 1,302,061,345.

The better one is to play forwards, starting with initial position $*\ldots*$ and turning over the top card when it is face down, running through all $(n-1)!$ permutations of $\{2, \ldots, n\}$ as cards are turned. If the bottom $n-m$ cards are known to be equal to $(m{+}1)(m{+}2)\ldots n$, in that order, at most $f(m)$ further moves are possible; thus we need not pursue a line of play any further if it cannot last long enough to be interesting. A permutation generator like Algorithm X allows us to share the computation for all permutations with the same prefix and to reject unimportant prefixes. The card in position j need not take the value j when it is turned. When $n = 13$ this method needs to consider only respectively $(1, 11, 940, 6960, 44745, 245083, 1118216, 4112676, 11798207, 26541611, 44380227, 37417359)$ branches at levels $(1, 2, \ldots, 12)$ and to make a total of only 482,663,902 forward moves. Although it repeats some lines of play, the early cutoffs of unprofitable branches make it run more than 11 times faster than the backward method when $n = 13$.

The unique way to attain length 80 is to start with 2 9 4 5 11 12 10 1 8 13 3 6 7.

108. This result holds for any game in which

$$a_1 \ldots a_n \to a_k a_{p(k,2)} \ldots a_{p(k,k-1)} a_1 a_{k+1} \ldots a_n$$

when $a_1 = k$, where $p(k,2) \ldots p(k,k-1)$ is an arbitrary permutation of $\{2, \ldots, k-1\}$. Suppose a_1 takes on exactly m distinct values $d(1) < \cdots < d(m)$ during a play of the game; we will prove that at most F_{m+1} permutations occur, including the initial shuffle. This assertion is obvious when $m = 1$.

Let $d(j)$ be the initial value of $a_{d(m)}$, where $j < m$, and suppose $a_{d(m)}$ changes on step r. If $d(j) = 1$, the number of permutations is $r + 1 \le F_m + 1 \le F_{m+1}$. Otherwise $r \le F_{m-1}$, and at most F_m further permutations follow step r. [*SIAM Review* **19** (1977), 239–241.]

The values of $f(n)$ for $1 \le n \le 16$ are (0, 1, 2, 4, 7, 10, 16, 22, 30, 38, 51, 65, 80, 101, 113, 139), and they are attainable in respectively (1, 1, 2, 2, 1, 5, 2, 1, 1, 1, 1, 1, 1, 4, 6, 1) ways. The unique longest-winded permutation for $n = 16$ is

$$9\ 12\ 6\ 7\ 2\ 14\ 8\ 1\ 11\ 13\ 5\ 4\ 15\ 16\ 10\ 3.$$

109. The forward method of answer 107 suggests that $f(n)$ probably grows at least as fast as $n \log n$ (by comparison with coupon collecting).

110. For $0 \le j \le 9$ construct the bit vectors $A_j = [a_j \in S_1] \ldots [a_j \in S_m]$ and $B_j = [j \in S_1] \ldots [j \in S_m]$. Then the number of j such that $A_j = v$ must equal the number of k such that $B_k = v$, for all bit vectors v. And if so, the values $\{a_j \mid A_j = v\}$ should be assigned to permutations of $\{k \mid B_k = v\}$ in all possible ways.

For example, the bit vectors in the given problem are

$$(A_0, \ldots, A_9) = (\mathtt{9,6,8,b,5,4,0,a,2,0}), \qquad (B_0, \ldots, B_9) = (\mathtt{5,0,8,6,2,a,4,b,9,0}),$$

in hexadecimal notation; hence $a_0 \ldots a_9 = 8327061549$ or 8327069541.

In a larger problem we would keep the bit vectors in a hash table. It would be better to give the answer in terms of equivalence classes, not permutations; indeed, this problem has comparatively little to do with permutations.

111. In the directed graph with $n!/2$ vertices $a_1 \ldots a_{n-2}$ and $n!$ arcs $a_1 \ldots a_{n-2} \to a_2 \ldots a_{n-1}$ (one for each permutation $a_1 \ldots a_n$), each vertex has in-degree 2 and out-degree 2. Furthermore, from paths like $a_1 \ldots a_{n-2} \to a_2 \ldots a_{n-1} \to a_3 \ldots a_n \to a_4 \ldots a_n a_2 \to a_5 \ldots a_n a_2 a_1 \to \cdots \to a_2 a_1 a_3 \ldots a_{n-2}$, we can see that any vertex is reachable from any other. Therefore an Eulerian trail exists by Theorem 2.3.4.2D, and such a trail clearly is equivalent to a universal cycle of permutations. The lexicographically smallest example when $n = 4$ is (123124132134214324314234).

112. By exercise 2.3.4.2–22 it suffices to count the oriented trees rooted at $1 2 \ldots (n{-}2)$, in the digraph of the preceding answer; and those trees can be counted by exercise 2.3.4.2–19. For $n \le 6$ the numbers U_n turn out to be tantalizingly simple: $U_2 = 1$, $U_3 = 3$, $U_4 = 2^7 \cdot 3$, $U_5 = 2^{33} \cdot 3^8 \cdot 5^3$, $U_6 = 2^{190} \cdot 3^{49} \cdot 5^{33}$. (Here we consider (121323) to be the same cycle as (213231), but different from (131232).)

