

It may be difficult to insert a new node here .

21. Algorithm A does the job in order $N \log N$ steps (see exercise 5); the following algorithm creates the same trees in $O(N)$ steps, using an interesting iterative rendition of a recursive method. We use three auxiliary lists:

- D_1, \dots, D_l (a binary counter that essentially controls the recursion);
- J_1, \dots, J_l (a list of pointers to "juncture nodes");
- T_1, \dots, T_l (a list of pointers to trees).

Here $l = \lceil \log_2 (N + 1) \rceil$. For convenience the algorithm also sets $D_0 \leftarrow 1$, $J_0 \leftarrow J_{l+1} \leftarrow \Lambda$.

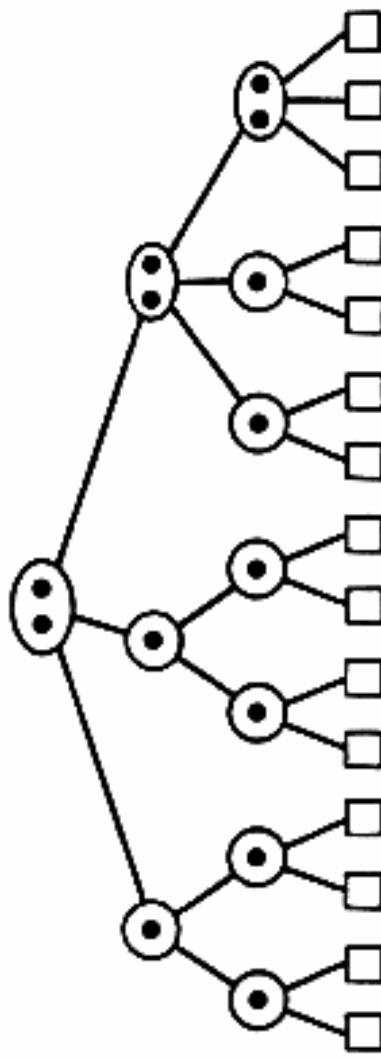
- G1. [Initialize.] Set $l \leftarrow 0$, $J_0 \leftarrow J_1 \leftarrow \Lambda$, $D_0 \leftarrow 1$.
- G2. [Get next item.] Let P point to the next input node. (We may invoke another coroutine in order to obtain P .) If there is no more input, go to G5. Otherwise, set $k \leftarrow 1$, $Q \leftarrow \Lambda$, and interchange $P \leftrightarrow J_1$.
- G3. [Carry.] If $k > l$ (equivalently if $P = \Lambda$), set $l \leftarrow l + 1$, $D_k \leftarrow 1$, $T_k \leftarrow Q$, $J_{k+1} \leftarrow \Lambda$, and return to G2. Otherwise set $D_k \leftarrow 1 - D_k$, interchange $Q \leftrightarrow T_k$, $P \leftrightarrow J_{k+1}$, and increase k by 1. If now $D_{k-1} = 0$, repeat this step.
- G4. [Concatenate.] Set $\text{LLINK}(P) \leftarrow T_k$, $\text{RLINK}(P) \leftarrow Q$, $B(P) \leftarrow 0$, $T_k \leftarrow P$, and return to G2.
- G5. [Finish up.] Set $\text{LLINK}(J_k) \leftarrow T_k$, $\text{RLINK}(J_k) \leftarrow J_{k-1}$, $B(J_k) \leftarrow 1 - D_{k-1}$, for $1 \leq k \leq l$. Then terminate the algorithm (J_l points to the root of the desired tree). ■

Step G3 is executed $2N - v(N)$ times, where $v(N)$ is the number of 1's in the binary representation of N .

22. The height of a weight-balanced tree with N internal nodes always lies between $\log_2 (N + 1)$ and $2 \log_2 (N + 1)$. To get this upper bound, note that the heavier subtree of the root has at most $(N + 1)/\sqrt{2}$ external nodes.
23. (a) Form a tree whose right subtree is a complete binary tree with $2^n - 1$ nodes, and whose left subtree is a Fibonacci tree with $F_{n+1} - 1$ nodes. (b) Form a weight balanced tree whose right subtree is about $2 \log_2 N$ levels high and whose left subtree is about $\log_2 N$ levels high (cf. exercise 22).
24. Consider a smallest tree which satisfies the condition but is not perfectly balanced. Then its left and right subtrees are perfectly balanced, so they have 2^l and 2^r external nodes, respectively, where $l \neq r$. But this contradicts the stated condition.
25. After inserting a node at the bottom of the tree, we work up from the bottom to check the weight balance at each node on the search path. Suppose imbalance occurs at node A in (1), after having inserted a new node in the right subtree, where B and its subtrees are weight-balanced. Then a single rotation will restore the balance unless

$(|\alpha| + |\beta|)/|\gamma| > \sqrt{2} + 1$, where $|x|$ denotes the number of external nodes in a tree x . But in this case it can be shown that a double rotation will suffice.

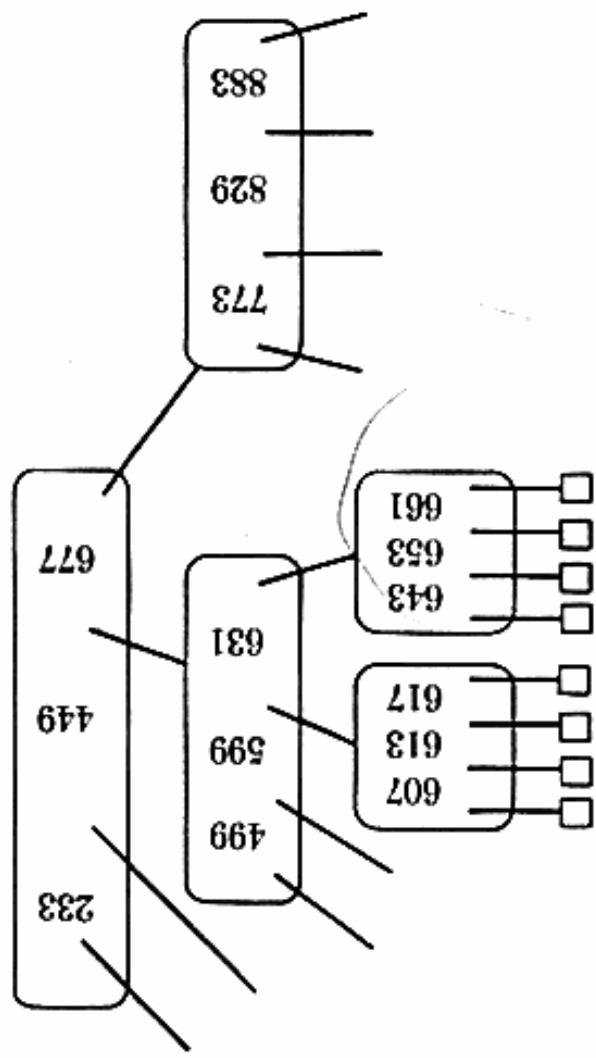
27. It is sometimes necessary to make two comparisons in nodes that contain two keys. The worst case occurs in a tree like the following, which sometimes needs $2 \log_2(N+2) - 2$ comparisons:



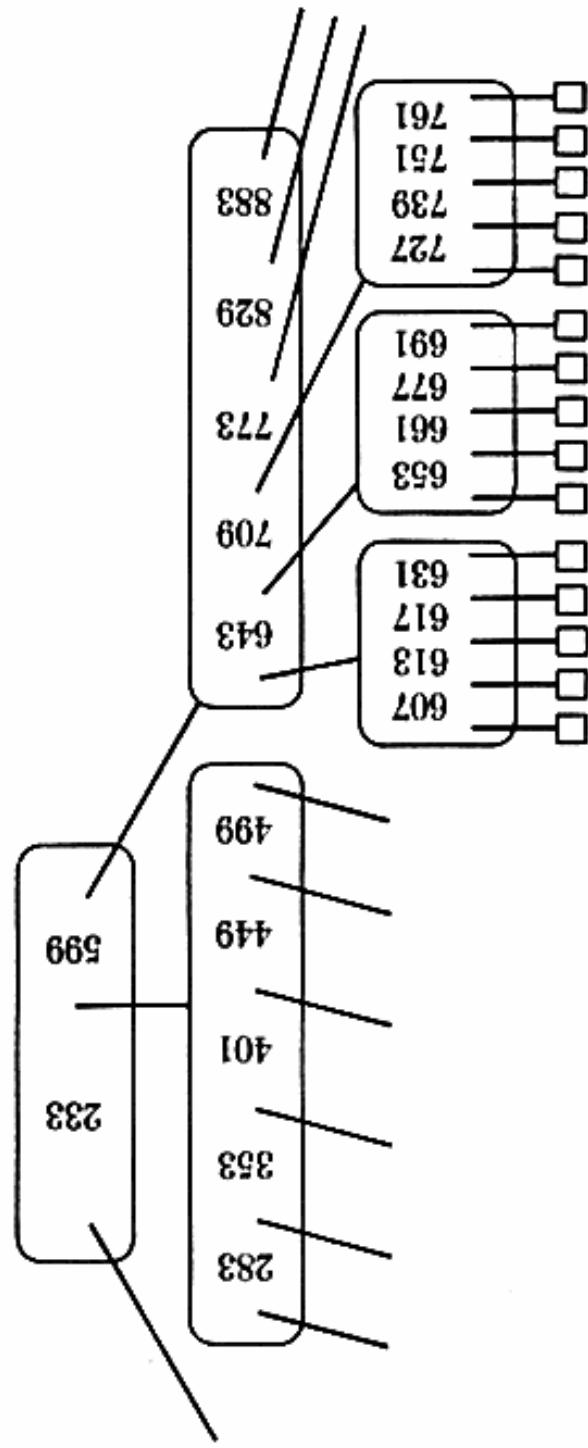
30. For best-fit, arrange the records in order of size, with an arbitrary rule to break ties in case of equality. (Cf. exercise 2.5–9.) For first fit, arrange the records in order of location, with an extra field in each node telling the size of the largest area in the left subtree of that node. This extra field can be maintained under insertions and deletions. (Although the running time is $O(\log n)$, it probably still doesn't beat the "ROVER" method of exercise 2.5–6 in practice.)

SECTION 6.2.4

1. Two nodes split:



2. Altered nodes:



(Of course a B^* -tree would have no 3-key nodes, although Fig. 30 does.)

3. (a) $1 + 2 \cdot 50 + 2 \cdot 51 \cdot 50 + 2 \cdot 51 \cdot 51 \cdot 50 = 2 \cdot 51^3 - 1 = 265301$. (b) $1 + 2 \cdot 50 + (2 \cdot 51 \cdot 100 - 100) + ((2 \cdot 51 \cdot 101 - 100) \cdot 100 - 100) = 101^3 = 1030301$. (c) $1 + 2 \cdot 66 + (2 \cdot 67 \cdot 66 + 2) + (2 \cdot 67 \cdot 67 \cdot 66 + 2 \cdot 67) = 601661$. (Less than (b)!)

4. Before splitting a non-root node, make sure that it has two full brothers, then split these three nodes into four. The root should split only when it has more than $3\lfloor(3m - 3)/4\rfloor$ keys.

5. 450. The worst case occurs if we have 1005 characters and the key to be passed to Father must be 50 characters long: 445 chars + ptr + 50-char key + ptr + 50-char key + ptr + 445 chars.

6. Yes, for example we could replace each K_i in (1) by i plus the number of keys in subtrees P_0, \dots, P_{i-1} . The search, insertion, and deletion algorithms can be modified appropriately.

7. If the key to be deleted is not on level $l - 1$, replace it by its successor and delete the successor. To delete a key on level $l - 1$, we simply erase it; if this makes the node too empty, we look at its right (or left) brother, and "underflow," i.e., move keys in from the brother so that both nodes have approximately the same amount of data. This underflow operation will fail only if the brother was minimally full, but in that case the two nodes can be collapsed into one (together with one key from their father); such a collapsing may cause the father in turn to underflow, etc.

9. Brief sketch: Extend the paging scheme so that exclusive access to buffers is given to one user at a time; the search, insertion, and deletion algorithms must be carefully modified so that such exclusive access is granted only for a limited time when absolutely necessary, and such that no deadlocks can occur. When a particular user needs to update a father page, it will be necessary for him to see if it has changed or moved since he last observed it.

10. Given a tree \mathfrak{J} with N internal nodes, let there be $a_k^{(j)}$ external nodes that require k accesses and whose father node belongs to a page containing j keys; and let $A^{(j)}(z)$ be the corresponding generating function. Thus $A^{(1)}(1) + \dots + A^{(M)}(1) = N + 1$. (Note that $a_k^{(j)}$ is a multiple of $j + 1$, for $1 \leq j < M$.) The next random insertion leads to $N + 1$ equally probable trees, whose generating functions are obtained by decreasing some coefficient $a_k^{(j)}$ by $j + 1$ and adding $j + 2$ to $a_k^{(j+1)}$; or (if $j = M$) by decreasing some $a_k^{(M)}$ by 1 and adding 2 to $a_{k+1}^{(1)}$. Now $B_N^{(j)}(z)$ is $(N + 1)^{-1}$ times the sum, over all \mathfrak{J} , of the generating function $A^{(j)}(z)$ for \mathfrak{J} times the probability that \mathfrak{J} occurs; the stated recurrence relations follow.

The recurrence has the form

$$(B_N^{(1)}(z), \dots, B_N^{(M)}(z))^T = (I + (N + 1)^{-1}\mathbf{W}(z))(B_{N-1}^{(1)}(z), \dots, B_{N-1}^{(M)}(z))^T = \dots = g_N(\mathbf{W}(z))(0, \dots, 0, 1)^T,$$

where

$$g_n(x) = (1 + x/(n + 1)) \cdots (1 + x/2) = \frac{(x+n+1)}{(n+1)} / (x + 1).$$

It follows that $C'_N = (1, \dots, 1)(B_N^{(1)'}(1), \dots, B_N^{(M)'}(1))^T = 2B_{N-1}^{(M)}(1)/(N + 1) + C'_{N-1} = 2f_N(\mathbf{W})_{MM}$, where $f_n(x) = g_{n-1}(x)/(n + 1) + \dots + g_0(x)/2 = (g_n(x) - 1)/x$, and $\mathbf{W} = \mathbf{W}(1)$. (The subscript MM denotes the lower right corner element of the matrix.) Now $\mathbf{W} = \mathbf{S}^{-1} \text{diag}(\lambda_1, \dots, \lambda_M) \mathbf{S}$, for some matrix \mathbf{S} , where $\text{diag}(\lambda_1, \dots, \lambda_M)$ denotes the diagonal matrix whose entries are the roots of $X(\lambda) = (\lambda + 2) \dots$

$\lambda + M + 1) - (M + 1)!$. (These roots are distinct, since $\chi(\lambda) = \chi'(\lambda) = 0$ implies $1/(\lambda + 2) + \dots + 1/(\lambda + M + 1) = 0$; the latter can hold only when λ is real, and $-M - 1 < \lambda < -2$, which implies that $|\lambda + 2| \dots |\lambda + M + 1| < (M + 1)!$, a contradiction.) If $p(x)$ is any polynomial, $p(\mathbf{W}) = p(\mathbf{S}^{-1} \mathbf{diag}(\lambda_1, \dots, \lambda_M) \mathbf{S}) = \mathbf{S}^{-1} \mathbf{diag}(p(\lambda_1), \dots, p(\lambda_M)) \mathbf{S}$; hence the lower right corner element of $p(\mathbf{W})$ has the form $c_1 p(\lambda_1) + \dots + c_M p(\lambda_M)$ for some constants c_1, \dots, c_M independent of p . These constants may be evaluated by setting $p(\lambda) = \chi(\lambda)/(\lambda - \lambda_i)$; since $(\mathbf{W}^k)_{MM} = (-2)^k$ for $0 \leq k \leq M - 1$, we have $p(\mathbf{W})_{MM} = p(-2) = (M + 1)!/(\lambda_i + 2) = c_i p(\lambda_i) = c_i \chi'(\lambda_i) = c_i (M + 1)! (1/\lambda_i + 2) + \dots + 1/(\lambda_i + M + 1)$; hence $c_i = (\lambda_i + 2)^{-1} ((1/\lambda_i + 2) + \dots + 1/(\lambda_i + M + 1))^{-1}$. This yields an “explicit” formula $C'_N = \sum_{1 \leq i \leq M} 2c_i f_N(\lambda_i)$; and it remains to study the roots λ_i . Note that $|\lambda_i + M + 1| \leq M + 1$ for all i , otherwise we would have $|\lambda_i + 2| \dots |\lambda_i + M + 1| > (M + 1)!$. Taking $\lambda_1 = 0$, this implies that $\Re(\lambda_i) < 0$ for $2 \leq i \leq M$. By Eq. 1.2.5–15, $\theta_n(x) \sim (n + 1)^x / \Gamma(x + 2)$ as $n \rightarrow \infty$; hence $g_n(\lambda_i) \rightarrow 0$ for $2 \leq i \leq M$. Consequently $C'_N = 2c_1 f_N(0) + O(1) = H_N / (H_{M+1} - 1) + O(1)$.

Notes: The above analysis is relevant also to the “samplesort” algorithm discussed briefly in Section 5.2.2. The calculations may readily be extended to show that $B_N^D(1) \sim (H_{M+1} - 1)^{-1}/(j + 2)$ for $1 \leq j < M$, $B_N^{M\Omega}(1) \sim (H_{M+1} - 1)^{-1}/2$. Hence the total number of interior nodes on unfilled pages is approximately $N(1/(3 \times 2) + 2/(4 \times 3) + \dots + (M - 1)/((M + 1) \times M)) / (H_{M+1} - 1) = N(1 - M/(M + 1)(H_{M+1} - 1))$; and the total number of pages used is approximately

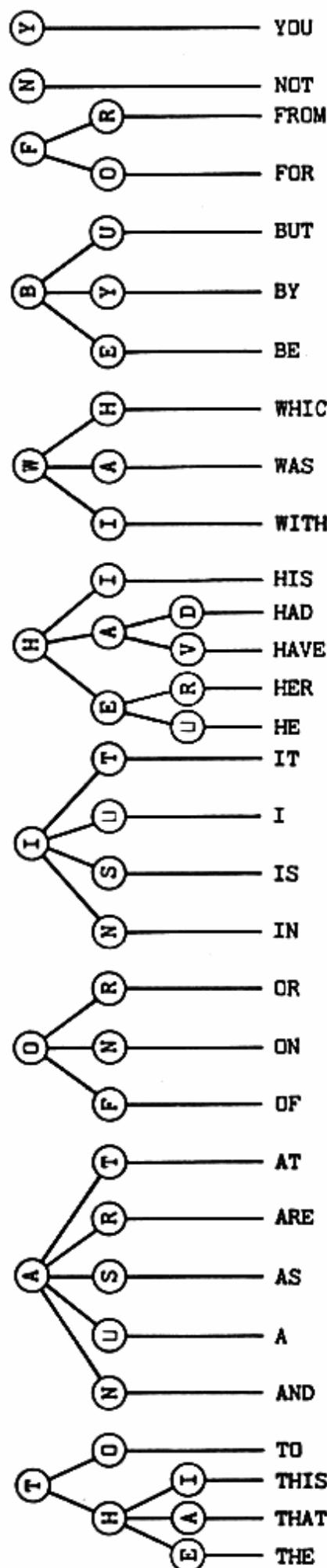
$$\begin{aligned} N(1/(3 \times 2) + 1/(4 \times 3) + \dots + 1/((M + 1) \times M) + 1/(M + 1)) / (H_{M+1} - 1) \\ = N/2(H_{M+1} - 1), \end{aligned}$$

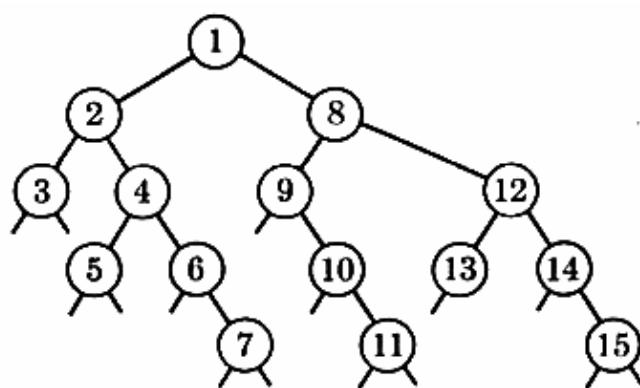
yielding an asymptotic storage utilization of $2(H_{M+1} - 1)/M$.

SECTION 6.3

1. Lieves (pl. of “lief”).
2. Perform Algorithm T using the new key as argument; it will terminate unsuccessfully in either step T3 or T4. If in T3, simply set table entry k of NODE(P) to K and terminate the insertion algorithm. Otherwise set this table entry to the address of a new node $Q \leftarrow \text{AVAIL}$, containing only null links, then set $P \leftarrow Q$. Now set k and k' to the respective next characters of K and X; if $k \neq k'$, store K in position k of NODE(P) and store X in position k' , but if $k = k'$ again make the k position point to a new node $Q \leftarrow \text{AVAIL}$, set $P \leftarrow Q$, and repeat the process until eventually $k \neq k'$. (We must assume that no key is a prefix of another.)
3. Replace the key by a null link, in the node where it appears. If this node is now “useless,” i.e. if all its entries are null except one which is a key X , delete the node and replace the corresponding pointer in its father node by X . If the father node is now useless, delete it in the same way.
4. Successful searches take place exactly as with the full table, but unsuccessful searches in the compressed table may go through several additional iterations. For example, an input argument such as IONIC will make Program T take seven iterations (more than five!); this is the worst case. It is necessary to verify that no infinite looping on blank sequences is possible.

5. In each family, test for the most probable outcome first, by arranging the letters from left to right in decreasing order of probability. The optimality of this arrangement can be proved as in Theorem 6.1S. [Cf. CACM 12 (1969), 72-76.]





7. For example, 8, 4, 12, 1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15. (No matter what sequence is used, the left subtree cannot contain more than two nodes on level 4, nor can the right subtree.) Even this “worst” tree is within 4 of the best possible tree, so we see that digital search trees aren’t very sensitive to the order of insertion.

8. Yes. The KEY fields now contain only a truncated key; leading bits implied by the node position are chopped off. (A similar modification of Algorithm T is possible.)

	9. START	LDX	K	1	<u>D1. Initialize.</u> (rX ≡ K)
		LD1	ROOT	1	P ← ROOT. (rI1 ≡ P)
		JMP	2F	1	
4H		LD2	0,1(RLINK)	C2	<u>D4. Move right.</u> Q ← RLINK(P).
		J2Z	5F	C2	To D5 if Q = Λ.
1H		ENT1	0,2	C - 1	P ← Q.
2H		CMPX	1,1	C	<u>D2. Compare.</u>
		JE	SUCCESS	C	Exit if K = KEY(P).
		SLB	1	C - S	Shift K left one bit.
		JAO	4B	C - S	To D4 if the detached bit was 1.
		LD2	0,1(LLINK)	C1	<u>D3. Move left.</u> Q ← LLINK(P).
		J2NZ	1B	C1	To D2 with P ← Q if Q ≠ Λ.
5H		Continue as in Program 6.2.2T, interchanging the roles of rA, rX.			

The running time for the searching phase of this program is $(10C - 3S + 4)u$, where $C - S$ is the number of bit inspections. For random data, the approximate average running times are therefore:

	Successful	Unsuccessful
Program 6.2.2T	$15 \ln N - 12.34$	$15 \ln N - 2.34$
This program	$14.4 \ln N - 6.17$	$14.4 \ln N + 1.26$

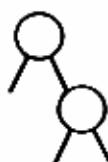
(Consequently Program 6.2.2T is a shade faster unless N is very large.)

10. Let \oplus denote the “exclusive or” operation on n -bit numbers, and let $f(x) = n - \lceil \log_2(x+1) \rceil$, the number of leading zero bits of x . One solution: (b) If a search via Algorithm T ends unsuccessfully in step T3, k is one less than the number of bit inspections made so far; otherwise if the search ends in step T4, $k = f(K \oplus X)$. (a, c) Do a regular search, but also keep track of the minimum value, x , of $K \oplus \text{KEY}(P)$ over all $\text{KEY}(P)$ compared with K during the search. Then $k = f(x)$. (Prove that no other key can have more bits in common with K than those compared to K . In case (a), the maximum k occurs for either the largest key $\leq K$ or the smallest key $> K$.)

11. Yes, since the keys of all nodes in the subtree of NODE(Q) begin with the appropriate sequence of bits. But major changes to the tree would be necessary if keys are

truncated as in exercises 8 and 9, so truncation should not be used together with deletion.

12. Insert three random numbers α, β, γ between 0 and 1 into an initially empty tree; then delete α with probability p , β with probability q , γ with probability r , using Algorithm 6.2.2D. The tree



is obtained with probability $\frac{3}{8}p + \frac{1}{2}q + \frac{1}{2}r$, and this is $\frac{1}{2}$ only if $p = 0$.

13. Add a KEY field to each node, and compare K with this key before looking at the vector element in step T2. Table 1 would change as follows: Nodes (1), ..., (12) would contain the respective keys THE, AND, BE, FOR, HIS, IN, OF, TO, WITH, HAVE, HE, THAT (if we inserted them in order of decreasing frequency), and these keys would be deleted from their previous positions. [The corresponding program would be slower and more complicated than Program T, in this case. A more direct M -ary generalization of Algorithm D would create a tree with N nodes, having one key and M links per node.]

14. If $j \leq n$, there is only one place, namely KEY(P). But if $j > n$, the set of all occurrences is found by traversing the subtree of node P: If there are r occurrences, this subtree contains $r - 1$ nodes (including node P), and so it has r link fields with TAG = 1; these link fields point to all the nodes that reference TEXT, positions matching K . (It isn't necessary to check the TEXT again at all.)

15. To begin forming the tree, set KEY(HEAD) to the first TEXT reference, and LLINK(HEAD) \leftarrow HEAD, LTAG(HEAD) \leftarrow 1. Further TEXT references can be entered into the tree using the following insertion algorithm:

Set K to the new key which we wish to enter. (This is the first reference the insertion algorithm makes to the TEXT array.) Perform Algorithm P; it must terminate unsuccessfully, since no key is allowed to be a prefix of another. (Step P6 makes the second reference to the TEXT; no more references will be needed!) Now suppose that the key located in step P6 agrees with the argument K in the first l bits, but differs from it in position $l + 1$, where K has the digit b and the key has $1 - b$. (Even though the search in Algorithm P might have let j get much greater than l , it is possible to prove that the procedure specified here will find the longest match between K and any existing key. Thus, all keys of the text which start with the first l bits of K have $1 - b$ as their $(l + 1)$ st bit.) Now repeat Algorithm P with K replaced by these leading l bits (i.e., $n \leftarrow l$). This time the search will be successful, so we needn't perform step P6. Now set R \leftarrow AVAIL, KEY(R) \leftarrow position of the new key in TEXT. If LLINK(Q) = P, set LLINK(Q) \leftarrow R, $t \leftarrow$ LTAG(Q), LTAG(Q) \leftarrow 0; otherwise set RLINK(Q) \leftarrow R, $t \leftarrow$ RTAG(Q), RTAG(Q) \leftarrow 0. If $b = 0$, set LTAG(R) \leftarrow 1, LLINK(R) \leftarrow R, RTAG(R) \leftarrow t , RLINK(R) \leftarrow P; otherwise set RTAG(R) \leftarrow 1, RLINK(R) \leftarrow R, LTAG(R) \leftarrow t , LLINK(R) \leftarrow P. If $t = 1$, set SKIP(R) \leftarrow $1 + l - j$; otherwise set SKIP(R) \leftarrow $1 + l - j +$ SKIP(P) and SKIP(P) \leftarrow $j - l - 1$.

16. The tree setup requires precisely one dotted link coming from below a node to that node; it comes from that part of the tree where this key first differs from all others. If there is no such part of the tree, the algorithms break down. We could simply drop keys that are prefixes of others, but then the algorithm of exercise 14 wouldn't have enough data to find all occurrences of the argument.

17. If we define $a_0 = a_1 = 0$, then

$$x_n = a_n + \sum_{k \geq 2} \binom{n}{k} (-1)^k \hat{a}_k / (m^{k-1} - 1) = \sum_{k \geq 2} \binom{n}{k} (-1)^k \hat{a}_k m^{k-1} / (m^{k-1} - 1).$$

18. To solve (4) we need the transform of $a_n = 1 - \delta_{n0} - \delta_{n1}$, namely $\hat{a}_n = \delta_{n0} - 1 + n$; hence for $N \geq 2$ we obtain $A_N = 1 - U_N + V_N$, where $U_N = K(N, 0, M)$ and $V_N = K(N, 1, M)$ in the notation of exercise 19. Similarly to solve (5), take $a_n = n - \delta_{n1} = \hat{a}_n$ and obtain $C_N = N + V_N$ for $N \geq 2$.

19. For $s = 1$, we have $V_n = K(n, 1, m) = n((\ln n + \gamma)/(\ln m) - \frac{1}{2} - f_0(n-1)) + O(1)$, and for $s \geq 2$ we have $K(n, s, m) = (-1)^s n(1/(\ln m) + f_{s-1}(n-s))/s(s-1) + O(1)$, where

$$f_s(n) = \frac{2}{\ln m} \sum_{k \geq 1} \Re (\Gamma(s - 2\pi i k / (\ln m)) \exp(2\pi i k \log_m n))$$

is a periodic function of $\log n$. [In this derivation we have

$$\begin{aligned} K(n+s, s, m) / (-1)^s \binom{n+s}{s} \\ = (n^{-s+1}/2\pi i) \int_{1/2-i\infty}^{1/2+i\infty} \Gamma(z) n^{s-1-z} dz / (m^{s-1-z} - 1) + O(n^{-s}). \end{aligned}$$

For small m and s , the f 's will be negligibly small; cf. exercise 5.2.2-46. Note that $f_s(n-a) = f_s(n) + O(n^{-1})$ for fixed a .]

20. For (a), let $a_n = 1 - \delta_{n0} - \delta_{n1} - \dots - \delta_{ns}$; for (b), let $a_n = n - \delta_{n1} - 2\delta_{n2} - \dots - s\delta_{ns}$; and for (c), we want to solve the recurrence

$$y_n = m^{1-n} \sum_k \binom{n}{k} (m-1)^{n-k} y_k \quad \text{for } n > s,$$

$$y_n = \binom{n+1}{2} \quad \text{for } n \leq s.$$

Setting $x_n = y_n - n$ yields a recurrence of the form of exercise 17, with

$$a_n = (1 - M^{-1}) \sum_{0 \leq k \leq s} \binom{k}{2} \delta_{nk}.$$

Therefore, in the notation of previous exercises, the answers are (a) $1 - K(N, 0, M) + K(N, 1, M) - \dots + (-1)^{s-1} K(N, s, M) = N / (s \ln M) - N(f_{s-1}(N) + f_0(n-1) + f_1(N-2)/2 + \dots + f_{s-1}(N-s)/s(s-1)) + O(1)$; (b) $N^{-1}(N + K(N, 1, M) - 2K(N, 2, M) + \dots + (-1)^{s-1} s K(N, s, M)) = (\ln N + \gamma - H_{s-1}) / (\ln M) + \frac{1}{2} - (f_0(N-1) + f_1(N-2)/1 + \dots + f_{s-1}(N-s)/(s-1)) + O(N^{-1})$; (c) $N^{-1}(N + (1 - M^{-1}) \sum_{2 \leq k \leq s} (-1)^k \binom{k}{2} K(N, k, M)) = 1 + \frac{1}{2}(1 - M^{-1})((s-1)/(\ln M) + f_1(N-2) + \dots + f_{s-1}(N-s)) + O(N^{-1})$.

21. Let there be A_N nodes in all. The number of nonnull pointers is $A_N - 1$, and the number of nonpointers is N , so the total number of null pointers is $MA_N - A_N +$

$1 - N$. To get the average number of these in any fixed position, divide by M . [The average value of A_N appears in exercise 20(a).]

22. There is a node for each of the M^l sequences of leading bits such that at least two keys have this bit pattern. The probability that exactly k keys have a particular bit pattern is

$$\binom{N}{k} M^{-lk} (1 - M^{-l})^{N-k},$$

so the average number of trie nodes on level l is $M^l(1 - (1 - M^{-l})^N) - N(1 - M^{-l})^{N-1}$.

23. More generally, consider the case of arbitrary s as in exercise 20. If there are a_l nodes on level l , they contain a_{l+1} links and $Ma_l - a_{l+1}$ places where the search might be unsuccessful. The average number of digit inspections will therefore be $\sum_{l \geq 0} (l+1)M^{-l-1}(Ma_l - a_{l+1}) = \sum_{l \geq 0} M^{-l}a_l$. Using the formula for a_l in a random trie, this equals $1 + (N+1)^{-1}(K(N+1, 1, M) - 2K(N+1, 2, M) + \dots + (-1)^s(s+1)K(N+1, s+1, M)) = (\ln N + \gamma - H_s)/(\ln M) + \frac{1}{2} - f_0(N) - f_1(N-1)/1 - \dots - f_s(N-s)/s + O(N^{-1})$.

24. We must solve the recurrences $x_0 = x_1 = y_0 = y_1 = 0$,

$$\begin{aligned} x_n &= m^{-n} \sum_{n_1+\dots+n_m=n} \binom{n}{n_1, \dots, n_m} \left(x_{n_1} + \dots + x_{n_m} + \sum_{1 \leq j \leq m} (1 - \delta_{n_j 0}) \right) \\ &= a_n + m^{1-n} \sum_k \binom{n}{k} x_k, \\ y_n &= m^{-n} \sum_{n_1+\dots+n_m=n} \binom{n}{n_1, \dots, n_m} \left(y_{n_1} + \dots + y_{n_m} + \sum_{1 \leq i < j \leq m} (1 - \delta_{n_i 0}) n_j \right) \\ &= b_n + m^{1-n} \sum_k \binom{n}{k} y_k, \end{aligned}$$

for $n \geq 2$, where $a_n = m(1 - (1 - 1/m)^n)$ and $b_n = \frac{1}{2}(m-1)n(1 - (1 - 1/m)^{n-1})$. By exercises 17, 18 the answers are (a) $x_N = N + V_N - U_N - \delta_{N1} = A_N + N - 1$ [a result which could have been obtained directly, since the number of nodes in the forest is always $N - 1$ more than the number in the corresponding trie!]; and (b) $y_N/N = \frac{1}{2}(M-1)V_N/N = \frac{1}{2}(M-1)((\ln N + \gamma)/(\ln M) - \frac{1}{2} - f_0(N-1)) + O(N^{-1})$.

25. (a) Let $A_N = M(N-1)/(M-1) - E_N$; then for $N \geq 2$, we have $(1 - M^{1-N})E_N = M - 1 - M(1 - 1/M)^{N-1} + M^{1-N} \sum_{0 < k < N} \binom{N}{k} (M-1)^{N-k} E_k$. Since $M - 1 - M(1 - 1/M)^{N-1} \geq 0$, we have $E_N \geq 0$ by induction. (b) By Theorem 1.2.7A with $x = 1/(M-1)$, $n = N-1$, we find $D_N = a_N + M^{1-N} \sum_k \binom{N}{k} (M-1)^{N-k} D_k$, where $a_1 = 0$ and $0 < a_N < M(1 - 1/M)^N / \ln M \leq (M-1)^2 / M \ln M$ for $N \geq 2$. Hence $0 \leq D_N \leq (M-1)^2 A_N / M \ln M \leq (M-1)(N-1) / \ln M$.

26. Taking $q = \frac{1}{2}$, $z = -\frac{1}{2}$ in the second identity of exercise 5.1.1-16, we get $1/3 - 1/(3 \cdot 7) + 1/(3 \cdot 7 \cdot 15) - \dots = 0.28879$; it's slightly faster to use $z = -\frac{1}{4}$ and take half of the result. Alternatively, Euler's formula from exercise 5.1.1-14 can be used,

involving only negative powers of 2. (John Wrench has computed the 40D value 0.28878 80950 86602 42127 88997 21929 23078 00889+.)

27. (For fun, the following derivation goes to $O(N^{-1})$.) In the notation of exercises 5.2.2-38, 48, we have $\bar{C}_N = U_N + N - 1 + V_{N+1}/(N+1) - \alpha N - \beta + \sum_{n \geq 2} (-1)^n 2^{-n(n+1)/2} (\prod_{1 \leq r \leq n} (1 - 2^{-r})^{-1}) (\sum_{m \geq 0} (2^{1-n})^m (1 - 2^{-m})^N)$, where $\alpha = 2/(1 \cdot 1) - 4/(3 \cdot 3 \cdot 1) + 8/(7 \cdot 7 \cdot 3 \cdot 1) - \dots \approx 1.60669 51524 15291 76378 33015 23190 92458 04806$ — and $\beta = 2/(1 \cdot 3 \cdot 1) - 4/(3 \cdot 7 \cdot 3 \cdot 1) + 8/(7 \cdot 15 \cdot 7 \cdot 3 \cdot 1) - \dots \approx 0.60670$. This numerical evaluation suggests that $\alpha = \beta + 1$, a fact which is not hard to prove. The value of $\sum_{m \geq 0} (2^{1-n})^m (1 - 2^{-m})^N$ is $O(N^{1-n})$, by exercise 5.2.2-46; and $V_{N+1}/(N+1) = U_{N+1} - U_N$. Hence $\bar{C}_N = U_{N+1} - (\alpha - 1)N - \alpha + O(N^{-1}) = (N+1) \log_2 (N+1) + N((\gamma - 1)/\ln 2 + \frac{1}{2} - \alpha + f_{-1}(N)) + \frac{1}{2} - 1/\ln 4 - \alpha - \frac{1}{2}f_1(N) + O(N^{-1})$, by exercise 5.2.2-50.

28. The derivations in the text and exercise 27 apply to general $M \geq 2$, if we substitute M for 2 in the obvious places. Hence the average number of digit inspections in a random successful search is $\bar{C}_N/N = U_{N+1} - \alpha_M + 1 + O(N^{-1}) = \log_M N + (\gamma - 1)/\ln M + \frac{1}{2} - \alpha_M + f_{-1}(N) + (\log_M N)/N + O(N^{-1})$; and for the unsuccessful case it is $\bar{C}_{N+1} - \bar{C}_N = V_{N+2}/(N+2) - \alpha_M + 1 + O(N^{-1}) = \log_M N + \gamma/\ln M + \frac{1}{2} - \alpha_M - f_0(N+1) + O(N^{-1})$. Here $f_s(n)$ is defined in exercise 19, and $\alpha_M = \sum_{j \geq 0} (-1)^j M^{j+1}/(M^{j+1} - 1)^2 (M^j - 1) \cdots (M - 1)$.

30. By iterating the recurrence, $h_n(z)$ is the sum of all possible terms of the form

$$\binom{n}{p_1} (z/(2^{p_1} - 1)) \binom{p_1}{p_2} (z/(2^{p_2} - 1)) \cdots (z/(2^{p_m} - 1)) \binom{p_m}{1},$$

for $n > p_1 > \cdots > p_m > 1$.

31. $h'_n(1) = V_n$ (cf. exercise 5.2.2-36b).

32. The sum of the SKIP fields is the number of nodes in the corresponding binary trie, so the answer is A_N (cf. exercise 20).

33. Here's how (18) was discovered: $A(2z) - 2A(z) = e^{2z} - 2e^z + 1 + A(z)(e^z - 1)$ can be transformed into $A(2z)/(e^{2z} - 1) = (e^z - 1)/(e^z + 1) + A(z)/(e^z - 1)$. Hence $A(z) = (e^z - 1) \sum_{j \geq 1} (e^{z/2^j} - 1)/(e^{z/2^j} + 1)$. Now if $f(z) = \sum c_n z^n$, $\sum_{j \geq 1} f(z/2^j) = \sum c_n z^n / (2^n - 1)$. In this case $f(z) = (e^z - 1)/(e^z + 1) = \tanh(z/2) = 1 - 2z^{-1} \times (z/(e^z - 1) - 2z/(e^{2z} - 1)) = \sum_{n \geq 1} B_{n+1} z^n (2^{n+1} - 1)/(n+1)!$. From this formula the route is apparent.

34. (a) Consider $\sum_{j \geq 1} \sum_{2 \leq k < n} \binom{n}{k} B_k 2^{j(k-1)}$; $1^{n-1} + \cdots + (m-1)^{n-1} = (B_n(m) - B_n)/n$ by exercise 1.2.11.2-4. (b) Consider $(B_n(m) - B_n)/nm^{n-1}$. (c) Argue as in Section 5.2.2 when $|x| < 2\pi$, then use analytic continuation. (d) $\frac{1}{2} \log_2 (n/\pi) + \gamma/(2 \ln 2) - \frac{3}{4} + f(n)$, where $f(n) = (2/\ln 2) \sum_{k \geq 1} \Re(\zeta(-2\pi ik/\ln 2) \Gamma(-2\pi ik/\ln 2) \exp(2\pi ik \log_2 n)) = (1/\ln 2) \sum_{k \geq 1} \Re(\zeta(1 + 2\pi ik/\ln 2) \exp(2\pi ik \log_2(n/\pi))/\cosh(\pi^2 k/\ln 2))$.

35. The keys must be $\{\alpha 0 \beta 0 \omega_1, \alpha 0 \beta 1 \omega_2, \alpha 1 \gamma 0 \omega_3, \alpha 1 \gamma 1 \delta 0 \omega_4, \alpha 1 \gamma 1 \delta 1 \omega_5\}$, where α, β, \dots are strings of 0's and 1's with $|\alpha| = a - 1$, $|\beta| = b - 1$, etc. The probability that five random keys have this form is $5! 2^{a-1+b-1+c-1+d-1} / 2^{a+b+a+b+a+c+a+d+a+c+d} = 5! / 2^{4a+4b+2c+4d+4}$.

36. Let n be the number of internal nodes. (a) $(n!/2^I) \prod (1/s(x)) = n! \prod (1/2^{s(x)-1} s(x))$, where I is the internal path length of the tree.

- (b) $((n+1)!/2^n) \prod (1/(2^{a(x)} - 1))$. (Consider summing the answer of exercise 35 over all $a, b, c, d \geq 1$.)
37. The smallest modified external path length is actually $2 - 1/2^{N-2}$, and it occurs only in a degenerate tree (whose external path length is *maximal*). [One can prove that the *largest* modified external path length occurs iff the external nodes appear on at most two adjacent levels! But it is not always true that a tree whose external path length is smaller than another has a larger modified external path length.]
38. Consider as subproblems the finding of k -node trees with parameters (α, β) , $(\alpha, \frac{1}{2}\beta)$, \dots , $(\alpha, 2^{k-n}\beta)$.
40. Let N/r be the true period length of the sequence. Form a Patricia-like tree, with $a_0a_1\dots$ as the TEXT and with N/r keys starting at positions $0, 1, \dots, N/r - 1$. (No key is a prefix of another, because of our choice of r .) Also include in each node a SIZE field, containing the number of tagged link fields in the subtree below that node. To do the specified operation, use Algorithm P; if the search is unsuccessful, the answer is 0, but if it is successful and $j \leq n$ the answer is $r \cdot \text{SIZE}(P)$. Finally if it is successful and $j > n$, the answer is $r \cdot \text{SIZE}(P)$.

SECTION 6.4

1. $-38 \leq r11 \leq 46$. Therefore the locations preceding and following TABLE must be guaranteed to contain no data that matches any given argument, e.g. their first byte could be zero. It would certainly be bad to store K in this range! [In a sense we could say that the method in exercise 6.3–4 uses less space, since the boundaries of that table are never exceeded.]

2. TOW. [Can the reader find ten “common” words of ≤ 5 letters that fill all the remaining gaps between -10 and 30 ?]

3. The alphabetic codes $A + T = I + N$ and $B - E = O - R$, so we would have either $f(AT) = f(IN)$ or $f(BE) = f(OR)$. Note that instructions 4 and 5 of Table 1 resolve this dilemma rather well, while keeping $r11$ from having too wide a range.

4. The smallest m such that

$$n^{-m} m! \sum_{0 \leq k \leq m} \binom{n}{m-k} \binom{m-k}{k} 2^{-k} < \frac{1}{2}$$

is 88. If you invite 88 people, the chance of a birthday trio is .5111, but if only 87 people come it is lowered to .4995.

5. The hash function is bad since it assumes at most 26 different values, and some of them occur much more often than the others. Even with double hashing (letting $h_2(K) = 1$ plus the second byte of K , say, and $M = 101$) the search will be slowed down more than the time saved by faster hashing. Also $M = 100$ is too small, since FORTRAN programs often have more than 100 distinct variables.

6. Not on MIX, since arithmetic overflow will almost always occur (dividend too large). [It would be nice to be able to compute $(wK) \bmod M$, especially if linear probing were being used with $c = 1$, but unfortunately most computers disallow this since the quotient overflows.]

7. If $R(x)$ is a multiple of $P(x)$, then $R(\alpha^j) = 0$ in $GF(2^k)$ for all $j \in S$. Let $R(x) = x^{a_1} + \dots + x^{a_s}$, where $a_1 > \dots > a_s \geq 0$ and $s \leq t$, and select $t - s$ further values a_{s+1}, \dots, a_t such that a_1, \dots, a_t are distinct nonnegative integers less than n . The Vandermonde matrix

$$\begin{pmatrix} \alpha^{a_1} & \dots & \alpha^{a_t} \\ \alpha^{2a_1} & \dots & \alpha^{2a_t} \\ \vdots & & \vdots \\ \alpha^{ta_1} & \dots & \alpha^{ta_t} \end{pmatrix}$$

is singular, since the sum of its first s columns is zero. But this contradicts the fact that $\alpha^{a_1}, \dots, \alpha^{a_t}$ are distinct elements of $GF(2^k)$. (See exercise 1.2.3-37.)

[The idea of polynomial hashing originated with M. Hanan, S. Muroga, F. P. Palermo, N. Raver, and G. Schay; see *IBM J. Research & Development* 7 (1963), 121-129; U.S. Patent 3311888 (March 28, 1967).]

8. By induction. The strong induction hypotheses can be supplemented by the fact that $\{(-1)^k(rq_k + q_{k-1})\theta\} = (-1)^k(r(q_k\theta - p_k) + (q_{k-1}\theta - p_{k-1}))$ for $0 \leq r \leq a_k$. The "record low" values of $\{n\theta\}$ occur for $n = q_1, q_2 + q_1, 2q_2 + q_1, \dots, a_2q_2 + q_1 = 0q_4 + q_3, q_4 + q_3, \dots, a_4q_4 + q_3 = 0q_6 + q_5, \dots$; the "record high" values occur for $n = q_0, q_1 + q_0, \dots, a_1q_1 + q_0 = 0q_3 + q_2, \dots$. These are the steps when interval number 0 of a new length is formed.

9. We have $\phi^{-1} = /1, 1, 1, \dots/$ and $\phi^{-2} = /2, 1, 1, \dots/$. Let $\theta = /a_1, a_2, \dots/$, and $\theta_k = /a_{k+1}, a_{k+2}, \dots/$, and $Q_k = q_k + q_{k-1}\theta_{k-2}$ in the notation of exercise 8. If $a_1 > 2$, the very first break is bad. The three sizes of intervals in exercise 8 are, respectively, $(1 - r\theta_{k-1})/Q_k, \theta_{k-1}/Q_k$, and $(1 - (r - 1)\theta_{k-1})/Q_k$, so the ratio of the first length to the second is $(a_k - r) + \theta_k$. This will be less than $\frac{1}{2}$ when $r = a_k$ and $a_{k+1} \geq 2$; hence $\{a_2, a_3, \dots\}$ must all equal 1 if there are to be no bad breaks. [For related theorems, cf. R. L. Graham and J. H. van Lint, *Canadian J. Math.* 20 (1968), 1020-1024, and the references cited there.]

11. There would be a problem if $K = 0$. If keys were required to be nonzero as in Program L, this change would be worth while, and we could also represent empty positions by 0.

12. We can store K in KEY[0], replacing lines 14-19 by

	STA	TABLE(KEY)	$A - S1$
	CMPA	TABLE,2(KEY)	$A - S1$
	JE	3F	$A - S1$
2H	ENT1	0,2	$C - 1 - S2$
	LD2	TABLE,1(LINK)	$C - 1 - S2$
	CMPA	TABLE,2(KEY)	$C - 1 - S2$
	JNE	2B	$C - 1 - S2$
3H	J2Z	5F	$A - S1$
	ENT1	0,2	$S2$
	JMP	SUCCESS	$S2$

The "savings" in time is $C - 1 - 5A + S + 4S1$, which is actually a net loss because C is rarely more than 5. (An inner loop shouldn't always be optimized!)

13. Let the table entries be of two distinguishable types, as in Algorithm C, with an additional one-bit TAG[i] field in each entry. This solution uses circular lists, with TAG[i] = 1 in the first word of each list.

- A1. [Initialize.] Set $i \leftarrow j \leftarrow h(K) + 1$, $Q \leftarrow q(K)$.
- A2. [Is there a list?] If TABLE[i] is empty, set TAG[i] $\leftarrow 1$ and go to A8. Otherwise if TAG[i] = 0, go to A7.
- A3. [Compare.] If $Q = \text{KEY}[i]$, the algorithm terminates successfully.
- A4. [Advance to next.] If LINK[i] $\neq j$, set $i \leftarrow \text{LINK}[i]$ and go back to A3.
- A5. [Find empty node.] Decrease R one or more times until finding a value such that TABLE[R] is empty. If $R = 0$, the algorithm terminates with overflow; otherwise set LINK[i] $\leftarrow R$.
- A6. [Prepare to insert.] Set $i \leftarrow R$, TAG[R] $\leftarrow 0$, and go to A8.
- A7. [Displace a record.] Repeatedly set $i \leftarrow \text{LINK}[i]$ one or more times until $\text{LINK}[i] = j$. Then do step A5. Then set TABLE[R] $\leftarrow \text{TABLE}[j]$, $i \leftarrow j$, TAG[j] $\leftarrow 1$.
- A8. [Insert new key.] Mark TABLE[i] as an occupied node, with KEY[i] $\leftarrow Q$, LINK[i] $\leftarrow j$. ■

(Note that if TABLE[i] is occupied it is possible to determine the corresponding full key K , given only the value of i . We have $q(K) = \text{KEY}[i]$, and then if we set $i \leftarrow \text{LINK}[i]$ zero or more times until TAG[i] = 1 we will have $h(K) = i - 1$.)

14. To insert a new key K : Set $Q \leftarrow \text{AVAIL}$, TAG(Q) $\leftarrow 1$, and store K in this word. [Alternatively, if keys are short, omit this and substitute K for Q in what follows.] Then set $R \leftarrow \text{AVAIL}$, TAG(R) $\leftarrow 1$, AUX(R) $\leftarrow Q$, LINK(R) $\leftarrow \Lambda$. Set $P \leftarrow h(K)$, and then:

- If TAG(P) = 0, set TAG(P) $\leftarrow 2$, AUX(P) $\leftarrow R$.
- If TAG(P) = 1, set $S \leftarrow \text{AVAIL}$, CONTENTS(S) $\leftarrow \text{CONTENTS}(P)$, TAG(P) $\leftarrow 2$, AUX(P) $\leftarrow R$, LINK(P) $\leftarrow S$.
- If TAG(P) = 2, set LINK(R) $\leftarrow \text{AUX}(P)$, AUX(P) $\leftarrow R$.

To retrieve a key K : Set $P \leftarrow h(K)$, and then:

- If TAG(P) $\neq 2$, K is not present.
- If TAG(P) = 2, set $P \leftarrow \text{AUX}(P)$. Then set $P \leftarrow \text{LINK}(P)$ zero or more times until either AUX(P) points to a word containing K [or if keys are short, AUX(P) = K] or LINK(P) = Λ .

(Elcock's original scheme, *Comp. J.* 8 (1965), 242–243, actually used TAG = 2 and TAG = 3 to distinguish between lists of length one (when we can save one word of space) and longer lists. This would be a worthwhile improvement, since we presumably have such a large scatter table that almost all lists have length one.)

15. Knowing that there is always an empty node makes the inner search loop faster, since we need not maintain a counter to determine how many times step L2 is performed. The shorter program amply compensates for this one wasted cell. [On the other hand, there is a neat way to avoid the variable N and to allow the table to become completely full, in Algorithm L, without slowing down the method appreci-

ably except when the table actually does overflow: Simply check whether $i < 0$ happens twice! This trick does not apply to Algorithm D.]

16. No; 0 always leads to SUCCESS, whether it has been inserted or not, and SUCCESS occurs with different values of i at different times.

17. The second probe would then always be to position 0.

18. The code in (31) costs about $3(A - S_1)$ units, compared to (30), and it saves $4u$ times the difference between (26), (27) and (28), (29). For a successful search, (31) is advantageous only when the table is more than about 94 percent full, and it never saves more than about $\frac{1}{2}u$ of time. For an unsuccessful search, (31) is advantageous when the table is more than about 71 percent full.

20. We want to show that

$$\binom{j}{2} \equiv \binom{k}{2} \pmod{2^m} \quad \text{and} \quad 1 \leq j \leq k \leq 2^m$$

implies $j = k$. Observe that $j(j-1) \equiv k(k-1) \pmod{2^{m+1}}$ implies $(k-j) \times (k+j-1) \equiv 0$. If $k-j$ is odd, $k+j-1$ must be a multiple of 2^{m+1} , but that's impossible since $2 \leq k+j-1 \leq 2^{m+1}-2$. Hence $k-j$ is even, so $k+j-1$ is odd, so $k-j$ is a multiple of 2^{m+1} , so $k = j$. [Conversely, if M is not a power of 2, this probe sequence does not work.]

The probe sequence has secondary clustering, and it increases the running time of Program D (as modified in (30)) by about $\frac{1}{2}(C-1)-(A-S_1)$ units since $B \approx \binom{C+1}{3}/M$ will now be negligible. This is a small improvement, until the table gets about 60 percent full.

21. If N is decreased, Algorithm D may fail since it may reach a state with no empty spaces and it will loop indefinitely. On the other hand, if N isn't decreased, Algorithm D may signal overflow when there still is room. The latter alternative is the lesser of the two evils, because rehashing can be used to get rid of deleted cells. (Note that in this case Algorithm D should increase N and test for overflow *only* when inserting an item into a previously *empty* position, since N represents the number of nonempty positions.) We could also have two counters.

22. Suppose that positions $j-1, j-2, \dots, j-k$ are occupied and $j-k-1$ is empty (modulo M). The keys which probe position j and find it occupied before being inserted are precisely those keys in positions $j-1$ through $j-k$ whose hash address does not lie between $j-1$ and $j-k$; these "problematical keys" appear in order of insertion. Algorithm R moves the first such key into position j , and repeats the process on a smaller range of problematical positions until no problematical keys remain.

23. If possible, replace $\text{KEY}[i]$ by a later key on the list that starts at $\text{LINK}[i]$, by recomputing the hash addresses and searching down the list for each such key. (Recall that the lists are almost always short.) Algorithm C should also be changed to use a doubly-linked list of available space instead of R.

24. $P(P-1)(P-2)P(P-1)P(P-1)/MP(MP-1)\dots(MP-6) = M^{-7} \times (1 - (5 - 21/M)P^{-1} + O(P^{-2}))$. In general, the probability of a hash sequence $a_1 \dots a_N$ is $(\prod_{0 \leq i < N} P^{b_i})/(MP)^N = M^{-N} + O(P^{-1})$, where b_i is the number of a_i that equal j .

25. Let the $(N + 1)$ st key hash to location a ; P_k is M^{-N} times the number of hash sequences that leave the k locations $a, a - 1, \dots, a - k + 1$ (modulo M) occupied and $a - k$ empty. The number of such sequences with $a + 1, \dots, a + t$ occupied and $a + t + 1$ empty is $g(M, N, t + k)$, by circular symmetry of the algorithm.

26. $\frac{9!}{2!2!4!1!} f(3, 2)f(3, 2)f(5, 4)f(2, 1) = 2^2 3^5 5^4 7 = 4252500.$

27. Following the hint,

$$s(n, x, y) = \sum_{k \geq 0} \binom{n}{k} x(x+k)^k (y-k)^{n-k-1} (y-n) \\ + n \sum_{k \geq 0} \binom{n-1}{k-1} (x+k)^k (y-k)^{n-k-1} (y-n).$$

In the first sum, replace k by $n - k$ and apply Abel's formula; in the second, replace k by $k + 1$. Now

$$g(M, N, k) = \binom{N}{k} (k+1)^{k-1} (M-k-1)^{N-k-1} (M-N-1),$$

and

$$M^N \sum (k+1)P_k = \sum_{k \geq 0} \binom{k+2}{2} g(M, N, k) \\ = \frac{1}{2} \left(\sum_{k \geq 0} (k+1)g(M, N, k) + \sum_{k \geq 0} (k+1)^2 g(M, N, k) \right).$$

The first sum is $M^N \sum P_k = M^N$, and the second is $s(N, 1, M-1) = M^N + N(2M^{N-1} + (N-1)(3M^{N-2} + \dots)) = M^N Q_1(M, N)$. [See J. Riordan, *Combinatorial Identities* (New York: Wiley, 1968), 18-23, for further study of sums like $s(n, x, y)$.]

28. Let $t(n, x, y) = \sum_{k \geq 0} \binom{n}{k} (x+k)^{k+2} (y-k)^{n-k-1} (y-n)$; then as in exercise 27 we find $t(n, x, y) = xs(n, x, y) + nt(n-1, x+1, y-1)$, $t(N, 1, M-1) = M^N (3Q_3(M, N) - 2Q_2(M, N))$. Thus $\sum (k+1)^2 P_k = M^{-N} \sum (\frac{1}{3}(k+1)^3 + \frac{1}{2}(k+1)^2 + \frac{1}{6}(k+1)) g(M, N, k) = Q_3(M, N) - \frac{2}{3}Q_2(M, N) + \frac{1}{2}Q_1(M, N) + \frac{1}{6}$. Subtracting $(C'_N)^2$ gives the variance, which is approximately $\frac{3}{4}(1-\alpha)^{-4} - \frac{2}{3}(1-\alpha)^{-3} - \frac{1}{12}$. The standard deviation is often larger than the mean; for example, when $\alpha = .9$ the mean is 50.5 and the standard deviation is $\frac{1}{2}\sqrt{27333} \approx 83$.

29. Let $M = m + 1$, $N = n$; the safe parking sequences are precisely those in which location 0 is empty when Algorithm L is applied to the hash sequence $(M - a_1) \dots (M - a_n)$. Hence the answer is $f(m+1, n) = (m+1)^n - n(m+1)^{n-1}$. [This parking problem originated with A. G. Konheim and B. Weiss, *SIAM J. Applied Math.* 14 (1966), 1266-1274, who were the first to publish a correct analysis of Algorithm L.]

30. Obviously if the cars get parked they define such a permutation. Conversely, if $p_1 p_2 \dots p_n$ exists, let $q_1 q_2 \dots q_n$ be the inverse permutation ($q_i = j$ iff $p_j = i$), and let b_i be the number of a_j that equal i . Every car will be parked if we can prove that

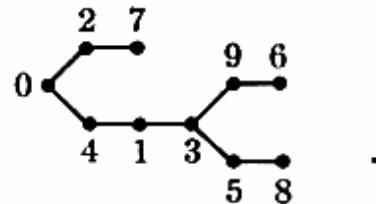
$b_n \leq 1$, $b_{n-1} + b_n \leq 2$, etc.; equivalently $b_1 \geq 1$, $b_1 + b_2 \geq 2$, etc. But this is clearly true, since the k elements a_{q_1}, \dots, a_{q_k} are all $\leq k$.

[Let r_j be the “left influence” of q_j , namely $r_j = k$ iff $q_{j-1} < q_j, \dots, q_{j-k+1} < q_j$ and either $j = k$ or $q_{j-k} > q_j$. Of all permutations $p_1 \dots p_n$ that dominate a given wakeup sequence $a_1 \dots a_n$, the “park immediately” algorithm finds the smallest one (in lexicographic order). Konheim and Weiss observed that the number of wakeup sequences leading to a given permutation $p_1 \dots p_n$ is $\prod_{1 \leq j \leq n} r_j$; it is remarkable that the sum of these products, taken over all permutations $q_1 \dots q_n$, is $(n+1)^{n-1}$.]

31. Many interesting connections are possible, and the following two are the author’s favorites:

a) In the notation of the previous answer, the counts b_1, b_2, \dots, b_n correspond to a full parking sequence iff $(b_1, b_2, \dots, b_n, 0)$ is a valid sequence of *degrees* of tree nodes in preorder. (Cf. 2.3.3–9, which illustrates postorder.) Each such tree corresponds to $n!/b_1! \dots b_n!$ distinct labeled free trees on $\{0, \dots, n\}$, since we can let 0 be the label of the root, and for $k = 1, 2, \dots, n$ we can successively choose the labels of the sons of the k th node in preorder in $(b_k + \dots + b_n)!/b_k!(b_{k+1} + \dots + b_n)!$ ways from the remaining unused labels, attaching labels from left to right in increasing order. And each such sequence of counts corresponds to $n!/b_1! \dots b_n!$ wakeup sequences.

b) Dominique Foata has given the following pretty one-to-one correspondence: Let $a_1 \dots a_n$ be a safe parking sequence, which leaves car q_j parked in space j . A labeled free tree on $\{0, 1, \dots, n\}$ is constructed by drawing a line from j to 0 when $a_j = 1$, and from j to q_{a_j-1} otherwise, for $1 \leq j \leq n$. (Think of the tree nodes as cars; car j is connected to the car that eventually winds up parked just before where wife j woke up.) Thus the example shown in exercise 29 corresponds to the free tree



The sequence of parked cars may be obtained from the tree by topological sorting, assuming that arrows emanate from the root 0, choosing the smallest “source” at each step. From this sequence, $a_1 \dots a_n$ can be reconstructed.

Actually the two constructions in (a) and (b) are strongly related. For further information, see Dominique Foata and John Riordan, “Mappings of acyclic and parking functions” (to appear).

32. Let subscripts be treated cyclically, so that $c_M = c_0$, $c_{M+1} = c_1$, etc. There is no solution with $c_j = b_j + c_{j+1} - 1$ for all j , since the sum over all j would give $\sum c_j = \sum b_j + \sum c_j - M < \sum c_j$. Hence every solution has $M - \sum b_j$ values of j such that $b_j = c_{j+1} = 0$. If (c'_0, \dots, c'_{M-1}) is a different solution we must have $c'_{j+1} > 0$ for all such j ; but this implies $c'_{j+2} > c_{j+2}, c'_{j+3} > c_{j+3}, \dots$, a contradiction. The solution can be found by defining c_{M-1}, c_{M-2}, \dots on the assumption that $c_0 = 0$; then if c_0 turns out to be greater than 0, it suffices to redefine c_{M-1}, c_{M-2}, \dots until no more changes are made. [A slightly more efficient algorithm, for computing $\sum c_j$ from the b ’s, has been given by T. C. Lowe and D. C. Roberts (to appear).]

33. The individual probabilities are not independent, since the condition $b_0 + b_1 + \dots + b_{M-1} = N$ was not taken into account; the derivation allows a nonzero probability that $\sum b_j$ has any given nonnegative value. Equations (46) are not strictly correct; they imply, for example, that q_k is positive for all k , contradicting the fact that c_j can never exceed $N - 1$.

34. (a) There are $\binom{N}{k}$ ways to choose the set of j such that a_j has a particular value, and $(M - 1)^{N-k}$ ways to assign values to the other a 's. Therefore

$$P_{Nk} = \binom{N}{k} (M - 1)^{N-k} / M^N.$$

(b) $P_N(z) = B(z)$ in (50). (c) $C'_N = \sum (k + \delta_{k0}) P_{Nk} = P'_N(1) + P_N(0)$. For C_N , consider the total number of probes to find all keys. A list of length k contributes $\binom{k+1}{2}$ to the total; hence

$$C_N = M \sum \binom{k+1}{2} P_{Nk} / N = (M/N)(\frac{1}{2}P''_N(1) + P'_N(1)).$$

Thus we obtain (18), (19).

35. $\sum (1 + \frac{1}{2}k - (k+1)^{-1} + \delta_{k0}) P_{Nk} = 1 + N/2M - M(1 - (1 - 1/M)^{N+1}) / (N+1) + (1 - 1/M)^N \approx 1 + \frac{1}{2}\alpha - (1 - e^{-\alpha})/\alpha + e^{-\alpha}$.

36. $\sum (\delta_{k0} + k)^2 P_{Nk} = \sum (\delta_{k0} + k^2) P_{Nk} = P(0) + P''(1) + P'(1)$. Subtracting C'_N gives the answer, $(M - 1)N/M^2 + (1 - 1/M)^N(1 - 2N/M - (1 - 1/M)^N) \approx \alpha + e^{-\alpha}(1 - 2\alpha - e^{-\alpha}) \leq 1 - e^{-1} - e^{-2} = 0.4968$.

37. Let S_N be the average value of $(C_N - 1)^2$; then

$$\begin{aligned} M^N N^2 S_N &= \sum_{k_1, \dots, k_M} \binom{N}{k_1, \dots, k_M} \left(\binom{k_1}{2} + \dots + \binom{k_M}{2} \right)^2 \\ &= M(M-1) \sum_{k_1, \dots, k_M} \binom{N}{k_1, \dots, k_M} \binom{k_1}{2} \binom{k_2}{2} + M \sum_{k_1, \dots, k_M} \binom{N}{k_1, \dots, k_M} \binom{k_1}{2}^2 \\ &= M(M-1) \sum_{k,j} \binom{N}{k} \binom{N-k}{j} \binom{k}{2} \binom{j}{2} (M-2)^{N-k-j} \\ &\quad + M \sum_{k,j} \binom{N}{k} \binom{k}{2}^2 (M-1)^{N-k} \\ &= M(M-1)(\frac{1}{4}M^{N-4}N^4) + M(M^{N-4})(\frac{1}{4}N^4 + N^3 + \frac{1}{2}N^2). \end{aligned}$$

The variance is $S_N - ((N-1)/2M)^2 = (M-1)(N-1)/2NM^2$. (Hence the standard deviation is $O(M^{-1/2})$ as $M \rightarrow \infty$.)

38. The average number of probes is $\sum P_{Nk} (2H_{k+1} - 2 + \delta_{k0})$ in the unsuccessful case, $(M/N) \sum P_{Nk} k (2(1 + 1/k)H_k - 3)$ in the successful case, by Eqs. 6.2.2-5, 6. These sums are respectively equal to $2f(N) + 2M(1 - (1 - 1/M)^{N+1})/(N+1) + (1 - 1/M)^N - 2$ and $2(M/N)f(N) + 2f(N-1) + 2M(1 - (1 - 1/M)^N)/N - 3$, where $f(N) = \sum P_{Nk} H_k$. Exercise 5.2.2-57 tells us that $f(N) = \ln \alpha + \gamma + E_1(\alpha) + O(M^{-1})$ when $N = \alpha M$, $M \rightarrow \infty$.

[Tree hashing was first proposed by P. F. Windley, *Comp. J.* 3 (1960), 84-88. The above analysis shows that tree hashing is not enough better than simple chaining

to justify the extra link fields (the lists are short anyway); and when M is small it is not enough better than pure tree search to justify the hashing time.]

39. $c(k_1, k_2, \dots) = (M - N + 1)c(k_1 - 1, k_2, \dots) + (k_1 + 1)c(k_1 + 1, k_2 - 1, k_3, \dots) + 2(k_2 + 1)c(k_1, k_2 + 1, k_3 - 1, \dots) + \dots$. [This is similar to the recurrence in exercise 1.2.5-21, but it does not seem to have a simple solution.] Hence

$$\begin{aligned} S_{N+1} &= (M - N) \sum_{j \geq 1, k_1 + 2k_2 + \dots = N+1} \binom{j}{2} k_j c(k_1 - 1, k_2, \dots) \\ &\quad + \sum_{j \geq 1, k_1 + 2k_2 + \dots = N+1} \binom{j}{2} k_j (k_1 + 1) c(k_1 + 1, k_2 - 1, \dots) + \dots \\ &= (M - N)S_N + \sum_{j \geq 1, k_1 + 2k_2 + \dots = N} c(k_1, k_2, \dots) \times \\ &\quad \left(\left(\binom{j}{2} k_j - \binom{1}{2} + \binom{2}{2} \right) k_1 + \left(\binom{j}{2} k_j - \binom{2}{2} + \binom{3}{2} \right) 2k_2 + \dots \right) \\ &= MS_N + \sum_{j \geq 1, k_1 + 2k_2 + \dots = N} j^2 k_j c(k_1, k_2, \dots) = MS_N + 2S_N + NM^N. \end{aligned}$$

Consequently $S_N = (N - 1)M^{N-1} + (N - 2)M^{N-2}(M + 2) + \dots + M(M + 2)^{N-2} = \frac{1}{4}(M(M + 2)^N - M^{N+1} - 2NM^N)$.

Consider the total number of probes in unsuccessful searches, summed over all M values of $h(K)$; each list of length k contributes $k + \delta_{k0} + \binom{k}{2}$ to the total, hence $M^{N+1}C'_N = M^{N+1} + S_N$.

40. Define U_N to be like S_N in exercise 39, but with $\binom{j}{2}$ replaced by $\binom{j+1}{3}$. We find $U_{N+1} = (M + 3)U_N + S_N + NM^N$, hence $U_N = \frac{1}{36}(M^N(M - 6N) - 9M(M + 2)^N + 8M(M + 3)^N)$. The variance is $2U_N/M^{N+1} + C'_N - C'^2_N$, which approaches $\frac{35}{144} - \frac{1}{12}\alpha - \frac{1}{4}\alpha^2 + (\frac{1}{4}\alpha - \frac{5}{8})e^{2\alpha} + \frac{4}{9}e^{3\alpha} - \frac{1}{16}e^{4\alpha}$ for $N = \alpha M$, $M \rightarrow \infty$. When $\alpha = 1$ this is about 4.50, so the standard deviation is bounded by about 2.12.

41. Let V_N be the average length of the block of occupied cells at the "high" end of the table. The probability that this block has length k is $A_{Nk}(M - 1 - k)^{\underline{N-k}}/M^N$, where A_{Nk} is the number of hash sequences (35) such that Algorithm C leaves the first $N - k$ and the last k cells occupied and such that the subsequence 1, 2, ..., $N - k$ appears in increasing order. Therefore

$$\begin{aligned} M^N V_N &= \sum k A_{Nk} (M - 1 - k)^{\underline{N-k}} = M^{N+1} - \sum (M - k) A_{Nk} (M - 1 - k)^{\underline{N-k}} \\ &= M^{N+1} - (M - N) \sum A_{Nk} (M - k)^{\underline{N-k}} = M^{N+1} - (M - N)(M + 1)^N. \end{aligned}$$

Now $T_N = (N/M)(1 + V_N - T_0 - \dots - T_{N-1})$, since $T_0 + \dots + T_{N-1}$ is the average number of times R has previously decreased and N/M is the probability that it decreases this time. The solution to this recurrence is $T_N = (N/M)(1 + 1/M)^N$. (Such a simple formula deserves a simpler proof!)

42. $S1_N$ is the number of items that were inserted with $A = 0$, divided by N .

44. Number the positions of the array from 1 to m , left to right. Considering the set of all $\binom{n}{k}$ sequences of operations with k "p-steps" and $n - k$ "q-steps" to be equally

likely, let $g(m, n+1, k, r)$ be $\binom{n}{k}$ times the probability that the first $r-1$ positions become occupied and the r th remains empty. Thus $g(m, l, k, r)$ is $(m-1)^{-l-1-k}$, times the sum over all configurations

$$1 \leq a_1 < \cdots < a_k < l; \quad (c_1, \dots, c_{l-1-k}), \quad 2 \leq c_i \leq m,$$

of the probability that the first empty location is r , when the a_j th operation is a p -step and the remaining $l-1-k$ operations are q -steps that begin by selecting positions c_1, \dots, c_{l-1-k} , respectively. By summing over all configurations subject to the further condition that the a_j th operation occupies position b_j , for specified

$$1 \leq b_1 < \cdots < b_k < r,$$

we obtain the recurrence

$$g(m, l, k+1, r) = \sum_{\substack{a < l \\ b < r \\ 1 \leq b \leq a}} \frac{(l-b-1)!}{(l-r)!} \frac{(m-r)!}{(m-b)!} (m-l+1) g(m, a, k, b);$$

$$g(m, l, 0, r) = \frac{(l-1)!}{(l-r)!} \frac{(m-r)!}{m!} (m-l+1) \left(P_l + (1 - \delta_{r1}) \frac{m}{l-1} (1 - P_l) \right),$$

where $P_l = (m/(m-1))^{l-1}$. It follows that if $G(m, l, k) = \sum_{1 \leq r \leq l} (m+1-r) \times g(m, l, k, r)$, we have

$$G(m, l, k+1) = \frac{m-l+1}{m-l+2} \sum_{1 \leq a < l} G(m, a, k); \quad G(m, l, 0) = \frac{m-l+1}{m-l+2} (m + P_l).$$

The answer to the stated problem is $m - \sum_{0 \leq k \leq n} p^k q^{n-k} G(m, n+1, k)$, which (after some maneuvering) equals $m - ((m-n)/(m-n+1))(Q_n + mR + pSR)$, where

$$Q_j = P_{j+1} q^j,$$

$$R = \left(1 - \frac{p}{m+1}\right) \left(1 - \frac{p}{m}\right) \cdots \left(1 - \frac{p}{m-n+2}\right) = \prod_{0 \leq j < n} \left(1 - \frac{p}{m+1-j}\right),$$

$$\begin{aligned} S &= \frac{\left(1 - \frac{1}{m+1}\right) Q_0}{\left(1 - \frac{p}{m+1}\right)} + \frac{\left(1 - \frac{1}{m}\right) Q_1}{\left(1 - \frac{p}{m+1}\right) \left(1 - \frac{p}{m}\right)} + \cdots + \frac{\left(1 - \frac{1}{m-n+2}\right) Q_{n-1}}{R} \\ &= \sum_{0 \leq j < n} \frac{(1 - 1/(m+1-j)) Q_j}{\prod_{0 \leq i \leq j} (1 - p/(m+1-i))}. \end{aligned}$$

When $p = 1/m$, $Q_j = 1$ for all j . Letting $w = m+1$, $n = \alpha w$, $w \rightarrow \infty$, we find $\ln R = -p(H_w - H_{w(1-\alpha)}) + O(p^2)$; hence $R = 1 + w^{-1} \ln(1-\alpha) + O(w^{-2})$; and similarly $S = \alpha w + O(1)$. Thus the answer is $(1-\alpha)^{-1} - 1 - \alpha - \ln(1-\alpha) + O(w^{-1})$.

Notes: The simpler problem "with probability p occupy the leftmost, otherwise occupy any randomly chosen empty position" is solved by taking $P_j = 1$ in the above, and the answer is $m - (m+1)(m-n)R/(m-n+1)$. To get C'_N for random probing with secondary clustering, set $n = N-1$, $m = M$ and add 1 to the above answer.

46. Define the numbers $\left[\left[\begin{matrix} n \\ k \end{matrix} \right] \right]$ for $k \geq 0$ by the rule

$$\sum_k \binom{x+k}{k} \left[\left[\begin{matrix} n \\ k \end{matrix} \right] \right] = (x+n+1)^n$$

for all x and all nonnegative integers n . Setting $x = -1, -2, \dots, -n-1$ implies that

$$\left[\left[\begin{matrix} n \\ k \end{matrix} \right] \right] = \sum_j \binom{k}{j} (-1)^j (n-j)^n \quad \text{for } 0 \leq k \leq n;$$

then setting $x = 0$ implies that we may take $\left[\left[\begin{matrix} n \\ k \end{matrix} \right] \right] = 0$ for all $k > n$, so the two sides of the defining equation are polynomials in x of degree n that agree on $n+1$ points. It follows that the numbers $\left[\left[\begin{matrix} n \\ k \end{matrix} \right] \right]$ have the stated property.

Let $f(N, r)$ be the number of hash sequences $a_1 \dots a_N$ that leave the first r locations occupied and the next one empty. There are $\binom{M-r-1}{N-r}$ possible patterns of occupied cells, and each pattern occurs as many times as there are sequences $a'_1 \dots a'_N$, $1 \leq a'_i \leq N$, that contain each of the numbers $r+1, r+2, \dots, N$ at least once. By inclusion-exclusion, there are $\left[\left[\begin{matrix} N \\ N-r \end{matrix} \right] \right]$ such sequences; hence

$$f(N, r) = \binom{M-r-1}{N-r} \left[\left[\begin{matrix} N \\ N-r \end{matrix} \right] \right].$$

Now $C'_N = 1 + M^{-N-1} \sum_{0 \leq r \leq N} f(N, r) (\sum_{0 \leq a < r} r + \sum_{r < a < M} ((N-r)/(M-r-1))(r+1)) = 1 + M^{-N-1} \sum_{0 \leq r \leq N} f(N, r) (N + (N-1)r)$. Let

$$S_n(x) = \sum_k k \binom{x+k}{k} \left[\left[\begin{matrix} n \\ k \end{matrix} \right] \right];$$

we have

$$(x+1)^{-1} S_n(x) + \sum_k \binom{x+k}{k} \left[\left[\begin{matrix} n \\ k \end{matrix} \right] \right] = \sum_k \binom{x+1+k}{k} \left[\left[\begin{matrix} n \\ k \end{matrix} \right] \right];$$

hence $S_n(x) = (x+1)((x+n+2)^n - (x+n+1)^n)$. It follows that $C'_N = N(1+1/M) - (N-1)(1-N/M)(1+1/M)^N \approx N(1 - (1-\alpha)e^\alpha)$; and $C_N = (N-1)((1+1/M)/2 + (1+1/M)^N) + (3M^2 + 6M + 2)((1+1/M)^N - 1)/N - (3M+2)(1+1/M)^N$, which is $(e-2.5)M + O(1)$ when $N = M-1$.

For further properties of the numbers $\left[\left[\begin{matrix} n \\ k \end{matrix} \right] \right]$, cf. John Riordan, *Combinatorial Identities* (New York: Wiley, 1968), 228.

47. The analysis of Algorithm L applies, almost word-for-word! Any probe sequence with cyclic symmetry, and which explores only positions adjacent to those previously examined, will have the same behavior.

48. $C'_N = 1 + p + p^2 + \dots$, where $p = N/M$ is the probability that a random location is filled; hence $C'_N = M/(M-N)$, and $C_N = N^{-1} \sum_{0 \leq k < N} C'_k = N^{-1} M(H_M - H_{M-N})$. These values are approximately equal to those for uniform

probing, but slightly higher because of the chance of multiple probes in the same place. Indeed, for $4 = N < M \leq 16$, linear probing is better!

In practice we wouldn't use infinitely many hash functions, some other scheme like linear probing would ultimately be used as a last resort. This method is inferior to those described in the text, but it is of historical importance because it suggested Morris's method which led to Algorithm D. See *CACM* 6 (1963), 101, where M. D. McIlroy credits the idea to V. A. Vyssotsky; the same technique had been discovered as early as 1956 by A. W. Holt, who used it successfully in the GPX system for the UNIVAC.

49. $C'_N - 1 = \sum_{k>b} (k-b)P_{Nk} \approx \sum_{k>b} (k-b)e^{-\alpha b}(\alpha b)^k/k! = \alpha b t_b(\alpha)$. [Note: In general if $P(z)$ is any probability generating function,

$$\sum_{b \geq 0} \left(\sum_{k>b} (k-b)P_k \right) z^b = P'(1)/(1-z) + z(P(z)-1)/(1-z)^2.]$$

$$\begin{aligned} C_N - 1 &= (M/N) \sum_{k>b} \binom{k-b+1}{2} P_{Nk} \\ &= (M/2N) \sum_{k>b} (k(k-1) - 2k(b-1) + b(b-1)) P_{Nk} \\ &\approx \frac{1}{2} e^{-b\alpha} (b\alpha)^b b!^{-1} (b + b\alpha - 2b + 2 + (b\alpha^2 - 2\alpha(b-1) + b-1) R(\alpha, b)). \end{aligned}$$

[The analysis of successful search with chaining was first carried out by W. P. Heising in 1957. The simple expressions in (57), (58) were found by J. A. van der Pool in 1971; he also considered how to minimize a function that represents the combined cost of storage space and number of accesses. Since $\sum_{k>b} (k-b)^2 P_{Nk} = (2N/M) \times (C_N - 1) - (C'_N - 1)$, we can determine the variance of C'_N and of the number of overflows per bucket. The variance of the total number of overflows may be approximated by M times the variance in a single bucket, but this is actually too high because the total number of records is constrained to be N . The true variance may be found as in exercise 37. Cf. also the derivation of the chi-square test in Section 3.3.1C.]

50. And next that $Q_0(M, N-1) = (M/N)(Q_0(M, N) - 1)$.

51. $R(\alpha, n) = \alpha^{-1} (n! e^{\alpha n} (\alpha n)^{-n} - Q_0(\alpha n, n))$.

52. Cf. Eq. 1.2.11.3-9 and exercise 3.1-14.

53. By Eq. 1.2.11.3-8, $\alpha(\alpha n)^n R(\alpha, n) = e^{\alpha n} \gamma(n+1, \alpha n)$; hence by the suggested exercise $R(\alpha, n) = (1-\alpha)^{-1} - (1-\alpha)^{-3} n^{-1} + O(n^{-2})$. [This asymptotic formula can be obtained more directly by the method of (43), by noting that the coefficient of α^k in $R(\alpha, n)$ is

$$1 - \binom{k+2}{2} n^{-1} + O(k^4 n^{-2}).$$

In fact, the coefficient of α^k is

$$\sum_{r \geq 0} (-1)^r n^{-r} \binom{r+k+1}{k+1}$$

by Eq. 1.2.9-28.]

55. If $B(z)C(z) = \sum s_i z^i$, we have $c_0 = s_0 + \dots + s_b$, $c_1 = s_{b+1}$, $c_2 = s_{b+2}, \dots$; hence $B(z)C(z) = z^b C(z) + Q(z)$. Now $P(z) = z^b$ has $b - 1$ roots $|q_j| < 1$, determined as the solutions to $e^{\alpha(q_j - 1)} = \omega^{-j} q_j$, $\omega = e^{2\pi i/b}$. To solve $e^{\alpha(q-1)} = \omega^{-1} q$, let $t = \alpha q$ and $z = \alpha \omega e^{-\alpha}$ so that $t = ze^\alpha$. By Lagrange's formula we get

$$\begin{aligned}\frac{1}{1-q} &= 1 + \sum_{r \geq 0} r \sum_{n \geq r} \frac{n^{n-r-1} \omega^n \alpha^{n-r} e^{-n\alpha}}{(n-r)!} \\ &= 1 + \sum_{r \geq 1} r \sum_{m \geq 0} \frac{\alpha^m}{m!} (-1)^m \sum_{n \geq r} \binom{m}{n-r} (-1)^{n-r} \omega^n n^{m-1}.\end{aligned}$$

By Abel's limit theorem, letting $|\omega| \rightarrow 1$ from inside the unit circle, this can be rearranged to

$$\frac{1 - \alpha\omega}{1 - \omega} + \sum_{m \geq 2} \frac{\alpha^m}{m!} (-1)^m \sum_{n \geq 0} \binom{m-2}{n} (-1)^n \omega^{n+1} (n+1)^{m-1}.$$

Now replacing ω by ω^j and summing for $1 \leq j < b$ reduces this to

$$\frac{b-1}{2} + \alpha \frac{b-1}{2} + \sum_{m \geq 2} \alpha^m \left(-\frac{1}{2} + \frac{(-1)^m}{m!} b \sum_{n \geq 1} \binom{m-2}{nb-1} (-1)^{nb-1} (nb)^{m-1} \right)$$

and the desired result follows after some more juggling since

$$t_n(\alpha) = (-1)^{n-1} (n! \alpha)^{-1} \sum_{m > n} (-n\alpha)^m / m(m-1)(m-n-1)!$$

This analysis was first begun by M. Tainiter, *JACM* 10 (1963), 307–315.

58. 0 1 2 3 4 and 0 2 4 1 3, plus additive shifts of 1 1 1 1 1 mod 5, each with probability $\frac{1}{10}$. [For $M = 6$, we need 30 permutations, and a solution exists starting with $\frac{1}{20} \times 0 1 2 3 4 5$, $\frac{1}{60} \times 0 1 3 2 5 4$, $\frac{1}{60} \times 0 2 4 3 1 5$, $\frac{1}{20} \times 0 2 3 4 5 1$, $\frac{1}{30} \times 0 3 4 1 2 5$. For $M = 7$, we need 49, and a solution is generated by $\frac{1}{35} \times 0 1 2 3 4 5 6$, $\frac{2}{105} \times 0 1 5 3 2 4 6$, $\frac{1}{35} \times 0 2 4 3 5 1 6$, $\frac{2}{105} \times 0 2 6 3 1 4 5$, $\frac{1}{35} \times 0 3 6 1 4 2 5$, $\frac{1}{105} \times 0 3 2 6 4 1 5$, $\frac{1}{105} \times 0 3 1 5 4 2 6$.]

59. No permutation can have a probability larger than

$$1 / \binom{M}{\lfloor M/2 \rfloor},$$

so there must be at least

$$\binom{M}{\lfloor M/2 \rfloor} = \exp(M \ln 2 + O(\log M))$$

permutations with nonzero probability.

63. MH_M , by exercise 3.3.2-8; the standard deviation is $\pi M / \sqrt{6}$. [This is the “mean time to failure” of the deletion method that simply marks cells “deleted.”]

65. The keys can be stored in a separate table, allocated sequentially (assuming that deletions, if any, are LIFO). The scatter table entries point to this “names table”,

e.g., $\text{TABLE}[i]$ might have the form

	$L[i]$	$KEY[i]$

where $L[i]$ is the number of words in the key stored at locations $KEY[i], KEY[i] + 1, \dots$. The rest of the scatter table entry might be used in any of several ways: (a) as a link for Algorithm C; (b) as part of the information associated with the key; or (c) as a “secondary hash code.” The latter idea, suggested by Robert Morris, sometimes speeds up a search [we take a careful look at the key in $KEY[i]$ only if $h_2(K)$ matches its secondary hash code, for some function $h_2(K)$].

SECTION 6.5

1. The path described in the hint can be converted by changing each downward step that runs from $(i - 1, j)$ to a “new record low” value $(i, j - 1)$ into an upward step. If c such changes are made, the path ends at $(n, n - 2t + 2c)$, where $c \geq 0$ and $c \geq 2t - n$; hence $n - 2t + 2c \geq n - 2k$. In the permutation corresponding to the changed path, the smallest c elements of list B correspond to the downward steps that changed, and list A contains the $t - c$ elements corresponding to downward steps that didn’t change.

When $t = k$ it is not difficult to see that the construction is reversible; hence exactly $\binom{n}{k}$ permutations are constructed. Note that, according to this proof, the contents of lists A and C may appear in arbitrary order.

Notes: We have counted these paths in another way in exercise 2.2.1–4. When $k = \lfloor n/2 \rfloor$ this construction proves *Sperner’s Lemma*, which states that it is impossible to have more than $\binom{n}{\lfloor n/2 \rfloor}$ subsets of $\{1, 2, \dots, n\}$ with no subset contained in another. [Emanuel Sperner, *Math. Zeitschrift* 27 (1928), 544–548.] For if we had such a collection of subsets, each of the $\binom{n}{k}$ permutations can have at most one of the subsets appearing in the initial positions, yet each subset appears in some permutation. The construction used here is a disguised form of a more general construction by which N. G. de Bruijn et al. [*Nieuw Archief voor Wiskunde* (2) 23 (1951), 191–193] proved the multiset generalization of Sperner’s Lemma: “Let M be a multiset containing n elements (counting multiplicities). The collection of all $\lfloor n/2 \rfloor$ -element submultisets of M is the largest possible collection such that no submultiset is contained in another.” For example, the largest such collection when $M = \{a, a, b, b, c, c\}$ consists of the seven submultisets $\{a, a, b\}$, $\{a, a, c\}$, $\{a, b, b\}$, $\{a, b, c\}$, $\{a, c, c\}$, $\{b, b, c\}$, $\{b, c, c\}$. This would correspond to seven permutations of six attributes $A_1, B_1, A_2, B_2, A_3, B_3$ in which all queries involving A_i also involve B_i .

2. Let a_{ijk} be a list of all references to records having (i, j, k) as the respective values of the three attributes, and assume that a_{011} is the shortest of the three lists a_{011} , a_{101} , a_{110} . Then a minimum-length list is $a_{001}a_{011}a_{111}a_{101}a_{100}a_{110}a_{111}a_{011}a_{010}$. However, if a_{011} is empty and so is either of a_{001} , a_{010} , or a_{100} , the length can be shortened by deleting one of the two occurrences of a_{111} .

3. (a) Anise seed, possibly in combination with nutmeg and/or vanilla extract.
(b) None.

4. Let p_t be the probability that the query involves exactly t bit positions, and let P_t be the probability that t given positions are all 1 in a random record. Then the

answer is $\sum_t p_t P_t$. By the principle of inclusion and exclusion,

$$P_t = \sum_{j \geq 0} (-1)^j \binom{t}{j} f(n-j, k, r) / f(n, k, r),$$

where $f(n, k, r)$ is the number of possible choices of r distinct k -bit attribute codes in an n -bit field, namely,

$$f(n, k, r) = \binom{\binom{n}{k}}{r}.$$

And (cf. exercise 1.3.3-26)

$$p_t = \sum_{i \geq 0} (-1)^i \binom{t+i}{t} \binom{n}{t+i} P_{t+i|_{r=q}} = \binom{n}{t} \sum_{j \geq 0} (-1)^j \binom{t}{j} f(t-j, k, q) / f(n, k, q).$$

Notes: The above calculations were first carried out, in more general form, by G. Orosz and L. Takács, *J. of Documentation* 12 (1956), 231–234. The mean $\sum t p_t$ is easily shown to be $n((1 - f(n-1, k, q)) / f(n, k, q))$. Another assumption, that the random attribute codes in records and queries are not necessarily distinct, as in the techniques of Harrison and Bloom, can be analyzed by the same method, setting $f(n, k, r) = \binom{n}{k}^r$. When the parameters are in appropriate ranges, we have $P_t \approx (1 - e^{-kr/n})^t$ and $\sum p_t P_t \approx P_{n(1-\exp(-kq/n))}$.

6. $L(k) = \sum_j \binom{m_1}{j} \binom{m_2}{k-j} L_1(j) L_2(k-j) / \binom{m_1+m_2}{k}$. [Hence if $L_1(k) \approx N_1 \alpha^{-k}$ and $L_2(k) \approx N_2 \alpha^{-k}$, then $L(k) \approx N_1 N_2 \alpha^{-k}$.]

7. (a) $L(1) = 3$, $L(2) = 1\frac{3}{4}$. (b) $L(1) = 3\frac{1}{4}$, $L(2) = 2\frac{1}{3}$, $L(3) = 1\frac{9}{16}$. [Note: Earl Sacerdoti has suggested the mapping

$$\begin{aligned} 0 & 0 * * \rightarrow 1 \\ 0 & 1 * * \rightarrow 2 \\ 1 & 0 * * \rightarrow 3 \\ 1 & 1 * * \rightarrow 4 \end{aligned}$$

which has a worse “worst case” but a better average case: $L(1) = 3$, $L(2) = 2\frac{1}{6}$, $L(3) = 1\frac{1}{2}$. A similar mapping exists whenever $m = 2^n$.]

10. (a) There must be $\frac{1}{6}v(v-1)$ triples, and x_v must occur in $\frac{1}{2}v$ of them. (b) Since v is odd, there is a unique triple $\{x_i, y_j, z\}$ for each i , and so S' is readily shown to be a Steiner triple system. The pairs missing in K' are $\{z, x_2\}, \{x_2, y_2\}, \{y_2, x_3\}, \{x_3, y_3\}, \dots, \{x_{v-1}, y_{v-1}\}, \{y_{v-1}, x_v\}, \{x_v, z\}$. (d) Starting with the case $v = 1$ and applying the operations $v \rightarrow 2v - 2$, $v \rightarrow 2v + 1$ yields all nonnegative numbers not of the form $3k + 2$, because the cases $6k + (0, 1, 3, 4)$ come respectively from the smaller cases $3k + (1, 0, 1, 3)$.

11. Take a Steiner triple system on $2v + 1$ objects. Call one of the objects z and name the other objects in such a way that the triples containing z are $\{z, x_i, \bar{x}_i\}$; delete these triples.

12. $\{k, (k+1) \bmod 14, (k+4) \bmod 14, (k+6) \bmod 14\}$, for $0 \leq k < 14$, where $(k+7) \bmod 14$ is the complement of k . [Complemented systems are a special case of so-called *group divisible* block designs; cf. Bose, Shrikhande, and Bhattacharya, *Ann. Math. Statistics* 24 (1953), 167–195.]

APPENDIX A

TABLES OF NUMERICAL QUANTITIES

Table 1

Quantities which are frequently used in standard subroutines and in analysis
of computer programs. (40 decimal places)

$\sqrt{2}$ =	1.41421	35623	73095	04880	16887	24209	69807	85697—
$\sqrt{3}$ =	1.73205	08075	68877	29352	74463	41505	87236	69428+
$\sqrt{5}$ =	2.23606	79774	99789	69640	91736	68731	27623	54406+
$\sqrt{10}$ =	3.16227	76601	68379	33199	88935	44432	71853	37196—
$\sqrt[3]{2}$ =	1.25992	10498	94873	16476	72106	07278	22835	05703—
$\sqrt[3]{3}$ =	1.44224	95703	07408	38232	16383	10780	10958	83919—
$\sqrt[3]{2}$ =	1.18920	71150	02721	06671	74999	70560	47591	52930—
$\ln 2$ =	0.69314	71805	59945	30941	72321	21458	17656	80755+
$\ln 3$ =	1.09861	22886	68109	69139	52452	36922	52570	46475—
$\ln 10$ =	2.30258	50929	94045	68401	79914	54684	36420	76011+
$1/\ln 2$ =	1.44269	50408	88963	40735	99246	81001	89213	74266+
$1/\ln 10$ =	0.43429	44819	03251	82765	11289	18916	60508	22944—
π =	3.14159	26535	89793	23846	26433	83279	50288	41972—
$1^\circ = \pi/180 =$	0.01745	32925	19943	29576	92369	07684	88612	71344+
$1/\pi =$	0.31830	98861	83790	67153	77675	26745	02872	40689+
$\pi^2 =$	9.86960	44010	89358	61883	44909	99876	15113	53137—
$\sqrt{\pi} = \Gamma(1/2) =$	1.77245	38509	05516	02729	81674	83341	14518	27975+
$\Gamma(1/3) =$	2.67893	85347	07747	63365	56929	40974	67764	41287—
$\Gamma(2/3) =$	1.35411	79394	26400	41694	52880	28154	51378	55193+
$e =$	2.71828	18284	59045	23536	02874	71352	66249	77572+
$1/e =$	0.36787	94411	71442	32159	55237	70161	46086	74458+
$e^2 =$	7.38905	60989	30650	22723	04274	60575	00781	31803+
$\gamma =$	0.57721	56649	01532	86060	65120	90082	40243	10422—
$\ln \pi =$	1.14472	98858	49400	17414	34273	51353	05871	16473—
$\phi =$	1.61803	39887	49894	84820	45868	34365	63811	77203+
$e^\gamma =$	1.78107	24179	90197	98523	65041	03107	17954	91696+
$e^{\pi/4} =$	2.19328	00507	38015	45655	97696	59278	73822	34616+
$\sin 1 =$	0.84147	09848	07896	50665	25023	21630	29899	96226—
$\cos 1 =$	0.54030	23058	68139	71740	09366	07442	97660	37323+
$\zeta(3) =$	1.20205	69031	59594	28539	97381	61511	44999	07650—
$\ln \phi =$	0.48121	18250	59603	44749	77589	13424	36842	31352—
$1/\ln \phi =$	2.07808	69212	35027	53760	13226	06117	79576	77422—
$-\ln \ln 2 =$	0.36651	29205	81664	32701	24391	58232	66946	94543—

Table 2

Quantities which are frequently used in standard subroutines and in analysis of computer programs, in *octal* notation. The name of each quantity, appearing at the left of the equal sign, is given in decimal notation.

0.1 =	0.06314 63146 31463 14631 46314 63146 31463 14631 4632
0.01 =	0.00507 53412 17270 24365 60507 53412 17270 24365 6051
0.001 =	0.00040 61115 64570 65176 76355 44264 16254 02030 4467
0.0001 =	0.00003 21556 13530 70414 54512 75170 33021 15002 3522
0.00001 =	0.00000 24761 32610 70664 36041 06077 17401 56063 3442
0.000001 =	0.00000 02061 57364 05536 66151 55323 07746 44470 2603
0.0000001 =	0.00000 00153 27745 15274 53644 12741 72312 20354 0215
0.00000001 =	0.00000 00012 57143 56106 04803 47374 77341 01512 6333
0.000000001 =	0.00000 00001 04560 27640 46655 12262 71426 40124 2174
0.0000000001 =	0.00000 00000 06676 33766 35367 55653 37265 34642 0163
$\sqrt{2}$ =	1.32404 74631 77167 46220 42627 66115 46725 12575 1744
$\sqrt{3}$ =	1.56663 65641 30231 25163 54453 50265 60361 34073 4222
$\sqrt{5}$ =	2.17067 36334 57722 47602 57471 63003 00563 55620 3202
$\sqrt{10}$ =	3.12305 40726 64555 22444 02242 57101 41466 33775 2253
$\sqrt[3]{2}$ =	1.20505 05746 15345 05342 10756 65334 25574 22415 0303
$\sqrt[3]{3}$ =	1.34233 50444 22175 73134 67363 76133 05334 31147 6012
$\sqrt[4]{2}$ =	1.14067 74050 61556 12455 72152 64430 60271 02755 7314
$\ln 2$ =	0.54271 02775 75071 73632 57117 07316 30007 71366 5364
$\ln 3$ =	1.06237 24752 55006 05227 32440 63065 25012 35574 5534
$\ln 10$ =	2.23273 06785 52524 25405 56512 66542 56026 46050 5071
$1/\ln 2$ =	1.34252 16624 53405 77027 35750 37766 40644 35175 0435
$1/\ln 10$ =	0.33626 75425 11562 41614 52825 33525 27655 14756 0622
π =	3.11037 55242 10264 30215 14230 63050 56006 70163 2112
$1^\circ = \pi/180$ =	0.01073 72152 11224 72344 25603 54276 63351 22056 1154
$1/\pi$ =	0.24276 30155 62344 20251 23760 47257 50765 15156 7007
π^2 =	11.67517 14467 62135 71322 25561 15466 30021 40654 3410
$\sqrt{\pi} = \Gamma(1/2)$ =	1.61337 61106 64736 65247 47035 40510 15273 34470 1776
$\Gamma(1/3)$ =	2.53347 35234 51013 61316 73106 47644 54653 00106 6605
$\Gamma(2/3)$ =	1.26523 57112 14154 74312 54572 37655 60126 23231 0245
e =	2.55760 52130 50535 51246 52773 42542 00471 72363 6166
$1/e$ =	0.27426 53066 13167 46761 52726 75436 02440 52371 0336
e^2 =	7.30714 45615 23355 33460 63507 35040 32664 25356 5022
γ =	0.44742 14770 67666 06172 23815 74376 01002 51313 2552
$\ln \pi$ =	1.11206 40443 47503 36413 65374 52661 52410 37511 4606
ϕ =	1.47433 57156 27751 23701 27634 71401 40271 66710 1501
e^γ =	1.61772 13452 61152 65761 22477 36553 53327 17554 2126
$e^{\pi/4}$ =	2.14275 31512 16162 52370 35530 11342 53525 44307 0217
$\sin 1$ =	0.65665 24436 04414 73402 03067 23644 11612 07474 1451
$\cos 1$ =	0.42450 50037 32406 42711 07022 14666 27320 70675 1232
$\zeta(3)$ =	1.14735 00023 60014 20470 15613 42561 31715 10177 0662
$\ln \phi$ =	0.36630 26256 61213 01145 13700 41004 52264 30700 4065
$1/\ln \phi$ =	2.04776 60111 17144 41512 11436 16575 00355 43630 4065
$-\ln \ln 2$ =	0.27351 71233 67265 63650 17401 56637 26334 31455 5701

Tables 1 and 2 contain several hitherto unpublished 40-digit values which have been furnished to the author by Dr. John W. Wrench, Jr.