Mark Cooke has discovered the following instructive way to compute these values efficiently: Notice first that a universal cycle of permutations is also equivalent to a *Hamiltonian* cycle on the Cayley graph with generators $\sigma = (1\ 2\ \ldots\ n)$ and $\rho = (1\ 2\ \ldots\ n{-}1)$. For example, the cycle in the previous answer for $n = 4$ corresponds to the cycle $\sigma^3\rho^2\sigma\rho\sigma^2\rho^2\sigma^3\rho\sigma^2\rho^2\sigma\rho\sigma^2\rho$.

Now consider the $n! \times n!$ matrix $M = 2I - R - S$, where $R_{\pi\pi'} = [\pi' = \pi\rho]$ and $S_{\pi\pi'} = [\pi' = \pi\sigma]$. There is a matrix H such that $H^- RH$ and $H^- SH$ each have block diagonal form consisting of k_λ copies of $k_\lambda \times k_\lambda$ matrices R_λ and S_λ, for each partition λ of n, where k_λ is $n!$ divided by the product of the hook lengths of shape λ (Theorem 5.1.4H), and where R_λ and S_λ are matrix representations of ρ and σ based on Young tableaux. [A proof can be found in Bruce Sagan, *The Symmetric Group* (Pacific Grove, Calif.: Wadsworth & Brooks/Cole, 1991).] For example, when $n = 3$ we have

$$R = \begin{pmatrix} 0&0&0&1&0&0\\ 0&0&0&0&0&1\\ 0&0&0&0&1&0\\ 1&0&0&0&0&0\\ 0&0&1&0&0&0\\ 0&1&0&0&0&0 \end{pmatrix}, \quad S = \begin{pmatrix} 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 1&0&0&0&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1\\ 0&0&0&1&0&0 \end{pmatrix}, \quad H = \begin{pmatrix} 1&1&1&-1&1&0\\ 1&1&-1&0&0&-1\\ 1&1&0&1&-1&1\\ 1&-1&-1&1&0&1\\ 1&-1&1&0&1&-1\\ 1&-1&0&-1&-1&0 \end{pmatrix},$$

$$H^-RH=\begin{pmatrix}1&0&0&0&0&0\\0&-1&0&0&0&0\\0&0&0&1&0&0\\0&0&1&0&0&0\\0&0&0&0&0&1\\0&0&0&0&1&0\end{pmatrix},\qquad H^-SH=\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&0&-1&0&0\\0&0&1&-1&0&0\\0&0&0&0&0&-1\\0&0&0&0&1&-1\end{pmatrix}$$

when rows and columns are indexed by the respective permutations 1, σ, σ^2, ρ, $\rho\sigma$, $\rho\sigma^2$; here $k_3=k_{111}=1$ and $k_{21}=2$. Therefore the eigenvalues of M are the union, over λ, of k_λ-fold repeated eigenvalues of the $k_\lambda\times k_\lambda$ matrices $2I-R_\lambda-S_\lambda$. In the example, the eigenvalues of (0), (2), and $\left(\begin{smallmatrix}2&0\\-2&3\end{smallmatrix}\right)$ twice are $\{0\}$, $\{2\}$, and $\{2,3\}$ twice.

The eigenvalues of M are directly related to those of the matrix A in exercise 2.3.4.2–19. Indeed, each eigenvector of A yields an eigenvector of M, if we equate the components for permutations π and $\pi\rho\sigma^-$, because rows π and $\pi\rho\sigma^-$ of $R+S$ are equal. For example,

$$A=\begin{pmatrix}2&-1&-1\\-1&2&-1\\-1&-1&2\end{pmatrix}\text{ has eigenvectors }\begin{pmatrix}1\\1\\1\end{pmatrix},\begin{pmatrix}1\\-1\\0\end{pmatrix},\begin{pmatrix}1\\0\\-1\end{pmatrix}\text{ for eigenvalues 0, 3, 3,}$$

yielding the eigenvectors $(1,1,1,1,1,1)^T$, $(1,-1,0,0,-1,1)^T$, $(1,0,-1,-1,0,1)^T$ of M for the same eigenvalues. And M has $n!/2$ additional eigenvectors, with all components zero except those indexed by π and $\pi\sigma^-\rho$ for some π, because only rows $\pi\rho^-$ and $\pi\sigma^-$ of $R+S$ have nonzero entries in columns π and $\pi\sigma^-\rho$; such vectors yield $n!/2$ additional eigenvalues, all equal to 2.

Therefore U_n, which is $2/n!$ times the product of the nonzero eigenvalues of A, is $2^{1-n!/2}/n!$ times the product of the nonzero eigenvalues of M.

Unfortunately the small-prime-factor phenomenon does not continue; U_7 equals $2^{1217}3^{123}5^{119}7^{5}11^{28}43^{35}73^{20}79^{21}109^{35}$, and U_9 is divisible by 59229013196333^{168}.

At least one of these cycles must almost surely be easy to describe and to compute, as we did for de Bruijn cycles in Section 7.2.1.1. But no simple construction has yet been found.

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.