For high-precision values of constants not found in this list, see J. Peters, *Ten Place Logarithms of the Numbers from 1 to 100000*, Appendix to Volume 1 (New York: F. Ungar Publ. Co., 1957); and *Handbook of Mathematical Functions*, ed. by M. Abramowitz and I. A. Stegun (Washington, D.C.: U. S. Govt. Printing Office, 1964), Chapter 1.

Table 3

Values of harmonic numbers, Bernoulli numbers, and Fibonacci numbers
for small values of n .

n	H_n	B_n	F_n	n
0	0	1	0	0
1	1	-1/2	1	1
2	3/2	1/6	1	2
3	11/6	0	2	3
4	25/12	-1/30	3	4
5	137/60	0	5	5
6	49/20	1/42	8	6
7	363/140	0	13	7
8	761/280	-1/30	21	8
9	7129/2520	0	34	9
10	7381/2520	5/66	55	10
11	83711/27720	0	89	11
12	86021/27720	-691/2730	144	12
13	1145993/360360	0	233	13
14	1171733/360360	7/6	377	14
15	1195757/360360	0	610	15
16	2436559/720720	-3617/510	987	16
17	42142223/12252240	0	1597	17
18	14274301/4084080	43867/798	2584	18
19	275295799/77597520	0	4181	19
20	55835135/15519504	-174611/330	6765	20
21	18858053/5173168	0	10946	21
22	19093197/5173168	854513/138	17711	22
23	444316699/118982864	0	28657	23
24	1347822955/356948592	-236364091/2730	46368	24
25	34052522467/8923714800	0	75025	25

For any x , let $H_x = \sum_{n \geq 1} \left(\frac{1}{n} - \frac{1}{n+x} \right)$. Then

$$H_{1/2} = 2 - 2 \ln 2,$$

$$H_{1/3} = 3 - \frac{1}{2}\pi/\sqrt{3} - \frac{3}{2} \ln 3,$$

$$H_{2/3} = \frac{3}{2} + \frac{1}{2}\pi/\sqrt{3} - \frac{3}{2} \ln 3,$$

$$H_{1/4} = 4 - \frac{1}{2}\pi - 3 \ln 2,$$

$$H_{3/4} = \frac{4}{3} + \frac{1}{2}\pi - 3 \ln 2,$$

$$H_{1/5} = 5 - \frac{1}{2}\pi\phi \sqrt{\frac{2+\phi}{5}} - \frac{1}{2}(3-\phi) \ln 5 - (\phi - \frac{1}{2}) \ln(2+\phi),$$

$$H_{2/5} = \frac{5}{2} - \frac{1}{2}\pi/\phi \sqrt{2+\phi} - \frac{1}{2}(2+\phi) \ln 5 + (\phi - \frac{1}{2}) \ln(2+\phi),$$

$$H_{3/5} = \frac{5}{3} + \frac{1}{2}\pi/\phi \sqrt{2+\phi} - \frac{1}{2}(2+\phi) \ln 5 + (\phi - \frac{1}{2}) \ln(2+\phi),$$

$$H_{4/5} = \frac{5}{4} + \frac{1}{2}\pi\phi \sqrt{\frac{2+\phi}{5}} - \frac{1}{2}(3-\phi) \ln 5 - (\phi - \frac{1}{2}) \ln(2+\phi),$$

$$H_{1/6} = 6 - \frac{1}{2}\pi\sqrt{3} - 2 \ln 2 - \frac{3}{2} \ln 3,$$

$$H_{5/6} = \frac{5}{3} + \frac{1}{2}\pi\sqrt{3} - 2 \ln 2 - \frac{3}{2} \ln 3,$$

and, in general, when $0 < p < q$ (cf. exercise 1.2.9-19),

$$H_{p/q} = \frac{q}{p} - \frac{1}{2}\pi \cot \frac{p}{q} \pi - \ln 2q + 2 \sum_{1 \leq n < q/2} \cos \frac{2\pi np}{q} \ln \sin \frac{n}{q} \pi.$$

INDEX TO NOTATIONS

In the following formulas, letters which are not further qualified have the following significance:

- j, k integer-valued arithmetic expression
- m, n nonnegative integer-valued arithmetic expression
- x, y, z real-valued arithmetic expression
- f real-valued function
- P pointer-valued expression, i.e., either Λ or an address within a computer.
- α string of symbols

Formal symbolism	Meaning	Section reference
$\text{NODE}(P)$	the node (group of variables which are individually distinguished by their field names) whose address is P , $P \neq \Lambda$	2.1
$F(P)$	the variable in $\text{NODE}(P)$ whose field name is F	2.1
$\text{CONTENTS}(P)$	contents of computer "word" whose address is P	2.1
$\text{LOC}(V)$	address of variable V within a computer	2.1
A_n or $A[n]$	the n th element of linear array A	1.1
A_{mn} or $A[m, n]$	the element in row m , column n of rectangular array A	1.1
$V \leftarrow E$	give variable V the value of expression E	1.1
$U \leftrightarrow V$	interchange the values of variables U and V	1.1
$P \Leftarrow \text{AVAIL}$	set the value of pointer variable P to the address of a new node, or signal memory overflow if there is no room for a new node	2.2.3
$\text{AVAIL} \Leftarrow P$	$\text{NODE}(P)$ is returned to free storage; all its fields lose their identity	2.2.3

Formal symbolism	Meaning	Section reference
$\text{top}(S)$	node at the top of a nonempty stack S	2.2.1
$X \leftarrow S$	pop up S to X : set $X \leftarrow \text{top}(S)$; delete $\text{top}(S)$ from nonempty stack S	2.2.1
$S \Leftarrow X$	push down X onto S : insert the value or group of values denoted by X as a new entry on the top of stack S	2.2.1
$(B \Rightarrow E_1; E_2)$	conditional expression: denotes E_1 if B is true, E_2 if B is false	8.1
δ_{jk}	Kronecker delta: $(j = k \Rightarrow 1; 0)$	1.2.6
$\sum_{R(k)} f(k)$	sum of all $f(k)$ such that k is an integer and relation $R(k)$ is true	1.2.3
$\prod_{R(k)} f(k)$	product of all $f(k)$ such that k is an integer and relation $R(k)$ is true	1.2.3
$\min_{R(k)} f(k)$	minimum value of all $f(k)$ such that k is an integer and relation $R(k)$ is true	1.2.3
$\max_{R(k)} f(k)$	maximum value of all $f(k)$ such that k is an integer and relation $R(k)$ is true	1.2.3
$j \backslash k$	j divides k : $k \bmod j = 0$	1.2.4
$U \setminus V$	set difference: $\{x \mid x \text{ is in } U \text{ but not in } V\}$	
$\gcd(j, k)$	greatest common divisor of j and k :	
	$(j = k = 0 \Rightarrow 0; \max_{d \mid j, d \mid k} d)$	1.1
$\det(A)$	determinant of square matrix A	1.2.3
A^T	transpose of rectangular array A :	
	$A^T[j, k] = A[k, j]$	1.2.3
x^y	x to the y power, x positive	1.2.2
α^R	left-right reversal of α	
x^k	x to the k th power:	
	$\left(k \geq 0 \Rightarrow \prod_{0 \leq j < k} x; \quad 1/x^{-k} \right)$	1.2.2
\bar{x}^k	x upper k :	
	$\left(k \geq 0 \Rightarrow x(x+1) \cdots (x+k-1) \right.$	
	$\quad \left. = \prod_{0 \leq j < k} (x+j); \quad 1/(x+k)^{-k} \right)$	1.2.6

Formal symbolism	Meaning	Section reference
x^k	x lower k : $\begin{aligned} k \geq 0 \Rightarrow x(x-1) \cdots (x-k+1) \\ = \prod_{0 \leq j < k} (x-j); \\ 1/(x-k) = (-1)^k (-x)^{\overline{k}} \end{aligned}$	1.2.6
$n!$	n factorial: $1 \cdot 2 \cdots n = n^n$	1.2.5
$\binom{x}{k}$	binomial coefficient: $(k < 0 \Rightarrow 0; x^k/k!)$	1.2.6
$\binom{n}{n_1, n_2, \dots, n_m}$	multinomial coefficient, $n = n_1 + n_2 + \cdots + n_m$	1.2.6
$\left[\begin{matrix} n \\ m \end{matrix} \right]$	Stirling number of first kind: $\sum_{0 < k_1 < k_2 < \cdots < k_{n-m} < n} k_1 k_2 \cdots k_{n-m}$	1.2.6
$\left\{ \begin{matrix} n \\ m \end{matrix} \right\}$	Stirling number of second kind: $\sum_{0 \leq k_1 \leq k_2 \leq \cdots \leq k_{n-m} \leq m} k_1 k_2 \cdots k_{n-m}$	1.2.6
$\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle$	Eulerian number	5.1.3.
$\{a \mid R(a)\}$	set of all a for which the relation $R(a)$ is true	
$\{a_1, \dots, a_n\}$	the set or multiset $\{a_k \mid 1 \leq k \leq n\}$	
$\{n_1 \cdot a_1, \dots, n_m \cdot a_m\}$	the multiset containing n_k occurrences of a_k	5.1.2
$\ M\ $	cardinality: the number of elements in the multiset M	
$ x $	absolute value of x : $(x < 0 \Rightarrow -x; x)$	
$ \alpha $	length of α	
$\lfloor x \rfloor$	floor of x , greatest integer function: $\max_{k \leq x} k$	1.2.4
$\lceil x \rceil$	ceiling of x , least integer function: $\min_{k \geq x} k$	1.2.4
$x \bmod y$	mod function: $(y = 0 \Rightarrow x; x - y \lfloor x/y \rfloor)$	1.2.4
$x \equiv y$ (modulo z)	relation of congruence: $x \bmod z = y \bmod z$	1.2.4

Formal symbolism	Meaning	Section reference
$\log_b x$	logarithm, base b , of x (real positive $b \neq 1$): $x = b^{\log_b x}$	1.2.2
$\ln x$	natural logarithm: $\log_e x$	1.2.2
$\exp x$	exponential of x : e^x	1.2.2
$\langle X_n \rangle$	the infinite sequence X_0, X_1, X_2, \dots (here n is a letter which is part of the symbol)	1.2.9
$f'(x)$	derivative of f at x	1.2.9
$f''(x)$	second derivative of f at x	1.2.10
$f^{(n)}(x)$	n th derivative: ($n = 0 \Rightarrow f(x)$); $g'(x) \quad \text{where} \quad g(x) = f^{(n-1)}(x)$	1.2.11.2
$f(x) _y^z$	$f(z) - f(y)$	
$H_n^{(x)}$	$1 + 1/2^x + \dots + 1/n^x = \sum_{1 \leq k \leq n} 1/k^x$	1.2.7
H_n	harmonic number: $H_n^{(1)}$	1.2.7
F_n	Fibonacci number: $(n \leq 1 \Rightarrow n; F_{n-1} + F_{n-2})$	1.2.8
B_n	Bernoulli number	1.2.11.2
$\Re(w)$	real part of the complex number w	
$\Im(w)$	imaginary part of the complex number w	
$\zeta(x)$	zeta function: $H_\infty^{(x)}$ when $x > 1$	1.2.7
$\Gamma(x)$	gamma function: $\gamma(x, \infty)$; $(x - 1)!$ when x is a positive integer	1.2.5
$\gamma(x, y)$	incomplete gamma function	1.2.11.3
γ	Euler's constant	1.2.7
e	base of natural logarithms: $\sum_{k \geq 0} 1/k!$	1.2.2
ϵ	empty string (of length zero)	
∞	infinity: larger than any number; or an artificially high key value	5
$M \uplus N$	sum of multisets M and N ; e.g., $\{a, a, b\} \uplus \{a, b, c\} = \{a, a, a, b, b, c\}$	4.6.3
Λ	null link (pointer to no address)	2.1
\emptyset	empty set (set with no elements)	
ϕ	golden ratio, $\frac{1}{2}(1 + \sqrt{5})$	1.2.8

Formal symbolism	Meaning	Section reference
$\varphi(n)$	Euler's totient function: $\sum_{\substack{0 \leq k < n \\ \gcd(k, n) = 1}} 1$	1.2.4
$O(f(n))$	big-oh of $f(n)$ as $n \rightarrow \infty$: a quantity whose magnitude is less than some constant times $f(n)$, for all large n	1.2.11.1
$O(f(x))$	big-oh of $f(x)$, for small x (or for x in some specified range)	1.2.11.1
(min x_1 , ave x_2 , max x_3 , dev x_4)	a random variable having minimum value x_1 , average ("expected") value x_2 , maximum value x_3 , standard deviation x_4	1.2.10
mean(g)	mean value of probability distribution represented by generating function g : $g'(1)$	1.2.10
var(g)	variance of probability distribution represented by generating function g :	
	$g''(1) + g'(1) - g'(1)^2$	1.2.10
P\$	address of symmetric order successor of NODE(P) in a binary tree	2.3.1
■	end of algorithm, program, or proof	1.1
□	one blank space	1.3.1
rA	register A (accumulator) of MIX	1.3.1
rX	register X (extension) of MIX	1.3.1
rI1, ..., rI6	(index) registers I1, ..., I6 of MIX	1.3.1
rJ	(jump) register J of MIX	1.3.1
(L:R)	partial field of MIX word, $0 \leq L \leq R \leq 5$	1.3.1
OP ADDRESS, I(F)	notation for MIX instruction	1.3.1, 1.3.2
u	unit of time in MIX	1.3.1
*	"self" in MIXAL	1.3.2
OF, 1F, 2F, ..., 9F	"forward" local symbol in MIXAL	1.3.2
OB, 1B, 2B, ..., 9B	"backward" local symbol in MIXAL	1.3.2
OH, 1H, 2H, ..., 9H	"here" local symbol in MIXAL	1.3.2

INDEX AND GLOSSARY

*If you don't find it in the Index,
look very carefully through the entire catalogue.*

—*Consumer's Guide, Sears, Roebuck and Co. (1897)*

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.

- Abbreviated keys, 401, 505, 543–544.
Abel, Niels Henrik, binomial formula, 545.
 limit theorem, 698.
Abraham, C. T., 567.
Abramowitz, Milton, 703.
Addition of polynomials, 166.
Addition to list, *see* Insertion.
Address calculation sorting, 99–102, 105,
 379–381, 388, 665.
Address table sorting, 74–75, 80.
Adel'son-Vel'skiĭ, Georgiĭ Maksimovich,
 451, 453.
Aho, Alfred Vaino, 470.
al-Khowārizmī, abu Ja'far Mohammed ibn
 Mūsā, 8.
Alekseyev, Vladimir Evgen'evich, 234, 239,
 242.
ALGOL, 423.
Algorithms, analysis of, *see* Analysis.
 comparison of, *see* Comparison.
 proof of, 50–52, 112–113, 317, 326,
 359–360.
Alphabetic code, 444–445.
Alternating runs, 46.
Amdahl, Gene M., 541.
American Library Association, 7.
AMM: American Mathematical Monthly.
Amphisbaenic sort, 351, 387.
Anagrams, 10.
Analysis of algorithms, 4, 78, 80, 82, 86–95,
 97, 101–105, 108–111, 119–123, 127–139,
 141, 153–159, 163, 168–169, 176, 178,
 260–266, 272–276, 286–288, 294–300,
 332–337, 343–346, 360–361, 382, 386,
 394–405, 409–414, 420–422, 427–433,
 447–449, 459–463, 470–471, 476, 480,
 493–505, 527–532, 536–540, 545–549,
 596; *see also* Complexity analysis.
AND (logical and), 135, 524.
André, Antoine Desire, 71, 588.
Anuyogadvarā-sutra, 23.
Apollonius Sophista, son of Archibius, 418.
Archimedes, 14.
Argument, 389.
Arisawa, Makoto, 563.
Arithmetic progression, 510.
Armerding, George W., 402.
Armstrong, Philip Nye, 227, 244, 642.
Arora, Sant Ram, 448.
Ashenhurst, Robert Lovett, 347, 352.
Associative law, 24, 454.
Associative memory, 389, 567.
Asymptotic methods, 41–43, 66–67, 72,
 129–139, 196, 287–288, 403, 470,
 501–503.
Atia, Rabbi Mordecai, 23.
Attributes, 550, *see* Secondary keys.
 binary, 557–564.
 compound, 557.
Automatic programming, 386.
Average cost, minimization of, 193–197,
 217–220, 410.
AVL trees, 465, *see* Balanced trees.

B-trees, 473–480.
Babbage, Charles, 180.
Baber, Robert Laurence, 667.
Babylonian mathematics, 417.
Backwards reading, *see* Read-backwards.
Baer, Robert M., 72.

- Balance factor, 452, 456.
Balanced filing, 564–566, 569.
Balanced merging, 247–250, 266, 299, 301,
314, 327–330, 336–339, 356, 385–386.
Balanced radix sort, 347, 352.
Balanced trees, 152, 451–471, 540.
 weight-balanced, 468, 471.
Ball, Walter William Rouse, 580.
Ballot problem, 64.
Barton, David Elliott, 44, 45, 585, 586, 588.
Basic query, 558, 563–564, 566, 569.
Batcher, Kenneth Edward, 111, 224–225,
227, 229–230, 232, 233, 236, 638.
 sorting method, *see* Merge exchange.
Batching, 98–99, 551.
Bayer, Rudolf, 469, 473, 477, 480.
Bayes, Anthony John, 350.
Bell, Colin J., 341.
Bell, David Arthur, 169, 387, 621.
Bell, James Richard, 524, 525.
Bellman, Richard Ernest, vii.
Bencher, Dennis Leo, 315, 318.
Bender, Edward Anton, vi, 588, 591, 663.
Bennett, Brian Thomas, 378.
Berge, Claude, 580.
Berners-Lee, Conway Maurice, 98, 446.
Bernoulli numbers, 499, 585, 703.
Bertrand, Joseph Louis François, 588.
Best-fit allocation, 471.
Best possible, 181.
Beta distribution, 573.
Betz, B. K., 268, 289.
Beus, H. Lynn, 245, 246.
Bháskara Áchárya, 23.
Bhattacharya, K. N., 700.
Bienaymé, Irénee Jules, 588.
BINAC, 385.
Binary attributes, 557–564.
Binary insertion sort, 83–84, 184, 186, 188,
374.
Binary merging, 205–206, 208.
Binary search, 83, 204–205, 406–414,
419–422, 432, 438, 538.
 uniform, 411–413, 420.
Binary search trees, 423–471, 501.
 optimum, 433–451.
 pessimum, 450.
Binary tree: Either empty, or a root node
 and its left and right binary subtrees;
 see also Complete binary tree, Extended
 binary tree.
 enumeration of, 63.
 triply-linked, 159, 467–468.
Binomial coefficients, 29–31, 88.
Binomial probability distribution, 101, 345,
532.
Binomial transform, 137, 501.
Birthday paradox, 506–507, 542.
Bisection, *see* Binary search.
Bitonic sort, 232–233, 237.
Bleier, Robert E., 567.
Block (on tape), 321–322.
Block designs, 565–569.
Bloom, Burton H., 561, 567, 700.
Blum, Manuel, 216.
Boehme McGraw, Elaine M., 541.
Boerner, Hermann, 639.
Boolean query, 550, 553–555.
Booth, Andrew Donald, 393, 397, 419, 447.
Boothroyd, John, 595.
Bose, Raj Chandra, 227–228, 567, 700.
Bottenbruch, Herman, 419, 422.
Bouricius, Willard Gail, 196.
Bourne, Charles Percy, 392, 567.
Brawn, Barbara S., 665.
Brent, Richard Peirce, 525, 526, 539.
Brock, Paul, 72.
Brown, William Stanley, 158.
Brute force, 66–67, 586, 589, 592, 599,
612–615, 666, 672, 691, 693, 698.
Bryce, James W., 384.
Bubble sort, 106–111, 134–135, 141, 223–224,
244–246, 352–353, 360, 379, 386, 388,
644.
Buchholz, Werner, 393, 541.
Buckets, 534–538, 541.
Buffering, 324–327, 343–346, 385–386.
 size of buffers, 335–336, 353, 370–371,
 377.
Bulk memory, *see* Direct-access storage.
Burge, William Herbert, 279, 299, 341.
Burton, Robert, v.
- CACM: Communications of the ACM.*
- Cancellation law, 24.
Card, edge-notched, 1, 558–559, 567.
Card sorters, 338, 381, 383–384.
Carlitz, Leonard, 38, 47, 599.
Caron, Jacques, 280.
 polyphase merge, 280–281, 288.
Carroll, Lewis (= Dodgson, Rev. Charles
Lutwidge), 209–211, 218, 571.
Carter, William Caswell, 279, 289, 298.
Cascade merge, 289–301, 310, 313, 329–331,
336–337, 342, 346.
 read-backwards, 302, 312, 331, 337.
 rewind overlap, 299, 301, 330, 336.
Cascade numbers, 295–298, 300.
Cascading pseudo-radix sort, 351.
Catalan numbers, 63, 296.
Catenated search, 404–405.
Cawdrey (=Cawdry), Robert, 418–419.
Cayley, Arthur, 605, 627.
Census, 381–385, 392.
Centered insertion, *see* Two-way insertion.
Césari, Yves, 195, 280.

- Chaining, 373, 513–518, 521, 535–540.
Chakravarti, Gurugovinda, 23.
Chandra, Ashok Kumar, 419.
Chang, Shi-Kuo, 451.
Channels, 338–339.
Chartres, Bruce Aylwin, 158.
Chase, Stephen Martin, 198.
Chebyshev, Pafnuti L'vovich, 392.
 polynomials, 297, 652.
Chow, David Kuo-kien, 566.
Church, Randolph, 640.
CI: MIX's comparison indicator.
Circular lists, 518.
Cliques, 10.
Closest match, search for, 9, 391, 405.
Coalescing lists, 514–518, 536.
COBOL, 342, 525.
Cocktail shaker sort, 110, 135, 360, 643, 661.
Coffman, E. G., Jr., 489.
Colin, Andrew J. T., 447.
Collating, 384–385, *see* Merging.
Collating sequence, 7, 417.
Collision resolution, 508, 513–526, 534–538.
Column sorting, 347, *see* Distribution
 sorting.
Comp. J.: The Computer Journal.
Comparator modules, 220–222, 236, 243.
Comparison counting sort, 76–79.
Comparison of algorithms, 152, 327–342,
 379–381, 538–540.
Comparison of keys, 4.
 minimizing, 181–220.
 multiprecision, 6.
 parallel (i.e., simultaneous), 114, 222–224,
 229–230, 233, 236, 243, 422.
 searching by, 396, 406–480, 540.
 sorting by, 76–123, 134–170, 181–198,
 220–346, 352–381.
Comparison trees, 182–183, 194–198.
Compiler techniques, 2–3, 423, 525.
Complement notation, 179.
Complements, finding all pairs, 9.
Complete binary tree: A binary tree with
 nodes numbered 1 to n , where
 node $\lfloor k/2 \rfloor$ is the father of node k ,
 144–145, 153–154, 256, 422, 460.
Complete P -ary tree, 365.
Complex partitions, 22.
Complexity analysis of algorithms, 170,
 179–246, 304–314, 357–360, 371–378,
 386–388, 422, 433–451, 532–534, 643.
Component of a graph, 190.
Compound attributes, 557.
Compound leaf of a tree, 654.
Compression of data, 401, 505.
Compromise merge, 299.
Computational complexity, *see* Complexity.
Computed entry table, *see* Scatter table.
Computer operator, skilled, 327, 341, 353.
Computer Sciences Corporation, 2.
Comrie, Leslie John, 171, 384.
Concatenation, 466, 469, 470.
Cookies, 556–560, 565–566.
Coroutine, 258.
Counting, sorting by, 76–80.
Covering, 236.
Coxeter, Harold Scott Macdonald, 580.
Cramer, Gabriel, 11.
Crane, Clark Allan, vi, 150, 151, 466–468,
 470, 676.
Creation, Book of, 23.
Criss-cross merge, 315.
Cross-indexing, *see* Secondary key retrieval.
Cross-reference routine, 7.
Cube, n -dimensional, linearized, 405.
Cundy, Henry Martyn, 580.
Cycle of a permutation, 25–32, 157, 596,
 605, 615, 631.
Cylinders, 361–362, 373, 376, 473, 554.
Daly, Lloyd William, 418.
Data base, 389.
Data structure, choice of, 96, 144, 152, 165,
 168, 171–172, 405, 452, 472, 552–554.
Dauer, Francis Watanabe, 399.
David, Florence Nightingale, 44, 585, 588.
Davidson, Leon, 392.
Davies, Donald Watts, 387.
Davis, David Robert, 566.
de Balbini, Guy, 521.
de Bruijn, Nicolaas Govert, 131, 138, 699.
de la Briandais, Rene, 483.
de Peyster, J. A., 538.
de Staël, Madame, *see* Staël-Holstein.
Deadlines, 404.
Dedekind, Richard, 240.
Degenerate tree, 426, 448, 674.
Degenerative addresses, 541.
Degree path length, 367–368, 371–373.
Deletion: Removing an item.
 from B -tree, 480.
 from balanced tree, 465–466, 470.
 from binary search tree, 428–432,
 448–449.
 from digital search tree, 501.
 from hash tables, 526–527, 544.
 from leftist tree, 159.
 from trie, 500.
Demuth, Howard B., 109, 185, 246, 352,
 357, 386–387, 643.
Dent, Warren Thomas, 448.
Determinant, 11, 20.
Diagram of partial order, 64–65, 184–185,
 189.
Digital searching, 481–505.
Digital sorting, 170, 347, *see* Distribution.

- Digital tree search, 489–490, 494–497, 500–504, 510.
Diminishing increment sort, 84, *see* Shellsort.
Dinsmore, Robert John, 258.
Direct-access storage, 361–378, 404, 472–480, 553–554.
Directed graphs, 10, 64–65, 185.
Discrete entropy, 374–375.
Discrete logarithms, 9.
Discriminant, 61, 69–70, 72.
Disks, 361, *see* Direct-access storage.
Disorder, measures of, 11, 105, 134.
Distribution counting sort, 78–80, 172, 178, 379, 381.
Distribution functions, *see* Probability.
Distribution patterns, 347–352.
Distribution sorting, 5–6, 78–80, 170–180, 347–352, 355, 374, 379–381, 386.
dual to merging, 349–352.
Dodd, Marisue, 512.
Dodgson, Rev. Charles Lutwidge, 209, *see* Carroll.
Doren, David G., 215, 219, 635.
Double-entry bookkeeping, 552.
Double hashing, 521–526.
Doubly exponential sequence, 459.
Douglas, Alexander Shafto, 98, 387.
Drake, Paul, 1.
Drums, 363, *see* Direct-access storage.
Dudeney, Henry Ernest, 576.
Dugundji, James, 244.
Dull, Brutus Cyclops, 6, 45, 542.
Dumey, Arnold Isaac, 254, 393, 419, 446, 541.
Dummy runs, 247, 270–279, 284–288, 290–294, 311–312, 345–346.
Dynamic storage allocation, 11, 471.
Dynamic table searching, 390, *see* Symbol table algorithms.

Eckert, John Presper, 385, 386.
Edge-notched cards, 1, 558–559, 567.
Editing routines, 3, 168.
Edmund, Norman Wilson, 1.
EDVAC, 384–385.
Efficiency of a graph, 189–193.
Eichelberger, Edward B., 667.
Elcock, Edward Wray, 544, 689.
Elementary symmetric functions, 240, 592.
Elevators, 150, 357–360, 374–378.
Ellery, Robert Lewis, 392.
Empirical data, 95.
Encoding surnames, 391–392.
English language, 1–2, 9–10, 418, 486–489, 541–542.
most common words of, 432–433, 481–482, 484, 489, 506–507.
Entropy, discrete, 374–375.
Enumeration of binary trees, 63.
Enumeration of permutations, 23–24, 27–32.
Enumeration sorting, 73, 76–78.
Equality of sets, testing, 209.
approximate, 9.
Eratosthenes, 617.
Erdélyi, Arthur, 132.
Erdős, Paul, 69.
Ershov, Andrei Petrovich, 541.
Euclid's algorithm, 509.
Euler, Leonhard, 9, 20–22, 35, 37, 38, 392, 685.
numbers (secant numbers), 35, 593.
summation formula, 67, 129–131, 138, 615, 666.
Eulerian numbers, 34–40, 45, 627.
Eve, James, 489.
Even permutation, 20, 198.
Exchange selection sort, 107, *see* Bubble sort.
Exchange sorting, 73, 105–139.
optimum, 198.
Exclusive or, 512, 682.
Exercises, notes on, vii–ix.
Exponential integral, 139.
Extended binary tree: Either a single “external” node, or an “internal” root node plus its left and right extended binary subtrees, 182.
External path length: Sum of the level numbers of all external nodes, *see* Path length.
External searching, 390, 399–401, 404, 471–480, 490–493, 534–541, 548, 553–554.
External sorting, 5, 7–10, 247–378.
summary, 327–342, 361–371.

Factorials, 23, 188.
Factorization of permutations, 25–32.
Fallacies in probability, 421.
Fast Fourier transform, 239.
Feature cards, 559, 567.
Feldman, Jerome Arthur, 567.
Feller, William, 507.
Ferguson, David Elton, vi, 291–292, 298, 300, 370, 419.
Feurzig, Wallace, 79.
Fibonacci, Leonardo, 420–421.
Fibonacci distributions, 268–271, 276–277, 286, 303.
Fibonacci hashing, xii, 510–512.
Fibonacci number system, 287, 350, 352, 421.
Fibonacci numbers, 93, 267, 414–415, 601, 653, 674, 703.
generalized, 267–268, 287, *see* Cascade numbers.

- Fibonacci search, 414.
Fibonacci trees, 414–415, 419–422, 450, 452–453, 465, 470, 674, 677.
Fibonaccian search, 414–416, 419–421.
Fifo (first-in-first-out) tree, 312–314, 349.
File, 4, 389.
partitioning, 505.
self-organizing, 398–399, 403, 514, 620.
Finite field, 542.
First-fit allocation, 471.
Fixed point of permutation, 65, 68.
Fletcher, William, 598.
Floating buffers, 324–327, 343–346.
Flores, Ivan, 387.
Floyd, Robert W, vi, 145–146, 158, 217–220, 227–228, 230, 237, 240, 298, 374, 378, 512, 605, 634, 635, 662.
Foata, Dominique Cyprien, 21, 24, 27, 33, 43, 45, 587, 692.
Ford, Donald Floyd, 392.
Ford, Lester Randolph, Jr., 185, 188.
Forecasting, 324–327, 345.
Forest, 483–485, 500.
FORTRAN, 2–3, 7, 423, 542.
Foster, Caxton Croxford, 3, 462, 675.
Fourier, Jean Baptiste Joseph, transform, 239.
Frame, James Sutherland, 63.
Frank, Robert Morris, 93.
Franklin, Fabian, 20, 22.
Frazer, William Donald, 123, 259, 646, 667.
Fredkin, Edward, 481.
Free group, 504–505.
Frequency of access, 396–405, 432–451, 504, 525.
Friend, Edward Harry, 79, 109, 142, 172, 254, 258, 327, 341, 351, 387, 624.
Frobenius, Ferdinand Georg, 61.
Front and rear compression, 505.

Gaines, Helen Fouché, 433.
Gale, David, 639.
Galen, Claudius, 418.
Galli, E. J., 489.
Gamma function, 132–134, 138, 503.
Gandz, Solomon, 23.
Gaps between prime numbers, 401–402.
Gaps on tape, 321–322, 333.
Gardner, Erle Stanley, 1.
Gardner, Martin, 576, 626.
Gassner, Betty Jane, 40–41, 262.
Gaudette, C. H., 351.
Gauss, Karl Friedrich, 392.
gcd: Greatest common divisor.
Generating functions, 15–22, 33–34, 37–43, 45–48, 65, 72, 103, 105, 136, 178, 195, 261, 264, 269, 272, 275–276, 286–288, 295–300, 344–346, 422, 448, 459–460, 480, 495–498, 502, 531–532, 546, 597, 607–611, 697.
Genoa, Giovanni di, 418.
Ghosh, Sakti Pada, 392, 477, 567–568.
Gibson, Kim Dean, 576.
Gilbert, Edgar Nelson, 445, 447.
Gilstad, Russell L., 303, 339.
Gleason, Andrew Mattei, 194, 622.
Goetz, Martin A., 299, 318–319, 341, 373, 387, 647.
Golden ratio, 510–511.
Goldenberg, Daniel, 386.
Goodwin, D. T., 304.
Gotlieb, Calvin Carl, 387, 439.
Graham, Ronald Lewis, vi, 199, 205, 207, 242, 543, 688.
Grasseli, Antonio, 640.
Gray, Harry Joshua, 566.
Greatest common divisor, 186, 509.
Green, Milton Webster, 229, 240, 638, 641.
Greniewski, Marek, 507.
Gross, Oliver Alfred, 196, 627.
Group, free, 504–505.
Growth ratio, 272.
Gruenberger, Fred Joseph, 402.
Guibas, Leonidas, vi, 518.
Gustafson, Richard Alexander, 562–563.
Gustavson, Frances Goertzel, 665.

Hadian, Abdollah, 187, 219, 241.
Hall, Marshall, Jr., 12, 504, 566.
Hall, Philip, 633.
Halpern, Mark Irwin, 419.
Hamilton, Douglas Alan, 674.
Hanan, Maurice, 688.
Hardy, Godfrey Harold, 667.
Hardy, Norman, 578.
Harmonic numbers, 610, 703–704.
Harper, Lawrence Hueston, 667.
Harrison, Malcolm Charles, 561, 700.
Harrison, Michael Alexander, iv.
Hash functions, 508–513, 521–524, 542–543.
combinatorial, 563–564, 568.
Hash sequences, 528, 545.
Hashing, 390, 506–549, 563–564, 568.
Heaps, 145–147, 150, 152–159, 252, 339–340, 667.
Heapsort, 145–149, 153–158, 339–340, 380–381, 388, 665.
Height of tree, 197, 452, 456.
Heilbronn, Hans, 392.
Heising, William Paul, 397, 697.
Heller, Robert Andrew, 505.
Hennie, Frederick Clair, 355–356, 360.
Hibbard, Thomas Nathaniel, 21, 93, 198, 227, 387, 410, 429, 447, 631.
Hilbert, David, 392.

- Hildebrandt, Paul, 128.
Hindenburg, Karl Friedrich, 14.
Hinton, Charles Howard, 579.
Hoare, Charles Antony Richard, 116,
 122–123, 136, 388.
Holberton, Frances Elizabeth Snyder, 327,
 385–386.
Hollerith, Herman, 382–384.
Holt Hopfenburg, Anatol Wolf, 697.
Homer, 418.
Homogeneous comparisons, 220–221.
Hooker, William Weston, 43.
Hooks, 62–63, 71.
Hopcroft, John Edward, vi, 245, 468.
Hopgood, Frank Robert Albert, 540.
Hosken, J. C., 387, 388.
Hu, Te Chiang, 439, 447, 450, 451.
Hu-Tucker algorithm, 439–444, 450–451.
Hubbard, George Underwood, 367, 388.
Huffman, David Albert, 365, 435, 440.
Hunt, Douglas H., 89.
Hurwitz, Henry, 610.
Hwang, Frank Kwangming, 189, 197,
 204–208.
- IBM Corporation, 171, 318, 383–384.
IBM 701, 541.
IBM 705, 83.
Iff: If and only if.
Inakibit-Anu of Uruk, 417.
Inclusion-exclusion principle, 573, 700.
Inclusive query, 558–563, 566.
Index for file partitioning, 505.
Index of a permutation, 16–18, 21, 33.
Index-sequential file organization, 473.
Index to this book, 552–553, 710–722.
Infinity, 5, 139.
Information retrieval, 389.
Information theory, 184.
Inner loop, *see* Main loop.
Insertion: Adding a new item, 18.
 into *B*-tree, 475–480.
 into balanced tree, 453–465, 471.
 into binary search tree, 424–427, 473, 480.
 into digital search tree, 490, 501.
 into hash tables, *see* Collision resolution.
 into leftist tree, 152.
 into trie, 500.
Insertion sorting, 73, 80–105, 180–188,
 194–195, 223, 379–381, 428.
Interblock gaps, 321–322, 333.
Intercalation product of permutations,
 24–34.
Internal (branch) node of binary tree, 182,
 see Extended binary tree.
Internal path length: Sum of the level
 numbers of all internal nodes, *see* Path
 length.
Internal searching, 390, 393–471, 481–534,
 538–555.
 summary, 538–540.
Internal sorting, 5, 73–246, 379–388.
 summary, 379–382.
Interpolation search, 416–419, 421–422.
Interpretive routine, 488.
Interval exchange sort, 128–129, 137.
Inverse of a permutation, 14–15, 19, 32, 76.
Inversions of a permutation, 11–22, 33, 78,
 82, 87–90, 103–104, 109, 157, 170, 198,
 352–353.
 table of, 12–13, 18–19, 108–109, 134–145,
 352–353, 597.
Inverted files, 552–559, 565–567.
Involutions, 19, 48–49, 65–67, 72.
Isaac, Earl J., 99.
Isbitz, Harold, 128.
Isidorus, St., of Seville, 418.
Isomorphisms, 10.
Iverson, Kenneth Eugene, 110, 144, 387,
 393, 419, 420, 447, 667.
- JACM: Journal of the ACM.*
- Jacobi, Carl Gustav Jacob, 21–22.
Jacobsen, William H., Jr., 488.
JAE (jump A even), 126, 500.
Jainism, 23.
JAO (jump A odd), 126.
Johnsen, Robert Lawrence, 505.
Johnsen, Thorstein Lunde, 299.
Johnson, Donald W., 624.
Johnson, Lyle Robert, 494, 566.
Johnson, Selmer Martin, 185, 188.
Johnson, Stephen Curtis, 548.
Josephus, Flavius, problem, 18–19, 579.
Jump operator of MIX, 6.
- Kaman, Charles Henry, 524, 525.
Kant, Immanuel, 392.
Karp, Richard Manning, vi, 105, 199, 288,
 304, 309, 312–314, 350, 356–360, 639,
 670.
Kaufman, Marc Thomas, 473.
Kautz, William Hall, 640.
Key, 4, 389.
Key sorting, 74, 338, 373–378.
Key transformation, 506–513, *see* Hashing.
Khizder, Leonid Abramovich, 470.
Kipling, Joseph Rudyard, 74.
Kircher, Athanasius, 23.
Kirchhoff, Gustav Robert, first law, 119,
 127.
Kirkman, Rev. Thomas Penyngton, 565,
 568–569.
Kislitsyn, Sergei Sergeyevich, 211–214, 219,
 635, 637.
Klarner, David Anthony, 620.

- Kleitman, Daniel J., 640.
Klerer, Melvin, 299, 387.
Knock-out tournament, 142, 252–253.
Knott, Gary Don, vi, 512, 522, 672.
Knuth, Donald Ervin, ii, vi, 60, 227–228,
265, 299, 384, 388, 392, 417, 419, 447,
529, 584, 586, 595.
Koch, Gary Grove, 567.
Konheim, Alan Gustave, 265, 376, 497, 691,
692.
Korn, Granino Arthur, 299, 387.
Kronmal, Richard Aaron, 99.
Kronrod, Mikhail Aleksandrovich, 169.
Krutar, Rudolph Allen, vi, 544.
Kwan, Lun Cheung, 630.
KWIC index, 437.
- Ladd, George Trumbull, 392.
Lagrange, Joseph Louis, comte, inversion
formula, 548.
Lamb, Sidney M., 488.
Lambert, Johann Heinrich, series, 617.
Lampson, Butler Wright, 518.
Landauer, Walter Isfried, 473, 566.
Lander, Leon Joseph, 9.
Landis, Evgenii Mikhailevich, 451, 453.
Largest-in-first-out, *see* Priority queue.
Latency time, 362–363, 376, 554.
Latin squares, 566.
Lattice, 20.
Lattice paths, 87–88, 103, 112–113.
Lazarus, Roger Ben, 93.
Leaf, 473.
Least-recently-used page replacement, 159,
478–479.
Lee, Tsai-hwa, 387.
Lefkowitz, David, 566.
Left-to-right maximum or minimum, *see*
Right-to-left.
Leftist trees, 151–152, 159.
Lehman, Alfred Baker, 667.
Level of a tree node: The distance to the
root.
LeVeque, William Judson, 571.
Lexicographic order, 5–6, 170, 173, 417.
Library card sorting, 7–9.
Library tape allocation, 399–400, 404.
Lifo (last-in-first-out) tree, 308–309,
311–312.
Lin, Andrew Damon, 541, 566.
Lin, Shen, 189, 197, 204–208, 220.
Linear algorithm for median, 216–217, 662.
Linear algorithms (average time) for sorting,
102, 178, 197.
Linear list representation, 96, 165, 247, 452,
463–468.
Linear order, 4.
- Linear probing, 518–521, 529–532, 536–539,
545–548.
Linear quotient method, *see* Double hashing.
Ling, Huei, 567.
Linked allocation, 74–75, 96, 99–102,
165–169, 172–176, 390, 396, 402, 485,
541.
Linn, John Charles, 422.
Lissajous, Jules Antoine, 392.
List insertion sort, 95–98, 104–105, 379–381,
428.
List merge sort, 165–169, 184, 380–381, 388.
List sorting, 74–75, 80, 388, 665; *see* List
insertion, List merge, Multiple list
insertion, Radix list sort, Tree insertion
sort.
Littlewood, Dudley Ernest, 593.
Littlewood, John Edensor, 667.
Litwin, S., 566.
Livius, Titus, v.
Lloyd, Stuart Phinney, 392.
Load factor, 517, 535.
Load point, 321, 323.
Logarithmic search, 407, *see* Binary search.
Logarithms, discrete, 9.
Logg, George Edward, 620, 675.
Logical tape unit numbers, 270, 291, 293.
Long runs, 47.
Loop optimization, 85–86, 104 (exercise 33),
137, 169 (exercise 15), 394–396, 415–416,
419, 422, 447, 543.
Lowe, Thomas C., 692.
Lozinski, Leonid Solomonovich, 621.
LSD: Least significant digit, 177.
Luhn, Hans Peter, 437, 540–541.
Łukasiewicz, Jan, 392.
Lum, Vincent Yu-sun, 512, 557, 567.
Lunnon, William Frederick, 640.
Lynch, William Charles, 648, 671.
- Machiavelli, Niccolò di Bernardo, 1.
MacLaren, Malcolm Donald, 177–179, 380,
596.
MacLeod, I. D. G., 595.
MacMahon, Maj. Percy Alexander, 8, 16,
21, 27, 33, 34, 43–46, 62, 64, 586.
Macro assembler, 450, 668.
Magnetic tape, 7–10, 247–250, 266–361,
399–405.
Main loop: Part of a program whose
instructions are performed much more
often than the neighboring parts, 163,
see Loop optimization.
Mallows, Colin Lingwood, 45, 586.
Maniac II, 188.
Mankin, Efrem S., 665.
Mann, Henry Berthold, 601.

- Markov, Andrei Andreyevich, process, 344.
Martin, Thomas Hughes, 477.
Mason, Perry, 1, 2.
Mathsort, 79, *see* Distribution counting.
Match, search for closest, 9, 391, 405,
 500–501, 555.
Matrix transposition, 7.
Mauchly, John William, 83, 350, 352, 385,
 386, 419.
Maximum, finding the, 141, 220.
McAllester, Robert Linné, 282–283.
McAndrew, M. H., 494.
McCall's Cookbook, 8, 556.
McCarthy, John, 8, 128, 169.
McCracken, Daniel Delbert, 387, 419.
McCreight, Edward Meyers, 471, 473, 477,
 479, 480.
McIlroy, Malcolm Douglas, 697.
McKellar, Archie Charles, 123, 378.
McKenna, James, 266.
McNamee, Carole Mattern, 602.
McNutt, Bruce, 555.
Measures of disorder, 11, 105, 134.
Median, 123, 137, 216–220, 662.
Median-of-three quicksort, 123, 136, 139,
 380–381.
Merge exchange sort, 111–114, 135, 224–226,
 380–381, 388.
Merge insertion sort, 185–188, 194–195, 380,
 388.
Merge patterns, 247–250, 266–346, 365–370,
 377; *see* Balanced merge, Cascade
 merge, Oscillating sort, Polyphase
 merge.
 dual to distribution patterns, 349–352.
 for direct-access storage, 365–373, 377.
 optimum, 304–314, 365–373, 377.
 summary, 327–342.
 tree representation of, 305–313, 365–373,
 377.
 vector representation of, 304–305,
 311–314.
Merge sorting, 99, 159–170, 384–386; *see*
 List merge, Natural merge, Straight
 merge.
 external, *see* Merge patterns.
 internal, 99, 159–170, 384, 665.
 k-way, 168, 241–242, 251–253, 262,
 323–327, 343–346.
 minimum comparisons, 198–209.
 networks for, 230–233, 237–240.
Merging priority queues, 151–152, 159, 444.
Middle-square method, 508.
Miles, Ernest Percy, Jr., 286.
Minimization of average cost, 193–197,
 217–220, 410.
Minimum-comparison algorithms, 181–246.
merging, 198–209.
networks, 220–246.
searching, 410, 433–451.
selection, 141, 209–220.
sorting, 181–198.
Minimum memory, 169, 388, 595–596.
Minimum time algorithms, 229–230, 233,
 236, 243, 422.
Minker, Jack, 567.
Misspelled names, 391–392.
MIX computer: Hypothetical machine
 defined in Section 1.3, 75–76, 382, 408.
MIXAL: The MIX assembly language, 423.
MIXT tape units, 320–323, 332, 334, 364.
MIXTEC disks and drums, 362–363,
 553–554.
Möbius, August Ferdinand, function, 33.
Modification, program self-, 85–86, 108.
Monotonic subsequences, 69, 605.
Mooers, Calvin Northrup, 559.
Moore, Edward Forrest, 254, 262, 445, 447.
Morris, Robert, 541, 697, 699.
Morrison, Donald Ross, 490.
Morse, Samuel Finley Breese, code, 601.
Mortenson, John Albert, 651.
Moser, Leo, 67.
Muir, Thomas, 11.
Multihead bubble sort, 244–245, 352.
Multilist system, 552–553, 566.
Multinomial coefficients, 23–24, 33, 494,
 693.
Multiple attribute retrieval, *see* Secondary
 keys.
Multiple list insertion sort, 99–105, 139,
 178, 197, 380–381, 513.
Multiplication of polynomials, 158.
Multiprecision comparison, 6.
Multireel files, 341, 346, 352, 361.
Multiset: Analogous to a set, but elements
 may appear more than once, 22–34,
 43–44, 46, 69, 213–214, 241–242,
 299–300.
Multiway trees, 446, 472–480, *see also* Tries.
Muntz, Richard, 473, 480.
Muroga, Saburo, 688.
Nagler, Harry, 83, 351, 621, 622.
Natural merge sort, 161–163, 168–169.
Natural selection, 259–260, 263–266.
Nelson, Raymond John, 227, 244.
Netto, Eugen, 287.
Networks, for merging, 230–233, 237–240.
 for permutations, 243.
 for selection, 233–235, 239.
 for sorting, 220–245.
 minimum delay, 229–230, 233, 236.
Newcomb, Simon, 43–44, 46.

- Newell, Allen, 518.
Newman, Donald Joseph, 497.
Nielsen, Jakob, 504.
Nievergelt, Jurg, 468, 471.
Nikitin, Andrei Ivanovich, 355.
Nitty gritty, 320–346, 361–371.
Norman, Robert Zane, 600.
Number-crunching computers, 176–177, 374, 380.
- O'Connor, Daniel J., 227.
Octahedron, truncated, 13–14, 19.
Odd-even merge, 224–226.
Odd-even transposition sort, 241.
Odell, Margaret K., 391.
Oderfeld, Jan, 511.
Olson, Charles A., 538.
On-line merge sort, 169.
One-tape sorting, 357–360.
Ones' complement notation, 179.
Open addressing, 517–534, 536–541, 544–549.
Operating systems, 150, 341.
Optimization of programs, *see* Loop optimization.
Optimum exchange sorting, 198.
Optimum open addressing, 532–534, 548–549.
Optimum permutations, 400–401, 403–405.
Optimum polyphase merge, 273–279, 287–288, 340.
Optimum sorting, 181–246.
Optimum trees, for merging, 304–314, 365–373, 376–377.
for searching, 433–451.
Oracles, 200–204, 209, 211–212, 220.
Order relation, 4.
Ordered table, searching in, 396, 406–422.
Orosz, Gábor, 700.
ORR (logical or), 522.
Oscillating sort, 314–320, 331–333, 342, 343, 351, 388.
Overflow, arithmetic, 572.
in *B-tree*, 477–480.
in scatter table, 515, 518, 519, 535, 540.
Own coding, 342–343.
- P*-operator sort (Bose-Nelson method), 227.
Paging, 159, 378, 472, 478–479, 540.
Painter, James Allan, 256.
Palermo, Frank Pantaleone, 688.
Papernov, A. A., 92, 93.
Parallel (simultaneous) computation, 114, 222–224, 229–230, 233, 236, 243, 422.
Parker, Ernest Tilden, 9.
Parkin, Thomas Randall, 9.
Parking problem, 545.
Partial fraction, 196.
Partial ordering, 64–65, 184–185, 189.
- Partition exchange sort, 116, *see* Quicksort.
Partitioning a file, 505.
Partitions of a number, 20–22.
Pass: Part of the execution of an algorithm, traversing all the data once, 272.
Patents, 227, 244, 254, 318–319, 391, 642.
Path length of tree, 194–198, 306–307, 348, 351, 367–368, 371–374, 410, 427, 448, 451, 495–498, 504.
Patricia, 490–493, 497–499, 501–504.
Patt, Yale Nance, 500.
Patterson, George William, 385, 419.
Pentagonal number, 16, 20.
Perfect distribution, 260, *see* Fibonacci distribution.
Perfect shuffle, 237–239.
Permutation network, 243.
Permutations, 11–72, 429–431, 532–534.
cycles of, 25–32, 157, 596, 605, 615, 631.
enumeration of, 23–24, 27–32.
even, 20, 198.
factorization of, 25–32.
fixed points of, 65, 68.
index of, 16–18, 21, 33.
intercalation of, 24–34.
inverse of, 14–15, 19, 32, 76.
inversions of, *see* Inversions.
multiset, 22–34, 43–44, 46, 69.
optimum, 400–401, 403–405.
readings of, 47, 625.
runs of, 34–48, 260–266.
Peter, Laurence Johnston, principle, 144.
Peters, Johann (=Jean) Theodor, 703.
Peterson, James Lyle, 93.
Peterson, William Wesley, 393, 419, 518, 527, 530, 541, 542.
Philco 2000, 256.
Picard, Claude François, 184, 197, 217.
Ping, Czen, 188.
Pipeline computers, 176–177, 374, 380.
PL/I, 342, 525.
Playing cards, 43–44, 46, 73, 170–171, 179–180.
Pocket sorting, 170, *see* Distribution sorting.
Pohl, Ira Sheldon, 219, 220.
Poisson, Siméon Denis, distribution, 548.
Polish prefix notation, 3.
Pólya, György (=George), 583, 667.
Polygon, regular, 290–291.
Polynomial arithmetic, 158, 166, 512–513.
Polynomial hashing, 512–513, 542–543.
Polyphase merge sorting, 266–288, 299, 302–306, 310, 314, 328–330, 336–337, 340, 342, 346, 349, 422.
Caron variation, 280–281, 288.
optimum, 273–279, 287–288, 340.
read-backwards, 302–306, 330, 337, 346.
tape-splitting, 282–286, 288, 329–330, 342.

- Polyphase radix sort, 350, 352.
Post-office tree, 555.
Posting, *see* Insertion.
Power of merge, 644.
Powers, James, 384.
Pr: Probability.
Pratt, Richard D., 312.
Pratt, Vaughan Ronald, vi, 92, 104, 217, 245, 601, 642.
 sorting method, 92, 104, 114, 236.
Prediction, 324–327, 345.
Prefix search, *see* Trie search.
Preorder merge, 309–310, 312, 340.
Prestet, Jean, 24.
Prime numbers, 138, 401–402, 509, 522.
Primitive sorting network, 241.
Pring, Edward John, 555.
Priority deques, 159.
Priority queues, 150–152, 158–159, 252, 444, 620, 667.
Probability distributions, 396–405, 432–451, 504, 525, 532, 548.
Proof of algorithms, 50–52, 112–113, 317, 326, 359–360.
Propagation sort, *see* Bubble sort.
Prywes, Noah Shmarya, 566.
- q*-nomial coefficients, 33.
Quadratic selection, 142.
Queries, 550–570.
Questionnaires, 184.
Queues, 135, 150, 158, 173, 247, 301, 313, 325–326.
Quicksort, 78, 114–123, 125, 128, 135–139, 149, 245, 353–355, 360, 369, 380–381, 388, 428, 665.
- Radix exchange sort, 123–129, 131, 137, 178, 355, 388, 493–494, 501, 665.
Radix insertion sort, 177–178.
Radix list sort, 173–179.
Radix sorting, *see* Distribution sorting.
Radke, Charles E., 299.
Railway switching, 169–170.
Random data for sorting, 21, 48, 75, 382.
Raney, George Neal, 298, 299.
Range queries, 550, 554–555.
Ranking, 182, *see* Sorting.
Raver, N., 688.
Ray Chaudhuri, Dwijendra Kumar, 567.
Read-back check, 365–366.
Read-backwards, 301–320, 323, 330–331, 336–337, 345–346, 354.
Readings of a permutation, 47, 625.
Real-time applications, 540.
Real-valued search tree, 198.
Reciprocals, 417.
Record, 4, 389.
- Recurrence relations, techniques for solving, 120–122, 135–137, 187, 207–208, 274–276, 360, 427, 459, 501, 614–615, 679–680, 685, 694–696.
Recursion, 354.
Recursion induction, 317.
Rehashing, 540.
Reingold, Edward Martin, 209, 468, 471.
Remington Rand Corporation, 384.
Removal, *see* Deletion.
Replacement selection, 40, 251–266, 328–340, 351, 352, 369.
Reversal of data, 312.
Rewinding tape, 279–282, 299, 301, 319, 322–323, 329–330, 333–336, 345–346, 405.
Rice, Stephan Oswald, vi, 138.
Riesel, Hans, 404.
Right-threaded tree, 447, 449.
Right-to-left (or left-to-right) maxima or minima, 12, 27, 82, 101, 139, 141, 157, 602, 646.
Riordan, John, 38, 47, 545, 691, 692, 696.
Rising, Hawley, 128.
Rivest, Ronald Linn, 216, 217, 563–564, 568.
Roberts, David C., 692.
Robinson, Gilbert de Beauregard, 60, 63.
Rochester, Nathaniel, 541.
Roebuck, Alvah, 710.
Rollett, Arthur Percy, 580.
Roselle, David Paul, 47.
Rothe, Heinrich August, 14, 15, 48, 65.
Rouché, Eugène, 647.
Rovner, Paul David, 567.
Royalties, use of, 405.
Runs of a permutation, 34–48, 260–266.
Russell, David Lewis, 93.
Russell, Robert C., 391.
Rutherford, Daniel Edwin, 48.
- Sable, Jerome David, 567.
Sacerdoti, Earl David, 700.
Sackman, Bertram Stanley, 279, 650.
Samplesort, 123, 680.
Samuel, Arthur Lee, 541.
Sandelius, Martin, 630.
Satellite information: Record minus key, 4.
Scatter storage, 506–549.
Schay, Geza, Jr., 399, 530–531, 548, 688.
Schensted, Craige Eugene, 59, 60, 69.
Schlegel, Stanislaus Ferdinand Victor, 269.
Schlumberger, Maurice Lorrain, 372–373, 377.
Schreier, Jozef, 211.
Schützenberger, Marcel Paul, 45, 58–60, 68, 72.
Schwartz, Eugene Sidney, 398, 489.

- Schwartz, Jules, 128.
Scoins, H. Ian, 577.
Scoville, Richard Arthur, 47.
Scrambling function, 510.
Searching, 389–570; *see* External searching, Internal searching; Comparison of keys, Digital searching, Hashing, Secondary key retrieval; Static table searching, Symbol table algorithms.
for closest match, 9, 391, 405, 500–501, 555.
methods, *see* B-trees, Balanced trees, Binary search, Chaining, Digital tree search, Fibonaccian search, Interpolation search, Open addressing, Patricia, Sequential search, Tree search, Trie search.
related to sorting, 2, 391, 406.
text, 504, 561.
Sears, Richard, 710.
Secondary clustering, 522–524, 547.
Secondary key retrieval, 392, 550–570.
Secondary storage, *see* External searching, External sorting.
Seek time, 362, 366–371, 554.
Sefer Yezirah, 23.
Selection of t th largest, 136–137, 141, 209–220, 223, 448, 464.
networks for, 233–235, 239.
Selection sorting, 57, 73, 139–159.
Selection trees, 142–144, 251–253.
Self-modifying program, 85–86, 108.
Self-organizing file, 398–399, 403, 514, 620.
Selfridge, John Lewis, 9.
Senko, Michael Edward, 477.
Sentinel, 5, 160, 252, 311, 386.
Separation sorting, 347, *see* Distribution.
Sequence search, *see* Digital tree search.
Sequential allocation, 96, 165, 173, 485.
Sequential file processing, 3, 7–10, 247.
Sequential searching, 393–406, 420.
Sets, testing equality, 209.
testing inclusion, 391.
Seward, Harold H., 79, 171, 254, 386, 640.
Sexagesimal numbers, 417.
Shackleton, P., 136.
Shannon, Claude Eldwood, 446.
Shapiro, Gerald Norris, 229–230, 243.
Shar, Leonard Eric, 413–414, 420.
Shell, Donald Lewis, 84, 93, 279.
Shellsort, 84–95, 102–105, 111, 149, 379, 381, 388, 639, 665.
Sherman, Philip Martin, 481.
Shockley, William Bradford, 639.
Sholmov, Leonid Ivanovich, 355.
Shrikhande, Sharadchandra Shankar, 700.
Shuffling, 7, 237–239.
Siegel, Shelby, 602.
Sift-up, 146–148, 154–158.
Sifting, 81, *see* Straight insertion.
Signed-magnitude notation, 179.
Silver, Roland Lazarus, 598.
Simulation, 150, 355–356.
Simultaneous comparisons, 114, 222–224, 229–230, 233, 236, 243, 422.
Singer, Theodore, 279.
Single hashing, 549.
Singleton, Richard Collom, 99, 116, 123, 136.
Sinking sort, 81, *see* Straight insertion.
SLB (shift left AX binary), 500.
Sloane, Neil James Alexander, 470.
Słupecki, Jerzy, 211.
Smallest-in-first-out, *see* Priority queue.
Smith, Alan Jay, 170, 662.
Smith, Alfred Emanuel, 389.
Smith, Cyril Stanley, 580.
Smith, Wayne Earl, 401, 404.
Snow job, 254–255, 259–264.
Sobel, Milton, 217, 219, 220, 241.
Sobel, Sheldon, 314, 319.
Software, 386.
Solitaire, 43–44, 46.
Sort generators, 342–343, 386–387.
Sorting (into order), 1–388; *see* External sorting, Internal sorting; Address calculation sorting, Distribution sorting, Enumeration sorting, Exchange sorting, Insertion sorting, Merge sorting, Selection sorting.
address table, 74–75, 80.
key-, 74, 338, 373–378.
library cards, 7–9, 624.
linear average time algorithms for, 102, 178, 197.
list, *see* List sorting.
methods, *see* Binary insertion sort, Bitonic sort, Bubble sort, Cocktail shaker sort, Comparison counting sort, Distribution counting sort, Heapsort, Interval exchange sort, List insertion sort, List merge sort, Median-of-three quicksort, Merge exchange sort, Merge insertion sort, Multiple list insertion sort, Natural merge sort, Odd-even transposition sort, P-operator sort, Pratt sort, Quicksort, Radix exchange sort, Radix insertion sort, Radix list sort, Samplesort, Shellsort, Straight insertion sort, Straight merge sort, Straight selection sort, Tree insertion sort, Tree selection sort, Two-way insertion sort; *see also* Merge patterns.
networks for, 220–245.
one-tape, 357–360.
related to searching, 2, 391, 406.

- stable, 4, 5, 25, 102, 134, 135, 157, 169, 178, 350–351, 388, 594, 622.
topological, 10, 32, 65, 69, 189, 390.
two-tape, 352–361.
- Sós, Vera Turán, 511.
- Soundex, 391–392.
- Speedup, *see* Loop optimization.
- Sperner, Emanuel, 699.
- Speroni, Joseph, 142.
- Splitting a list, 466–470.
- Spruth, Wilhelm Gustav Bernhard, 530–531, 548.
- SRB (shift right AX binary), 126, 135, 408.
- Stable sorting, 4, 5, 25, 102, 134, 135, 157, 169, 178, 350–351, 388, 594, 622.
- Stacks, 115–119, 124–127, 135, 149, 158, 169, 178, 247, 301, 354, 454.
- Staél-Holstein, Anne Louise Germaine Necker, Baronne de, 576.
- Standard sorting network, 236.
- Stanfel, Larry E., 450.
- Stanley, Richard Peter, vi, 584.
- Stasevich, G. V., 92, 93.
- Static table searching, 390, 406–423, 433–451, 473, 481–486, 506–507.
- Stearns, Richard Edwin, 355–356, 360.
- Stegun, Irene Anne, 703.
- Steiner, Jacob, triple systems, 564–565, 568–569.
- Steinhaus, Hugo Dynoizy, 188, 211, 419, 511, 579.
- Step-downs, 161.
- Stirling, James, approximation, 66, 88, 130, 183, 199.
- Stirling numbers, 35, 37–38, 45, 448, 647, 697.
- Stone, Harold Stuart, 237, 239, 422.
- Stop-start time, 322, 333–335, 346.
- Straight insertion sort, 80–84, 96, 102, 106, 110, 117, 127, 141, 149, 165, 223–224, 379, 381, 388, 643.
- Straight merge sort, 163–164, 168–169.
- Straight selection sort, 111, 139–141, 149, 157, 380–381, 386, 388.
- Stratum, 361, *see* Cylinder.
- String: A sequence of items, 22, 27, 274–275, 284–285, 311.
- String: An ordered block, 247, *see* Run.
- Successful search, 389, 394.
- Sue, Jeffrey Yen, 660.
- Summing factor, 121.
- Superimposed coding, 559–563, 568.
- Surnames, encoding, 391–392.
- Sussenguth, Edward Henry, Jr., 486.
- Swierczkowski, Stanisław Sławomir, 511.
- Swift, Jonathan, vi.
- Sylvester, James Joseph, 583.
- Symbol table algorithms, 3, 399, 423–433, 447–505, 513–549.
- Symmetric functions, 240, 591–592.
- Symmetric order: Left subtree, then root, then right subtree, 409, 631.
- T-C algorithm, *see* Hu-Tucker algorithm.
- T-fifo tree, 312–314, 349.
- T-lifo tree, 308–309, 311–312.
- Table, 389.
- Tableau, 48–65, 67–72.
- Tag sorting, *see* Key sorting.
- Tainiter, Melvin, 698.
- Takács, Lajos, 700.
- Tan, Kok-Chye, 451.
- Tangent numbers, 585, 593.
- Tape, *see* Magnetic tape.
- Tape controller, 323.
- Tape searching, 399–401, 405.
- Tape splitting, 281–282.
polyphase merge, 282–286, 288, 299, 329–330, 336, 342.
- Tape units, reliability of, 340.
- Tardiness, 404.
- Tarjan, Robert Endre, 216, 217, 624.
- Tartar, Michael E., 99.
- Tennis tournament, 209–212, 218.
- Terquem, Olry, 578.
- Text searching, 504, 561.
- Thiel, L. H., 567.
- Thrall, Robert McDowell, 63, 71.
- Threaded tree, 447, 449.
- Three-distance theorem, 511, 543.
- Tomlinson, Robert L., Jr., 602.
- Topological sorting, 10, 32, 65, 69, 189, 390.
- Total order, 4.
- Touchard, Jacques, 627.
- Tournament, 142–143, 209–212, 218, 252–253.
- Transitive law, 4–6, 19–20, 209, 449.
- Transmission time, 362–365.
- Transposing a matrix, 7.
- Transposition sorting, *see* Exchange sorting.
- Tree hashing, 546.
- Tree insertion sort, 98, 388, 428, 446–447.
- Tree representation of distribution pattern, 348, 352.
- Tree representation of merge pattern, 305–313, 365–373, 377.
- Tree selection sort, 142–145, 168, 184, 212–214.
- Tree search, 422–433, 447–449, 473, 480, 503–504, 539–540; *see also* Digital tree search.
- Tree traversal, 138, 428.
- Tribolet, Charles Siegfried, 602.
- Trichotomy law, 4–6.
- Trie search, 481–490, 493–494, 499–502, 505.
- Triple systems, 564–565, 568–569.
- Triply-linked binary tree, 159, 467–468.

- Truesdell, Leon Edgar, 383.
Truncated octahedron, 13–14, 19.
Trybuła, Stanisław, 188.
Tucker, Alan Curtiss, 439, 447, 450.
Tumble instruction, 83.
Turán Sós, Vera, 511.
Turing, Alan Mathison, machine, 643.
Turski, Władysław, 507.
Twin primes, 522.
Two's complement notation, 179.
Two-tape sorting, 352–361.
Two-way insertion sort, 83–84.
Typesetting, 562.
- Ullman, Jeffrey David, 532, 533.
Uniform binary search, 411–414, 420.
Uniform hashing, 523, 527–528, 532–534, 548.
Uniform sorting, 245–246.
UNIVAC I, 385, 697.
UNIVAC Larc, 2.
Unsuccessful search, 389, 394.
Updating a file, 3, 168.
Uzgalis, Robert, 473–480.
- van der Pool, Jan Albertus, vi, 697.
van Emden, Maarten Herman, 128, 137, 610, 613.
van Lint, Jacobus Hendricus, 688.
Van Valkenburg, Mac Elwyn, 9.
Van Voorhis, David Curtis, 229, 230, 243.
van Wijngaarden, Adriaan, 9.
Vandermonde, Alexander Théophile, 9, 593.
Varga, Richard Steven, iv.
Variable-length code, 444–445.
Variable-length keys, 266, 477, 549.
Vector representation of merge patterns, 304–305, 311–314.
Venn, J. L., 304.
Virtual memory, 378, 540.
von Mises, Richard, elder, 507.
von Neumann, John, 9, 161, 384.
Vuillemin, Jean Etienne, 372–373, 377.
Vyssotsky, Victor Alexander, 697.
- Waks, David Jeffrey, 343.
Waksman, Abraham, 228, 641.
Walker, Ewing S., 370.
- Walker, Wayne A., 439.
Wallis, John, 24.
Walters, John R., Jr., 256.
Ward, Morgan, 640.
Waters, Samuel Joseph, 370.
Wedekind, Hartmut, 315.
Weighing, 182.
Weight-balanced trees, 468, 471.
Weighted path length, 198, 218.
Weisert, Conrad, 281.
Weiss, Benjamin, 691, 692.
Weiss, Harold, 387.
Weissblum, Walter, 47.
Wells, Mark Brimhall, 188, 192.
Wheeler, David John, 98, 446.
Whirlwind, 386.
Williams, Francis A., Jr., 514.
Williams, John William Joseph, 145–146, 150, 157, 159, 388.
Windley, P. F., 447, 672, 693.
Wong, Chak-Kuen, 259, 451, 468, 471, 646.
Woodall, Arthur David, 168.
Woodrum, Luther Jay, 168, 343.
Worst binary search trees, 450.
Wrench, John William, Jr., vi, 40, 41, 43, 156, 158, 703.
Wright, Edward Maitland, 581.
Wyman, Max, 67.
- Yamada, Hiseo M., 489.
Yoash, Nahal Ben (pseud. of Gideon Yuval), 353.
Youden, William Wallace, 437.
Young, Alfred, 48.
tableaux, 48–65, 67–72.
Yuen, Pasteur Shih Teh, 512.
- Zalk, Martin M., 256.
Zave, Derek Alan, 279, 649.
Zeckendorf, Edouard, 648.
Zero-one principle, 224–225.
Zeta function, 612.
Zipf, George Kingsley, 397.
Zipf's law, 397–399, 432, 449.
Zoberbier, W., 341.
Zolnowsky, John Edward, 581.
- 3-2 trees, 468–469, 471, 475.
80-20 law, 397–398, 403, 449.

Any inaccuracies in this index may be explained by the fact that it has been sorted with the help of a computer. For additional definitions of computer terminology, see Volume 1 and the *IFIP-ICC Vocabulary of Information Processing* (Amsterdam: North-Holland Publ. Co., 1966).

THE ART OF COMPUTER PROGRAMMING

PRE-FASCICLE 2A

A DRAFT OF SECTION 7.2.1.1: GENERATING ALL n -TUPLES

DONALD E. KNUTH *Stanford University*

ADDISON-WESLEY



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

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

Copyright © 2001 by Addison-Wesley

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

Zeroth printing (revision 6), 30 March 2002

PREFACE

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

— PETER J. GOMES (1996)

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

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

To put the material in context, this is Section 7.2.1.1 of a long, long chapter on combinatorial algorithms. Chapter 7 will eventually fill three volumes (namely Volumes 4A, 4B, and 4C), assuming that I'm able to remain healthy. It will begin with a short review of graph theory, with emphasis on some highlights of significant graphs in The Stanford GraphBase (from which I will be drawing many examples). Then comes Section 7.1, which deals with the topic of bitwise manipulations. (I drafted about 60 pages about that subject in 1977, but those pages need extensive revision; meanwhile I've decided to work for awhile on the material that follows it, so that I can get a better feel for how much to cut.) Section 7.2 is about generating all possibilities, and it begins with Section 7.2.1: Generating Basic Combinatorial Patterns. That sets the stage for the main contents of this booklet, Section 7.2.1.1, where I get the ball rolling at last by dealing with the generation of n -tuples. Then will come Section 7.2.1.2 (about permutations), Section 7.2.1.3 (about combinations), etc. Section 7.2.2 will deal with backtracking in general. And so it will go on, if all goes well; an outline of the entire Chapter 7 as currently envisaged appears on the [taocp](#) webpage that is cited on page ii.

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

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

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

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

Happy reading!

*Stanford, California
August 2001 (revised, September 2001)*

D. E. K.

7.2. GENERATING ALL POSSIBILITIES

All present or accounted for, sir.

— Traditional American military saying

All present and correct, sir.

— Traditional British military saying

7.2.1. Generating Basic Combinatorial Patterns

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

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

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

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

He's got 'em on the list—

he's got 'em on the list;

And they'll none of 'em be missed—

they'll none of 'em be missed.

— WILLIAM S. GILBERT, *The Mikado* (1885)

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

Of course such a problem has an absurdly simple solution. All we need to do is start with the binary number $(0\dots00)_2 = 0$ and repeatedly add 1 until we reach $(1\dots11)_2 = 2^n - 1$. We will see, however, that even this utterly trivial problem has astonishing points of interest when we look into it more deeply. And our study of n -tuples will pay off later when we turn to the generation of more difficult kinds of patterns.

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

$$0 \leq a_j < m_j \quad \text{for } 1 \leq j \leq n, \quad (1)$$

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

$$\begin{bmatrix} a_1, & a_2, & \dots, & a_n \\ m_1, & m_2, & \dots, & m_n \end{bmatrix} \quad (2)$$

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

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

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

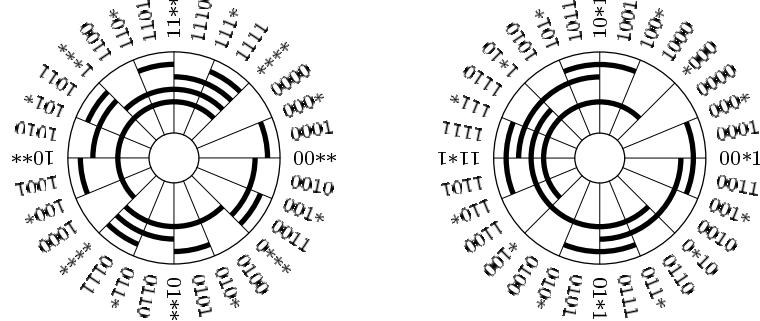
- M1.** [Initialize.] Set $a_j \leftarrow 0$ for $0 \leq j \leq n$, and set $m_0 \leftarrow 2$.
- M2.** [Visit.] Visit the n -tuple (a_1, \dots, a_n) . (The program that wants to examine all n -tuples now does its thing.)
- M3.** [Prepare to add one.] Set $j \leftarrow n$.
- M4.** [Carry if necessary.] If $a_j = m_j - 1$, set $a_j \leftarrow 0$, $j \leftarrow j - 1$, and repeat this step.
- M5.** [Increase, unless done.] If $j = 0$, terminate the algorithm. Otherwise set $a_j \leftarrow a_j + 1$ and go back to step M2. ▀

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

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

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

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



(a) Lexicographic binary code.

(b) Gray binary code.

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

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

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

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

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

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

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

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

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

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

$$a_j = b_j \oplus b_{j+1}, \quad \text{for } j \geq 0. \quad (7)$$

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

$$b_j = a_j \oplus a_{j+1} \oplus a_{j+2} \oplus \dots, \quad \text{for } j \geq 0; \quad (8)$$

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

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

$$g(k) = k \oplus \lfloor k/2 \rfloor. \quad (9)$$

Similarly, the inverse function in (8) satisfies

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

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

$$g(k \oplus k') = g(k) \oplus g(k'). \quad (11)$$

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

$$\Gamma_{n+1} = 0\Gamma_n, 1(\Gamma_n \oplus 10 \dots 0);$$

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

$$\Gamma_n^R = \Gamma_n \oplus \overbrace{10 \dots 0}^{n-1}. \quad (12)$$

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

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

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

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

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

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

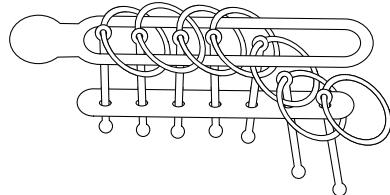


Fig. 11.
The Chinese ring puzzle.

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

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

When the rings are in any state other than 00...0 or 10...0, exactly two moves are possible, one of type (a) and one of type (b). Only one of these moves advances toward the desired goal; the other is a step backward that will need to be undone. A type (a) move changes k to $k \oplus 1$; thus we want to do it when k is odd, since this will decrease k . A type (b) move from a position that ends in $(10^{j-1})_2$ for $1 \leq j < n$ changes k to $k \oplus (1^{j+1})_2 = k \oplus (2^{j+1} - 1)$. When k is

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

even, we want to decrease k by 1, which means that k must be a multiple of 2^j but not a multiple of 2^{j+1} ; in other words,

$$j = \rho(k), \quad (13)$$

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

Going backwards, successively putting rings on or off until we reach the ultimate state $10\dots0$ (which, as John Wallis observed in 1693, is more difficult to reach than the supposedly harder state $11\dots1$), yields an algorithm for counting in Gray binary code:

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

$$a_\infty = a_{n-1} \oplus \dots \oplus a_1 \oplus a_0. \quad (14)$$

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

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

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

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

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

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

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

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

or

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

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

It is impossible, of course, to remove all ambiguity in the lowest-order digit except by a scheme like one the Irish railways are said to have used of removing the last car of every train because it is too susceptible to collision damage.

— G. R. STIBITZ and J. A. LARRIVEE, *Mathematics and Computers* (1957)

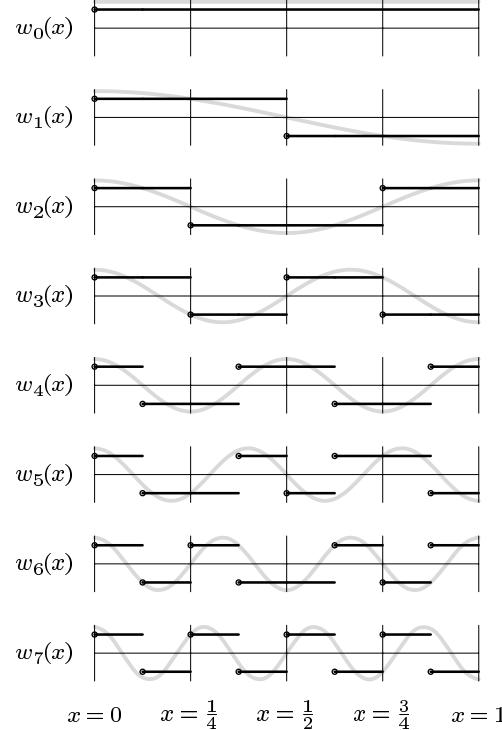


Fig. 12. Walsh functions $w_k(x)$ for $0 \leq k < 8$, with the analogous trigonometric functions $\sqrt{2} \cos k\pi x$ shown in gray for comparison.

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

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

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

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

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

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

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

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

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

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

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

Equation (17) implies the handy formula

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

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

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

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

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

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

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

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

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

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

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

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

- L1.** [Initialize.] Set $a_j \leftarrow 0$ and $f_j \leftarrow j$ for $0 \leq j < n$; also set $f_n \leftarrow n$. (A loopless algorithm is allowed to have loops in its initialization step, as long as the initial setup is reasonably efficient; after all, every program needs to be loaded and launched.)
- L2.** [Visit.] Visit the n -tuple $(a_{n-1}, \dots, a_1, a_0)$.
- L3.** [Choose j .] Set $j \leftarrow f_0$, $f_0 \leftarrow 0$. (If this is the k th time we are performing the present step, j is now equal to $\rho(k)$.) Terminate if $j = n$; otherwise set $f_j \leftarrow f_{j+1}$ and $f_{j+1} \leftarrow j + 1$.
- L4.** [Complement coordinate j .] Set $a_j \leftarrow 1 - a_j$ and return to L2. ▀

For example, the computation proceeds as follows when $n = 4$. Elements a_j have been underlined in this table if the corresponding bit b_j is 1 in the binary string $b_3 b_2 b_1 b_0$ such that $a_3 a_2 a_1 a_0 = g(b_3 b_2 b_1 b_0)$:

a_3	0	0	0	0	0	0	0	0	<u>1</u>							
a_2	0	0	0	0	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	1	1	1	1	0	0	0	0
a_1	0	0	<u>1</u>	<u>1</u>	1	1	<u>0</u>	<u>0</u>	0	0	<u>1</u>	<u>1</u>	1	1	<u>0</u>	<u>0</u>
a_0	0	<u>1</u>	1	<u>0</u>	0	<u>1</u>	1	<u>0</u>	0	<u>1</u>	1	0	0	<u>1</u>	1	0
f_3	3	3	3	3	3	3	3	3	4	4	4	4	3	3	3	3
f_2	2	2	2	2	3	3	2	2	2	2	2	2	4	4	2	2
f_1	1	1	2	1	1	1	3	1	1	1	2	1	1	1	4	1
f_0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4

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

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

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

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

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

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

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

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

- W1.** [Check the difference.] Set $z \leftarrow w \oplus w'$. Reject the word pair (w, w') if $((z - m) \oplus z \oplus m) \wedge m' \neq 0$, where $m = 2^{20} + 2^{15} + 2^{10} + 2^5 + 1$ and $m' = 2^5 m$; this test eliminates cases where w and w' have a common letter in some position. (See 7.1–(oo); it turns out that 10,614,085 of the 16,568,646 word pairs have no such common letters.)
- W2.** [Form individual masks.] Set $m_0 \leftarrow z \wedge (2^5 - 1)$, $m_1 \leftarrow z \wedge (2^{10} - 2^5)$, $m_2 \leftarrow z \wedge (2^{15} - 2^{10})$, $m_3 \leftarrow z \wedge (2^{20} - 2^{15})$, and $m_4 \leftarrow z \wedge (2^{25} - 2^{20})$, in preparation for the next step.
- W3.** [Count words.] Set $l \leftarrow 1$ and $A_0 \leftarrow w$; variable l will count the number of words we have found so far, starting with w . Then perform the operations *swap*(4) defined below.
- W4.** [Print a record-setting solution.] If l exceeds or equals the current maximum, print A_j for $0 \leq j < l$. ■

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

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

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

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

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

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

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

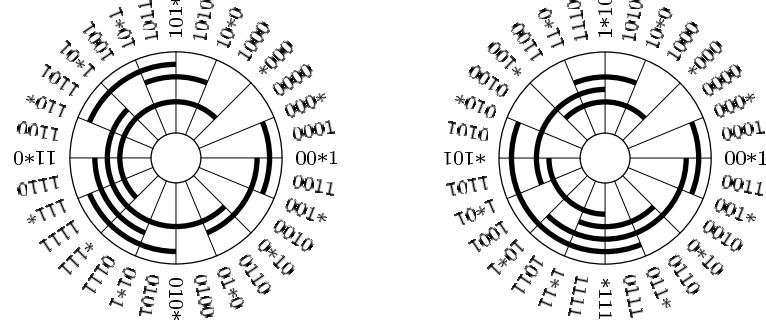
ducks	ducky	duces	dunes	dunks	dinks	dinky	
dines	dices	dicey	dicky	dicks	picks	picky	(23)
pines	piney	pinky	pinks	punks	punky	pucks	

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

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

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

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



(a) Complementary Gray code.

(b) Balanced Gray code.

terminology of graph theory, a Gray code is an oriented Hamiltonian circuit on the n -cube. We can assume that subscripts have been chosen so that $v_0 = 0 \dots 0$.

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

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

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

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

$$00; \quad 0101; \quad 01020102 \quad \text{and} \quad 01210121. \quad (25)$$

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

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

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

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

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

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

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

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

For example, if we start with the sequence 01020102 for $n = 3$ and let the three underlined elements be j_1, j_2, j_3 , the new sequence (27) for a 5-bit code is

$$01410301020131024201043401020103. \quad (28)$$

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

Starting with 000...0 = 00 v_{10} , we proceed to obtain the vertices

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

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

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

then comes 00 v_{30} , etc., and we eventually reach 11 v_{lm_l} . The glorious finale then uses the third line of (27) to generate all the missing vertices 01 $v_{lm_l}, \dots, 01v_{10}$ and take us back to 000...0. ■

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

Corollary B. For all $n \geq 4$, there is an n -bit Gray code with transition counts $(c_0, c_1, \dots, c_{n-1})$ that satisfy the condition

$$|c_j - c_k| \leq 2 \quad \text{for } 0 \leq j < k < n. \quad (29)$$

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

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

Proof. Given a delta sequence for an n -bit Gray code with transition counts (c_0, \dots, c_{n-1}) , the counts for code (27) are obtained by starting with the values $(c'_0, \dots, c'_{n-1}, c'_n, c'_{n+1}) = (4c_0, \dots, 4c_{n-1}, l+1, l+1)$, then subtracting 2 from c'_{j_k} for $1 \leq k < l$ and subtracting 4 from c'_{j_l} . For example, when $n = 3$ we can obtain a balanced 5-bit Gray code having transition counts $(8 - 2, 16 - 10, 8, 6, 6) = (6, 6, 8, 6, 6)$ if we apply Theorem D to the delta sequence 01210121. Exercise 51 works out the details for other values of n . ■

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

$$(0123042103210423)^2. \quad (30)$$

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

***Binary Gray paths.** We have defined an n -bit Gray code as a periodic sequence v_0, v_1, \dots , in which v_k is adjacent to v_{k+1} in the n -cube for all $k \geq 0$, each binary n -tuple occurs in $(v_0, v_1, \dots, v_{2^n-1})$, and $v_{k+2^n} = v_k$. The periodic property is nice, but not always essential; and sometimes we can do better without it. Therefore we say that an n -bit *Gray path* is any ordering $v_0, v_1, \dots, v_{2^n-1}$ of the 2^n possible n -tuples in such a way that v_k is adjacent to v_{k+1} for $0 \leq k < 2^n - 1$. In other words, a Gray code is a Hamiltonian *circuit* on the vertices of the n -cube, but a Gray path is simply a Hamiltonian *path* on that graph.

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

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

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

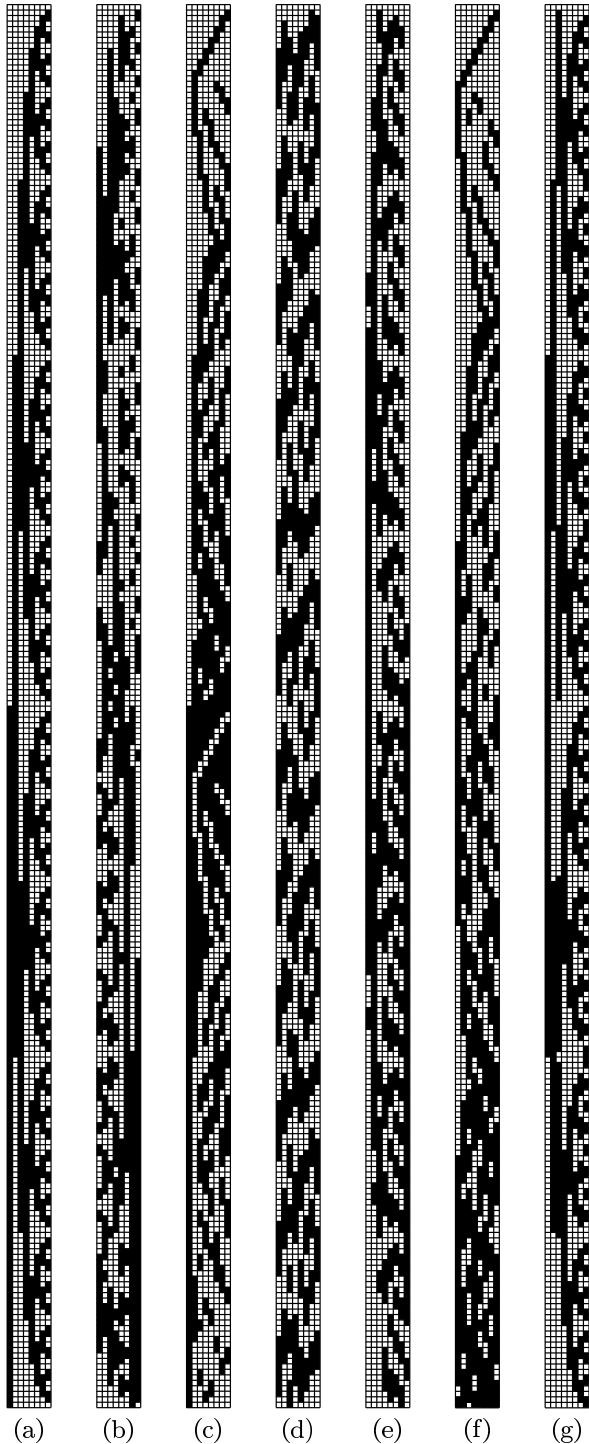


Fig. 14. Examples of 8-bit Gray paths:

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

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

$$000, 001, 011, 010, 110, 100, 101, 111; \quad (32)$$

$$001, 000, 010, 110, 100, 101, 111, 011. \quad (33)$$

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

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

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

Carla Savage and Peter Winkler [*J. Combinatorial Theory A* **70** (1995), 230–248] found an elegant way to construct monotonic binary Gray paths for all $n > 0$. Such paths are necessarily built from subpaths P_{nj} in which all transitions are between n -tuples of weights j and $j + 1$. Savage and Winkler defined suitable subpaths recursively by letting $P_{10} = 0, 1$ and, for all $n > 0$,

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

$$P_{nj} = \emptyset \quad \text{if } j < 0 \text{ or } j \geq n. \quad (35)$$

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

$$P_{20} = 0 P_{10} = 00, 01 \quad (36)$$

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

$$P_{21} = 1 P_{10}^{\pi_1} = 10, 11 \quad (37)$$

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

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

$$\omega_{nj} = 0^{n-j-1}1^{j+1}, \quad \text{for } 0 \leq j < n; \quad (38)$$

$$\alpha_{n0} = 0^n, \quad \text{for } n > 0; \quad (39)$$

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

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

$$P_{n0}, P_{n1}^R, P_{n2}, P_{n3}^R, \dots \quad (41)$$

$$\text{and } P_{n0}^R, P_{n1}, P_{n2}^R, P_{n3}, \dots \quad (42)$$

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

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

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

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

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

$$\pi_n = \sigma_n \pi_{n-1}^2, \quad (45)$$

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

$$\begin{aligned} \pi_1 &= (0), & \pi_4 &= (03), \\ \pi_2 &= (01), & \pi_5 &= (04321), \\ \pi_3 &= (021), & \pi_6 &= (052413); \end{aligned}$$

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

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

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

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

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

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

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

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

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

$$k = \begin{bmatrix} b_{n-1}, \dots, b_1, & b_0 \\ m_{n-1}, \dots, m_1, m_0 \end{bmatrix} = b_{n-1}m_{n-2}\dots m_1m_0 + \dots + b_1m_0 + b_0 \quad (46)$$

into its reflected-Gray equivalent

$$rg(k) = \begin{bmatrix} a_{n-1}, \dots, a_1, & a_0 \\ m_{n-1}, \dots, m_1, m_0 \end{bmatrix} = a_{n-1}m_{n-2}\dots m_1m_0 + \dots + a_1m_0 + a_0, \quad (47)$$

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

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

so that

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

The rule connecting the a 's and b 's is not difficult to derive by induction:

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

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

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

Curiously, rule (50) and its inverse in (51) are exactly the same when all of the radices m_j are odd. In Gray ternary code, for example, when $m_0 = m_1 = \dots = 3$, we have $rg((10010211012)_3) = (12210211010)_3$ and also $rg((12210211010)_3) = (10010211012)_3$. Exercise 78 proves (50) and (51), and discusses similar formulas

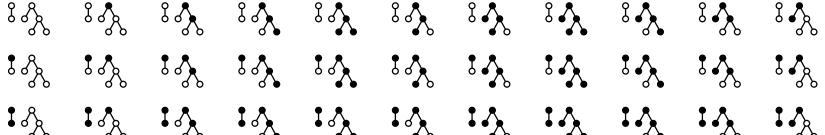
that hold in the modular case. People often call these sequences “Gray codes,” although strictly speaking they might only be Gray paths (see exercise 79).

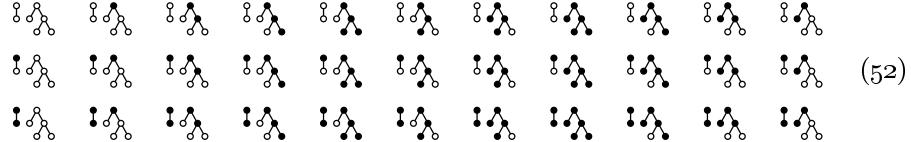
We can in fact generate such Gray sequences looplessly, generalizing Algorithms M and L:

Algorithm H (*Loopless reflected mixed-radix Gray generation*). This algorithm visits all n -tuples (a_{n-1}, \dots, a_0) such that $0 \leq a_j < m_j$ for $0 \leq j < n$, changing only one coordinate by ± 1 at each step. It maintains an array of focus pointers (f_n, \dots, f_0) to control the actions as in Algorithm L, together with an array of directions (d_{n-1}, \dots, d_0) . We assume that each radix m_j is ≥ 2 .

- H1.** [Initialize.] Set $a_j \leftarrow 0$, $f_j \leftarrow j$, and $d_j \leftarrow 1$, for $0 \leq j < n$; also set $f_n \leftarrow n$.
- H2.** [Visit.] Visit the n -tuple $(a_{n-1}, \dots, a_1, a_0)$.
- H3.** [Choose j .] Set $j \leftarrow f_0$ and $f_0 \leftarrow 0$. (As in Algorithm L, j was the rightmost active coordinate; all elements to its right have now been activated.)
- H4.** [Change coordinate j .] Terminate if $j = n$; otherwise set $a_j \leftarrow a_j + d_j$.
- H5.** [Reflect?] If $a_j = 0$ or $a_j = m_j - 1$, set $d_j \leftarrow -d_j$, $f_j \leftarrow f_{j+1}$, and $f_{j+1} \leftarrow j + 1$. (Coordinate j has thus become passive.) Return to H2. ▀

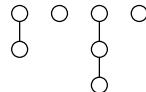
A similar algorithm generates the modular variation (see exercise 77).

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



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

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

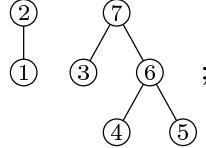


has $3 \times 2 \times 4 \times 2$ principal subforests, corresponding to 4-tuples (x_1, x_2, x_3, x_4) such that $0 \leq x_1 < 3$, $0 \leq x_2 < 2$, $0 \leq x_3 < 4$, and $0 \leq x_4 < 2$; the value of x_j is the number of nodes selected in the j th forest. When the algorithm of Koda

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

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

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



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

- K1.** [Initialize.] Set $a_j \leftarrow 0$ and $f_j \leftarrow j$ for $1 \leq j \leq n$, thereby making the initial subforest empty and all nodes active. Set $f_0 \leftarrow 0$, $l_0 \leftarrow n$, $r_n \leftarrow 0$, $r_0 \leftarrow c_0$, and $l_{c_0} \leftarrow 0$, thereby putting all roots into the current fringe.
- K2.** [Visit.] Visit the subforest defined by (a_1, \dots, a_n) .
- K3.** [Choose p .] Set $q \leftarrow l_0$, $p \leftarrow f_q$. (Now p is the rightmost active node of the fringe.) Also set $f_q \leftarrow q$ (thereby activating all nodes to p 's right).
- K4.** [Check a_p .] Terminate the algorithm if $p = 0$. Otherwise go to K6 if $a_p = 1$.
- K5.** [Insert p 's children.] Set $a_p \leftarrow 1$. Then, if $c_p \neq 0$, set $q \leftarrow r_p$, $l_q \leftarrow p - 1$, $r_{p-1} \leftarrow q$, $r_p \leftarrow c_p$, $l_{c_p} \leftarrow p$ (thereby putting p 's children to the right of p in the fringe). Go to K7.
- K6.** [Delete p 's children.] Set $a_p \leftarrow 0$. Then, if $c_p \neq 0$, set $q \leftarrow r_{p-1}$, $r_p \leftarrow q$, $l_q \leftarrow p$ (thereby removing p 's children from the fringe).
- K7.** [Make p passive.] (At this point we know that p is active.) Set $f_p \leftarrow f_{l_p}$ and $f_{l_p} \leftarrow l_p$. Return to K2. ■

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

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

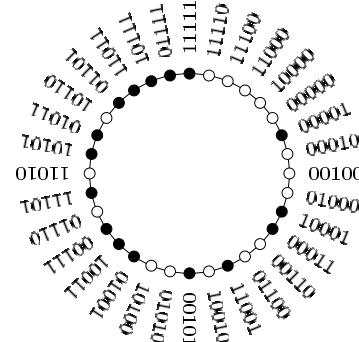


Fig. 15.

A de Bruijn cycle
for 5-bit numbers.

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

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

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

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

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

$$00000100011001010011101011011111; \quad (53)$$

and the first m^2 digits of the infinite sequence

$$0011021220313233041424344\dots \quad (54)$$

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

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

Table 1
PARAMETERS FOR ALGORITHM A

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

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

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

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

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

$$x^n - c_n x^{n-1} - \dots - c_2 x - c_1 \quad (56)$$

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

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

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

- A1.** [Initialize.] Set $(x_0, x_1, \dots, x_{n-1}) \leftarrow (1, 0, \dots, 0)$ and $k \leftarrow 0$, $j \leftarrow s$. In Case 2, also set $i \leftarrow t$ and $h \leftarrow s + t$.
- A2.** [Visit.] Visit the n -tuple $(x_{k-1}, \dots, x_0, x_{n-1}, \dots, x_{k+1}, x_k)$.
- A3.** [Test for end.] If $x_k \neq 0$, set $r \leftarrow 0$; otherwise set $r \leftarrow r + 1$, and go to A6 if $r = n - 1$. (We have just seen r consecutive zeros.)
- A4.** [Shift.] Set $k \leftarrow (k - 1) \bmod n$ and $j \leftarrow (j - 1) \bmod n$. In Case 2 also set $i \leftarrow (i - 1) \bmod n$ and $h \leftarrow (h - 1) \bmod n$.
- A5.** [Compute a new bit.] Set $x_k \leftarrow x_k \oplus x_j$ [Case 1] or $x_k \leftarrow x_k \oplus x_j \oplus x_i \oplus x_h$ [Case 2]. Return to A2.
- A6.** [Finish.] Visit $(0, \dots, 0)$ and terminate. ■

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

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

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

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

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

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

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

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

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

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

$$001102122\ 001102122\ 0\dots, \quad (57)$$

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

$$y = 00102122001110212200102122\ 001\dots$$

$$t = 001001000012340010000100100\ 001\dots$$

$$x = 000110102220120020211122121\ 0001\dots$$

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

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

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

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

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

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

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

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

- D4.** [Invoke g .] Set $y' \leftarrow g()$.
- D5.** [Count repeats.] If $x' \neq y'$, set $x' \leftarrow y'$ and $t' \leftarrow 1$. Otherwise set $t' \leftarrow t' + 1$.
- D6.** [Possibly reject g .] If $t' = n$ and $x' < r$ and either $t < n$ or $x' < x$, go to D4. If $t' = n$ and $x' < r$ and $x' = x$, go to D3.
- D7.** [Output from g .] Output the current value of x' . Return to D3 if $t' = n$ and $x' < r$; otherwise return to D1. ■

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

t in step D2: 12 31211112 12312111 12123121 11121231 21111212 ...
 x in step D3: 00001102122 00011021 22000110 21220001 102122000 ...
 t' in step D6: 121211112121211112121211112121211112121 ...
 x' in step D7: 0 11021220 11021220 11021220 11021220 1 ...;

so the 81-cycle produced in steps D3 and D7 is 00001011012...222200001....

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

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

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

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

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

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

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

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

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

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

$$\sum_{d|n} dL_m(d) = m^n, \quad \text{for all } m, n \geq 1. \quad (59)$$

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

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

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

Definition Q. *A string is preprime if it is a nonempty prefix of a prime.*

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

Proof. See exercise 105. ■

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

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

F1. [Initialize.] Set $a_0 \leftarrow -1$, $a_1 \leftarrow \dots \leftarrow a_n \leftarrow 0$, and $j \leftarrow 1$.

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

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

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

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

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

0000	0011	0022	0111	0122	0212	1111	1212	
0001	0012	0101	0112	0202	0220	1112	1221	
0002	0020	0102	0120	0210	0221	1121	1222	
0010	0021	0110	0121	0211	0222	1122	2222	

(The digits preceding ‘ \wedge ’ are the prime strings 0, 0001, 0002, 001, 0011, ..., 2.)

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

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

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

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

There is a sense in which the outputs of this procedure are actually equivalent to the “granddaddy” of all de Bruijn cycle constructions that work for all m and n , namely the construction first published by M. H. Martin in *Bull. Amer. Math. Soc.* **40** (1934), 859–864: Martin’s original cycle for $m = 3$ and $n = 4$ was 2222122202211...10000, the twos’ complement of (62). In fact, Fredricksen and Maiorana discovered Algorithm F almost by accident while looking for a

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

- i) The average value of $n - j$ in steps F3 and F5 is approximately $m/(m-1)^2$.
- ii) The total running time to produce a de Bruijn cycle like (62) is $O(m^n)$.

EXERCISES

1. [10] Explain how to generate all n -tuples (a_1, \dots, a_n) in which $l_j \leq a_j \leq u_j$, given lower bounds l_j and upper bounds u_j for each coordinate. (Assume that $l_j \leq u_j$.)
2. [15] What is the 1000000th n -tuple visited by Algorithm M if $n = 10$ and $m_j = j$ for $1 \leq j \leq n$? Hint: $\begin{bmatrix} 0, 0, 1, 2, 3, 0, 2, 7, 1, 0 \\ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \end{bmatrix} = 1000000$.
- ▶ 3. [M20] How many times does Algorithm M perform step M4?
- ▶ 4. [18] On most computers it is faster to count down to 0 rather than up to m . Revise Algorithm M so that it visits all n -tuples in the opposite order, starting with $(m_1 - 1, \dots, m_n - 1)$ and finishing with $(0, \dots, 0)$.
- ▶ 5. [20] Algorithms such as the “fast Fourier transform” (exercise 4.6.4–14) often end with an array of answers in bit-reflected order, having $A[(b_0 \dots b_{n-1})_2]$ in the place where $A[(b_{n-1} \dots b_0)_2]$ is desired. What is a good way to rearrange the answers into proper order? [Hint: Reflect Algorithm M.]
6. [M17] Prove (7), the basic formula for Gray binary code.
7. [20] Fig. 10(b) shows the Gray binary code for a disk that is divided into 16 sectors. What would be a good Gray-like code to use if the number of sectors were 12 or 60 (for hours or minutes on a clock), or 360 (for degrees in a circle)?
8. [15] What's an easy way to run through all n -bit strings of even parity, changing only two bits at each step?
9. [16] What move should follow Fig. 11, when solving the Chinese ring puzzle?
- ▶ 10. [M21] Find a simple formula for the total number of steps A_n or B_n in which a ring is (a) removed or (b) replaced, in the shortest procedure for removing n Chinese rings. For example, $A_3 = 4$ and $B_3 = 1$.
11. [M22] (H. J. Purkiss, 1865.) The two smallest rings of the Chinese ring puzzle can actually be taken on or off the bar simultaneously. How many steps does the puzzle require when such accelerated moves are permitted?
- ▶ 12. [23] The *compositions* of n are the sequences of positive integers that sum to n . For example, the compositions of 4 are 1111, 112, 121, 13, 211, 22, 31, and 4. An integer n has exactly 2^{n-1} compositions, corresponding to all subsets of the points $\{1, \dots, n-1\}$ that might be used to break the interval $(0 \dots n)$ into integer-sized subintervals.
 - a) Design a loopless algorithm to generate all compositions of n , representing each composition as a sequential array of integers $s_1 s_2 \dots s_j$.
 - b) Similarly, design a loopless algorithm that represents the compositions implicitly in an array of pointers $q_0 q_1 \dots q_t$, where the elements of the composition are $(q_0 - q_1)(q_1 - q_2) \dots (q_{t-1} - q_t)$ and we have $q_0 = n$, $q_t = 0$. For example, the composition 211 would be represented under this scheme by the pointers $q_0 = 4$, $q_1 = 2$, $q_2 = 1$, $q_3 = 0$, and with $t = 3$.

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

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

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

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

$$\text{LINK}(0:0) = 1:0, \quad \text{LINK}(1:0) = 2:1, \quad \text{LINK}(2:1) = 3:0, \quad \text{LINK}(3:0) = \Lambda,$$

where $0:0$ is a special header node; the other **LINK** fields can have any convenient values.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- a) Explain how to compute $C(\alpha)$, the number of Gray binary clusters of the subcube defined by a given string α of asterisks, 0s, and 1s. What is $C(**10**0*)$?
 b) Prove that $C(\alpha)$ always lies between $2^{t'-1}$ and $2^{t'}$, inclusive.
 c) What is the average value of $C(\alpha)$, over all $2^{n-t} \binom{n}{t}$ possible t -subcubes?

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

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

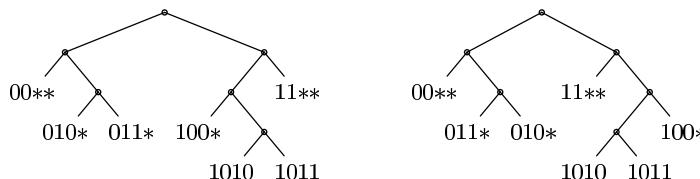


Fig. 16. (a) Normal binary trie.

(b) Gray binary trie.

- 23.** [20] Suppose $g(k) \oplus 2^j = g(l)$. What is a simple way to find l , given j and k ?
- 24.** [M21] Consider extending the Gray binary function g to all 2-adic integers (see exercise 4.1–31). What is the corresponding inverse function $g^{[-1]}$?
- **25.** [M25] Prove that if $g(k)$ and $g(l)$ differ in $t > 0$ bits, and if $0 \leq k, l < 2^n$, then $\lceil 2^t/3 \rceil \leq |k - l| \leq 2^n - \lceil 2^t/3 \rceil$.
- 26.** [25] (Frank Ruskey.) For which integers N is it possible to generate all of the nonnegative integers less than N in such a way that only one bit of the binary representation changes at each step?
- **27.** [20] Let $S_0 = \{1\}$ and $S_{n+1} = 1/(2 + S_n) \cup 1/(2 - S_n)$; thus, for example,

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

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

- 28.** [M26] A *median* of n -bit strings $\{\alpha_1, \dots, \alpha_t\}$, where α_k has the binary representation $\alpha_k = a_{k(n-1)} \dots a_{k0}$, is a string $\hat{\alpha} = a_{n-1} \dots a_0$ whose bits a_j for $0 \leq j < n$ agree with the majority of the bits a_{kj} for $1 \leq k \leq t$. (If t is even and the bits α_{kj} are half 0 and half 1, the median bit a_j can be either 0 or 1.) For example, the strings $\{0010, 0100, 0101, 1110\}$ have two medians, 0100 and 0110, which we can denote by 01*0.

- a) Find a simple way to describe the medians of $G_t = \{g(0), \dots, g(t-1)\}$, the first t Gray binary strings, when $0 < t \leq 2^n$.
- b) Prove that if $\alpha = a_{n-1} \dots a_0$ is such a median, and if $\alpha' = a'_{n-1} \dots a'_0$ is any element of G_t with $a'_j \neq a_j$, then the string β obtained from α by changing a_j to a'_j is also an element of G_t .
- 29.** [M24] If integer values k are transmitted as n -bit Gray binary codes $g(k)$ and received with errors described by a bit pattern $p = (p_{n-1} \dots p_0)_2$, the average numerical error is

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

leads to the sum-of-four-squares identity

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

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

b) Generalize to H_4 and higher-order matrices.

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

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

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

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

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

44. [M35] How many of the delta sequences for n -bit Gray codes have the property that exactly (a) one or (b) two of the coordinate names occur only twice? Express your answers in terms of $d(n - 1)$ and $d(n - 2)$.

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

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

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

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

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

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

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

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

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

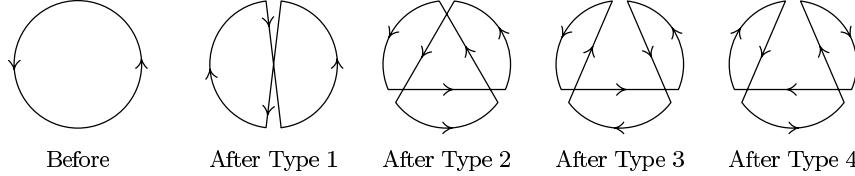
54. [M20] (H. S. Shapiro, 1953.) Show that if a sequence of integers (a_1, \dots, a_{2^n}) contains only n distinct values, then there is a subsequence whose product $a_{k+1}a_{k+2} \dots a_l$ is a perfect square, for some $0 \leq k < l \leq 2^n$. However, this conclusion might not be true if we disallow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

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

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

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

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



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

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

- **58.** [21] Let α be the delta sequence of an n -bit Gray code, and obtain β from α by changing q occurrences of 0 to n , where q is odd. Prove that $\beta\beta$ is the delta sequence of an $(n+1)$ -bit Gray code.
- 59.** [22] The 5-bit Gray code of (30) is *nonlocal* in the sense that no 2^t consecutive elements belong to a single t -subcube, for $1 < t < n$. Prove that nonlocal n -bit Gray codes exist for all $n \geq 5$. [Hint: See the previous exercise.]
- 60.** [20] Show that the run-length-bound function satisfies $r(n+1) \geq r(n)$.
- 61.** [M30] Show that $r(m+n) \geq r(m) + r(n) - 1$ if (a) $m = 2$ and $2 < r(n) < 8$; or (b) $m \leq n$ and $r(n) \leq 2^{m-3}$.
- 62.** [46] Does $r(8) = 6$?
- 63.** [30] (Luis Goddyn.) Prove that $r(10) \geq 8$.
- **64.** [HM35] (L. Goddyn and P. Gvozdjak.) An n -bit *Gray stream* is a sequence of permutations $(\sigma_0, \sigma_1, \dots, \sigma_{l-1})$ where each σ_k is a permutation of the vertices of the n -cube, taking every vertex to one of its neighbors.
 - a) Suppose (u_0, \dots, u_{2^m-1}) is an m -bit Gray code and $(\sigma_0, \sigma_1, \dots, \sigma_{2^m-1})$ is an n -bit Gray stream. Let $v_0 = 0\dots 0$ and $v_{k+1} = v_k \sigma_k$, where $\sigma_k = \sigma_k \bmod 2^m$ if $k \geq 2^m$. Under what conditions is the sequence

$$W = (u_0v_0, u_0v_1, u_1v_1, u_1v_2, \dots, u_{2^{m+n-1}-1}v_{2^{m+n-1}-1}, u_{2^{m+n-1}-1}v_{2^{m+n-1}})$$
 an $(m+n)$ -bit Gray code?
 - b) Show that if m is sufficiently large, there is an n -bit Gray stream satisfying the conditions of (a) for which all run lengths of the sequence (v_0, v_1, \dots) are $\geq n-2$.
 - c) Apply these results to prove that $r(n) \geq n - O(\log n)$.
- 65.** [30] (Brett Stevens.) In Samuel Beckett's play *Quad*, the stage begins and ends empty; n actors enter and exit one at a time, running through all 2^n possible subsets, and the actor who leaves is always the one whose previous entrance was earliest. When $n = 4$, as in the actual play, some subsets are necessarily repeated. Show, however, that there is a perfect pattern with exactly 2^n entrances and exits when $n = 5$.

- 66.** [40] Is there a perfect Beckett–Gray pattern for 8 actors?
- 67.** [20] Sometimes it is desirable to run through all n -bit binary strings by changing as many bits as possible from one step to the next, for example when testing a physical circuit for reliable behavior in worst-case conditions. Explain how to traverse all binary n -tuples in such a way that each step changes n or $n - 1$ bits, alternately.
- 68.** [21] Peter Q. Perverse decided to construct an *anti-Gray* ternary code, in which each n -trit number differs from its neighbors in *every* digit position. Is such a code possible for all n ?
- **69.** [M25] Modify the definition of Gray binary code (7) by letting
- $$h(k) = (\dots(b_6 \oplus b_5)(b_5 \oplus b_4)(b_4 \oplus b_3 \oplus b_2 \oplus b_0)(b_3 \oplus b_0)(b_2 \oplus b_1 \oplus b_0)b_1)_2,$$
- when $k = (\dots b_5 b_4 b_3 b_2 b_1 b_0)_2$.
- Show that the sequence $h(0), h(1), \dots, h(2^n - 1)$ runs through all n -bit numbers in such a way that exactly 3 bits change each time, when $n > 3$.
 - Generalize this rule to obtain sequences in which exactly t bits change at each step, when t is odd and $n > t$.
- 70.** [21] How many monotonic n -bit Gray paths exist for $n = 5$ and $n = 6$?
- 71.** [M22] Derive (45), the recurrence that defines the Savage–Winkler permutations.
- 72.** [20] What is the Savage–Winkler path from 00000 to 11111?
- **73.** [32] Design an efficient algorithm to construct the delta sequence of an n -bit monotonic Gray path.
- 74.** [M25] (Savage and Winkler.) How far apart can adjacent vertices of the n -cube be, in a monotonic Gray path?
- 75.** [32] Find all 5-bit Gray paths v_0, \dots, v_{31} that are *trend-free*, in the sense that $\sum_{k=0}^{31} k(-1)^{v_{kj}} = 0$ in each coordinate position j .
- 76.** [M25] Prove that trend-free n -bit Gray paths exist for all $n \geq 5$.
- 77.** [21] Modify Algorithm H in order to visit mixed-radix n -tuples in *modular* Gray order.
- 78.** [M26] Prove the conversion formulas (50) and (51) for reflected mixed-radix Gray paths, and derive analogous formulas for the modular case.
- **79.** [M22] When is the last n -tuple of the (a) reflected (b) modular mixed-radix Gray path adjacent to the first?
- 80.** [M20] Explain how to run through all divisors of a number, given its prime factorization $p_1^{e_1} \dots p_t^{e_t}$, repeatedly multiplying or dividing by a single prime at each step.
- 81.** [M21] Let $(a_0, b_0), (a_1, b_1), \dots, (a_{m^2-1}, b_{m^2-1})$ be the 2-digit m -ary modular Gray code. Show that, if $m > 2$, every edge $(x, y) — (x, (y + 1) \bmod m)$ and $(x, y) — ((x + 1) \bmod m, y)$ occurs in one of the two cycles
- $$(a_0, b_0) — (a_1, b_1) — \dots — (a_{m^2-1}, b_{m^2-1}) — (a_0, b_0),$$
- $$(b_0, a_0) — (b_1, a_1) — \dots — (b_{m^2-1}, a_{m^2-1}) — (b_0, a_0).$$
- **82.** [M25] (G. Ringel, 1956.) Use the previous exercise to deduce that there exist four 8-bit Gray codes that, together, cover all edges of the 8-cube.
- 83.** [M46] Can four *balanced* 8-bit Gray codes cover all edges of the 8-cube?

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

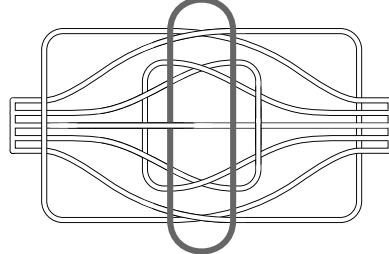


Fig. 17. The Loony Loop puzzle.

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

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

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

- 86. [26] Define an infinite Gray path that runs through all possible nonnegative integer n -tuples (a_1, \dots, a_n) in such a way that $\max(a_1, \dots, a_n) \leq \max(a'_1, \dots, a'_n)$ when (a_1, \dots, a_n) is followed by (a'_1, \dots, a'_n) .
- 87. [27] Continuing the previous exercise, define an infinite Gray path that runs through *all* integer n -tuples (a_1, \dots, a_n) , in such a way that $\max(|a_1|, \dots, |a_n|) \leq \max(|a'_1|, \dots, |a'_n|)$ when (a_1, \dots, a_n) is followed by (a'_1, \dots, a'_n) .
- 88. [25] After Algorithm K has terminated in step K4, what would happen if we immediately restarted it in step K2?
- 89. [25] (*Gray code for Morse code.*) The Morse code words of length n (exercise 4.5.3–32) are strings of dots and dashes, where n is the number of dots plus twice the number of dashes.
- Show that it is possible to generate all Morse code words of length n by successively changing a dash to two dots or vice versa. For example, the path for $n = 3$ must be $\bullet\bullet$, $\bullet\bullet\bullet$, $\bullet\bullet\bullet\bullet$ or its reverse.
 - What string follows $\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet$ in your sequence for $n = 15$?
90. [26] For what values of n can the Morse code words be arranged in a *cycle*, under the ground rules of exercise 89? [Hint: The number of code words is F_{n+1} .]
- 91. [34] Design a loopless algorithm to visit all binary n -tuples (a_1, \dots, a_n) such that $a_1 \leq a_2 \geq a_3 \leq a_4 \geq \dots$. [The number of such n -tuples is F_{n+2} .]
92. [M30] Is there an infinite sequence Φ_n whose first m^n elements form an m -ary de Bruijn cycle, for all m ? [The case $n = 2$ is solved in (54).]
- 93. [M28] Prove that Algorithm R outputs a de Bruijn cycle as advertised.
94. [22] What is the output of Algorithm D when $m = 5$, $n = 1$, $r = 3$, and both $f()$ and $g()$ are the trivial cycles 01234 01234 01...?

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

$$c_0c_1c_2c_3c_4c_5\dots = a_0b_0a_1b_1a_2b_2\dots$$

- a) Under what circumstances does $c_0c_1c_2\dots$ have period pq ? (The “period” of a sequence $a_0a_1a_2\dots$, for the purposes of this exercise, is the smallest integer $p > 0$ such that $a_k = a_{k+p}$ for all $k \geq 0$.)
 - b) Which $2n$ -tuples would occur as consecutive outputs of Algorithm D if step D6 were changed to say simply “If $t' = n$ and $x' < r$, go to D4”?
 - c) Prove that Algorithm D outputs a de Bruijn cycle as advertised.
- 96. [M23] Suppose a family of coroutines has been set up to generate a de Bruijn cycle of length m^n using Algorithms R and D, based recursively on simple routines for the base case $n = 2$.
- a) How many coroutines of each type will there be?
 - b) What is the maximum number of coroutine activations needed to get one top-level digit of output?

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

- a) Write out the cycles $(f_n(0)\dots f_n(2^n-1))$ for $n = 2, 3, 4$, and 5 .
- b) Prove that, for all even values of n , there is a number $\delta_n = \pm 1$ such that we have

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

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

- c) Let $(c_n(0)c_n(1)\dots c_n(2^{2n}-5))$ be the cycle produced when the simplified version of Algorithm D in exercise 95(b) is applied to $f_n()$. Where do the $(2n-1)$ -tuples 1^{2n-1} and $(01)^{n-1}0$ occur in this cycle?
- d) Use the results of (c) to express $f_{2n}(k)$ in terms of $f_n()$.
- e) Find a (somewhat) simple formula for j_n as a function of n .

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

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

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

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

- a) Prove that if λ and λ' are prime, then $\lambda\lambda'$ is prime if $\lambda < \lambda'$.
- b) Consequently every string α can be written in the form

$$\alpha = \lambda_1\lambda_2\dots\lambda_t, \quad \lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_t, \quad \text{where each } \lambda_j \text{ is prime.}$$

- c) In fact, only one such factorization is possible. Hint: Show that λ_t must be the lexicographically smallest nonempty suffix of α .
- d) True or false: λ_1 is the longest prime prefix of α .
- e) What are the prime factors of $3141592653589793238462643383279502884197$?

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

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

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

105. [M31] Let α be a preprime of length n .

- a) Show that if the final letter of α is increased, the resulting string is prime.
- b) If α has been factored as in exercise 101, show that it is the n -extension of λ_1 .
- c) Furthermore α cannot be the n -extension of two different primes.

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

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

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

- a) Show that if $a_1 \dots a_n = \alpha\beta$ is distinct from all its cyclic shifts, and if $\beta\alpha = \lambda_k$ is prime, then $\alpha\beta$ is a substring of $\lambda_k\lambda_{k+1}$, unless $\alpha = 9^j$ for some $j \geq 1$.
- b) Where does $\alpha\beta$ appear in $\lambda_1 \dots \lambda_t$ if $\beta\alpha$ is prime and α consists of all 9s? Hint: Show that if $a_{n+1-j} \dots a_n = 9^l$ in step F2 for some $l > 0$, and if j is not a divisor of n , the previous step F2 had $a_{n-l} \dots a_n = 9^{l+1}$.
- c) Now consider n -tuples of the form $(\alpha\beta)^d$, where $d > 1$ is a divisor of n and $\beta\alpha = \lambda_k$ is prime.
- d) Identify the positions of 899135, 997879, 913131, 090909, 909090, and 911911 when $n = 6$.

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

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

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

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

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

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

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

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

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

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

- a) What is the smallest positive integer that cannot be represented in such a way?
- b) Consider also inserting signs into the 10-digit sequence 9876543210.

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

ANSWERS TO EXERCISES

*All that heard him were astonished
at his understanding and answers.*

— Luke 2:47

SECTION 7.2.1.1

1. Let $m_j = u_j - l_j + 1$, and visit $(a_1 + l_1, \dots, a_n + l_n)$ instead of visiting (a_1, \dots, a_n) in Algorithm M. Or, change ' $a_j \leftarrow 0$ ' to ' $a_j \leftarrow l_j$ ' and ' $a_j = m_j - 1$ ' to ' $a_j = u_j$ ' in that algorithm, and set $l_0 \leftarrow 0$, $u_0 \leftarrow 1$ in step M1.
2. $(0, 0, 1, 2, 3, 0, 2, 7, 0, 9)$.
3. Step M4 is performed $m_1 m_2 \dots m_k$ times when $j = k$; therefore the total is $\sum_{k=0}^n \prod_{j=1}^k m_j = m_1 \dots m_n (1 + 1/m_n + 1/m_n m_{n-1} + \dots + 1/m_n \dots m_1)$. If all m_j are 2 or more, this is less than $2m_1 \dots m_n$. [Thus, we should keep in mind that fancy Gray-code methods, which change only one digit per visit, actually reduce the total number of digit changes by at most a factor of 2.]
4. **N1.** [Initialize.] Set $a_j \leftarrow m_j - 1$ for $0 \leq j \leq n$, where $m_0 = 2$.
N2. [Visit.] Visit the n -tuple (a_1, \dots, a_n) .
N3. [Prepare to subtract one.] Set $j \leftarrow n$.
N4. [Borrow if necessary.] If $a_j = 0$, set $a_j \leftarrow m_j - 1$, $j \leftarrow j - 1$, and repeat this step.
N5. [Decrease, unless done.] If $j = 0$, terminate the algorithm. Otherwise set $a_j \leftarrow a_j - 1$ and go back to step N2. ▀
5. Bit reflection is easy on a machine like MMIX, but on other computers we can proceed as follows:
 - R1.** [Initialize.] Set $j \leftarrow k \leftarrow 0$.
 - R2.** [Swap.] Interchange $A[j+1] \leftrightarrow A[k+2^{n-1}]$. Also, if $j > k$, interchange $A[j] \leftrightarrow A[k]$ and $A[j+2^{n-1}+1] \leftrightarrow A[k+2^{n-1}+1]$.
 - R3.** [Advance k .] Set $k \leftarrow k + 2$, and terminate if $k \geq 2^{n-1}$.
 - R4.** [Advance j .] Set $h \leftarrow 2^{n-2}$. If $j \geq h$, repeatedly set $j \leftarrow j - h$ and $h \leftarrow h/2$ until $j < h$. Then set $j \leftarrow j + h$. (Now $j = (b_0 \dots b_{n-1})_2$ if $k = (b_{n-1} \dots b_0)_2$.) Return to R2. ▀
6. If $g((0b_{n-1} \dots b_1 b_0)_2) = (0(b_{n-1}) \dots (b_2 \oplus b_1)(b_1 \oplus b_0))_2$ then $g((1b_{n-1} \dots b_1 b_0)_2) = 2^n + g((\bar{b}_{n-1} \dots \bar{b}_1 \bar{b}_0)_2) = (1(\bar{b}_{n-1}) \dots (\bar{b}_2 \oplus \bar{b}_1)(\bar{b}_1 \oplus \bar{b}_0))_2$, where $\bar{b} = b \oplus 1$.

7. To accommodate $2r$ sectors one can use $g(k)$ for $2^n - r \leq k < 2^n + r$, where $n = \lceil \lg r \rceil$, because $g(2^n - r) \oplus g(2^n + r - 1) = 2^n$ by (5). [G. C. Tootill, *Proc. IEE* **103**, Part B Supplement (1956), 434.] See also exercise 26.

8. Use Algorithm G with $n \leftarrow n - 1$ and include the parity bit a_∞ at the right. (This yields $g(0), g(2), g(4), \dots$)

9. Replace the rightmost ring, since $\nu(1011000)$ is odd.

10. $A_n + B_n = g^{[-1]}(2^n - 1) = \lfloor 2^{n+1}/3 \rfloor$ and $A_n = B_n + n$. Hence $A_n = \lfloor 2^n/3 + n/2 \rfloor$ and $B_n = \lfloor 2^n/3 - n/2 \rfloor$.

Historical notes: The early Japanese mathematician Yoriyuki Arima (1714–1783) treated this problem in his *Shūki Sanpō* (1769), Problem 44, observing that the n -ring puzzle reduces to an $(n - 1)$ -ring puzzle after a certain number of steps. Let $C_n = A_n - A_{n-1} = B_n - B_{n-1} + 1$ be the number of rings removed during this reduction. Arima noticed that $C_n = 2C_{n-1} - [n \text{ even}]$; thus he could compute $A_n = C_1 + C_2 + \dots + C_n$ for $n = 9$ without actually knowing the formula $C_n = \lceil 2^{n-1}/3 \rceil$.

More than two centuries earlier, Cardano had already mentioned the “complicati annuli” in his *De Subtilitate Libri XXI* (Nuremberg: 1550), Book 15. He wrote that they are “useless yet admirably subtle,” stating erroneously that 95 moves are needed to remove seven rings and 95 more to put them back. John Wallis devoted seven pages to this puzzle in the Latin edition of his *Algebra 2* (Oxford: 1693), Chapter 111, presenting detailed but nonoptimum methods for the nine-ring case. He included the operation of sliding a ring through the bar as well as putting it on or off, and he hinted that shortcuts were available, but he did not attempt to find a shortest solution.

11. The solution to $S_n = S_{n-2} + 1 + S_{n-2} + S_{n-1}$ when $S_1 = S_2 = 1$ is $S_n = 2^{n-1} - [n \text{ even}]$. [*Math. Quest. Educational Times* **3** (1865), 66–67.]

12. (a) The theory of $n - 1$ Chinese rings proves that Gray binary code yields the compositions in a convenient order (4, 31, 211, 22, 112, 1111, 121, 13):

A1. [Initialize.] Set $t \leftarrow 0, j \leftarrow 1, s_1 \leftarrow n$. (We assume that $n > 1$.)

A2. [Visit.] Visit $s_1 \dots s_j$. Then set $t \leftarrow 1 - t$, and go to A4 if $t = 0$.

A3. [Odd step.] If $s_j > 1$, set $s_j \leftarrow s_j - 1, s_{j+1} \leftarrow 1, j \leftarrow j + 1$; otherwise set $j \leftarrow j - 1$ and $s_j \leftarrow s_j + 1$. Return to A2.

A4. [Even step.] If $s_{j-1} > 1$, set $s_{j-1} \leftarrow s_{j-1} - 1, s_{j+1} \leftarrow s_j, s_j \leftarrow 1, j \leftarrow j + 1$; otherwise set $j \leftarrow j - 1, s_j \leftarrow s_{j+1}, s_{j-1} \leftarrow s_{j-1} + 1$ (but terminate if $j - 1 = 0$). Return to A2. ■

(b) Now q_1, \dots, q_{t-1} represent rings on the bar:

B1. [Initialize.] Set $t \leftarrow 1, q_0 \leftarrow n$. (We assume that $n > 1$.)

B2. [Visit.] Set $q_t \leftarrow 0$ and visit $(q_0 - q_1) \dots (q_{t-1} - q_t)$. Go to B4 if t is even.

B3. [Odd step.] If $q_{t-1} = 1$, set $t \leftarrow t - 1$; otherwise set $q_t \leftarrow 1$ and $t \leftarrow t + 1$. Return to step B2.

B4. [Even step.] If $q_{t-2} = q_{t-1} + 1$, set $q_{t-2} \leftarrow q_{t-1}$ and $t \leftarrow t - 1$ (but terminate if $t = 2$); otherwise set $q_t \leftarrow q_{t-1}, q_{t-1} \leftarrow q_t + 1, t \leftarrow t + 1$. Return to B2. ■

These algorithms [see J. Misra, *ACM Trans. Math. Software* **1** (1975), 285] are loopless even in their initialization steps.

13. In step A1, also set $C \leftarrow 1$. In step A3, set $C \leftarrow s_j C$ if $s_j > 1$, otherwise $C \leftarrow C/(s_{j-1} + 1)$. In step A4, set $C \leftarrow s_{j-1} C$ if $s_{j-1} > 1$, otherwise $C \leftarrow C/(s_{j-2} + 1)$.

Similar modifications apply to steps B1, B3, B4. Sufficient precision is needed to accommodate the value $C = n!$ for the composition $1\dots 1$; we are stretching the definition of looplessness by assuming that arithmetic operations take unit time.

14. S1. [Initialize.] Set $j \leftarrow 0$.

S2. [Visit.] Visit the string $a_1 \dots a_j$.

S3. [Lengthen.] If $j < n$, set $j \leftarrow j + 1$, $a_j \leftarrow 0$, and return to S2.

S4. [Increase.] If $a_j < m_j - 1$, set $a_j \leftarrow a_j + 1$ and return to S2.

S5. [Shorten.] Set $j \leftarrow j - 1$, and return to S4 if $j > 0$. ▀

15. T1. [Initialize.] Set $j \leftarrow 0$.

T2. [Even visit.] If j is even, visit the string $a_1 \dots a_j$.

T3. [Lengthen.] If $j < n$, set $j \leftarrow j + 1$, $a_j \leftarrow 0$, and return to T2.

T4. [Odd visit.] If j is odd, visit the string $a_1 \dots a_j$.

T5. [Increase.] If $a_j < m_j - 1$, set $a_j \leftarrow a_j + 1$ and return to T2.

T6. [Shorten.] Set $j \leftarrow j - 1$, and return to T4 if $j > 0$. ▀

This algorithm is loopless, although it may appear at first glance to contain loops, because four steps separate consecutive visits. The basic idea is related to exercise 2.3.1–5 and to “prepostorder” traversal (exercise 7.2.1.3–00).

16. Suppose $\text{LINK}(j-1:b) = j:(b \oplus a_j)$ for $1 \leq j \leq n$, and $\text{LINK}(n:b) = \Lambda$. These links represent (b_1, \dots, b_n) if and only if $g(b_1 \dots b_n) = a_1 \dots a_n$, so we can use a loopless Gray binary generator to achieve the desired result.

17. Put the concatenation of 3-bit codes $(g(j), g(k))$ in row j and column k , for $0 \leq j, k < 8$. [It is not difficult to prove that this is essentially the *only* solution, except for permuting and/or complementing coordinates and/or rotating rows, because the coordinate that changes when moving north or south depends only on the row, and a similar statement applies to columns. Karnaugh’s isomorphism between the 4-cube and the 4×4 torus can be traced back to *The Design of Switching Circuits* by W. Keister, A. E. Ritchie, and S. H. Washburn (1951), page 174. Incidentally, Keister went on to design an ingenious variant of Chinese rings called SpinOut, and a generalization called The Hexadecimal Puzzle, U.S. Patents 3637215–3637216 (1972).]

18. Use 2-bit Gray code to represent the digits $u_j = (0, 1, 2, 3)$ respectively as the bit pairs $u'_{2j-1}u'_{2j} = (00, 01, 11, 10)$. [C. Y. Lee introduced his metric in *IEEE Trans. IT-4* (1958), 77–82. A similar $m/2$ -bit encoding works for even values of m ; for example, when $m = 8$ we can represent $(0, 1, 2, 3, 4, 5, 6, 7)$ by $(0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000)$. But such a scheme leaves out some of the binary patterns when $m > 4$.]

19. (a) A modular Gray quaternary algorithm needs slightly less computation than Algorithm M, but it doesn’t matter because 256 is so small. The result is $z_0^8 + z_1^8 + z_2^8 + z_3^8 + 14(z_0^4z_2^4 + z_1^4z_3^4) + 56z_0z_1z_2z_3(z_0^2 + z_2^2)(z_1^2 + z_3^2)$.

(b) Replacing (z_0, z_1, z_2, z_3) by $(1, z, z^2, z)$ gives $1 + 112z^6 + 30z^8 + 112z^{10} + z^{16}$; thus all of the nonzero Lee weights are ≥ 6 . Now use the construction in the previous exercise to convert each $(u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_\infty)$ into a 16-bit number.

20. Recover the quaternary vector $(u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_\infty)$ from u' , and use Algorithm 4.6.1D to find the remainder of $u_0 + u_1x + \dots + u_6x^6$ divided by $g(x)$, mod 4; that algorithm can be used in spite of the fact that the coefficients do not belong to a field, because $g(x)$ is monic. Express the remainder as $x^j + 2x^k$ (modulo $g(x)$ and 4), and let $d = (k - j) \bmod 4$, $s = (u_0 + \dots + u_6 + u_\infty) \bmod 4$.

Case 1, $s = 1$: If $k = \infty$, the error was x^j (in other words, the correct vector has $u_j \leftarrow (u_j - 1) \bmod 4$); otherwise there were three or more errors.

Case 2, $s = 3$: If $j = k$ the error was x^j ; otherwise ≥ 3 errors occurred.

Case 3, $s = 0$: If $j = k = \infty$, no errors were made; if $j = \infty$ and $k < \infty$, at least four errors were made. Otherwise the errors were $x^a - x^b$, where $a = (j + (\infty, 3, 6, 1, 5, 4, 2, 0)) \bmod 7$ according as $d = (0, 1, 2, 3, 4, 5, 6, \infty)$, and $b = (j+2d) \bmod 7$.

Case 4, $s = 2$: If $j = \infty$ the errors were $2x^k$. Otherwise the errors were

$$\begin{aligned} &x^j + x^\infty, \text{ if } k = \infty; \\ &-x^j - x^\infty, \text{ if } d = 0; \\ &x^a + x^b, \text{ if } d \in \{1, 2, 4\}, a = (j - 3d) \bmod 7, b = (j - 2d) \bmod 7; \\ &-x^a - x^b, \text{ if } d \in \{3, 5, 6\}, a = (j - 3d) \bmod 7, b = (j - d) \bmod 7. \end{aligned}$$

Given $u' = (1100100100001111)_2$, we have $u = (2, 0, 3, 1, 0, 0, 2, 2)$ and $2 + 3x^2 + x^3 + 2x^6 \equiv 1 + 3x + 3x^2 \equiv x^5 + 2x^6$; also $s = 2$. Thus the errors are $x^2 + x^3$, and the nearest errorfree codeword is $(2, 0, 2, 0, 0, 0, 2, 2)$. Algorithm 4.6.1D tells us that $2 + 2x^2 + 2x^6 \equiv (2 + 2x + 2x^3)g(x)$ (modulo 4); so the eight information bits are $(11110011)_2$, corresponding to $(v_0, v_1, v_2, v_3) = (2, 2, 0, 2)$. [A more intelligent algorithm would also say, “Aha: The first 16 bits of π .”]

For generalizations to other efficient coding schemes based on quaternary vectors, see the classic paper by Hammons, Kumar, Calderbank, Sloane, and Solé, *IEEE Trans. IT-40* (1994), 301–319.

21. (a) $C(\epsilon) = 1$, $C(0\alpha) = C(1\alpha) = C(\alpha)$, and $C(*\alpha) = 2C(\alpha) - [10\dots 0 \in \alpha]$. Iterating this recurrence gives $C(\alpha) = 2^t - 2^{t-1}e_t - 2^{t-2}e_{t-1} - \dots - 2^0e_1$, where $e_j = [10\dots 0 \in \alpha_j]$ and α_j is the suffix of α following the j th asterisk. In the example we have $\alpha_1 = *10**0*$, $\alpha_2 = 10**0*$, ..., $\alpha_5 = \epsilon$; thus $e_1 = 0$, $e_2 = 1$, $e_3 = 1$, $e_4 = 0$, and $e_5 = 1$ (by convention), hence $C(**10**0*) = 2^5 - 2^4 - 2^2 - 2^1 = 10$.

(b) We may remove trailing asterisks so that $t = t'$. Then $e_t = 1$ implies $e_{t-1} = \dots = e_1 = 0$. [The case $C(\alpha) = 2^{t'-1}$ occurs if and only if α ends in 10^{j+k} .]

(c) To compute the sum of $C(\alpha)$ over all t -subcubes, note that $\binom{n}{t}$ clusters begin at the n -tuple $0\dots 0$, and $\binom{n-1}{t}$ begin at each succeeding n -tuple (namely one cluster for each t -subcube containing that n -tuple and specifying the bit that changed). Thus the average is $((\binom{n}{t} + (2^n - 1)\binom{n-1}{t})/2^{n-t}\binom{n}{t}) = 2^t(1 - t/n) + 2^{t-n}(t/n)$. [The formula in (c) holds for any n -bit Gray path, but (a) and (b) are specific to the reflected Gray binary code. These results are due to C. Faloutsos, *IEEE Trans. SE-14* (1988), 1381–1393.]

22. Let α^{*j} and β^{*k} be consecutive lieves of a Gray binary trie, where α and β are binary strings and $j \leq k$. Then the last $k - j$ bits of α are a string α' such that α and $\beta\alpha'$ are consecutive elements of Gray binary code, hence adjacent. [Interesting applications of this property to cube-connected message-passing concurrent computers are discussed in *A VLSI Architecture for Concurrent Data Structures* by William J. Dally (Kluwer, 1987), Chapter 3.]

23. $2^j = g(k) \oplus g(l) = g(k \oplus l)$ implies that $l = k \oplus g^{[-1]}(2^j) = k \oplus (2^{j+1} - 1)$. In other words, if $k = (b_{n-1} \dots b_0)_2$ we have $l = (b_{n-1} \dots b_{j+1} \bar{b}_j \dots \bar{b}_0)_2$.

24. Defining $g(k) = k \oplus \lfloor k/2 \rfloor$ as usual, we find $g(k) = g(-1 - k)$; hence there are two 2-adic integers k such that $g(k)$ has a given 2-adic value l . One of them is even, the other is odd. We can conveniently define $g^{[-1]}$ to be the solution that is even; then (8) is replaced by $b_j = a_{j-1} \oplus \dots \oplus a_0$, for $j \geq 0$. For example, $g^{[-1]}(1) = -2$ by this definition; when l is a normal integer, the “sign” of $g^{[-1]}(l)$ is the parity of l .

25. Let $p = k \oplus l$; exercise 7.1–00 tells us that $2^{\lfloor \lg p \rfloor + 1} - p \leq |k - l| \leq p$. We have $\nu(g(p)) = \nu(g(k) \oplus g(l)) = t$ if and only if there are positive integers j_1, \dots, j_t such that $p = (1^{j_1} 0^{j_2} 1^{j_3} \dots (0 \text{ or } 1)^{j_t})_2$. The largest possible $p < 2^n$ occurs when $j_1 = n + 1 - t$ and $j_2 = \dots = j_t = 1$, yielding $p = 2^n - \lceil 2^t / 3 \rceil$. The smallest possible $2^{\lfloor \lg p \rfloor + 1} - p = (1^{j_2} 0^{j_3} \dots (1 \text{ or } 0)^{j_t})_2 + 1$ occurs when $j_2 = \dots = j_t = 1$, yielding $p = \lceil 2^t / 3 \rceil$. [C. K. Yuen, *IEEE Trans. IT-20* (1974), 668; S. R. Cavior, *IEEE Trans. IT-21* (1975), 596.]

26. Let $N = 2^{n_t} + \dots + 2^{n_1}$ where $n_t > \dots > n_1 \geq 0$; also, let Γ_n be any Gray code for $\{0, 1, \dots, 2^n - 1\}$ that begins at 0 and ends at 1, except that Γ_0 is simply 0. Use

$$\begin{aligned} \Gamma_{n_t}^R, 2^{n_t} + \Gamma_{n_{t-1}}, \dots, 2^{n_t} + \dots + 2^{n_3} + \Gamma_{n_2}^R, 2^{n_t} + \dots + 2^{n_2} + \Gamma_{n_1}, &\text{ if } t \text{ is even;} \\ \Gamma_{n_t}, 2^{n_t} + \Gamma_{n_{t-1}}^R, \dots, 2^{n_t} + \dots + 2^{n_3} + \Gamma_{n_2}^R, 2^{n_t} + \dots + 2^{n_2} + \Gamma_{n_1}, &\text{ if } t \text{ is odd.} \end{aligned}$$

27. In general, if $k = (b_{n-1} \dots b_0)_2$, the $(k+1)$ st largest element of S_n is equal to

$$1/(2 - (-1)^{a_{n-1}}/(2 - \dots/(2 - (-1)^{a_1}/(2 - (-1)^{a_0})) \dots)),$$

corresponding to the sign pattern $g(k) = (a_{n-1} \dots a_0)_2$. Thus we can compute any element of S_n in $O(n)$ steps, given its rank. Setting $k = 2^{100} - 10^{10}$ and $n = 100$ yields the answer 373065177/1113604409. [Whenever $f(x)$ is a positive and monotonic function, the 2^n elements $f(\pm f(\dots \pm f(\pm x) \dots))$ are ordered according to Gray binary code, as observed by H. E. Salzer, *CACM* **16** (1973), 180. In this particular case there is, however, another way to get the answer, because we also have $S_n = //2, \pm 2, \dots, \pm 2, \pm 1//$ using the notation of Section 4.5.3; continued fractions in this form are ordered by complementing alternate bits of k .]

28. (a) As $t = 1, 2, \dots$, bit a_j of $\text{median}(G_t)$ runs through the periodic sequence

$$0, \dots, 0, *, 1, \dots, 1, *, 0, \dots, 0, *, \dots$$

with asterisks at every 2^{1+j} th step. Thus the strings that correspond to the binary representations of $\lfloor (t-1)/2 \rfloor$ and $\lfloor t/2 \rfloor$ are medians. And those strings are in fact “extreme” cases, in the sense that all medians agree with the common bits of $\lfloor (t-1)/2 \rfloor$ and $\lfloor t/2 \rfloor$, hence asterisks appear where they disagree. For example, when $t = 100 = (01100100)_2$ and $n = 8$, we have $\text{median}(G_{100}) = 001100**$.

(b) If α is $g(p)$ and β is $g(q)$ in Gray binary, we have $p = (p_{n-1} \dots p_0)_2$ and $q = (p_{n-1} \dots p_{j+1} \bar{p}_j \dots \bar{p}_0)_2$, where $p_{j+1} \oplus p_j = a'_j \neq a_j$. By (a), either t or $t-1$ lies in $(a_{n-1} \dots a_0*)_2$. Now if $p_j = 1$, clearly $q < p < t$. And if $p_j = 0$ we have $p_{j+1} = \bar{a}_j$; hence $p < t$ implies $q < t$. [See A. J. Bernstein, K. Steiglitz, and J. E. Hopcroft, *IEEE Trans. IT-12* (1966), 425–430.]

29. Assuming that $p \neq 0$, let $l = \lfloor \lg p \rfloor$ and $S_a = \{s \mid 2^l a \leq s < 2^l(a+1)\}$ for $0 \leq a < 2^{n-l}$. Then $(k \oplus p) - k$ has a constant sign for all $k \in S_a$, and

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

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

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

[See Morgan M. Buchner, Jr., *Bell System Tech. J.* **48** (1969), 3113–3130.]

30. The cycle containing $k > 1$ has length $2^{\lfloor \lg \lg k \rfloor + 1}$, because it is easy to show from Eq.(7) that if $k = (b_{n-1} \dots b_0)_2$ we have

$$g^{[2^l]}(k) = (c_{n-1} \dots c_0)_2, \quad \text{where } c_j = b_j \oplus b_{j+l+1}.$$

To permute all elements k such that $\lfloor \lg k \rfloor = t$, there are two cases: If t is a power of 2, the cycle containing $2\lfloor k/2 \rfloor$ also contains $2\lfloor k/2 \rfloor + 1$, so we must double the cycle leaders for $t - 1$. Otherwise the cycle containing $2\lfloor k/2 \rfloor$ is disjoint from the cycle containing $2\lfloor k/2 \rfloor + 1$, so $L_t = (2L_{t-1}) \cup (2L_{t-1} + 1) = (L_{t-1}*)_2$. This argument, discovered by Jörg Arndt in 2001, establishes the hint and yields the following algorithm:

- P1.** [Initialize.] Set $t \leftarrow 1$, $m \leftarrow 0$. (We may assume that $n \geq 2$.)
- P2.** [Loop through leaders.] Set $r \leftarrow m$. Perform Algorithm Q with $k = 2^t + r$; then if $r > 0$, set $r \leftarrow (r-1) \wedge m$ and repeat until $r = 0$. [See exercise 7.1–00.]
- P3.** [Increase $\lg k$.] Set $t \leftarrow t + 1$. Terminate if t is now equal to n ; otherwise set $m \leftarrow 2m + [t \wedge (t-1) \neq 0]$ and return to P2. ▀
- Q1.** [Begin a cycle.] Set $s \leftarrow X_k$, $l \leftarrow k$, $j \leftarrow l \oplus \lfloor l/2 \rfloor$.
- Q2.** [Follow the cycle.] If $j \neq k$ set $X_l \leftarrow X_j$, $l \leftarrow j$, $j \leftarrow l \oplus \lfloor l/2 \rfloor$, and repeat until $j = k$. Then set $X_l \leftarrow s$. ▀

31. We get a field from f_n if and only if we get one from $f_n^{[2]}$, which takes $(a_{n-1} \dots a_0)_2$ to $((a_{n-1} \oplus a_{n-2})(a_{n-1} \oplus a_{n-3})(a_{n-2} \oplus a_{n-4}) \dots (a_2 \oplus a_0)(a_1))_2$. Let $c_n(x)$ be the characteristic polynomial of the matrix A defining this transformation, mod 2; then $c_1(x) = x + 1$, $c_2(x) = x^2 + x + 1$, and $c_{n+1}(x) = xc_n(x) + c_{n-1}(x)$. Since $c_n(A)$ is the zero matrix, by the Cayley–Hamilton theorem, a field is obtained if and only if $c_n(x)$ is a primitive polynomial, and this condition can be tested as in Section 3.2.2. The first such values of n are 1, 2, 3, 5, 6, 9, 11, 14, 23, 29, 30, 33, 35, 39, 41, 51, 53, 65, 69, 74, 81, 83, 86, 89, 90, 95.

[Running the recurrence backwards shows that $c_{-j-2}(x) = c_j(x)$, hence $c_j(x)$ divides $c_{(2j+1)k+j}(x)$; for example, $c_{3k+1}(x)$ is always a multiple of $x+1$. All numbers n of the form $2jk + j + k$ are therefore excluded when $j > 0$ and $k > 0$. The polynomials $c_{18}(x)$, $c_{50}(x)$, $c_{98}(x)$, and $c_{99}(x)$ are irreducible but not primitive.]

32. Mostly true, but false at the points where x changes sign. (Walsh originally suggested that $w_k(x)$ should be zero at such points; but the convention adopted here is better, because it makes simple formulas like (15)–(19) valid for all x .)

33. By induction on k , we have

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

for $0 \leq x < \frac{1}{2}$, because $r_j(2x) = r_{j+1}(x)$ and $r_1(x) = 1$ in this range. And when $\frac{1}{2} \leq x < 1$,

$$\begin{aligned} w_k(x) &= (-1)^{\lceil k/2 \rceil} w_{\lfloor k/2 \rfloor}(2x-1) = r_1(x)^{b_0+b_1} r_1(2x-1)^{b_1+b_2} r_2(2x-1)^{b_2+b_3} \dots \\ &= r_1(x)^{b_0+b_1} r_2(x)^{b_1+b_2} r_3(x)^{b_2+b_3} \dots \end{aligned}$$

because $\lceil k/2 \rceil \equiv b_0 + b_1$ (modulo 2) and $r_j(2x-1) = r_{j+1}(x - \frac{1}{2}) = r_{j+1}(x)$ for $j \geq 1$.

34. $p_k(x) = \prod_{j \geq 0} r_{j+1}^{b_j}$; hence $w_k(x) = p_k(x) p_{\lfloor k/2 \rfloor}(x) = p_{g(k)}(x)$. [R. E. A. C. Paley, *Proc. London Math. Soc.* (2) **34** (1932), 241–279.]

35. If $j = (a_{n-1} \dots a_0)_2$ and $k = (b_{n-1} \dots b_0)_2$, the element in row j and column k is $(-1)^{f(j,k)}$, where $f(j,k)$ is the sum of all $a_r b_s$ such that: $r = s$ (Hadamard); $r+s = n-1$ (Paley); $r+s = n$ or $n-1$ (Walsh).

Let R_n , F_n , and G_n be permutation matrices for the permutations that take $j = (a_{n-1} \dots a_0)_2$ to $k = (a_0 \dots a_{n-1})_2$, $k = 2^n - 1 - j = (\bar{a}_{n-1} \dots \bar{a}_0)_2$, and $k = g^{[-1]}(j) = ((a_{n-1}) \dots (a_{n-1} \oplus \dots \oplus a_0))_2$, respectively. Then, using the Kronecker product of matrices, we have the recursive formulas

$$\begin{aligned} R_{n+1} &= \begin{pmatrix} R_n \otimes (1 \ 0) \\ R_n \otimes (0 \ 1) \end{pmatrix}, & F_{n+1} &= F_n \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, & G_{n+1} &= \begin{pmatrix} G_n & 0 \\ 0 & G_n F_n \end{pmatrix}, \\ H_{n+1} &= H_n \otimes \begin{pmatrix} 1 & 1 \\ 1 & \bar{1} \end{pmatrix}, & P_{n+1} &= \begin{pmatrix} P_n \otimes (1 \ 1) \\ P_n \otimes (1 \ \bar{1}) \end{pmatrix}, & W_{n+1} &= \begin{pmatrix} W_n \otimes (1 \ 1) \\ F_n W_n \otimes (1 \ \bar{1}) \end{pmatrix}. \end{aligned}$$

Thus $W_n = G_n^T P_n = P_n G_n$; $H_n = P_n R_n = R_n P_n$; and $P_n = W_n G_n^T = G_n W_n = H_n R_n = R_n H_n$.

36. W1. [Hadamard transform.] For $k = 0, 1, \dots, n-1$, replace the pair (X_j, X_{j+2^k}) by $(X_j + X_{j+2^k}, X_j - X_{j+2^k})$ for all j with $\lfloor j/2^k \rfloor$ even, $0 \leq j < 2^n$. (These operations effectively set $X^T \leftarrow H_n X^T$.)

W2. [Bit reversal.] Apply the algorithm of exercise 5 to the vector X . (These operations effectively set $X^T \leftarrow R_n X^T$, in the notation of exercise 35.)

W3. [Gray binary permutation.] Apply the algorithm of exercise 30 to the vector X . (These operations effectively set $X^T \leftarrow G_n^T X^T$). ■

If n has one of the special values in exercise 31, it may be faster to combine steps W2 and W3 into a single permutation step.

37. If $k = 2^{e_1} + \dots + 2^{e_t}$ with $e_1 > \dots > e_t \geq 0$, the sign changes occur at $S_{e_1} \cup \dots \cup S_{e_t}$, where

$$S_0 = \left\{ \frac{1}{2} \right\}, \quad S_1 = \left\{ \frac{1}{4}, \frac{3}{4} \right\}, \quad \dots, \quad S_e = \left\{ \frac{2j+1}{2^e} \mid 0 \leq j < 2^e \right\}.$$

Therefore the number of sign changes in $(0 \dots x)$ is $\sum_{j=1}^t \lfloor 2^{e_j} x + \frac{1}{2} \rfloor$. Setting $x = l/(k+1)$ gives $l+O(t)$ changes; so the l th is at a distance of at most $O(\nu(k))/2^{\lfloor \lg k \rfloor}$ from $l/(k+1)$.

[This argument makes it plausible that infinitely many pairs (k, l) exist with $|z_{kl} - l/(k+1)| = \Omega((\log k)/k)$. But no explicit construction of such “bad” pairs is immediately apparent.]

38. Let $t_0(x) = 1$ and $t_k(x) = \omega^{\lfloor 3x \rfloor \lceil 2k/3 \rceil} t_{\lfloor k/3 \rfloor}(3x)$, where $\omega = e^{2\pi i/3}$. Then $t_k(x)$ winds around the origin $\frac{2}{3}k$ times as x increases from 0 to 1. If $s_k(x) = \omega^{\lfloor 3^k x \rfloor}$ is the ternary analog of the Rademacher function $r_k(x)$, we have $t_k(x) = \prod_{j \geq 0} s_{j+1}(x)^{b_j - b_{j+1}}$ when $k = (b_{n-1} \dots b_0)_3$, as in the modular ternary Gray code.

39. Let’s call the symbols $\{x_0, x_1, \dots, x_7\}$ instead of $\{a, b, c, d, e, f, g, h\}$. We want to find a permutation p of $\{0, 1, \dots, 7\}$ such that the matrix with $(-1)^{j \cdot k} x_{p(j) \oplus k}$ in row j and column k has orthogonal rows; this condition is equivalent to requiring that

$$(j + j') \cdot (p(j) + p(j')) \equiv 1 \pmod{2}, \quad \text{for } 0 \leq j < j' < 8.$$

One solution is $p(0) \dots p(7) = 01725634$, yielding the identity $(a^2 + b^2 + c^2 + d^2 + e^2 + f^2 + g^2 + h^2)(A^2 + B^2 + C^2 + D^2 + E^2 + F^2 + G^2 + H^2) = \mathcal{A}^2 + \mathcal{B}^2 + \mathcal{C}^2 + \mathcal{D}^2 +$

$\mathcal{E}^2 + \mathcal{F}^2 + \mathcal{G}^2 + \mathcal{H}^2$, where

$$\begin{pmatrix} \mathcal{A} \\ \mathcal{B} \\ \mathcal{C} \\ \mathcal{D} \\ \mathcal{E} \\ \mathcal{F} \\ \mathcal{G} \\ \mathcal{H} \end{pmatrix} = \begin{pmatrix} a & b & c & d & e & f & g & h \\ b & -a & d & -c & f & -e & h & -g \\ c & g & -f & -e & d & c & -b & -a \\ d & -d & -a & b & g & -h & -e & f \\ e & f & e & g & -b & -a & -d & -c \\ f & g & -h & e & -f & -c & d & -a \\ g & d & c & -b & -a & -h & -g & f \\ h & e & -f & -g & h & -a & b & c \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{pmatrix}.$$

(b) There is no 16×16 solution. The closest one can come is

$$p(0) \dots p(15) = 0 1 11 2 14 15 13 4 9 10 7 12 5 6 3 8,$$

which fails if and only if $j \oplus j' = 5$. [See *Philos. Mag.* **34** (1867), 461–475. In §9, §10, §11, and §13 of this paper, Sylvester stated and proved the basic results about what has somehow come to be known as the Hadamard transform—although Hadamard himself gave credit to Sylvester [*Bull. des Sciences Mathématiques* (2) **17** (1893), 240–246]. Moreover, Sylvester introduced transforms of m^n elements in §14, using m th roots of unity.]

40. Yes; this change would in fact run through the swapped subsets in lexicographic binary order rather than in Gray binary order. (Any 5×5 matrix of 0s and 1s that is nonsingular mod 2 will generate all 32 possibilities when we run through all linear combinations of its rows.) The most important thing is the appearance of the ruler function, or some other Gray code delta sequence, not the fact that only one a_j changes per step, in cases like this where any number of the a_j can be changed simultaneously at the same cost.

41. At most 16; for example, **fired**, **fires**, **finds**, **fines**, **fined**, **fares**, **fared**, **wares**, **wards**, **wands**, **wanes**, **waned**, **wines**, **winds**, **wires**, **wired**. We also get 16 from **paced**/**links** and **paled**/**mints**; perhaps also from a word mixed with an antipodal nonword.

42. Suppose $n \leq 2^r + r + 1$, and let $s = 2^r$. We use an auxiliary table of 2^{r+s} bits f_{jk} for $0 \leq j < 2^s$ and $0 \leq k < s$, representing focus pointers as in Algorithm L, together with an auxiliary s -bit “register” $j = (j_{s-1} \dots j_0)_2$ and an $(r+2)$ -bit “program counter” $p = (p_{r+1} \dots p_0)_2$. At each step we examine the program counter and possibly the j register and one of the f bits; then, based on the bits seen, we complement a bit of the Gray code, complement a bit of the program counter, and possibly change a j or f bit, thereby emulating step L3 with respect to the most significant $n - r - 2$ bits.

For example, here is the construction when $r = 1$:

$p_2 p_1 p_0$	Change	Set	$p_2 p_1 p_0$	Change	Set
0 0 0	a_0, p_0	$j_0 \leftarrow f_{00}$	1 1 0	a_0, p_0	$f_{j0} \leftarrow f_{(j+1)0}$
0 0 1	a_1, p_1	$j_1 \leftarrow f_{01}$	1 1 1	a_1, p_1	$f_{j1} \leftarrow f_{(j+1)1}$
0 1 1	a_0, p_0	$f_{00} \leftarrow 0$	1 0 1	a_0, p_0	$f_{(j+1)0} \leftarrow (j+1)_0$
0 1 0	a_2, p_2	$f_{01} \leftarrow 0$	1 0 0	a_{j+3}, p_2	$f_{(j+1)1} \leftarrow (j+1)_1$

The process stops when it attempts to change bit a_n .

[In fact, we need change only *one* auxiliary bit per step if we allow ourselves to examine some Gray binary bits as well as the auxiliary bits, because $p_r \dots p_0 = a_r \dots a_0$, and we can set $f_0 \leftarrow 0$ in a more clever way when j doesn’t have its final value $2^s - 1$. This construction, suggested by Fredman in 2001, improves on another that he had

published in *SICOMP* **7** (1978), 134–146. With a more elaborate construction it is possible to reduce the number of auxiliary bits to $O(n)$.]

43. This number was estimated by Silverman, Vickers, and Sampson [*IEEE Trans. IT-29* (1983), 894–901] to be about 7×10^{22} . Exact calculation might be feasible because every 6-bit Gray code has only five or fewer segments that lie in a 5-cube corresponding to at least one of the six coordinates. (In unpublished work, Steve Winkler had used a similar idea to evaluate $d(5)$ in less than 15 minutes on a “generic” computer in 1972.)

44. All $(n+1)$ -bit delta sequences with just two occurrences of the coordinate j are produced by the following construction: Take n -bit delta sequences $\delta_0 \dots \delta_{2^n-1}$ and $\varepsilon_0 \dots \varepsilon_{2^n-1}$ and find an index k with $\delta_k = \varepsilon_0$. Form the cycle

$$\delta_0 \dots \delta_{k-1} n \varepsilon_1 \dots \varepsilon_{2^n-1} n \delta_{k+1} \dots \delta_{2^n-1}$$

and then interchange $n \leftrightarrow j$.

All $(n-2)$ -bit delta sequences with just two occurrences of coordinates h and j (with h before j) are, similarly, produced from four n -bit sequences $\delta_0 \dots \delta_{2^n-1}, \dots, \eta_0 \dots \eta_{2^n-1}$ and an index k with $\delta_k = \varepsilon_0 = \zeta_0 = \eta_0$, by interchanging $n \leftrightarrow h$ and $n+1 \leftrightarrow j$ in

$$\delta_0 \dots \delta_{k-1} n \varepsilon_1 \dots \varepsilon_{2^n-1} (n+1) \zeta_1 \dots \zeta_{2^n-1} n \eta_1 \dots \eta_{2^n-1} (n+1) \delta_{k+1} \dots \delta_{2^n-1}.$$

Let $a(n)$ and $b(n)$ be the answers to (a) and (b), for $n \geq 1$. The first construction shows that $a(n+1) + 2b(n+1) = 2^n(n+1)d(n)^2/n$, because it produces the delta sequences enumerated by $b(n+1)$ in two ways. The second construction shows that $b(n+2) = 2^n(n+2)(n+1)d(n)^4/n^3$.

45. We have $d(n+1) \geq 2^n d(n)^2/n$, because $2^n d(n)^2/n$ is the number of $(n+1)$ -bit delta sequences with exactly two appearances of 0. Hence $d(n) > \frac{5}{32}2^{2n}$ for $n \geq 5$, by induction on n .

Indeed, we can establish even faster growth by using the previous exercise, because $d(n+1) \geq a(n+1) + b(n+1)$ and $b(n+1) \leq \frac{25}{64}(n+1)d(n)^2/n$ for $n \geq 5$. Hence $d(n+1) \geq (2^n - \frac{25}{64})(n+1)d(n)^2/n$ for $n \geq 5$, and iteration of this relation shows that

$$\lim_{n \rightarrow \infty} d(n)^{1/2^n} \geq d(5)^{1/32} \prod_{n=5}^{\infty} \left(2^n - \frac{25}{64}\right)^{1/2^{n+1}} \left(\frac{n+1}{n}\right)^{1/2^{n+1}} \approx 2.3606.$$

[See R. J. Douglas, *Disc. Math.* **17** (1977), 143–146; M. Mollard, *European J. Comb.* **9** (1988), 49–52.] But the true value of this limit is probably ∞ .

46. Leo Moser (unpublished) has conjectured that it is $\sim n/e$. So far only an upper bound of about $n/\sqrt{2}$ has been established; see the references in the previous answer.

48. If $d(n, k, v)$ of the codes begin with $g(0) \dots g(k-1)v$, the conjecture implies that $d(n, k, v) \leq d(n, k, g(k))$, because the reverse of a Gray code is a Gray code. Thus the hint follows from $d(n) = d(n, 1)$ and

$$d(n, k) = \sum_v \{ d(n, k, v) \mid v \text{--- } g(k-1), v \notin S_k \} \leq c_{nk} d(n, k, g(k)) = d(n, k+1).$$

Finally, $d(n, 2^n) = 1$, hence $d(n) \leq \prod_{k=1}^{2^n-1} c_{nk} = \prod_{k=1}^n k^{\binom{n}{k}} = n \prod_{k=1}^{n-1} (k(n-k))^{\binom{n}{k}/2} \leq n \prod_{k=1}^{n-1} (n/2)^{\binom{n}{k}} = n(n/2)^{2^n-2}$. [*IEEE Trans. IT-29* (1983), 894–901.]

49. Take any Hamiltonian path P from 0...0 to 1...1 in the $(2n-1)$ -cube, such as the Savage–Winkler path, and use $0P, 1P^R$. (All such codes are obtained by this construction when $n = 1$ or $n = 2$, but many more possibilities exist when $n > 2$.)

50. $\alpha_1(n+1)\alpha_1^R n \alpha_1 j_1 \alpha_2 n \alpha_2^R(n+1)\alpha_2 \dots j_{l-1} \alpha_l n \alpha_l^R(n+1)\alpha_l n \alpha_l^R j_{l-1} \dots j_1 \alpha_1^R n$.

51. We can assume that $n > 3$ and that we have an n -bit Gray code with transition counts $c_j = 2\lfloor(2^{n-1} + j)/n\rfloor$; we want to construct an $(n+2)$ -bit code with transition counts $c'_j = 2\lfloor(2^{n+1} + j)/(n+2)\rfloor$. If $2^{n+1} \bmod (n+2) \geq 2$, we can use Theorem D with $l = 2\lfloor 2^{n+1}/(n+2) \rfloor + 1$, underlining b_j copies of j where $b_j = 4\lfloor(2^{n-1} + j)/n\rfloor - \lfloor(2^{n+1} + j)/(n+2)\rfloor - [j=0]$ and putting an underlined 0 last. This is always easy to do because $|b_j - 2^{n+2}/n(n+2)| < 5$. A similar construction works if $2^{n+1} \bmod (n+2) \leq n$, with $l = 2\lfloor 2^{n+1}/(n+2) \rfloor - 1$ and $b_j = 4\lfloor(2^{n-1} + j)/n\rfloor - \lfloor(2^{n+1} + j+2)/(n+2)\rfloor - [j=0]$. In fact, $2^{n+1} \bmod (n+2)$ is always $\leq n$ [see K. Kedlaya, *Electronic J. Combinatorics* **3** (1996), comment on #R25 (9 April 1997)]. The basic idea of this proof is due to J. P. Robinson and M. Cohn [*IEEE Trans. C-30* (1981), 17–23].

52. The number of different code patterns in the smallest j coordinate positions is at most $c_0 + \dots + c_{j-1}$.

53. Notice that Theorem D produces only codes with $c_j = c_{j+1}$ for some j , so it cannot produce the counts $(2, 4, 6, 8, 12)$. The extension in exercise 50 gives also $c_j = c_{j+1} - 2$, but it cannot produce $(6, 10, 14, 18, 22, 26, 32)$. The sets of numbers satisfying the conditions of exercise 52 are precisely those obtainable by starting with $\{2, 2, 4, \dots, 2^{n-1}\}$ and repeatedly replacing some pair $\{c_j, c_k\}$ for which $c_j < c_k$ by the pair $\{c_j + 2, c_k - 2\}$.

54. Suppose the values are $\{p_1, \dots, p_n\}$, and let x_{jk} be the number of times p_j occurs in (a_1, \dots, a_k) . We must have $(x_{1k}, \dots, x_{nk}) \equiv (x_{1l}, \dots, x_{nl})$ (modulo 2) for some $k < l$. But if the p 's are prime numbers, varying as the delta sequence of an n -bit Gray code, the only solution is $k = 0$ and $l = 2^n$. [*AMM* **60** (1953), 418; **83** (1976), 54.]

56. [*Bell System Tech. J.* **37** (1958), 815–826.] The 112 canonical delta sequences yield

Class	Example	t	Class	Example	t	Class	Example	t
A	0102101302012023	2	D	0102013201020132	4	G	0102030201020302	8
B	0102303132101232	2	E	0102032021202302	4	H	0102101301021013	8
C	0102030130321013	2	F	0102013102010232	4	I	0102013121012132	1

Here B is the balanced code (Fig. 13(b)), G is standard Gray binary (Fig. 10(b)), and H is the complementary code (Fig. 13(a)). Class H is also equivalent to the modular $(4, 4)$ Gray code under the correspondence of exercise 18. A class with t automorphisms corresponds to $32 \times 24/t$ of the 2688 different delta sequences $\delta_0 \delta_1 \dots \delta_{15}$.

Similarly (see exercise 7.2.3–00), the 5-bit Gray codes fall into 237,675 different equivalence classes.

57. With Type 1 only, 480 vertices are isolated, namely those of classes D, F, G in the previous answer. With Type 2 only, the graph has 384 components, 288 of which are isolated vertices of classes F and G . There are 64 components of size 9, each containing 3 vertices from E and 6 from A ; 16 components of size 30, each with 6 from H and 24 from C ; and 16 components of size 84, each with 12 from D , 24 from B , 48 from I . With Type 3 (or Type 4) only, the entire graph is connected. [Similarly, all 91,392 of the 4-bit Gray paths are connected if path $\alpha\beta$ is considered adjacent to path $\alpha^R\beta$. Vickers and Silverman, *IEEE Trans. C-29* (1980), 329–331, have conjectured that Type 3 changes will suffice to connect the graph of n -bit Gray codes for all $n \geq 3$.]

58. If some nonempty substring of $\beta\beta$ involves each coordinate an even number of times, that substring cannot have length $|\beta|$, so some cyclic shift of β has a prefix γ

with the same evenness property. But then α doesn't define a Gray code, because we could change each n of γ back to 0.

59. If α is nonlocal in exercise 58, so is $\beta\beta$, provided that $q > 1$ and that 0 occurs more than $q + 1$ times in α . Therefore, starting with the α of (30) but with 0 and 1 interchanged, we obtain nonlocal codes for $n \geq 5$ in which coordinate 0 changes exactly 6 times. [Mark Ramras, *Discrete Math.* **85** (1990), 329–331.] On the other hand, a 4-bit Gray code cannot be nonlocal because it always has a run of length 2; if $\delta_k = \delta_{k+2}$, elements $\{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}$ form a 2-subcube.

60. Use the construction of exercise 58 with $q = 1$.

61. The idea is to interleave an m -bit code $U = (u_0, u_1, u_2, \dots)$ with an n -bit code $V = (v_0, v_1, v_2, \dots)$, by forming concatenations

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

where $a_0a_1a_2\dots$ is a periodic string of control bits $\alpha\alpha\alpha\dots$; we advance to the next element of U when $a_k = 0$, otherwise to the next element of V .

If α is any string of length $2^m \leq 2^n$, containing s bits that are 0 and $t = 2^m - s$ bits that are 1, W will be an $(m+n)$ -bit Gray code if s and t are odd. For we have $i_{k+l} \equiv i_k$ (modulo 2^m) and $j_{k+l} \equiv j_k$ (modulo 2^n) only if l is a multiple of 2^m , since $i_k + j_k = k$. Suppose $l = 2^mc$; then $j_{k+l} = j_k + tc$, so c is a multiple of 2^n .

(a) Let $\alpha = 0111$; then runs of length 8 occur in the left 2 bits and runs of length $\geq \lfloor \frac{4}{3}r(n) \rfloor$ occur in the right n bits.

(b) Let s be the largest odd number $\leq 2^m r(m)/(r(m) + r(n))$. Also let $t = 2^m - s$ and $a_k = \lfloor (k+1)t/2^m \rfloor - \lfloor kt/2^m \rfloor$, so that $i_k = \lceil ks/2^m \rceil$ and $j_k = \lfloor kt/2^m \rfloor$. If a run of length l occurs in the left m bits, we have $i_{k+l+1} \geq i_k + r(m) + 1$, hence $l+1 > 2^m r(m)/s \geq r(m) + r(n)$. And if it occurs in the right n bits we have $j_{k+l+1} \geq j_k + r(n) + 1$, hence

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

because $r(m) \leq r(n)$.

The construction often works also in less restricted cases. See the paper that introduced the study of Gray code runs: L. Goddyn, G. M. Lawrence, and E. Nemeth, *Utilitas Math.* **34** (1988), 179–192.

63. Set $a_k \leftarrow k \bmod 4$ for $0 \leq k < 2^{10}$, except that $a_k = 4$ when $k \bmod 16 = 15$ or $k \bmod 64 = 42$ or $k \bmod 256 = 133$. Also set $(j_0, j_1, j_2, j_3, j_4) \leftarrow (0, 2, 4, 6, 8)$. Then for $k = 0, 1, \dots, 1023$, set $\delta_k \leftarrow j_{a_k}$ and $j_{a_k} \leftarrow 1 + 4a_k - j_{a_k}$. (This construction generalizes the method of exercise 61.)

64. (a) Each element u_k appears together with $\{v_k, v_{k+2^m}, \dots, v_{k+2^{m(2^{n-1}-1)}}\}$ and $\{v_{k+1}, v_{k+1+2^m}, \dots, v_{k+1+2^{m(2^{n-1}-1)}}\}$. Thus the permutation $\sigma_0 \dots \sigma_{2^m-1}$ must be a 2^{n-1} -cycle containing the n -bit vertices of even parity, times an arbitrary permutation of the other vertices. This condition is also sufficient.

(b) Let τ_j be the permutation that takes $v \mapsto v \oplus 2^j$, and let $\pi_j(u, w)$ be the permutation $(uw)\tau_j$. If $u \oplus w = 2^i + 2^j$ then $\pi_j(u, w)$ takes $u \mapsto u \oplus 2^i$ and $w \mapsto w \oplus 2^i$, while $v \mapsto v \oplus 2^j$ for all other vertices v , so it takes each vertex to a neighbor.

If S is any set $\subseteq \{0, \dots, n-1\}$, let $\sigma(S)$ be the stream of all permutations τ_j for all $j \in \{0, \dots, n-1\} \setminus S$, in increasing order of j , repeated twice; for example, if $n = 5$

we have $\sigma(\{1, 2\}) = \tau_0 \tau_3 \tau_4 \tau_0 \tau_3 \tau_4$. Then the Gray stream

$$\Sigma(i, j, u) = \sigma(\{i, j\}) \pi_j(u, u \oplus 2^i \oplus 2^j) \sigma(\{i, j\}) \tau_j \sigma(\{j\})$$

consists of $6n - 8$ permutations whose product is the transposition $(u \ u \oplus 2^i \oplus 2^j)$. Moreover, when this stream is applied to any n -bit vertex v , its runs all have length $\geq n - 2$.

We may assume that $n \geq 5$. Let $\delta_0 \dots \delta_{2^n-1}$ be the delta sequence for an n -bit Gray code $(v_0, v_1, \dots, v_{2^n-1})$ with all runs of length 3 or more. Then the product of all permutations in

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

is $(v_1 v_3)(v_2 v_4) \dots (v_{2^n-3} v_{2^n-1})(v_{2^n-2} v_0) = (v_{2^n-1} \dots v_1)(v_{2^n-2} \dots v_0)$, so it satisfies the cycle condition of (a).

Moreover, all powers $(\sigma(\emptyset)\Sigma)^t$ produce runs of length $\geq n - 2$ when applied to any vertex v . By repeating individual factors $\sigma(\{i, j\})$ or $\sigma(\{j\})$ in Σ as many times as we wish, we can adjust the length of $\sigma(\emptyset)\Sigma$, obtaining $2n + (2^{n-1} - 1)(12n - 16) + 2(n-2)a + 2(n-1)b$ for any integers $a, b \geq 0$; thus we can increase its length to exactly 2^m , provided that $2^m \geq 2n + (2^{n-1} - 1)(12n - 16) + 2(n^2 - 5n + 6)$, by exercise 5.2.1–21.

(c) The bound $r(n) \geq n - 4\lg n + 8$ can be proved for $n \geq 5$ as follows. First we observe that it holds for $5 \leq n < 33$ by the methods of exercises 60–63. Then we observe that every integer $N \geq 33$ can be written as $N = m + n$ or $N = m + n + 1$, for some $m \geq 20$, where

$$n = m - \lfloor 4\lg m \rfloor + 10.$$

If $m \geq 20$, 2^m is sufficiently large for the construction in part (b) to be valid; so we have

$$\begin{aligned} r(N) &\geq r(m+n) \geq 2 \min(r(m), n-2) \geq 2(m - \lfloor 4\lg m \rfloor + 8) \\ &= m + n + 1 - \lfloor 4\lg(m+n) - 1 + \epsilon \rfloor + 8 \\ &\geq N - 4\lg N + 8 \end{aligned}$$

where $\epsilon = 4\lg(2m/(m+n)) < 1$. [To appear.] Recursive use of (b) gives, in fact, $r(1024) \geq 1000$.

65. A computer search reveals that eight essentially different patterns (and their reverses) are possible. One of them has the delta sequence 01020314203024041234 214103234103, and it is close to two of the others.

66. (Solution by Mark Cooke.) One suitable delta sequence is 012345607012132435 65760710213534626701537412362567017314262065701342146560573102464537 57102043537614073630464273703564027132750541210275641502403654250136 02541615604312576032572043157624321760452041751635476703564757062543 7242132624161523417514367143164314.

67. Let $v_{2k+1} = \bar{v}_{2k}$ and $v_{2k} = 0u_k$, where $(u_0, u_1, \dots, u_{2^n-1})$ is any $(n - 1)$ -bit Gray code. [See Robinson and Cohn, *IEEE Trans. C-30* (1981), 17–23.]

68. Yes. The simplest way is probably to take $(n - 1)$ -trit modular Gray ternary code and add 0...0, 1...1, 2...2 to each string (modulo 3). For example, when $n = 3$ the code is 000, 111, 222, 001, 112, 220, 002, 110, 221, 012, 120, 201, ..., 020, 101, 212.

69. (a) We need only verify the change in h when bits $b_{j-1} \dots b_0$ are simultaneously complemented, for $j = 1, 2, \dots$; and these changes are respectively $(1110)_2, (1101)_2, (0111)_2, (1011)_2, (10011)_2, (100011)_2, \dots$. To prove that every n -tuple occurs, note that $0 \leq h(k) < 2^n$ when $0 \leq k < 2^n$ and $n > 3$; also $h^{[-1]}((a_{n-1} \dots a_0)_2) = (b_{n-1} \dots b_0)_2$, where $b_0 = a_0 \oplus a_1 \oplus a_2 \oplus \dots$, $b_1 = a_0$, $b_2 = a_2 \oplus a_3 \oplus a_4 \oplus \dots$, $b_3 = a_0 \oplus a_1 \oplus a_3 \oplus \dots$, and $b_j = a_j \oplus a_{j+1} \oplus \dots$ for $j \geq 4$.

(b) Let $h(k) = (\dots a_2 a_1 a_0)_2$ where $a_j = b_j \oplus b_{j+1} \oplus b_0[j \leq t] \oplus b_{t-1}[t-1 \leq j \leq t]$.

70. As in (32) and (33), we can remove a factor of $n!$ by assuming that the strings of weight 1 occur in order. Then there are 14 solutions for $n = 5$ starting with 00000, and 21 starting with 00001. When $n = 6$ there are 46,935 of each type (related by reversal and complementation). When $n = 7$ the number is much, much larger, yet very small by comparison with the total number of 7-bit Gray paths.

71. Suppose that $\alpha_{n(j+1)}$ differs from α_{nj} in coordinate t_j , for $0 \leq j < n-1$. Then $t_j = j\pi_n$, by (44) and (38). Now Eq. (34) tells us that $t_0 = n-1$; and if $0 < j < n-1$ we have $t_j = ((j-1)\pi_{n-1})\pi_{n-1}$ by (40). Thus $t_j = j\sigma_n\pi_{n-1}^2$ for $0 \leq j < n-1$, and the value of $(n-1)\pi_n$ is whatever is left. (Notations for permutations are notoriously confusing, so it is always wise to check a few small cases carefully.)

72. The delta sequence is 0102132430201234012313041021323.

73. Let $Q_{nj} = P_{nj}^R$ and denote the sequences (41), (42) by S_n and T_n . Thus $S_n = P_{n0}Q_{n1}P_{n2}\dots$ and $T_n = Q_{n0}P_{n1}Q_{n2}\dots$, if we omit the commas; and we have

$$S_{n+1} = 0P_{n0} 0Q_{n1} 1Q_{n0}^\pi 1P_{n1}^\pi 0P_{n2} 0Q_{n3} 1Q_{n2}^\pi 1P_{n3}^\pi 0P_{n4} \dots,$$

$$T_{n+1} = 0Q_{n0} 1P_{n0}^\pi 0P_{n1} 0Q_{n2} 1Q_{n1}^\pi 1P_{n2}^\pi 0P_{n3} 0Q_{n4} 1Q_{n3}^\pi \dots,$$

where $\pi = \pi_n$, revealing a reasonably simple joint recursion between the delta sequences Δ_n and E_n of S_n and T_n . Namely, if we write

$$\Delta_n = \phi_1 a_1 \phi_2 a_2 \dots \phi_{n-1} a_{n-1} \phi_n, \quad E_n = \psi_1 b_1 \psi_2 b_2 \dots \psi_{n-1} b_{n-1} \psi_n,$$

where each ϕ_j and ψ_j is a string of length $2\binom{n-1}{j-1} - 1$, the next sequences are

$$\Delta_{n+1} = \phi_1 a_1 \phi_2 n \psi_1 \pi b_1 \pi \psi_2 \pi n \phi_3 a_3 \phi_4 n \psi_3 \pi b_3 \pi \psi_4 \pi n \dots$$

$$E_{n+1} = \psi_1 n \phi_1 \pi n \psi_2 b_2 \psi_3 n \phi_2 \pi a_2 \pi \phi_3 \pi n \psi_4 b_4 \psi_5 n \phi_4 \pi a_4 \pi \phi_5 \pi n \dots$$

For example, we have $\Delta_3 = 0\underline{1}021\underline{0}1$ and $E_3 = 0\underline{2}120\underline{2}1$, if we underline the a 's and b 's to distinguish them from the ϕ 's and ψ 's; and

$$\Delta_4 = 0102130\pi 2\pi 1\pi 2\pi 0\pi 3131\pi = 0\underline{1}02132\underline{1}01231\underline{3}0,$$

$$E_4 = 030\pi 31202130\pi 2\pi 1\pi 0\pi 1\pi = 0\underline{3}23120\underline{2}13210\underline{2}0;$$

here $a_3\phi_4$ and $b_3\psi_4$ are empty. Elements have been underlined for the next step.

Thus we can compute the delta sequences in memory as follows. Here $p[j] = j\pi_n$ for $1 \leq j < n$; $s_k = \delta_k$, $t_k = \varepsilon_k$, and $u_k = [\delta_k \text{ and } \varepsilon_k \text{ are underlined}]$, for $0 \leq k < 2^n - 1$.

R1. [Initialize.] Set $n \leftarrow 1$, $p[0] \leftarrow 0$, $s_0 \leftarrow t_0 \leftarrow u_0 \leftarrow 0$.

R2. [Advance n .] Perform Algorithm S below, which computes the arrays s' , t' , and u' for the next value of n ; then set $n \leftarrow n + 1$.

R3. [Ready?] If n is sufficiently large, the desired delta sequence Δ_n is in array s' ; terminate. Otherwise keep going.

R4. [Compute π_n .] Set $p'[0] = n - 1$, and $p'[j] = p[p[j - 1]]$ for $1 \leq j < n$.

R5. [Prepare to advance.] Set $p[j] \leftarrow p'[j]$ for $0 \leq j < n$; set $s_k \leftarrow s'_k$, $t_k \leftarrow t'_k$, and $u_k \leftarrow u'_k$ for $0 \leq k < 2^n - 1$. Return to R2. ▀

In the following steps, “Transmit stuff(l, j) while $u_j = 0$ ” is an abbreviation for “If $u_j = 0$, repeatedly stuff(l, j), $l \leftarrow l + 1$, $j \leftarrow j + 1$, until $u_j \neq 0$.“

- S1.** [Prepare to compute Δ_{n+1} .] Set $j \leftarrow k \leftarrow l \leftarrow 0$ and $u_{2^n-1} \leftarrow -1$.
- S2.** [Advance j .] Transmit $s'_l \leftarrow s_j$ and $u'_l \leftarrow 0$ while $u_j = 0$. Then go to S5 if $u_j < 0$.
- S3.** [Advance j and k .] Set $s'_l \leftarrow s_j$, $u'_l \leftarrow 1$, $l \leftarrow l + 1$, $j \leftarrow j + 1$. Then transmit $s'_l \leftarrow s_j$ and $u'_l \leftarrow 0$ while $u_j = 0$. Then set $s'_l \leftarrow n$, $u'_l \leftarrow 0$, $l \leftarrow l + 1$. Then transmit $s'_l \leftarrow p[t_k]$ and $u'_l \leftarrow 0$ while $u_k = 0$. Then set $s'_l \leftarrow p[t_k]$, $u'_l \leftarrow 1$, $l \leftarrow l + 1$, $k \leftarrow k + 1$. And once again transmit $s'_l \leftarrow p[t_k]$ and $u'_l \leftarrow 0$ while $u_k = 0$.
- S4.** [Done with Δ_{n+1} ?] If $u_k < 0$, go to S6. Otherwise set $s'_l \leftarrow n$, $u'_l \leftarrow 0$, $l \leftarrow l + 1$, $j \leftarrow j + 1$, $k \leftarrow k + 1$, and return to S2.
- S5.** [Finish Δ_{n+1} .] Set $s'_l \leftarrow n$, $u'_l \leftarrow 1$, $l \leftarrow l + 1$. Then transmit $s'_l \leftarrow p[t[k]]$ and $u'_l \leftarrow 0$ while $u_k = 0$.
- S6.** [Prepare to compute E_{n+1} .] Set $j \leftarrow k \leftarrow l \leftarrow 0$. Transmit $t'_l \leftarrow t_k$ while $u_k = 0$. Then set $t'_l \leftarrow n$, $l \leftarrow l + 1$.
- S7.** [Advance j .] Transmit $t'_l \leftarrow p[s_j]$ while $u_j = 0$. Then terminate if $u_j < 0$; otherwise set $t'_l \leftarrow n$, $l \leftarrow l + 1$, $j \leftarrow j + 1$, $k \leftarrow k + 1$.
- S8.** [Advance k .] Transmit $t'_l \leftarrow t_k$ while $u_k = 0$. Then go to S10 if $u_k < 0$.
- S9.** [Advance k and j .] Set $t'_l \leftarrow t_k$, $l \leftarrow l + 1$, $k \leftarrow k + 1$. Then transmit $t'_l \leftarrow t_k$ while $u_k = 0$. Then set $t'_l \leftarrow n$, $l \leftarrow l + 1$. Then transmit $t'_l \leftarrow p[s_j]$ while $u_j = 0$. Then set $t'_l \leftarrow p[s_j]$, $l \leftarrow l + 1$, $j \leftarrow j + 1$. Return to S7.
- S10.** [Finish E_{n+1} .] Set $t'_l \leftarrow n$, $l \leftarrow l + 1$. Then transmit $t'_l \leftarrow p[s_j]$ while $u_j = 0$. ▀

To generate the monotonic Savage–Winkler path for fairly large n , one can first generate Δ_{10} and E_{10} , say, or even Δ_{20} and E_{20} . Using these tables, a suitable recursive procedure will then be able to reach higher values of n with very little computational overhead per step, on the average.

- 74.** If the monotonic path is v_0, \dots, v_{2^n-1} and if v_k has weight j , we have

$$2 \sum_{t>0} \binom{n}{j-2t} + ((j + \nu(v_0)) \bmod 2) \leq k \leq 2 \sum_{t\geq 0} \binom{n}{j-2t} + ((j + \nu(v_0)) \bmod 2) - 2.$$

Therefore the maximum distance between vertices of respective weights j and $j + 1$ is $2(\binom{n-1}{j-1} + \binom{n-1}{j} + \binom{n-1}{j+1}) - 1$. The maximum value, approximately $3 \cdot 2^n / \sqrt{2\pi n}$, occurs when j is approximately $n/2$. [This is only about three times the smallest value achievable in *any* ordering of the vertices, which is $\sum_{j=0}^{n-1} \binom{j}{\lfloor j/2 \rfloor}$ by exercise 7.10–00.]

- 75.** There are only five essentially distinct solutions, all of which turn out in fact to be Gray *codes*. The delta sequences are

```
0123012421032101210321040123012(1)
0123012421032101301230141032103(1)
0123012421032102032103242301230(2)
0123012423012302012301242301230(2)
0123410121030143210301410123410(3)
```

76. If v_0, \dots, v_{2^n-1} is trend-free, so is the $(n+1)$ -bit code $0v_0, 1v_0, 1v_1, 0v_1, 0v_2, 1v_2, \dots, 1v_{2^n-1}, 0v_{2^n-1}$. Fig. 14(g) shows a somewhat more interesting construction, which generalizes the first solution of exercise 75 to an $(n+2)$ -bit code

$$00\Gamma''^R, 01\Gamma'^R, 11\Gamma', 10\Gamma'', 10\Gamma, 11\Gamma''', 01\Gamma'''^R, 00\Gamma^R$$

where Γ is the n -bit sequences $g(1), \dots, g(2^{n-1})$ and $\Gamma' = \Gamma \oplus g(1)$, $\Gamma'' = \Gamma \oplus g(2^{n-1})$, $\Gamma''' = \Gamma \oplus g(2^{n-1} + 1)$. [An n -bit trend-free design that is *almost* a Gray code, having just four steps in which $\nu(v_k \oplus v_{k+1}) = 2$, was found for all $n \geq 3$ by C. S. Cheng, *Proc. Berkeley Conf. Neyman and Kiefer 2* (Hayward, Calif.: Inst. of Math. Statistics, 1985), 619–633.]

77. Replace the array (d_{n-1}, \dots, d_0) by an array of sentinel values (s_{n-1}, \dots, s_0) , with $s_j \leftarrow m_j - 1$ in step H1. Set $a_j \leftarrow (a_j + 1) \bmod m_j$ in step H4. If $a_j = s_j$ in step H5, set $s_j \leftarrow (s_j - 1) \bmod m_j$, $f_j \leftarrow f_{j+1}$, $f_{j+1} \leftarrow j + 1$.

78. For (50), notice that B_{j+1} is the number of times reflection has occurred in coordinate j , because we bypass coordinate j on steps that are multiples of $m_j \dots m_0$. Hence, if $b_j < m_j$, an increase of b_j by 1 causes a_j to increase or decrease by 1 as appropriate. Furthermore, if $b_i = m_i - 1$ for $0 \leq i < j$, changing all these b_i to 0 when incrementing b_j will increase each of B_0, \dots, B_j by 1, thereby leaving the values a_0, \dots, a_{j-1} unchanged in (50).

For (51), note that $B_j = m_j B_{j+1} + b_j \equiv m_j B_{j+1} + a_j + (m_j - 1) B_{j+1} \equiv a_j + B_{j+1}$ (modulo 2); hence $B_j \equiv a_j + a_{j+1} + \dots$, and (51) is obviously equivalent to (50).

In the modular Gray code for general radices (m_{n-1}, \dots, m_0) , let

$$mg(k) = \begin{bmatrix} a_{n-1}, \dots, a_2, a_1, a_0 \\ m_{n-1}, \dots, m_2, m_1, m_0 \end{bmatrix}$$

when k is given by (46). Then $a_j = (b_j - B_{j+1}) \bmod m_j$, because coordinate j has increased modulo m_j exactly $B_j - B_{j+1}$ times if we start at $(0, \dots, 0)$. The inverse function, which determines the b 's from the modular Gray a 's, is $b_j = (a_j + a_{j+1} + a_{j+2} + \dots) \bmod m_j$ in the special case that each m_j is a divisor of m_{j+1} (for example, if all m_j are equal). But the inverse has no simple form in general; it can be computed by using the recurrences $b_j = (a_j + B_{j+1}) \bmod m_j$, $B_j = m_j B_{j+1} + b_j$ for $j = n-1, \dots, 0$, starting with $B_n = 0$.

[Reflected Gray codes for radix $m > 2$ were introduced by Ivan Flores in *IRE Trans. EC-5* (1956), 79–82; he derived (50) and (51) in the case that all m_j are equal. Modular Gray codes with general mixed radices were implicitly discussed by Joseph Rosenbaum in *AMM 45* (1938), 694–696, but without the conversion formulas; conversion formulas when all m_j have a common value m were published by Martin Cohn, *Info. and Control 6* (1963), 70–78.]

79. (a) The last n -tuple always has $a_{n-1} = m_{n-1} - 1$, so it is one step from $(0, \dots, 0)$ only if $m_{n-1} = 2$. And this condition suffices to make the final n -tuple $(1, 0, \dots, 0)$. [Similarly, the final subforest output by Algorithm K is adjacent to the initial one if and only if the leftmost tree is an isolated vertex.]

(b) The last n -tuple is $(m_{n-1}-1, 0, \dots, 0)$ if and only if $m_{n-1} \dots m_{j+1} \bmod m_j = 0$ for $0 \leq j < n-1$, because $b_j = m_j - 1$ and $B_j = m_{n-1} \dots m_j - 1$.

80. Run through $p_1^{a_1} \dots p_t^{a_t}$ using reflected Gray code with radices $m_j = e_j + 1$.

81. The first cycle contains the edge from (x, y) to $(x, (y+1) \bmod m)$ if and only if $(x+y) \bmod m \neq m-1$ if and only if the second cycle contains the edge from (x, y) to $((x+1) \bmod m, y)$.

82. There are two 4-bit Gray codes (u_0, \dots, u_{15}) and (v_0, \dots, v_{15}) that cover all edges of the 4-cube. (Indeed, the non-edges of classes A, B, D, H, and I in exercise 56 form Gray codes, belonging to the same class as their complement.) Therefore with 16-ary modular Gray code we can form the four desired cycles $(u_0u_0, u_0u_1, \dots, u_0u_{15}, u_1u_{15}, \dots, u_{15}u_0)$, $(u_0u_0, u_1u_0, \dots, u_{15}u_0, u_{15}u_1, \dots, u_0u_{15})$, $(v_0v_0, \dots, v_{15}v_0)$, $(v_0v_0, \dots, v_0v_{15})$.

In a similar way we can show that $n/2$ edge-disjoint n -bit Gray codes exist when n is 16, 32, 64, etc. [Abhandlungen Math. Sem. Hamburg **20** (1956), 13–16.] J. Aubert and B. Schneider [*Discrete Math.* **38** (1982), 7–16] have proved that the same property holds for all even values of $n \geq 4$, but no simple construction is known.

84. Calling the initial position $(2, 2)$, the 8-step solution in Fig. A-1 shows how the sequence progresses down to $(0, 0)$. In the first move, for example, the front half of the cord passes around and behind the right comb, then through the large right loop. The middle line should be read from right to left. The generalization to n pairs of loops would, similarly, take $3^n - 1$ steps.

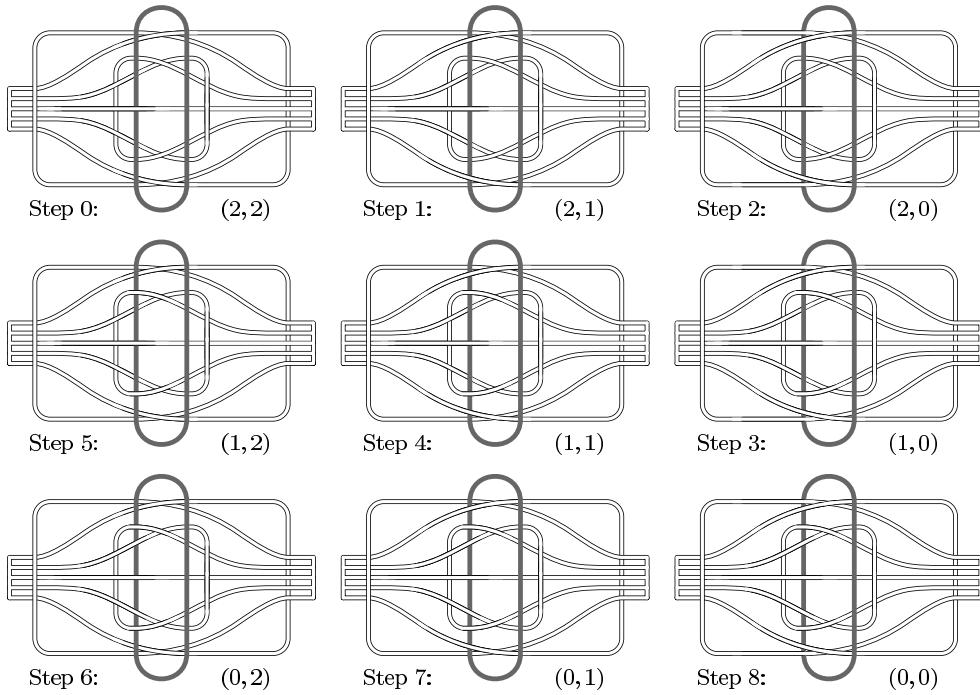


Fig. A-1.

[The origin of this delightful puzzle is obscure. *The Book of Ingenious & Diabolical Puzzles* by Jerry Slocum and Jack Botermans (1994) shows a 2-loop version carved from horn, probably made in China about 1850 [page 101], and a modern 6-loop version made in Malaysia about 1988 [page 93]. Slocum also owns a 4-loop version made from bamboo in England about 1884. He has found it listed in Henry Novra's *Catalogue of Conjuring Tricks and Puzzles* (1858 or 1859) and W. H. Cremer's *Games, Amusements, Pastimes and Magic* (1867), as well as in Hamleys' catalog of 1895, under the name "Marvelous Canoe Puzzle." Dyckman noted its connection to reflected Gray ternary in a letter to Martin Gardner, dated 2 August 1972.]

85. By (50), element $\left[\begin{smallmatrix} b, & b' \\ t, & t' \end{smallmatrix} \right]$ of $\Gamma \boxtimes \Gamma'$ is $\alpha_a \alpha'_{a'}$ if $rg\left(\left[\begin{smallmatrix} b, & b' \\ t, & t' \end{smallmatrix} \right]\right) = \left[\begin{smallmatrix} a, & a' \\ t, & t' \end{smallmatrix} \right]$ in the reflected Gray code for radices (t, t') . We can now show that element $\left[\begin{smallmatrix} b, & b', & b'' \\ t, & t', & t'' \end{smallmatrix} \right]$ of both $(\Gamma \boxtimes \Gamma') \boxtimes \Gamma''$ and $\Gamma \boxtimes (\Gamma' \boxtimes \Gamma'')$ is $\alpha_a \alpha'_{a'} \alpha''_{a''}$ if $rg\left(\left[\begin{smallmatrix} b, & b', & b'' \\ t, & t', & t'' \end{smallmatrix} \right]\right) = \left[\begin{smallmatrix} a, & a', & a'' \\ t, & t', & t'' \end{smallmatrix} \right]$ in the reflected Gray code for radices (t, t', t'') . See exercise 4.1–10, and note also the mixed-radix law

$$m_1 \dots m_n - 1 - \left[\begin{smallmatrix} x_1, & \dots, & x_n \\ m_1, & \dots, & m_n \end{smallmatrix} \right] = \left[\begin{smallmatrix} m_1 - 1 - x_1, & \dots, & m_n - 1 - x_n \\ m_1, & \dots, & m_n \end{smallmatrix} \right].$$

In general, the reflected Gray code for radices (m_1, \dots, m_n) is $(0, \dots, m_1 - 1) \boxtimes \dots \boxtimes (0, \dots, m_n - 1)$. [*Information Processing Letters* **22** (1986), 201–205.]

86. Let Γ_{mn} be the reflected m -ary Gray path, which can be defined by $\Gamma_{m0} = \epsilon$ and

$$\Gamma_{m(n+1)} = (0, 1, \dots, m-1) \boxtimes \Gamma_{mn}, \quad n \geq 0.$$

This path runs from $(0, 0, \dots, 0)$ to $(m-1, 0, \dots, 0)$ when m is even. Consider the Gray path Π_{mn} defined by $\Pi_{m0} = \emptyset$ and

$$\Pi_{m(n+1)} = \begin{cases} (0, 1, \dots, m-1) \boxtimes \Pi_{mn}, m\Gamma_{(m+1)n}^R, & \text{if } m \text{ is odd;} \\ (0, 1, \dots, m) \boxtimes \Pi_{mn}, m\Gamma_{mn}^R, & \text{if } m \text{ is even.} \end{cases}$$

This path traverses all of the $(m+1)^n - m^n$ nonnegative integer n -tuples for which $\max(a_1, \dots, a_n) = m$, starting with $(0, \dots, 0, m)$ and ending with $(m, 0, \dots, 0)$. The desired infinite Gray path is $\Pi_{0n}, \Pi_{1n}^R, \Pi_{2n}, \Pi_{3n}^R, \dots$

87. This is impossible when n is odd, because the n -tuples with $\max(|a_1|, \dots, |a_n|) = 1$ include $\frac{1}{2}(3^n + 1)$ with odd parity and $\frac{1}{2}(3^n - 3)$ with even parity. When $n = 2$ we can use a spiral $\Sigma_0, \Sigma_1, \Sigma_2, \dots$, where Σ_m winds clockwise from $(m, 1-m)$ to $(m, -m)$ when $m > 0$. For even values of $n \geq 2$, if T_m is a path of n -tuples from $(m, 1-m, m-1, 1-m, \dots, m-1, 1-m)$ to $(m, -m, m, -m, \dots, m, -m)$, we can use $\Sigma_m \boxtimes (T_0, \dots, T_{m-1}), (\Sigma_0, \dots, \Sigma_m)^R \boxtimes T_m$ for $(n+2)$ -tuples with the same property, where \boxtimes is the dual operation

$$\Gamma \boxtimes \Gamma' = (\alpha_0 \alpha'_0, \dots, \alpha_{t-1} \alpha'_0, \alpha_{t-1} \alpha'_1, \dots, \alpha_0 \alpha'_1, \alpha_0 \alpha'_2, \dots, \alpha_{t-1} \alpha'_2, \alpha_{t-1} \alpha'_3, \dots).$$

[Infinite n -dimensional Gray paths *without* the magnitude constraint were first constructed by E. Vázsonyi, *Acta Litterarum ac Scientiarum, sectio Scientiarum Mathematicarum* **9** (Szeged: 1938), 163–173.]

88. It would visit all the subforests again, but in reverse order, ending with $(0, \dots, 0)$ and returning to the state it had after the initialization step K1. (This reflection principle is, in fact, the key to understanding how Algorithm K works.)

89. (a) Let $M_0 = \epsilon$, $M_1 = \bullet$, and $M_{n+2} = \bullet M_{n+1}^R, -M_n^R$. This construction works because the last element of M_{n+1}^R is the first element of M_{n+1} , namely a dot followed by the first element of M_n^R .

(b) Given a string $d_1 \dots d_l$ where each d_j is \bullet or $-$, we can find its successor by letting $k = l - [d_l = \bullet]$ and proceeding as follows: If k is odd and $d_k = \bullet$, change $d_k d_{k+1}$ to $-$; if k is even and $d_k = -$, change d_k to $\bullet\bullet$; otherwise decrease k by 1 and repeat until either making a change or reaching $k = 0$. The successor of the given word is $\bullet - - \bullet \bullet \bullet - - \bullet \bullet$.

90. A cycle can exist only when the number of code words is even, since the number of dashes changes by ± 1 at each step. Thus we must have $n \bmod 3 = 2$. The Gray paths M_n of exercise 89 are not suitable; they begin with $(\bullet -)^{\lfloor n/3 \rfloor} \bullet^{n \bmod 3}$ and end

with $(-\bullet)^{\lfloor n/3 \rfloor} \bullet^{[n \bmod 3=1]} -^{[n \bmod 3=2]}$. But $M_{3k+1}\bullet, M_{3k}^R-$ is a Hamiltonian circuit in the Morse code graph when $n = 3k + 2$.

91. Equivalently, the n -tuples $a_1\bar{a}_2a_3\bar{a}_4\dots$ have no two consecutive 1s. Such n -tuples correspond to Morse code sequences of length $n+1$, if we append 0 and then represent \bullet and $\bullet-$ respectively by 0 and 10. Under this correspondence we can convert the path M_{n+1} of exercise 89 into a procedure like Algorithm K, with the fringe containing the indices where each dot or dash begins (except for a final dot).

Q1. [Initialize.] Set $a_j \leftarrow \lfloor ((j-1) \bmod 6)/3 \rfloor$ and $f_j \leftarrow j$ for $1 \leq j \leq n$. Also set $f_0 \leftarrow 0, r_0 \leftarrow 1, l_1 \leftarrow 0, r_j \leftarrow j + (j \bmod 3)$ and $l_{j+(j \bmod 3)} \leftarrow j$ for $1 \leq j \leq n$, except if $j + (j \bmod 3) > n$ set $r_j \leftarrow 0$ and $l_0 \leftarrow j$. (The “fringe” now contains 1, 2, 4, 5, 7, 8, ...)

Q2. [Visit.] Visit the n -tuple (a_1, \dots, a_n) .

Q3. [Choose p .] Set $q \leftarrow l_0, p \leftarrow f_q, f_q \leftarrow q$.

Q4. [Check a_p .] Terminate the algorithm if $p = 0$. Otherwise set $a_p \leftarrow 1 - a_p$ and go to Q6 if $a_p + p$ is now even.

Q5. [Insert $p+1$.] If $p < n$, set $q \leftarrow r_p, l_q \leftarrow p+1, r_{p+1} \leftarrow q, r_p \leftarrow p+1, l_{p+1} \leftarrow p$. Go to Q7.

Q6. [Delete $p+1$.] If $p < n$, set $q \leftarrow r_{p+1}, r_p \leftarrow q, l_q \leftarrow p$.

Q7. [Make p passive.] Set $f_p \leftarrow f_{l_p}$ and $f_{l_p} \leftarrow l_p$. Return to Q2. ▀

This algorithm can also be derived as a special case of a considerably more general method due to Gang Li, Frank Ruskey, and D. E. Knuth, which extends Algorithm K by allowing the user to specify either $a_p \geq a_q$ or $a_p \leq a_q$ for each (parent, child) pair (p, q) . [See Knuth and Ruskey, “Deconstructing coroutines,” to appear.] A generalization in another direction, which produces all strings of length n that do not contain certain substrings, has been discovered by M. B. Squire, *Electronic J. Combinatorics* **3** (1996), #R17, 1–29.

92. Yes, because the digraph of all $(n-1)$ -tuples (x_1, \dots, x_{n-1}) with $x_1, \dots, x_{n-1} \leq m$ and with arcs $(x_1, \dots, x_{n-1}) \rightarrow (x_2, \dots, x_n)$ whenever $\max(x_1, \dots, x_n) = m$ is connected and balanced; see Theorem 2.3.4.2G. Indeed, we get such a sequence from Algorithm F if we note that the final k^n elements of the prime strings of length dividing n , when subtracted from $m-1$, are the same for all $m \geq k$. When $n=4$, for example, the first 81 digits of the sequence Φ_4 are $2 - \alpha^R = 00001010011\dots$, where α is the string (62). [There also are infinite m -ary sequences whose first m^n elements are de Bruijn cycles for all n , given any fixed $m \geq 3$. See L. J. Cummings and D. Wiedemann, *Cong. Numerantium* **53** (1986), 155–160.]

93. The cycle generated by $f()$ is a cyclic permutation of $\alpha 1$, where α has length $m^n - 1$ and ends with 1^{n-1} . The cycle generated by Algorithm R is a cyclic permutation of $\gamma = c_0 \dots c_{m^{n+1}-1}$, where $c_k = (c_0 + b_0 + \dots + b_{k-1}) \bmod m$ and $b_0 \dots b_{m^{n+1}-1} = \beta = \alpha^m 1^m$.

If $x_0 \dots x_n$ occurs in γ , say $x_j = c_{k+j}$ for $0 \leq j \leq n$, then $y_j = b_{k+j}$ for $0 \leq j < n$, where $y_j = (x_{j+1} - x_j) \bmod m$. [This is the connection with modular m -ary Gray code; see exercise 78.] Now if $y_0 \dots y_{n-1} = 1^n$ we have $m^{n+1} - m - n < k \leq m^{n+1} - n$; otherwise there is an index k' such that $-n < k' < m^n - n$ and $y_0 \dots y_{n-1}$ occurs in β at positions $k = (k' + r(m^n - 1)) \bmod m^{n+1}$ for $0 \leq r < m$. In both cases the m choices of k have different values of x_0 , because the sum of all elements in α is $m-1$

(modulo m) when $n \geq 2$. [Algorithm R is valid also for $n = 1$ if $m \bmod 4 \neq 2$, because $m \perp \sum \alpha$ in that case.]

94. 0010203041121314223243344. (The underlined digits are effectively inserted into the interleaving of 00112234 with 34. Algorithm D can be used in general when $n = 1$ and $r = m - 2 \geq 0$; but it is pointless to do so, in view of (54).)

95. (a) Let $c_0 c_1 c_2 \dots$ have period r . If r is odd we have $p = q = r$, so $r = pq$ only in the trivial case when $p = q = 1$ and $a_0 = b_0$. Otherwise $r/2 = \text{lcm}(p, q) = pq/\gcd(p, q)$ by 4.5.2-(10), hence $\gcd(p, q) = 2$. In the latter case the $2n$ -tuples $c_i c_{i+1} \dots c_{i+2n-1}$ that occur are $a_j b_k \dots a_{j+n-1} b_{k+n-1}$ for $0 \leq j < p$, $0 \leq k < q$, $j \equiv k$ (modulo 2), and $b_k a_j \dots b_{k+n-1} a_{j+n-1}$ for $0 \leq j < p$, $0 \leq k < q$, $j \not\equiv k$ (modulo 2).

(b) The output would interleave two sequences $a_0 a_1 \dots$ and $b_0 b_1 \dots$ whose periods are respectively $m^n + r$ and $m^n - r$; the a 's are the cycle of $f()$ with x^n changed to x^{n+1} and the b 's are the cycle of $g()$ with x^n changed to x^{n-1} , for $0 \leq x < r$. By (58) and part (a), the period length is $m^{2n} - r^2$, and every $2n$ -tuple occurs with the exception of $(xy)^n$ for $0 \leq x, y < r$.

(c) The real step D6 alters the behavior of (b) by going to D3 when $t \geq n$ and $0 \leq x' = x < r$; this emits an extra x at the time when x^{2n-1} has just been output and b is about to be emitted, where b is the digit following x^n in g 's cycle. D6 also allows control to pass to D7 and then D3 with $t' = n$ in the case that $t \geq n$ and $x < x' < r$; this emits an extra $x'x$ at the time when $(xx')^{n-1}x$ has just been output and b will be next. These r^2 extra bits provide the r^2 missing $2n$ -tuples of (b).

96. (a) The recurrences $S_2 = 1$, $S_{2n+1} = S_{2n} = 2S_n$, $R_2 = 0$, $R_{2n+1} = 1 + R_{2n}$, $R_{2n} = 2R_n$, $D_2 = 0$, $D_{2n+1} = D_{2n} = 1 + 2D_n$ have the solution $S_n = 2^{\lfloor \lg n \rfloor - 1}$, $R_n = n - 2S_n$, $D_n = S_n - 1$. Thus $S_n + R_n + D_n = n - 1$.

(b) Each top-level output usually involves $\lfloor \lg n \rfloor - 1$ D-activations and $\nu(n) - 1$ R-activations, plus one basic activation at the bottom level. But there are exceptions: Algorithm R might invoke its $f()$ twice, if the first activation completed a sequence 1^n ; and sometimes Algorithm R doesn't need to invoke $f()$ at all. Algorithm D might invoke its $g()$ twice, if the first activation completed a sequence $(x')^n$; but sometimes Algorithm D doesn't need to invoke either $f()$ or $g()$.

Algorithm R completes a sequence x^{n+1} if and only if its child $f()$ has just completed a sequence 0^n . Algorithm D completes a sequence x^{2n} for $x < r$ if and only if it has just jumped from D6 to D3 without invoking any child.

From these observations we can conclude that at most $\lfloor \lg n \rfloor + \nu(n) + 1$ activations are possible per top-level output, if $r > 1$; such a case happens when Algorithm D for $n = 6$ goes from D6 to D4. But when $r = 1$ we can have as many as $2\lfloor \lg n \rfloor + 3$ activations, for example when Algorithm R for $n = 25$ goes from R4 to R2.

97. (a) (0011), (00011101), (0000101001111011), and (00000110001011011111 001110101001). Thus $j_2 = 2$, $j_3 = 3$, $j_4 = 9$, $j_5 = 15$.

(b) We obviously have $f_{n+1}(k) = \Sigma f_n(k) \bmod 2$ for $0 \leq k < j_n + n$. The next value, $f_{n+1}(j_n + n)$, depends on whether step R4 jumps to R2 after computing $y = f_n(j_n + n - 1)$. If it does (namely, if $f_{n+1}(j_n + n - 1) \neq 0$), we have $f_{n+1}(k) \equiv 1 + \Sigma(k+1)$ for $j_n + n \leq k < 2^n + j_n + n$; otherwise we have $f_{n+1}(k) \equiv 1 + \Sigma(k - 1)$ for those values of k . In particular, $f_{n+1}(k) = 1$ when $2^n \leq k + \delta_n \leq 2^n + n$. The stated formula, which has simpler ranges for the index k , holds because $1 + \Sigma(k \pm 1) \equiv \Sigma(k)$ when $j_n < k < j_n + n$ or $2^n + j_n < k < 2^n + j_n + n$.

(c) The interleaved cycle has $c_n(2k) = f_n^+(k)$ and $c_n(2k + 1) = f_n^-(k)$, where

$$f_n^+(k) = \begin{cases} f_n(k-1), & \text{if } 0 < k \leq j_n + 1; \\ f_n(k-2), & \text{if } j_n + 1 < k \leq 2^n + 2; \end{cases} \quad f_n^-(k) = \begin{cases} f_n(k+1), & \text{if } 0 \leq k < j_n; \\ f_n(k+2), & \text{if } j_n \leq k < 2^n - 2; \end{cases}$$

$f_n^+(k) = f_n^+(k \bmod (2^n + 2))$, $f_n^-(k) = f_n^-(k \bmod (2^n - 2))$. Therefore the subsequence 1^{2n-1} begins at position $k_n = (2^{n-1} - 2)(2^n + 2) + 2j_n + 2$ in the c_n cycle; this will make j_{2n} odd. The subsequence $(01)^{n-1}0$ begins at position $l_n = (2^{n-1} + 1)(j_n - 1)$ if $j_n \bmod 4 = 1$, at $l_n = (2^{n-1} + 1)(2^n + j_n - 3)$ if $j_n \bmod 4 = 3$. Also $k_2 = 6$, $l_2 = 2$.

(d) Algorithm D inserts four elements into the c_n cycle; hence

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

(e) Consequently $j_{2n} = k_n + 1 + 2[j_n \bmod 4 < 3]$. Indeed, the elements preceding 1^{2n} consist of $2^{n-2} - 1$ complete periods of $f_n^+(\cdot)$ interleaved with 2^{n-2} complete periods of $f_n^-(\cdot)$, with one 0 inserted and also with 10 inserted if $l_n < k_n$, followed by $f_n(1)f_n(1)f_n(2)f_n(2)\dots f_n(j_n - 1)f_n(j_n - 1)$. The sum of all these elements is odd, unless $l_n < k_n$; therefore $\delta_{2n} = 1 - 2[j_n \bmod 4 = 3]$.

Let $n = 2^t q$, where q is odd and $n > 2$. The recurrences imply that, if $q = 1$, we have $j_n = 2^{n-1} + b_t$ where $b_t = 2^t/3 - (-1)^t/3$. And if $q > 1$ we have $j_n = 2^{n-1} \pm b_{t+2}$, where the + sign is chosen if and only if $\lfloor \lg q \rfloor + \lfloor [4q/2^{\lfloor \lg q \rfloor}] \rfloor = 5$ is even.

98. If $f(k) = g(k)$ when k lies in a certain range, there's a constant C such that $\Sigma f(k) = C + \Sigma g(k)$ for k in that range. We can therefore continue almost mindlessly to derive additional recurrences: If $n > 1$ we have

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

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

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

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

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

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

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

And then, aha, there is closure:

$$\Sigma \Sigma c_n(2k) = \Sigma f_n^+(k), \quad \Sigma \Sigma c_n(2k + 1) = \Sigma f_n^-(k).$$

If $n = 2^t q$ where q is odd, the running time to evaluate $f_n(k)$ by this system of recursive formulas is $O(t + S(q))$, where $S(1) = 1$, $S(2k) = 1 + 2S(k)$, and $S(2k+1) = 1 + S(k)$. Clearly $S(k) < 2k$, so the evaluations involve at most $O(n)$ simple operations on n -bit numbers. In fact, the method is often significantly faster: If we average $S(k)$ over all k with $\lfloor \lg k \rfloor = s$ we get $(3^{s+1} - 2^{s+1})/2^s$, which is less than $3k^{\lg(3/2)} < 3k^{0.59}$. (Incidentally, if $k = 2^{s+1} - 1 - (2^{s-e_1} + 2^{s-e_2} + \dots + 2^{s-e_t})$ we have $S(k) = s + 1 + e_t + 2e_{t-1} + 4e_{t-2} + \dots + 2^t e_1$.)

99. A string that starts at position k in $f_n()$ starts at position $k^+ = k + 1 + [k > j_n]$ in $f_n^+()$ and at position $k^- = k - 1 - [k > j_n]$ in $f_n^-()$, except that 0^n and 1^n occur twice in $f_n^+()$ but not at all in $f_n^-()$.

To find $\gamma = a_0 b_0 \dots a_{n-1} b_{n-1}$ in the cycle $f_{2n}()$, let $\alpha = a_0 \dots a_{n-1}$ and $\beta = b_0 \dots b_{n-1}$. Suppose α starts at position j and β at position k in $f_n()$, and assume that neither α nor β is 0^n or 1^n . If $j^+ \equiv k^+$ (modulo 2), let $l/2$ be a solution to the equation $j^+ + (2^n + 2)x = k^- + (2^n - 2)y$; we may take $l/2 = k + (2^n - 2)(2^{n-3}(j - k) \bmod (2^{n-1} + 1))$ if $j \geq k$, otherwise $l/2 = j + (2^n + 2)(2^{n-3}(k - j) \bmod (2^{n-1} - 1))$. Otherwise let $(l - 1)/2 = k^+ + (2^n + 2)x = j^- + (2^n - 2)y$. Then γ starts at position l in the cycle $c_n()$; hence it starts at position $l + 1 + [l \geq k_n] + 2[l \geq l_n]$ in the cycle $f_{2n}()$. Similar formulas hold when $\alpha \in \{0^n, 1^n\}$ or $\beta \in \{0^n, 1^n\}$ (but not both). Finally, 0^{2n} , 1^{2n} , $(01)^n$, and $(10)^n$ start respectively in positions 0, j_{2n} , $l_n + 1 + [k_n < l_n]$, and $l_n + 2 + [k_n < l_n]$.

To find $\beta = b_0 b_1 \dots b_n$ in $f_{n+1}()$ when n is even, suppose that the n -bit string $(b_0 \oplus b_1) \dots (b_{n-1} \oplus b_n)$ starts at position j in $f_n()$. Then β starts at position $k = j - \delta_n[j \geq j_n] + 2^n[j = j_n][\delta_n = 1]$ if $f_{n+1}(k) = b_0$, otherwise at position $k + (2^n - \delta_n, \delta_n, 2^n + \delta_n)$ according as $(j < j_n, j = j_n, j > j_n)$.

The running time of this recursion satisfies $T(n) = O(n) + 2T(\lfloor n/2 \rfloor)$, so it is $O(n \log n)$. [Exercises 97–99 are based on the work of J. Tuliani, who also has developed methods for certain larger values of m ; see *Discrete Math.* **226** (2001), 313–336.]

100. No obvious defects are apparent, but extensive testing should be done before any sequence can be recommended. By contrast, the de Bruijn cycle produced implicitly by Algorithm F is a terrible source of supposedly random bits, even though it is n -distributed in the sense of Definition 3.5D, because 0s predominate at the beginning. Indeed, when n is prime, bits $tn + 1$ of that sequence are zero for $0 \leq t < (2^n - 2)/n$.

101. (a) Let β be a proper suffix of $\lambda\lambda'$ with $\beta \leq \lambda\lambda'$. Either β is a suffix of λ' , whence $\lambda < \lambda' \leq \beta$, or $\beta = \alpha\lambda'$ and we have $\lambda < \alpha < \beta$.

Now $\lambda < \beta \leq \lambda\lambda'$ implies that $\beta = \lambda\gamma$ for some $\gamma \leq \lambda'$. But γ is a suffix of β with $1 \leq |\gamma| = |\beta| - |\lambda| < |\lambda'|$; hence γ is a proper suffix of λ' , and $\lambda' < \gamma$. Contradiction.

(b) Any string of length 1 is prime. Combine adjacent primes by (a), in any order, until no further combination is possible. [See the more general results of M. P. Schützenberger, *Proc. Amer. Math. Soc.* **16** (1965), 21–24.]

(c) If $t \neq 0$, let λ be the smallest suffix of $\lambda_1 \dots \lambda_t$. Then λ is prime by definition, and it has the form $\beta\gamma$ where β is a nonempty suffix of some λ_j . Therefore $\lambda_t \leq \lambda_j \leq \beta \leq \beta\gamma = \lambda \leq \lambda_t$, so we must have $\lambda = \lambda_t$. Remove λ_t and repeat until $t = 0$.

(d) True. For if we had $\alpha = \lambda\beta$ for some prime λ with $|\lambda| > |\lambda_1|$, we could append the factors of β to obtain another factorization of α .

(e) $3 \cdot 1415926535897932384626433832795 \cdot 02884197$. (Knowing more digits of π would not change the first two factors. The infinite decimal expansion of any number that is “normal” in the sense of Borel (see Section 3.5) factors into primes of finite length.)

102. We must have $1/(1 - mz) = 1/\prod_{n=1}^{\infty}(1 - z^n)^{L_m(n)}$. This implies (6o) as in exercise 4.6.2–4.

103. When $n = p$ is prime, (59) tells us that $L_m(1) + pL_m(p) = m^p$, and we also have $L_m(1) = m$. [This combinatorial proof provides an interesting contrast to the traditional algebraic proof of Theorem 1.2.4F.]

104. The 4483 nonprimes are **abaca**, **agora**, **ahead**, ...; the 1274 primes are ..., **rusts**, **rusty**, **rutty**. (Since **prime** isn't prime, we should perhaps call prime strings **lowly**.)

105. (a) Let α' be α with its last letter increased, and suppose $\alpha' = \beta\gamma'$ where $\alpha = \beta\gamma$ and $\beta \neq \epsilon$, $\gamma \neq \epsilon$. Let θ be the prefix of α with $|\theta| = |\gamma|$. By hypothesis there is a string ω such that $\alpha\omega$ is prime; hence $\theta \leq \alpha\omega < \gamma\omega$, so we must have $\theta \leq \gamma$. Consequently $\theta < \gamma'$, and we have $\alpha' < \gamma'$.

(b) Let $\alpha = \lambda_1\beta = a_1\dots a_n$ where $\lambda_1\beta\omega$ is prime. The condition $\lambda_1\beta\omega < \beta\omega$ implies that $a_j \leq a_{j+r}$ for $1 \leq j \leq n-r$, where $r = |\lambda_1|$. But we cannot have $a_j < a_{j+r}$; otherwise α would begin with a prime longer than λ_1 , contradicting exercise 101(d).

(c) If α is the n -extension of both λ and λ' , where $|\lambda| > |\lambda'|$, we must have $\lambda = (\lambda')^q\theta$ where θ is a proper prefix of λ' . But then $\theta < \lambda' < \lambda < \theta$.

106. **B1.** [Initialize.] Set $a_1 \leftarrow \dots \leftarrow a_n \leftarrow m-1$, $a_{n+1} \leftarrow -1$, and $j \leftarrow 1$.

B2. [Visit.] Visit (a_1, \dots, a_n) with index j .

B3. [Subtract one.] Terminate if $a_j = 0$. Otherwise set $a_j \leftarrow a_j - 1$, and $a_k \leftarrow m-1$ for $j < k \leq n$.

B4. [Prepare to factor.] (According to exercise 105(b), we now want to find the first prime factor λ_1 of $a_1 \dots a_n$.) Set $j \leftarrow 1$ and $k \leftarrow 2$.

B5. [Find the new j .] (Now $a_1 \dots a_{k-1}$ is the $(k-1)$ -extension of the prime $a_1 \dots a_j$.) If $a_{k-j} > a_k$, return to B2. Otherwise, if $a_{k-j} < a_k$, set $j \leftarrow k$. Then increase k by 1 and repeat this step. ■

The efficient factoring algorithm in steps B4 and B5 is due to J. P. Duval, *J. Algorithms* **4** (1983), 363–381. For further information, see Cattell, Ruskey, Sawada, Serra, and Miers, *J. Algorithms* **37** (2000), 267–282.

107. The number of n -tuples visited is $P_m(n) = \sum_{j=1}^n L_m(j)$. Since $L_m(n) = \frac{1}{n}m^n + O(m^{n/2}/n)$, we have $P_m(n) = Q(m, n) + O(Q(\sqrt{m}, n))$, where

$$\begin{aligned} Q(m, n) &= \sum_{k=1}^n \frac{m^k}{k} = \frac{m^n}{n} R(m, n); \\ R(m, n) &= \sum_{k=0}^{n-1} \frac{m^{-k}}{1-k/n} = \sum_{k=0}^{n/2} \frac{m^{-k}}{1-k/n} + O(nm^{-n/2}) \\ &= \frac{m}{m-1} \sum_{j=0}^{t-1} \frac{1}{n^j} \sum_l \binom{j}{l} \frac{m^l}{(m-1)^j} + O(n^{-t}). \end{aligned}$$

The main contributions to the running time come from the loops in steps F3 and F5, which cost $n-j$ for each prime of length j , hence a total of $nP(n) - \sum_{j=1}^n jL_m(j) = m^{n+1}/((m-1)^2 n) + O(1/(mn^2))$. This is less than the time needed to output the m^n individual digits of the de Bruijn cycle.

108. (a) If $\alpha \neq 9\dots 9$, we have $\lambda_{k+1} \leq \beta 9^{|\alpha|}$, because the latter is prime.

(b) We can assume that β is not all 0s, since $9^j 0^{n-j}$ is a substring of $\lambda_{t-1} \lambda_t \lambda_1 \lambda_2 = 89^n 0^n 1$. Let k be maximal with $\beta \leq \lambda_k$; then $\lambda_k \leq \beta\alpha$, so β is a prefix of λ_k . Since β is a preprime, it is the $|\beta|$ -extension of some prime $\beta' \leq \beta$. The preprime visited by Algorithm F just before β' is $(\beta' - 1)9^{n-|\beta'|}$, by exercise 106, where $\beta' - 1$ denotes the decimal number that is one less than β' . Thus, if β' is not λ_{k-1} , the hint (which also follows from exercise 106) implies that λ_{k-1} ends with at least $n - |\beta'| \geq n - |\beta|$ 9s, and α is a suffix of λ_{k-1} . On the other hand if $\beta' = \lambda_{k-1}$, α is a suffix of λ_{k-2} , and β is a prefix of $\lambda_{k-1} \lambda_k$.

(c) If $\alpha \neq 9\dots 9$, we have $\lambda_{k+1} \leq (\beta\alpha)^{d-1} \beta 9^{|\alpha|}$, because the latter is prime. Otherwise λ_{k-1} ends with at least $(d-1)|\beta\alpha|$ 9s, and $\lambda_{k+1} \leq (\beta\alpha)^{d-1} 9^{|\beta\alpha|}$, so $(\alpha\beta)^d$ is a substring of $\lambda_{k-1} \lambda_k \lambda_{k+1}$.

(d) Within the primes 135899135914, 787899787979, 12999913131314, 09090911, 089999090911, 089999119119122.

[In all cases, the position of $a_1 \dots a_n$ precedes the position of $a_1 \dots a_{n-1}(a_n + 1)$, if $0 \leq a_n < 9$ (and if we assume that strings like $9^j 0^{n-j}$ occur at the beginning). Furthermore $9^j 0^{n-j-1}$ occurs only after $9^{j-1} 0^{n-j} a$ has appeared for $1 \leq a \leq 9$, so we must not place 0 after $9^{j-1} 0^{n-j-1}$. Therefore no m -ary de Bruijn cycle of length m^n can be lexicographically smaller than $\lambda_1 \dots \lambda_t$.]

109. Suppose we want to locate the submatrix

$$\begin{pmatrix} (w_{n-1} \dots w_1 w_0)_2 & (x_{n-1} \dots x_1 x_0)_2 \\ (y_{n-1} \dots y_1 y_0)_2 & (z_{n-1} \dots z_1 z_0)_2 \end{pmatrix}.$$

The binary case $n = 1$ is the given example, and if $n > 1$ we can assume by induction that we only need to determine the leading bits a_{2n-1} , a_{2n-2} , b_{2n-1} , and b_{2n-2} . The case $n = 3$ is typical: We must solve

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

here $b'_5 = b_5 \oplus b_4 b_3 b_2 b_1$ takes account of carrying when j becomes $j + 1$.

110. Let $a_0 a_1 \dots a_{m^2-1}$ be an m -ary de Bruijn cycle, such as the first m^2 elements of (54). If m is odd, let $a_{ij} = a_j$ when i is even, $a_{ij} = a_{(j+(i-1)/2) \bmod m^2}$ when i is odd. [The first of many people to discover this construction seems to have been John C. Cock, who also constructed de Bruijn toruses of other shapes and sizes in *Discrete Math.* **70** (1988), 209–210.]

If $m = m'm''$ where $m' \perp m''$, we use the Chinese remainder theorem to define

$$a_{ij} \equiv a'_{ij} \pmod{m'} \quad \text{and} \quad a_{ij} \equiv a''_{ij} \pmod{m''}$$

in terms of matrices that solve the problem for m' and m'' . Thus the previous exercise leads to a solution for arbitrary m .

Another interesting solution for even values of m was found by Zoltán Tóth [*2nd Conf. Automata, Languages, and Programming Systems* (1988), 165–172; see also Hurlbert and Isaak, *Contemp. Math.* **178** (1994), 153–160]. The first m^2 elements a_j of the infinite sequence

0011 021331203223 04152435534251405445 0617263746577564 ... 0766708 ...

define a de Bruijn cycle with the property that the distance between the appearances of ab and ba is always even. Then we can let $a_{ij} = a_j$ if $i + j$ is even, $a_{ij} = a_i$ if $i + j$ is odd. For example, when $m = 4$ we have

$$\left(\begin{array}{c} 0010021220302232 \\ 0001020320212223 \\ 0111031321312333 \\ 1011121330313233 \\ 0010021220302232 \\ 0203000122232021 \\ 0111031321312333 \\ 1213101132333031 \\ 0010021220302232 \\ 2021222300010203 \\ 0111031321312333 \\ 30313233310111213 \\ 0010021220302232 \\ 2223202102030001 \\ 0111031321312333 \\ 3233303112131011 \end{array} \right) \text{ (exercise 109); } \left(\begin{array}{c} 0010001030203020 \\ 0001020301000203 \\ 0111011131213121 \\ 1011121311101213 \\ 0010001030203020 \\ 2021222321202223 \\ 0111011131213121 \\ 3031323331303233 \\ 0313031333233323 \\ 1011121311101213 \\ 0212021232223222 \\ 0001020301000203 \\ 0313031333233323 \\ 2021222321202223 \\ 0212021232223222 \\ 3031323331303233 \end{array} \right) \text{ (Tóth).}$$

111. (a) Let $d_j = j$ and $0 \leq a_j < 3$ for $1 \leq j \leq 9$, $a_9 \neq 0$. Form sequences s_j, t_j by the rules $s_1 = 0$, $t_1 = d_1$; $t_{j+1} = d_{j+1} + 10t_j[a_j = 0]$ for $1 \leq j < 9$; $s_{j+1} = s_j + (0, t_j, -t_j)$ for $a_j = (0, 1, 2)$ and $1 \leq j \leq 9$. Then s_{10} is a possible result; we need only remember the smallish values that occur. More than half the work is saved by disallowing $a_k = 2$ when $s_k = 0$, then using $|s_{10}|$ instead of s_{10} . Since fewer than $3^8 = 6561$ possibilities need to be tried, brute force via the ternary version of Algorithm M works well; fewer than 24,000 mems and 1600 multiplications are needed to deduce that all integers less than 211 are representable, but 211 is not.

Another approach, using Gray code to vary the signs after breaking the digits into blocks in 2^8 possible ways, reduces the number of multiplications to 255, but at the cost of about 500 additional mems. Therefore Gray code is not advantageous in this application.

(b) Now (with 73,000 mems and 4900 multiplications) we can reach all numbers less than 241, but not 241. There are 46 ways to represent 100, including the remarkable $9 - 87 + 6 + 5 - 43 + 210$.

[H. E. Dudeney introduced his “century” problem in *The Weekly Dispatch* (4 and 18 June 1899); see also *The Numerology of Dr. Matrix* by Martin Gardner, Chapter 6.]

112. The method of exercise 111 now needs more than 167 million mems and 10 million multiplications, because 3^{16} is so much larger than 3^8 . We can do much better (10.4 million mems, 1100 mults) by first tabulating the possibilities obtainable from the first k and last k digits, for $1 \leq k < 9$, then considering all blocks of digits that use the 9. There are 60,318 ways to represent 100, and the first unreachable number is 16,040.

INDEX AND GLOSSARY

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

- 2-adic numbers, 31.
4-cube, 42, 54.
8-cube, 17, 35.
 $\nu(k)$, *see* Lee weight, Sideways sum.
 π (circle ratio), 43, 60.
 $\rho(k)$, *see* Ruler function.
- Almost-linear recurrence, 23.
Analog-to-digital conversion, 3–4, 15.
Analysis of algorithms, 28, 37, 38.
Anti-Gray code, 35.
Antipodal words, 11.
Arima, Yoriyuki (有馬頬箇), 41.
Arndt, Jörg, 45.
Artificial intelligence, 43.
Aubert, Jacques, 54.
Automorphisms, 49.
- Balanced Gray code, 14–17, 35, 49.
Bandwidth of n -cube, 35.
baud: One transmission unit (e.g., one bit) per second, 4.
Baudot, Jean Maurice Émile, 4–5.
Beckett, Samuel Barclay, 34–35.
Bennett, William Ralph, 4.
Bernstein, Arthur Jay, 44.
Binary Gray codes, 12–17, 33–35.
Binary number system, 1, 4.
Binary recurrences, 43, 59.
Binary trie, 30.
Bit reversal, 28, 31.
Bitwise operations, 4, 11–12, 32, 45.
Borel, Émile Félix Édouard Justin, 60.
Borrow, 40.
Botermans, Jacobus (= Jack) Petrus Hermana, 55.
Boustrophedon product, 36, 56.
Bruijn, Nicolaas Govert de, 22.
 cycles, 22–27, 36–38, 62.
 toruses, 38.
Buchner, Morgan Mallory, Jr., 44.
Calderbank, Arthur Robert, 43.
Canoe puzzle, 55.
Canonical delta sequence, 13, 49.
Cardano, Girolamo (= Hieronymus Cardanus), 41.
Carry, 2, 62.
Castown, Rudolph W., 11.
Cattell, Kevin Michael, 61.
Cavior, Stephan Robert, 44.
Cayley, Arthur, Hamilton theorem, 45.
Center of gravity, 17.
- Characteristic polynomial, 45.
Chen, Kuo-Tsai (陳國才), 26.
Chinese remainder theorem, 62.
Chinese ring puzzle, 5–6, 28, 41–42.
Cheng, Ching-Shui (鄭清水), 54.
Cock, John Crowle, 62.
Cohn, Martin, 49, 51, 54.
Complementary Gray codes, 13, 16–17,
 33, 49.
Compositions, 28–29.
Concatenation, 25, 35, 49.
Concurrent computing, 43.
Connected components, 34.
Cooke, Raymond Mark, 51.
Coordinates, 13.
Coroutines, recursive, 24–25.
Cremer, William Henry, Jr., 55.
Cube, *see* n -cube.
Cube-connected computers, 43.
Cummings, Larry Jean, 57.
Cycle leaders, 31.
Cyclic shifts, 26.
- Dally, William James, 43.
de Bruijn, Nicolaas Govert, 22.
 cycles, 22–27, 36–38, 62.
 toruses, 38.
Decimal number system, 2, 18–19, 39.
Delta sequence, 13.
Dilation of embedded graph, 35.
Discrete Fourier transform, 9, 27, 47.
Divisors of a number, 35.
Doubly linked list, 21, 56–57.
Douglas, Robert James, 48.
Dual boustrophedon product, 56.
Dudeney, Henry Ernest, 5, 63.
Duval, Jean Pierre, 61.
Dyckman, Howard Lloyd, 36, 55.
- Edge covering, 35.
Ehrlich, Gideon (גדיין אַרְלִיךְ), 9.
Enumeration, 1.
Error-correcting codes, 30.
Etzion, Tuvi (טוֹבִי הָוְלָצֵר, טובי עציוני), born 25.
Extension, 26.
- Factorization of strings, 37.
 algorithm for, 61.
Faloutsos, Christos (Φαλούτσος, Χρήστος), 43.
Fast Fourier transform, 28.
Fast Walsh transform, 32.
Fermat, Pierre de, theorem, 38.
Fibonacci, Leonardo, of Pisa, numbers, 36.
Field, finite, 32.

- Five-letter words, 11, 32–33, 38.
 Flores, Ivan, 54.
 Focus pointers, 10–11, 20–21, 56–57.
 Forest, 20–21.
 Fourier, Jean Baptiste Joseph, series, 7.
 transform, discrete, 9, 28, 47.
 Fox, Ralph Hartzler, 26.
 Fredman, Michael Lawrence, 33, 47.
 Fredricksen, Harold Marvin, 26, 27.
 Fringe, 21, 56–57.
 Gardner, Martin, 55, 63.
 Generating functions, 61.
 Generation, 1.
 constant amortized time, 40.
 loopless, 9–12, 20, 28, 29, 36, 42.
 Gilbert, Edgar Nelson, 33.
 Gilbert, William Schwenck, 1.
 Goddyn de la Vega, Luis Armando, 34, 50.
 Gomes, Peter John, iii.
 Gordian Knot puzzle, 35.
 Gray, Elisha, 5.
 Gray, Frank, 4.
 Gray binary code, 2–12, 16, 28–33, 36.
 permutation, 3, 31.
 Gray binary trie, 30.
 Gray code: A cycle of adjacent objects, 12, 20.
 Gray code for n -tuples, 12.
 advantages of, 6, 11–12.
 binary, *see* Gray binary code.
 limitations of, 40, 63.
 nonbinary, 18–20, 35–36.
 Gray fields, 31.
 Gray path: A sequence of adjacent objects, 15, 20.
 Gray stream, 34.
 Gray ternary code, 19, 36.
 Gros, Luc Agathon Louis, 5.
 Gvozdljak, Pavol, 34.
 Hadamard, Jacques Salomon, 47.
 transform, 9, 32, 46, 47.
 Hamilton, William Rowan, *see* Cayley.
 circuit, 13, 34.
 path, 15.
 Hamley, William, and sons, 55.
 Hammons, Arthur Roger, Jr., 43.
 Harmuth, Henning Friedolf, 7.
 Hexadecimal puzzle, 42.
 Hopcroft, John Edward, 44.
 Hurlbert, Glenn Howland, 60.
 in situ permutation, 28, 31.
 in situ transformation, 9.
 Inclusion and exclusion principle, 6.
 Inline expansion, 11–12.
 Interleaving, 37, 50, 62.
 Internet, ii, iii.
- Inverse function, 4, 31.
 Isaak, Garth Timothy, 62.
 Isomorphic Gray codes, 33–34.
 Iteration of functions, 32, 45.
 Japanese mathematics, 41.
 Karnaugh, Maurice, 29.
 Kedlaya, Kiran Sridhara, 49.
 Keister, William, 42.
 Kiefer, Jack Carl, 54.
 Knuth, Donald Ervin (高德纳), i, iv, 57.
 Koda, Yasunori (黄田保憲), 20–21.
 Kronecker, Leopold, product, 46.
 Kumar, Panganamala Vijay (కుమార్ విజయ), 43.
 Larrivee, Jules Alphonse, 6.
 Lawrence, George Melvin, 15, 50.
 Lee, Chester C. Y., 42.
 distance, 29.
 weight, 29.
 Lempel, Abraham (לֶמְפל מַרְבָּאָן), 25.
 Lexicographic order, 2–3, 25, 29, 47.
 Li, Gang (= Kenny) (李钢), 57.
 Lieves, 30.
 Linked allocation, 28.
 Listing, 1.
 Loony Loop, 35–36.
 Loopless generation, 9–12, 20, 28, 29, 36, 42.
 Luke, Saint (Άγιος Λουκᾶς ὁ Εὐαγγελιστής), 40.
 Lyndon, Roger Conant, 26.
 words, 26.
 m -ary digit: An integer between 0 and $m - 1$, inclusive, 2, 22.
 Macro-processor, 11.
 Maiorana, James Anthony, 26, 27.
 Mantel, Willem, 23.
 Martin, Monroe Harnish, 27–28.
 Matching, 33.
 Matrix (Bush), Irving Joshua, 63.
 McClintock, William Edward, 15.
 Median, 31.
 Miers, Charles Robert, 61.
 Military sayings, 1.
 Misra, Jayadev (ଜୟାଦେବ ମିଶ୍ର), 41.
 Mitchell, Christopher John, 25.
 Mixed-radix number system, 2, 19–21,
 35, 54, 55.
 MMIX, 40.
 Modular Gray codes, 19–20, 35, 54.
 decimal, 19.
 m -ary, 24, 54, 57.
 quaternary, 42, 49.
 ternary, 46, 51.
 Mollard, Michel, 48.
 Monic polynomial, 42.
 Monotonic binary Gray path, 15–18, 35.
 Morse, Samuel Finley Breese, code, 36, 57.

- Moser, Leo, 48.
 Multinomial coefficient, 29.
n-cube: The graph of *n*-bit strings,
 adjacent when they differ in only one
 position, 13, 15, 33–34.
 subcubes of, 30–31.
n-distributed sequence, 60.
n-extension, 26.
n-tuple: a sequence or string of
 length *n*, 1–2.
 Nemeth, Evelyn (= Evi) Hollister Pratt, 50.
 Neyman, Jerzy, 54.
 Nonbinary Gray codes, 18–20, 35–36.
 Nonlocal Gray codes, 16–17, 34.
 Nordstrom, Alan Wayne, 30.
 Normal numbers, 60.
 Novra, Henry, 55.
 Octacode, 30.
 Orthogonal vectors, 8, 32.
 Ouroboruses, 38–39.
 Paley, Raymond Edward Alan Christopher,
 45.
 functions, 32.
 Parity bit, 6, 28, 29.
 Paterson, Kenneth Graham, 25.
 Perverse, Peter Quentin, 35.
 Pi (π), 43, 60.
 Prefix of a string, 25.
 Prepostorder, 42.
 Preprime string, 26–28, 37.
 Prime string, 25–28, 37.
 factorization, 37, 61.
 Primitive polynomial modulo *p*, 23, 45.
 Principal subforest, 20–21.
 Proper prefix or suffix, 25.
 Pseudorandom bits, 37.
 Pulse code modulation, 4.
 Purkiss, Henry John, 28.
 Quaternary *n*-tuples, 29, 49.
 R&D method, 25, 37.
 Rademacher, Hans, 8.
 functions, 8, 32, 46.
 Ramras, Mark Bernard, 50.
 Random number generation, 37.
 Reflected Gray codes, 19–21, 35, 54, 55.
 decimal, 19.
 ternary, 36.
 Richards, Dana Scott, 36.
 Right subcube, 30.
 Ringel, Gerhard, 35.
 Ritchie, Alistair English, 42.
 Robinson, John Paul, 30, 49, 51.
 Rosenbaum, Joseph, 54.
 Ruler function, 6, 8, 12, 13, 47.
 decimal, 19.
 Run lengths, 15–17, 34, 50.
 Ruskey, Frank, 20, 21, 28, 31, 33, 57, 61.
 Salzer, Herbert Ellis, 44.
 Sampson, John Laurence, 33, 48.
 Savage, Carla Diane, 17–18, 28, 33, 35, 48.
 Sawada, Joseph James, 61.
 Schäffler, Otto, 5.
 Schneider, Bernadette, 54.
 Schützenberger, Marcel Paul, 60.
 Sequency, 7.
 Serra, Micaela, 61.
 Shapiro, Harold Seymour, 33.
 Shift register sequences, 22–28, 36–38.
 Sideways sum, 15, 44.
 Silverman, Jerry, 33, 48, 49.
 Sloane, Neil James Alexander, 43.
 Slocum, Gerald Kenneth (= Jerry), 55.
 Solé, Patrick, 43.
 SpinOut puzzle, 42.
 Squire, Matthew Blaze, 57.
 Stahnke, Wayne Lee, 23.
 Standard sequences, 26.
 Stanford GraphBase, ii, iii, 11, 32–33, 38.
 Steiglitz, Kenneth, 44.
 Stevens, Brett, 34.
 Stewart, Ian Nicholas, 38.
 Stibitz, George Robert, 4, 6.
 Subcubes, 30–31.
 Subforests, 20–21, 36.
 Subsets, 1, 6.
 Suffix of a string, 25.
 Sums of squares, 32.
 Sylvester, James Joseph, 32, 47.
 Tangle puzzle, *see* Loony Loop.
 Taylor, Lloyd William, 5.
 Telephone, 5.
 Television, 4.
 Ternary *n*-tuples, 19, 26–27, 34, 36,
 45, 50, 63.
 Tiring irons, 5.
 Tootill, Geoffrey Colin, 14, 41.
 Torture test, 35.
 Torus, 29, 38, 42.
 Tóth, Zoltán, 62.
 Transition counts, 14, 33.
 Traversal, 1.
 Trend-free Gray path, 16–17, 35.
 Trie, 30.
 Tuliani, Jonathan R., 60.
 Tuple: A sequence containing a given
 number of elements.
 Up-down sequence, 36.
 Vázsonyi, Endre, 56.
 Vickers, Virgil Eugene, 33, 48, 49.
 Visitation, 1.

- Wallis, John, 6, 41.
Walsh, Joseph Leonard, 7, 8, 45.
 functions, 7–9, 32.
 transform, 8–9, 32.
Wang, Terry Min Yih (王珉懿), 28.
Washburn, Seth Harwood, 42.
Weight enumeration, 42.
- Wiedemann, Douglas Henry, 57.
Winker, Steven Karl, 48.
Winkler, Peter Mann, 17–18, 34, 48.
Wrapping around, 19, 29, 38.
- Yates, Frank, 9.
Yuen, Chung Kwong (阮宗光), 44.

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.

Copyright © 2002 by Addison-Wesley

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

Zeroth printing (revision 5), 30 March 2002

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. Those 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 Prefascicle 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 111 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 108 and 111; I’ve also implicitly posed additional unsolved questions in the answers to exercises 28, 58, 63, 67, 88, 99, 105, and 111. 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, 75, 88, 98, and/or 103.

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:—)

Happy reading!

*Stanford, California
31 December 2001*

D. E. K.

*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 $k + 1 \rightarrow k$ and examine the digits of k for undesirable equalities.

This gives us the permutations in dictionary order too!

On second sober thought ... we do need to think of something else.

— D. H. LEHMER (1957)

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

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

$$a_1 \leq a_2 \leq \dots \leq a_n, \tag{1}$$

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

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

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

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

- L2.** [Find j .] Set $j \leftarrow n - 1$. If $a_j \geq a_{j+1}$, decrease j by 1 repeatedly until $a_j < a_{j+1}$. Terminate the algorithm if $j = 0$. (At this point j is the largest subscript such that we have already visited all permutations beginning with $a_1 \dots a_j$. 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 $a_{j+1} \geq \dots \geq a_n$, element a_l is the smallest element greater than a_j that can legitimately follow $a_1 \dots a_{j-1}$ in a permutation. Before the interchange we had $a_{j+1} \geq \dots \geq a_{l-1} \geq a_l > a_j \geq a_{l+1} \geq \dots \geq a_n$; after the interchange, we have $a_{j+1} \geq \dots \geq a_{l-1} \geq a_j > a_l \geq a_{l+1} \geq \dots \geq a_n$.)
- L4.** [Reverse $a_{j+1} \dots a_n$.] 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 at least to the 18th century, 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 $a_1 \dots a_n$ 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 $a_1 \dots a_j$ 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 $a_{j+1} \geq \dots \geq a_n$ at the beginning of step L4. Therefore the first permutation beginning with the current prefix $a_1 \dots a_j$ is $a_1 \dots a_j a_n \dots a_{j+1}$, 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 $\{a_1, a_2, \dots, a_n\}$.

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 $a_1 \dots a_n$ each time.

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

$$1122 — 1212 \begin{array}{c} \swarrow \\[-1ex] \searrow \end{array}^{2112} 2121 — 2211; \quad (2)$$

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 circuit analogous to Gray binary code.

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

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

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: $(1234, 1243, 1423, 4123, 4132, 1432, 1342, 1324, 3124, 3142, \dots, 2143, 2134)$.

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 $c_1 \dots c_n$ of exercise 5.1.1–7, where c_j is the number of elements lying to the right of j that are less than j , we find that the permutations in (3) have the following inversion tables:

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

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 $a_1a_2\dots a_n$ of n distinct elements, this algorithm generates all of their permutations by repeatedly interchanging adjacent pairs. It uses an auxiliary array $c_1c_2\dots c_n$, which represents inversions as described above, running through all sequences of integers such that

$$0 \leq c_j < j \quad \text{for } 1 \leq j \leq n. \quad (5)$$

Another array $d_1d_2\dots d_n$ governs the directions by which the entries c_j change.

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

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

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 $q \leftarrow c_j + d_j$. 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 $d_j \leftarrow -d_j$, $j \leftarrow j - 1$, and go back to P4. ▀

This procedure, which clearly works for all $n \geq 1$, 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 (1668)

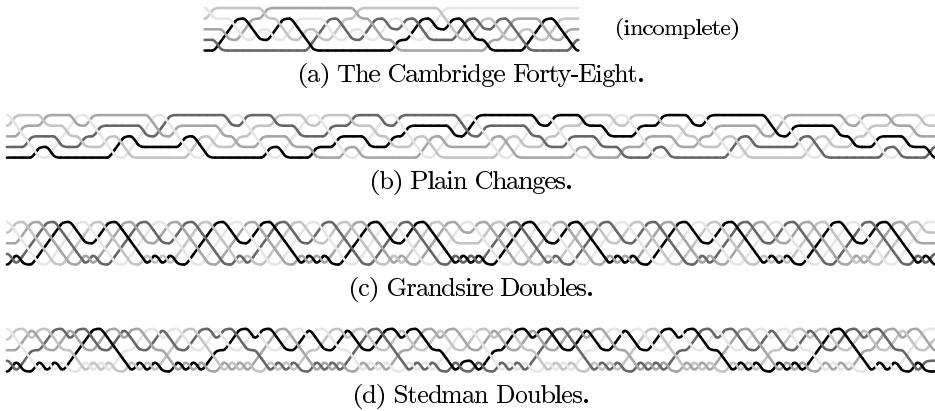


Fig. 18. Four patterns used to ring church-bells in 17th-century England. Pattern (b) corresponds to Algorithm P.

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 d tables make it easy to keep track of the current total number of inversions, $c_1 + \dots + c_n$, 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 $t[1], t[2], \dots, t[n! - 1]$ such that the actions of Algorithm P are equivalent to the successive interchanges $a_{t[k]} \leftrightarrow a_{t[k]+1}$ for $1 \leq k < n!$. We assume that $n \geq 2$.

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 $(t[1], t[2], \dots, t[23]) = (3, 2, 1, 3, 1, 2, 3, 1, 3, 2, 1, 3, 1, 2, 3, 1, 3, 2, 1, 3, 1, 2, 3)$.

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

$$\begin{array}{r} \text{SEND} \\ + \text{MORE} \\ \hline \text{MONEY} \end{array} \quad (6)$$

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

$$\text{VIOLIN} + \text{VIOLIN} + \text{VIOLA} = \text{TRIO} + \text{SONATA}.$$

Equivalently, we want to solve puzzles such as

$$2(\text{VIOLIN}) + \text{VIOLA} - \text{TRIO} - \text{SONATA} = 0, \quad (7)$$

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 $(1, 2, \dots, 10)$ to the letters (V, I, O, L, N, A, T, R, S, X), the respective signatures corresponding to (7) are

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

The problem then is to find all permutations $a_1 \dots a_{10}$ of $\{0, 1, \dots, 9\}$ such that

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

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

$$a_j \neq 0 \quad \text{for all } j \in F; \quad (10)$$

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 $t[j]$. Then, for each problem defined by a signature sequence (s_1, \dots, s_{10}) and a first-letter set F , we can exhaustively look for solutions as follows:

- A1.** [Initialize.] Set $a_1 a_2 \dots a_{10} \leftarrow 01 \dots 9$, $v \leftarrow \sum_{j=1}^{10} (j-1)s_j$, $k \leftarrow 1$, and $\delta_j \leftarrow s_{j+1} - s_j$ for $1 \leq j < 10$.
- A2.** [Test.] If $v = 0$ and if (10) holds, output the solution $a_1 \dots a_{10}$.
- 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 $s_{m+1} = \dots = s_{10} = 0$, and each of its solutions will be found $(10 - m)!$ times unless we insist that, say, $a_{m+1} < \dots < a_{10}$.

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 $a_0a_1\dots a_{n-1}$ for permutations of $\{0, 1, \dots, n-1\}$ instead of writing $a_1a_2\dots a_n$ for permutations of $\{1, 2, \dots, n\}$. More importantly, we will consider schemes for generating permutations in which most of the action takes place at the *left*, so that all permutations of $\{0, 1, \dots, k-1\}$ will be generated during the first $k!$ steps, for $1 \leq k \leq n$. For example, one such scheme for $n = 4$ is

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

this is called “reverse colex order,” because if we reflect the strings from right to left we get 3210, 3201, 3120, ..., 0123, the reverse of lexicographic order. Another way to think of (11) is to view the entries as $(n-a_n)\dots(n-a_2)(n-a_1)$, where $a_1a_2\dots a_n$ runs lexicographically through the permutations of $\{1, 2, \dots, n\}$.

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

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

or in the more compact cycle form

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

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

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

or in the cycle form

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

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

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

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

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

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

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

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

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

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

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

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

is a permutation of G that fixes $n-1$. The set G' of all such permutations is a group, and S_1, \dots, S_{n-2} is a Sims table for G' ; therefore the result follows by induction on n . ■

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

$$\begin{aligned}
 S_f &= \{(), (01)(23)(45)(67)(89)(ab)(cd)(ef), \dots, \\
 &\quad (0f)(1e)(2d)(3c)(4b)(5a)(69)(78)\}; \\
 S_e &= \{(), (12)(56)(9a)(de), (14)(36)(9c)(be), (18)(3a)(5c)(7e)\}; \\
 S_d &= \{(), (24)(35)(ac)(bd), (28)(39)(6c)(7d)\}; \\
 S_c &= \{()\}; \\
 S_b &= \{(), (48)(59)(6a)(7b)\}; \\
 S_a &= S_9 = \dots = S_1 = \{()\};
 \end{aligned} \tag{14}$$

here S_f contains 16 permutations σ_{fj} for $0 \leq j \leq 15$, which respectively take $i \mapsto i \oplus (15 - j)$ for $0 \leq i \leq 15$. The set S_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_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 $\sigma_f, \sigma_e, \sigma_d, \sigma_c$, and σ_b .

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

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

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

Using the general framework. Our chief concern is the group of *all* permutations on $\{0, 1, \dots, n - 1\}$, and in this case every set S_k of a Sims table will contain $k + 1$ elements $\{\sigma(k, 0), \sigma(k, 1), \dots, \sigma(k, k)\}$, where $\sigma(k, 0)$ is the identity and the others take k to the values $\{0, \dots, k - 1\}$ in some order (fixing all elements greater than k). Every such Sims table leads to a permutation generator, according to the following outline:

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

- G1.** [Initialize.] Set $a_j \leftarrow j$ and $c_{j+1} \leftarrow 0$ for $0 \leq j < n$.
- G2.** [Visit.] (At this point the mixed-radix number $\left[\begin{smallmatrix} c_{n-1}, \dots, c_2, c_1 \\ n, \dots, 3, 2 \end{smallmatrix} \right]$ is the number of permutations visited so far.) Visit the permutation $a_0 a_1 \dots a_{n-1}$.
- G3.** [Add 1 to $c_n \dots c_2 c_1$.] 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 $a_0 a_1 \dots a_{n-1}$, as explained below, and return to G2. ■

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

$$\tau(k, j) = \sigma(k, j) \sigma(k, j-1)^- \quad \text{for } 1 \leq j \leq k; \quad (15)$$

$$\omega(k) = \sigma(1, 1) \dots \sigma(k, k). \quad (16)$$

Then steps G3 and G4 maintain the property that

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

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

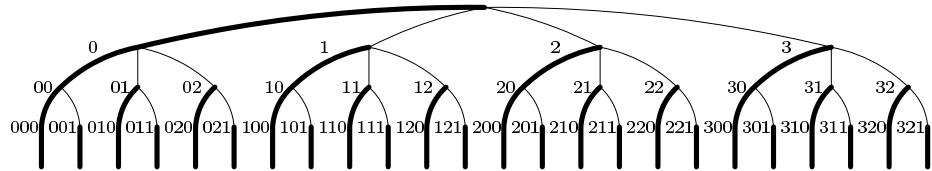


Fig. 19. Algorithm G implicitly traverses this tree when $n = 4$.

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 $0 \leq c_3 \leq 3$, $0 \leq c_2 \leq 2$, and $0 \leq c_1 \leq 1$. 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 permutation in reverse colex order as a special case of Algorithm G (see (11)), by letting $\sigma(k,j)$ be the $(j+1)$ -cycle

$$\sigma(k,j) = (k-j \ k-j+1 \ \dots \ k). \quad (18)$$

The reason is that $\sigma(k,j)$ should be the permutation that corresponds to $c_n \dots c_1$ in reverse colex order when $c_k = j$ and $c_i = 0$ for $i \neq k$, and this permutation $a_0 a_1 \dots a_{n-1}$ is $01 \dots (k-j-1)(k-j+1) \dots (k)(k-j)(k+1) \dots (n-1)$. For example, when $n = 8$ and $c_n \dots c_1 = 00030000$ 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

$$\tau(k,j) = (k-j \ k); \quad (19)$$

$$\omega(k) = (01)(012) \dots (01 \dots k) = (0k)(1k-1)(2k-2) \dots = \phi(k); \quad (20)$$

here $\phi(k)$ is the “ $(k+1)$ -flip” that changes $a_0 \dots a_k$ to $a_k \dots a_0$. 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

$$(k-j \ k)\phi(k-1) = \phi(k-1)(j-1 \ k) \quad (21)$$

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

$$\pi^- (j_1 \ j_2 \ \dots \ j_t) \pi = (j_1 \pi \ j_2 \pi \ \dots \ j_t \pi) \quad (22)$$

valid for *any* permutation π and any t -cycle $(j_1 \ j_2 \ \dots \ j_t)$. 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 whatever, 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

$$\sigma(k, j) = (k \dots 1 0)^j. \quad (23)$$

Now it is clear from (15) that

$$\tau(k, j) = (k \dots 1 0); \quad (24)$$

and once again we have

$$\omega(k) = (0 k)(1 k-1)(2 k-2)\dots = \phi(k), \quad (25)$$

because $\sigma(k, k) = (0 1 \dots k)$ 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 :

$$\tau(k, j)\omega(k-1)^- = (k \dots 1 0)\phi(k-1)^- = \phi(k). \quad (26)$$

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

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

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

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 $k = 3, 5, \dots$; 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 $a_0 \dots a_{k-1}$ 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 $a_k \dots a_{n-1}$ 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 $a_k \dots a_{n-1}$ 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 $a_0 \dots a_{n-1}$, 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 $a_0 \dots a_{n-1}$, and return to G2.0.

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

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.

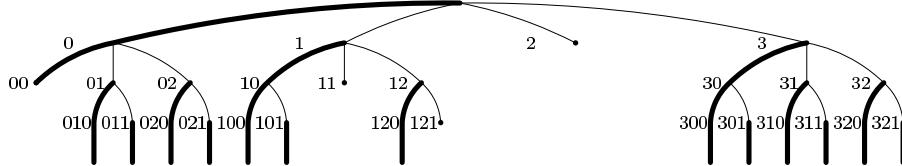


Fig. 20. Unwanted branches can be pruned from the tree of Fig. 19, if Algorithm G is suitably extended.

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 $a_k \dots a_{n-1}$ of the permutation $a_0 \dots a_{n-1}$ corresponds to a prefix $c_n \dots c_k$ of the control string $c_n \dots c_1$, because the permutations $\sigma(1, c_1) \dots \sigma(k-1, c_{k-1})$ do not affect $a_k \dots a_{n-1}$.) Step G2.1 premultiplies by $\tau(k, j)$ to move from node $c_{n-1} \dots c_{k+1}j$ to its right sibling $c_{n-1} \dots c_{k+1}(j+1)$, and it premultiplies by $\sigma(k, k)^-$ to move up from node $c_{n-1} \dots c_{k+1}k$ to its parent $c_{n-1} \dots c_{k+1}$. 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 $a_0a_1 \dots a_{n-1}$ that has suffix $a_1 \dots a_{n-1}$.) After node 2 is rejected, $\tau(3, 3)$ takes us to node 3, etc.

Notice, incidentally, that bypassing a suffix $a_k \dots a_{n-1}$ in this extension of Algorithm G is essentially the same as bypassing a prefix $a_1 \dots a_j$ in our original notation, if we go back to the idea of generating permutations $a_1 \dots a_n$ of $\{1, \dots, n\}$ and doing most of the work at the right-hand end. Our original notation corresponds to choosing a_1 first, then a_2, \dots , then a_n ; the notation in Algorithm G essentially chooses a_{n-1} first, then a_{n-2}, \dots , 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 $(D + E - Y) \bmod 10 = 0$ 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 $\{0, 1, \dots, 9\}$ so that $r_0 \geq r_1 \geq \dots \geq r_9$. For example, to solve the trio sonata problem (7), we could use $(0, 1, \dots, 9)$ respectively for $(X, S, V, A, R, I, L, T, O, N)$, obtaining the signatures

$$\begin{aligned} s_0 &= 0, & s_1 &= -100000, & s_2 &= 210000, & s_3 &= -100, & s_4 &= -100, \\ s_5 &= 21010, & s_6 &= 210, & s_7 &= -1010, & s_8 &= -7901, & s_9 &= -998; \end{aligned}$$

hence $(r_0, \dots, r_9) = (\infty, 5, 4, 2, 2, 1, 1, 1, 0, 0)$. 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 $a_k \dots a_9$ is unacceptable unless $a_k s_k + \dots + a_9 s_9$ is a multiple of $10^{r_{k-1}}$. Also, (10) tells us that $a_k \dots a_9$ 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 kilomems! 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 $\tau(k, 1), \tau(k, 2), \dots, \tau(k, k)$.

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

$$(551, 110, 14, 8, 350, 84, 153, 1598) \text{ kilomems with Heap's method;} \quad (28)$$

$$(429, 84, 10, 5, 256, 63, 117, 1189) \text{ kilomems with reverse colex.}$$

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 $c_n \dots c_1$ used by Algorithm G. And a closer look at Algorithm L shows that we can improve its behavior when permutations are frequently being skipped, by using a linked list instead of a sequential array. The improved algorithm is well-suited to a wide variety of algorithms that wish to generate restricted classes of permutations:

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

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

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

$$\{1, \dots, n\} \setminus \{a_1, \dots, a_k\} = \{b_1, \dots, b_{n-k}\}, \quad \text{where } b_1 < \dots < b_{n-k}, \quad (29)$$

then we have

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

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

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

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

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

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

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

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

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

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

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).

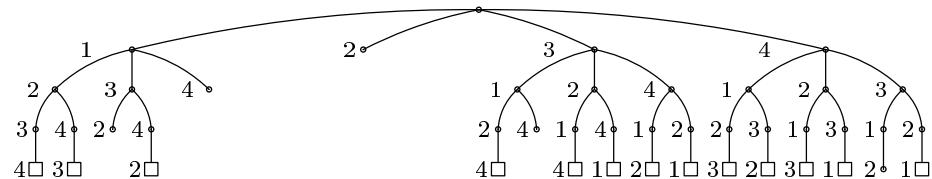


Fig. 21. The tree implicitly traversed by Algorithm X when $n = 4$, if all permutations are visited except those beginning with 132, 14, 2, 314, or 4312.

***Dual methods.** If S_1, \dots, 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 $\sigma_1 \dots \sigma_{n-1}$, where $\sigma_k \in S_k$; see (13). Exercise 50 shows that every element α can also be expressed uniquely in the dual form

$$\alpha = \sigma_{n-1}^- \dots \sigma_2^- \sigma_1^-, \quad \text{where } \sigma_k \in S_k \text{ for } 1 \leq k < n, \quad (32)$$

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

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

where $0 \leq c_k \leq k$ for $1 \leq k < n$ 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 $a_0 \dots a_{n-1}$ of $\{0, \dots, n-1\}$, using an auxiliary table $c_0 \dots c_{n-1}$.

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

H2. [Visit.] (At this point the mixed-radix number $[c_1, c_2, \dots, c_{n-1}]$ is the number of permutations visited so far.) Visit the permutation $a_0 a_1 \dots a_{n-1}$.

H3. [Add 1 to $c_0 c_1 \dots c_{n-1}$.] 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 $a_0 a_1 \dots a_{n-1}$, 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

$$\tau(k, j) = \sigma(k, j)^- \sigma(k, j-1), \quad \text{for } 1 \leq j \leq k, \quad (34)$$

$$\omega(k) = \sigma(n-1, n-1)^- \sigma(n-2, n-2)^- \dots \sigma(k, k)^-. \quad (35)$$

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

$$\sigma(k, j) = (k-j \ k-j+1 \ \dots \ k) \quad (36)$$

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 $\sigma(3, 3)^- = (bc)(ab)(3\ a) = (3\ abc)$; 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 $(3\ c)(3\ abc) = (abc)$. 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 $(3\ abc)(a\ c\ b) = (3\ c)$. 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

$$\sigma(k, j) = (k \dots 1\ 0)^j \quad (37)$$

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

$$\tau(k, j) = (0\ 1 \dots k), \quad (38)$$

and this fact is particularly advantageous in Algorithm H because it allows us to dispense with the control table $c_0c_1\dots c_{n-1}$. 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 $a_1 \dots a_n$ of the distinct elements $\{x_1, \dots, x_n\}$.

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

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

C3. [Shift.] Replace $a_1a_2\dots a_k$ by the cyclic shift $a_2\dots a_k a_1$, and return to C2 if $a_k \neq x_k$.

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

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

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

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 $c_0 c_1 \dots c_{k-1}$ is usually not of importance. The special case (36) is, however, sometimes an exception, because the $n!/k!$ permutations with $c_0 c_1 \dots c_{k-1} = 00 \dots 0$ in that case are precisely those $a_0 a_1 \dots a_{n-1}$ 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 $c_1 \dots c_{n-1}$. 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 $a_0 \dots a_{n-1}$ by using auxiliary tables $b_0 \dots b_{n-1}$ and $c_1 \dots c_n$.

- E1.** [Initialize.] Set $b_j \leftarrow j$ and $c_{j+1} \leftarrow 0$ for $0 \leq j < n$.
- E2.** [Visit.] Visit the permutation $a_0 \dots a_{n-1}$.
- 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 $j \geq k$. 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 $t[1], t[2], \dots, t[n! - 1]$. 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

$$E_\infty = 121213212123121213212124313132131312\dots \quad (39)$$

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

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

***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 $a_1a_2a_3\dots a_n$ at each step by either $a_2a_3\dots a_na_1$ or $a_2a_1a_3\dots a_n$, and they wondered whether such a method exists for all n .

In general, if G is any group of permutations and if $\alpha_1, \dots, \alpha_k$ are elements of G , the *Cayley graph* for G with generators $(\alpha_1, \dots, \alpha_k)$ is the directed graph whose vertices are the permutations π of G and whose arcs go from π to $\alpha_1\pi, \dots, \alpha_k\pi$. [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 $\{1, 2, \dots, n\}$, with generators σ and τ where σ is the cyclic permutation $(1 2 \dots n)$ 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 circuit:

Theorem R. *Let G be a group consisting of g permutations. If the Cayley graph for G with generators (α, β) has a Hamiltonian circuit, and if the permutations $(\alpha, \beta, \alpha\beta^{-1})$ 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 72. ■

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

construct when $n = 4$, because we can join up the 12-cycles

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

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 $\sigma\tau$ graph for $n = 5$ and discovered that it has five essentially distinct Hamiltonian circuits, 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 $\sigma\tau$ 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 circuits 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 n th 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. Fig. 22(b) illustrates the case $n = 5$ of this method, which was motivated by examples of bell-ringing.

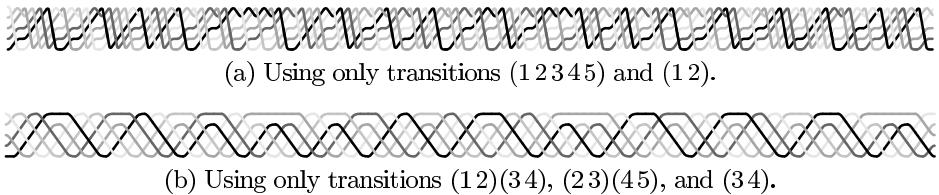


Fig. 22. Hamiltonian circuits for $5!$ permutations.

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

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

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

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

(42)

(Here A0 is the address of octabyte a_0 , etc.) A complete permutation program, which takes care of getting the right things into u , v , and w , appears in exercise 76, but the other instructions are less important because they need to be performed only $\frac{1}{6}$ of the time. The total cost per permutation, not counting the $4v$ needed for PUSHJ and POP on each call to Visit, comes to approximately $2.77\mu + 5.69v$ with this approach. If we use four registers u , v , w , x , and if we expand (42) to 24 calls on Visit, the running time per permutation drops to about $2.19\mu + 3.07v$. And with r registers and $r!$ Visits, exercise 77 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:

$$a_0a_1 \dots a_{n-2}a_{n-1}a_0a_1 \dots a_{n-2}. \quad (43)$$

If we now let $a_0a_1 \dots a_{n-2}$ 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

$$a_0a_1 \dots a_{n-1}, \quad a_1 \dots a_{n-1}a_0, \quad \dots, \quad a_{n-1}a_0 \dots a_{n-2}, \quad (44)$$

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 $\{0, 1, \dots, n - 1\}$. 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 $(t[1], \dots, t[5039])$ 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 $0 \leq j < 252$, 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       $p \leftarrow$  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       $c \leftarrow c \gg 3$ .
2H   AND t,c,#1c    $t \leftarrow c \wedge (1110)_2$ .          (45)
        PBNZ t,1B      Branch if  $t \neq 0$ .
        ADD p,p,8
3H   LDO c,p,0      $c \leftarrow$  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 78 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 LD0U a,0B           Set a  $\leftarrow$  #...3210.
        JMP 2F
1H   SRU a,a,4*(16-N)     $a \leftarrow \lfloor a/16^{16-n} \rfloor$ .
        OR a,a,t               $a \leftarrow a \vee t$ .          (46)
2H   ⟨Visit the permutation in register a⟩
        SRU t,a,4*(N-1)        $t \leftarrow \lfloor a/16^{n-1} \rfloor$ .
        SLU a,a,4*(17-N)        $a \leftarrow 16^{17-n} a \bmod 16^{16}$ .
        PBNZ t,1B              To 1B if  $t \neq 0$ .
        ⟨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 80 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 $\{1, \dots, n\}$, 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

$$1234, 1243, 2134, 2143, 2413. \quad (47)$$

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 $x_1 \prec y_1, \dots, x_m \prec y_m$, 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 satisfied 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 $12\dots n$ 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 $\{1, \dots, 8\}$ 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 $a_1 \dots a_n$ if and only if $a'_x < a'_y$ in the inverse permutation $a'_1 \dots a'_n$. Therefore the algorithm we are about to study will also find all permutations $a'_1 \dots a'_n$ 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 $\{1, \dots, n\}$ 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 $a'_1 \dots a'_9$ such that

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

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 $\{1, \dots, 2n\}$ into n pairs. There are $(2n-1)(2n-3)\dots(1) = (2n)!/(2^n n!)$ ways to do this, and they correspond to permutations that satisfy

$$a'_1 < a'_2, \quad a'_3 < a'_4, \quad \dots, \quad a'_{2n-1} < a'_{2n}, \quad a'_1 < a'_3 < \dots < a'_{2n-1}. \quad (49)$$

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 $\{1, \dots, n-1\}$ topologically, so that $a_1 \dots a_{n-1}$ 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 $a_1 \dots a_{n-1}$: We simply start with $a_1 \dots a_{n-1} n$, 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 $\{1, \dots, n\}$ with the property that $x \prec y$ implies $x < y$, this algorithm generates all permutations $a_1 \dots a_n$ and their inverses $a'_1 \dots a'_n$ 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 $1 \leq k \leq n$.

- V1.** [Initialize.] Set $a_j \leftarrow j$ and $a'_j \leftarrow j$ for $0 \leq j \leq n$.
- V2.** [Visit.] Visit the permutation $a_1 \dots a_n$ and its inverse $a'_1 \dots a'_n$. 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 $k > 0$. \blacksquare

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

$$\boxed{a'_1 a'_2 a'_3 \\ a'_4 a'_5 a'_6 \\ a'_7 a'_8 a'_9}, \quad (50)$$

we get the following 42 results:

123	123	123	123	123	123	124	124	124	124	124	125	125	125	125
456	457	458	467	468	356	357	358	367	368	367	368	346	347	347
789	689	679	589	579	789	689	679	589	579	489	479	789	689	
125	126	126	127	126	126	127	134	134	134	134	134	135	135	135
348	347	348	348	357	358	358	256	257	258	267	268	267	268	
679	589	579	569	489	479	469	789	689	679	589	579	489	479	
145	145	135	135	135	136	136	137	136	136	137	146	146	147	
267	268	246	247	248	247	248	248	257	258	258	257	258	258	
389	379	789	689	679	589	579	569	489	479	469	389	379	369	

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

$$M = t_n + \dots + t_1 \quad \text{and} \quad N = t_n. \quad (51)$$

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

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

$$2 \prec 3, \quad 3 \prec 4, \quad \dots, \quad n - 1 \prec n, \quad (52)$$

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

$$1 \prec 2, \quad 2 \prec 3, \quad \dots, \quad n - 2 \prec n - 1, \quad (53)$$

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

Exercise 88 shows that a simple preprocessing step will find element labels so that a slight modification of Algorithm V is able to generate all topological sorts in $O(N + n)$ steps. Thus topological sorting can always be done efficiently.

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

EXERCISES

- 1. [20] Explain how to make Algorithm L run faster, by streamlining its operations when the value of j is near n .
- 2. [20] Rewrite Algorithm L so that it produces all permutations of $a_1 \dots a_n$ in reverse colex order. (In other words, the values of the reflections $a_n \dots a_1$ should be lexicographically decreasing, as in (11). This form of the algorithm is often simpler and faster than the original, because fewer calculations depend on the value of n .)

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

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

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

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

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

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

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

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

$$\begin{aligned} & \epsilon, 1, 12, 122, 1223, 123, 1232, 13, 132, 1322, \\ & 2, 21, 212, 2123, 213, 2132, 22, 221, 2213, 223, 2231, 23, 231, 2312, 232, 2321, \\ & 3, 31, 312, 3122, 32, 321, 3212, 322, 3221. \end{aligned}$$

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

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

10. [20] What are the values of $a_1 a_2 \dots a_n$, $c_1 c_2 \dots c_n$, and $d_1 d_2 \dots d_n$ at the end of Algorithm P, if $a_1 a_2 \dots a_n = 12 \dots n$ at the beginning?

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

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

13. [M21] (Martin Gardner, 1974.) True or false: If $a_1 a_2 \dots a_n$ is initially $12 \dots n$, Algorithm P begins by visiting all $n!/2$ permutations in which 1 precedes 2; then the next permutation is $n \dots 21$.

14. [M22] True or false: If $a_1 a_2 \dots a_n$ is initially $x_1 x_2 \dots x_n$ in Algorithm P, we always have $a_{j-c_j+s} = x_j$ at the beginning of step P5.

- 15.** [M23] (Selmer Johnson, 1963.) Show that the offset variable s never exceeds 2 in Algorithm P.
- 16.** [21] Explain how to make Algorithm P run faster, by streamlining its operations when the value of j is near n . (This problem is analogous to exercise 1.)
- **17.** [20] Extend Algorithm P so that the *inverse permutation* $a'_1 \dots a'_n$ is available for processing when $a_1 \dots a_n$ 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 $\{1, \dots, n\}$; that is, no cyclic shift of $a_1 \dots a_n$ or $a_n \dots a_1$ will be generated if $a_1 \dots a_n$ 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.L.
- **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 $000 \mapsto 001$, $001 \mapsto 011$, ..., $111 \mapsto 110$. Design a simple algorithm that generates all signed permutations of $\{1, 2, \dots, n\}$, 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:
- SEND + A + TAD + MORE = MONEY. (Peter Macdonald, 1977)
 - ZEROES + ONES = BINARY. (Willy Enggren, 1972)
 - DCLIX + DLXVI = MCCXXV. (Michael R. W. Buckley, 1977)
 - COUPLE + COUPLE = QUARTET. (Bob Vinnicombe, 1978)
 - FISH + N + CHIPS = SUPPER. (Willy Enggren, 1968)
 - SATURN + URANUS + NEPTUNE + PLUTO = PLANETS. (Herman Nijon, 1977)
 - EARTH + AIR + FIRE + WATER = NATURE. (Herman Nijon, 1977)
 - AN + ACCELERATING + INFERENTIAL + ENGINEERING + TALE + ELITE + GRANT + FEE + ET + CETERA = ARTIFICIAL + INTELLIGENCE.
 - 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 $a_1 \dots a_{10}$ of $\{0, \dots, 9\}$, given the signature vector $s = (s_1, \dots, s_{10})$ 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

$$\text{NIHUAU} \pm \text{KAUAI} \pm \text{OAHU} \pm \text{MOLOKAI} \pm \text{LANAI} \pm \text{MAUI} \pm \text{HAWAII} = 0$$
- 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 $n = n_1 + \dots + n_t$ with $n_1 \geq \dots \geq n_t > 0$. Such a partition is called *doubly true* if $\alpha(n) = \alpha(n_1) + \dots + \alpha(n_t)$ is also a pure alphametic, where $\alpha(n)$ is the “name” of n in some language. Doubly true partitions were introduced by Alan Wayne in *AMM* 54 (1947), 38, 412–414, where he suggested solving TWENTY = SEVEN + SEVEN + SIX and a few others.

- a) Find all partitions that are doubly true in English when $1 \leq n \leq 20$.
 - b) Wayne also gave the example EIGHTY = FIFTY + TWENTY + NINE + ONE. Find all doubly true partitions for $1 \leq n \leq 100$ in which the parts are *distinct*, using the names ONE, TWO, ..., NINETYNINE, ONEHUNDRED.
- **29.** [M25] Continuing the previous exercise, find all equations of the form $n_1 + \dots + n_t = n'_1 + \dots + n'_{t'}$ that are both mathematically and alphametically true in English, when $\{n_1, \dots, n_t, n'_1, \dots, n'_{t'}\}$ are distinct positive integers less than 20. For example,

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

the alphametics should all be pure.

- 30.** [25] Solve these multiplicative alphametics by hand or by computer:
- a) TWO \times TWO = SQUARE. (H. E. Dudeney, 1929)
 - b) HIP \times HIP = HURRAY. (Willy Enggren, 1970)
 - c) PI \times R \times R = AREA. (Brian Barwell, 1981)
 - d) NORTH/SOUTH = EAST/WEST. (Nob Yoshigahara, 1995)

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

- 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 $\{1, \dots, 9\}$. For example, $100 = 91 + 5742/638$. The plus sign should precede the slash.

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

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

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

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

0	1	2	3
4	5	6	7
8	9	a	b
c	d	e	f

,

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

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

- 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.** [M31] Design an algorithm that generates all r -variations of $\{0, 1, \dots, n - 1\}$ 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 $a_{n-r} \dots a_{n-1}$ of an array $a_0 \dots a_{n-1}$. 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, 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 $1 \leq j \leq k$ 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 $1 \leq j \leq k$ 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 $u_1 \dots u_9$ when Algorithm X visits the permutation 314592687? (b) What permutation is visited when $u_1 \dots u_9 = 314157700$?
- 46.** [20] True or false: When Algorithm X visits $a_1 \dots a_n$, we have $u_k > u_{k+1}$ if and only if $a_k > a_{k+1}$, for $1 \leq k < n$.
- **47.** [M21] Express the number of times that each step of Algorithm X is performed in terms of the numbers N_0, N_1, \dots, N_n , where N_k is the number of prefixes $a_1 \dots a_k$ that satisfy $t_j(a_1, \dots, a_j)$ for $1 \leq j \leq k$.
- **48.** [M25] Compare the running times of Algorithm X and Algorithm L, in the case when the tests $t_1(a_1), t_2(a_1, a_2), \dots, t_n(a_1, a_2, \dots, a_n)$ always are true.
- **49.** [28] The text's suggested method for solving additive alphametics with Algorithm X essentially chooses digits from right to left; in other words, it assigns tentative values to the least significant digits before considering digits that correspond to higher powers of 10.

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

- 50.** [M15] Explain why the dual formula (32) follows from (13).
- 51.** [M16] True or false: If the sets $S_k = \{\sigma(k, 0), \dots, \sigma(k, k)\}$ form a Sims table for the group of all permutations, so also do the sets $S_k^- = \{\sigma(k, 0)^-, \dots, \sigma(k, k)^-\}$.
- **52.** [M22] What permutations $\tau(k, j)$ and $\omega(k)$ arise when Algorithm H is used with the Sims table (36)? Compare the resulting generator with Algorithm P.
- **53.** [M26] (F. M. Ives.) Construct a Sims table for which Algorithm H will generate all permutations by making only $n! + O((n-2)!)$ transpositions.
- 54.** [20] Would Algorithm C work properly if step C3 did a right-cyclic shift, setting $a_1 \dots a_{k-1} a_k \leftarrow a_k a_1 \dots a_{k-1}$, 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 $j \leq k$. Let α_0 and β_0 be the identity permutation, and for $m > 0$ define

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

For example, if σ_k is the flip operation $(1\ k-1)(2\ k-2)\dots = (0\ k)\phi(k)$ and if $\tau_k = (0\ k)$, and if Algorithm E is started with $a_j = j$ for $0 \leq j < n$, then α_m and β_m are the contents of $a_0\dots a_{n-1}$ and $b_0\dots b_{n-1}$ 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 $j = k - 2, k - 4, \dots, (2 \text{ or } 1)$. 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 $a_0\dots a_{n-1} = x_1\dots x_n$ 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 code for permutations* is a cycle $(\pi_0, \pi_1, \dots, \pi_{n!-1})$ that includes every permutation of $\{1, 2, \dots, n\}$ 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 circuit on the Cayley graph for the group of all permutations on $\{1, 2, \dots, n\}$, with the $n - 1$ generators $((1\ 2), (2\ 3), \dots, (n-1\ n))$. The *delta sequence* of such a Gray code is the sequence of integers $\delta_0\delta_1\dots\delta_{n!-1}$ 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 code defined by plain changes when $n = 4$; its delta sequence is $(32131231)^3$.

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

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

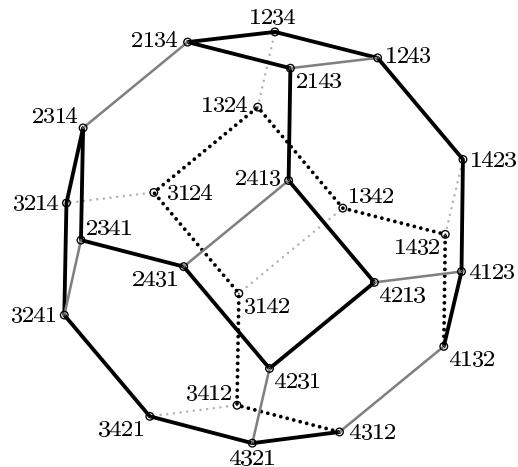


Fig. 23. Algorithm P traces out this Hamiltonian circuit on the truncated octahedron of Fig. 5-1.

61. [21] Continuing the previous exercise, a *Gray path for permutations* is like a Gray code 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 paths for $n = 4$ that start with 1234.

► **62.** [M23] What permutations can be reached as the final element of a Gray path that starts at 12... n ?

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

64. [23] A “doubly Gray” code for permutations is a Gray code 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 $\{1, \dots, n\}$? (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 code for permutations, in which the $n - 1$ generators are the star transpositions $(1 2), (1 3), \dots, (1 n)$. For example, Fig. 24 shows the relevant graph when $n = 4$. Analyze the Hamiltonian circuits of this graph.

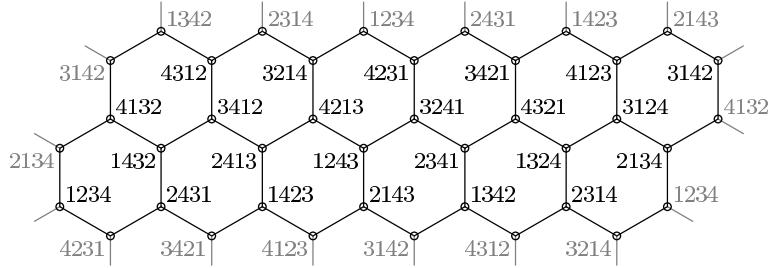


Fig. 24. The Cayley graph for permutations of $\{1, 2, 3, 4\}$, generated by the star transpositions $(1 2), (1 3)$, and $(1 4)$, drawn as a twisted torus.

67. [26] Continuing the previous exercise, find a first-element-swap Gray code 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 $\{1, \dots, n\}$, with generators $(\alpha_1, \dots, \alpha_k)$ where each α_j is a transposition $(u_j v_j)$; also let A be the graph with vertices $\{1, \dots, n\}$ and edges $u_j — v_j$ for $1 \leq j \leq k$. Prove that G has a Hamiltonian circuit if and only if A is connected. (Fig. 23 is the special case when A is a path; Fig. 24 is the special case when A is a “star.”)

► **69.** [28] If $n \geq 4$, the following algorithm generates all permutations $A_1 A_2 A_3 \dots A_n$ of $\{1, 2, 3, \dots, n\}$ using only three transformations,

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

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

Z1. [Initialize.] Set $A_j \leftarrow j$ for $1 \leq j \leq n$. Also set $a_j \leftarrow 2j$ for $j \leq n/2$ and $a_{n-j} \leftarrow 2j + 1$ for $j < n/2$. Then invoke Algorithm P, but with parameter $n - 1$ instead of n . We will treat that algorithm as a coroutine, which should

return control to us whenever it “visits” $a_1 \dots a_{n-1}$ in step P2. We will also share its variables (except n).

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

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

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

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

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

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

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

$$\rho' = (1 \ n)(2 \ n-1)\dots, \quad \sigma' = (2 \ n)(3 \ n-1)\dots, \quad \tau' = (2 \ n-1)(3 \ n-2)\dots$$

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

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

$$\begin{aligned} 1134 &\rightarrow 1341 \rightarrow 3411 \rightarrow 4311 \rightarrow 3114 \rightarrow 1143 \rightarrow 1431 \\ &\rightarrow 4131 \rightarrow 1314 \rightarrow 3141 \rightarrow 1413 \rightarrow 4113 \rightarrow 1134. \end{aligned}$$

Replacing $\{1, 1\}$ by $\{1, 2\}$ yields disjoint cycles, and we obtained a Hamiltonian path by jumping from one to the other. Can a $\sigma\tau$ 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.** [M21] Given a Cayley graph with generators $(\alpha_1, \dots, \alpha_k)$, assume that each α_j takes $x \mapsto y$. (For example, both σ and τ take $1 \mapsto 2$ when $\sigma = (1 \ 2 \ \dots \ n)$ and $\tau = (1 \ 2)$.) Prove that any Hamiltonian path starting at $12\dots n$ in G must end at a permutation that takes $y \mapsto x$.

- **72.** [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.

- 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.
- Derive Theorem R from part (a).

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

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

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

- 75.** [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 $m, n \geq 3$?

Fig. 25. A northeasterly knight's tour.

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

- **76.** [22] Complete the MMIX program whose inner loop appears in (42), using Heap's method (27).
- 77.** [M23] Analyze the running time of the program in exercise 76, generalizing it so that the inner loop does $r!$ visits (with $a_0 \dots a_{r-1}$ in global registers).
- 78.** [20] What seven MMIX instructions will \langle Swap the nybbles ... \rangle 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.
- 79.** [21] Solve the previous exercise with only five MMIX instructions. *Hint:* Use MXOR.
- **80.** [22] Complete the MMIX program (46) by specifying how to \langle Continue with Langdon's method \rangle .
- 81.** [M21] Analyze the running time of the program in exercise 80.
- 82.** [22] Use the $\sigma\tau$ path of exercise 70 to design an MMIX routine analogous to (42) that generates all permutations of #123456 in register **a**.
- 83.** [20] Suggest a good way to generate all $n!$ permutations of $\{1, \dots, n\}$ on p processors that are running in parallel.
- **84.** [25] Assume that n is small enough that $n!$ fits in a computer word. What's a good way to convert a given permutation $\alpha = a_1 \dots a_n$ of $\{1, \dots, n\}$ into an integer $k = r(\alpha)$ in the range $0 \leq k < n!$? Both functions $k = r(\alpha)$ and $\alpha = r^{-1}(k)$ should be computable in only $O(n)$ steps.
- 85.** [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 $x \prec z$.
- 86.** [20] (F. Ruskey.) Consider the inversion tables $c_1 \dots c_n$ of the permutations visited by Algorithm V. What noteworthy property do they have? (Compare with the inversion tables (4) in Algorithm P.)

- 87.** [21] Show that Algorithm V can be used to generate all ways to partition the digits $\{0, 1, \dots, 9\}$ into two 3-element sets and two 2-element sets.
- **88.** [M30] Consider the numbers t_0, t_1, \dots, t_n in (51). Clearly $t_0 = t_1 = 1$.
- 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.
 - Analyze the running time of the modified algorithm. What formulas replace (51)?
 - Say that the interval $[j \dots k]$ is not a chain if we do not have $l \prec l+1$ for $j \leq l < k$. Prove that in such a case $t_k \geq 2t_{j-1}$.
 - Every inverse topological sort $a'_1 \dots a'_n$ defines a labeling that corresponds to relations $a'_{j_1} \prec a'_{k_1}, \dots, a'_{j_m} \prec a'_{k_m}$, which are equivalent to the original relations $j_1 \prec k_1, \dots, j_m \prec k_m$. Explain how to find a labeling such that $[j \dots k]$ is not a chain when j and k are consecutive nontrivial indices.
 - Prove that with such a labeling, $M < 4N$ in the formulas of part (b).
- 89.** [M21] Algorithm V can be used to produce all permutations that are h -ordered for all h in a given set, namely all $a'_1 \dots a'_n$ such that $a'_j < a'_{j+h}$ for $1 \leq j \leq n-h$ (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.
- 90.** [HM21] Analyze the running time of Algorithm V when it is used with the relations (49) to find matchings.
- 91.** [M18] How many permutations is Algorithm V likely to visit, in a “random” case? Let P_n be the number of partial orderings on $\{1, \dots, n\}$, 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 .
- 92.** [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.)
- **93.** [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.
- 94.** [21] Discuss how to generate all *up-down permutations* of $\{1, \dots, n\}$, namely those $a_1 \dots a_n$ such that $a_1 < a_2 > a_3 < a_4 > \dots$.
- 95.** [21] Discuss how to generate all *cyclic permutations* of $\{1, \dots, n\}$, namely those $a_1 \dots a_n$ whose cycle representation consists of a single n -cycle.
- 96.** [21] Discuss how to generate all *derangements* of $\{1, \dots, n\}$, namely those $a_1 \dots a_n$ such that $a_1 \neq 1, a_2 \neq 2, a_3 \neq 3, \dots$.
- 97.** [HM23] Analyze the asymptotic running time of the method in the previous exercise.
- 98.** [M30] Given $n \geq 3$, show that all derangements of $\{1, \dots, n\}$ can be generated by making at most two transpositions between visits.
- 99.** [21] Discuss how to generate all *indecomposable* permutations of $\{1, \dots, n\}$, namely those $a_1 \dots a_n$ such that $\{a_1, \dots, a_j\} \neq \{1, \dots, j\}$ for $1 \leq j < n$.
- 100.** [21] Discuss how to generate all *involutions* of $\{1, \dots, n\}$, namely those permutations $a_1 \dots a_n$ with $a_{a_1} \dots a_{a_n} = 1 \dots n$.

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

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

► **103.** [M22] A permutation $a_1 \dots a_n$ of $\{1, \dots, n\}$ 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 $a_1 \dots a_n$ is well-balanced, so are its reversal $a_n \dots a_1$, its complement $(n+1-a_1) \dots (n+1-a_n)$, and its inverse $a'_1 \dots a'_n$.
- c) Determine the number of well-balanced permutations for small values of n .

► **104.** [26] A *weak order* is a relation \preceq that is transitive ($x \preceq y$ and $y \preceq z$ implies $x \preceq z$) and complete ($x \preceq y$ or $y \preceq x$ always holds). We can write $x \equiv y$ if $x \preceq y$ and $y \preceq x$; $x \prec y$ if $x \preceq y$ and $y \not\preceq x$. There are thirteen weak orders on three elements $\{1, 2, 3\}$, namely

$$1 \equiv 2 \equiv 3, \quad 1 \equiv 2 \prec 3, \quad 1 \prec 2 \equiv 3, \quad 1 \prec 2 \prec 3, \quad 1 \equiv 3 \prec 2, \quad 1 \prec 3 \prec 2, \\ 2 \prec 1 \equiv 3, \quad 2 \prec 1 \prec 3, \quad 2 \equiv 3 \prec 1, \quad 2 \prec 3 \prec 1, \quad 3 \prec 1 \equiv 2, \quad 3 \prec 1 \prec 2, \quad 3 \prec 2 \prec 1.$$

- a) Explain how to generate all weak orders of $\{1, \dots, n\}$ systematically, as sequences of digits separated by the symbols \equiv or \prec .
- b) A weak order can also be represented as a sequence $a_1 \dots a_n$ 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 .

105. [M40] Can exercise 104(b) be solved with a Gray-like path?

► **106.** [30] (John H. Conway, 1973.) To play the solitaire game of “topswops,” start by shuffling a pack of n cards labeled $\{1, \dots, n\}$ 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 $a_1 \dots a_n$ to $a_k \dots a_1 a_{k+1} \dots a_n$. Continue until the top card is 1. For example, the 7-step sequence

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

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

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

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

► **109.** [25] Find all permutations $a_0 \dots a_9$ of $\{0, \dots, 9\}$ 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.

► **110.** [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 $\{1, \dots, n\}$ 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 $n \geq 2$.

111. [M46] Exactly how many universal cycles exist, for permutations of order n ?

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 $a_{j+1} \dots a_{n-1}$.] 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 $a_1 \leq a_2 \leq \dots \leq a_n$ 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 $a_1 a_2 \dots a_n$.

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 $a_1 \dots a_{j-1}$.] 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 $C_1 \dots C_n = c_{a_1} \dots c_{a_n}$ be the inversion table, as in exercise 5.1.1–7. Then $\text{rank}(a_1 \dots a_n)$ is the mixed-radix number $[C_n, \dots, C_2, C_1]_n$. [See H. A. Rothe, *Sammlung combinatorisch-analytischer Abhandlungen* **2** (1800), 263–264.] For example, 314592687 has rank $[2, 0, 1, 1, 4, 0, 0, 1, 0]_9 = 2 \cdot 8! + 6! + 5! + 4 \cdot 4! + 1! = 81577$; this is the factorial number system featured in Eq. 4.1–(10).

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

$$\frac{3}{9} (2, 1, 1, 1, 2, 1, 1) + 0 + \frac{2}{7} (1, 1, 1, 2, 1, 1) + 0 + \frac{1}{5} (1, 2, 1, 1) + \frac{3}{4} (1, 1, 1, 1) + 0 + \frac{1}{2} (1, 1) = 30991.$$

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 $a_{n-t+1} > \dots > a_n$, namely $[t \leq n]/t!$. Therefore the mean is $\sum k(q_k - q_{k+1}) = q_1 + \dots + q_n = [n!e]/n! - 1 \approx e - 1 \approx 1.718$, and the variance is

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

[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 $\sum \lfloor k/2 \rfloor (q_k - q_{k+1}) = q_2 + q_4 + \dots \approx \cosh 1 - 1 = (e + e^{-1} - 2)/2 \approx 0.543$, 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 $1 \leq k < n$ 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 $a_1 \dots a_j$ is $j! [z^j] e_{n_1}(z) \dots e_{n_t}(z)$. This is $N = \binom{n}{n_1, \dots, n_t}$ times the probability q_{n-j} that at least $n-j$ comparisons are made in step L2. Therefore the mean is $\frac{1}{N} w(e_{n_1}(z) \dots e_{n_t}(z)) - 1$, where $w(\sum x_k z^k/k!) = \sum x_k$. In the binary case the mean is $M/\binom{n}{m} - 1$, where $M = \sum_{l=0}^m \sum_{k=l}^{n-m+l} \binom{k}{l} = \sum_{l=0}^m \binom{n-m+l+1}{l+1} = \binom{n+2}{m+1} - 1 = \binom{n}{m} (2 + \frac{m}{n-m+1} + \frac{n-m}{m+1}) - 1$.

(b) If $\{a_1, \dots, a_j\} = \{n'_1 \cdot x_1, \dots, n'_t \cdot x_t\}$, the prefix $a_1 \dots a_j$ contributes altogether $\sum_{1 \leq k < l \leq t} (n_k - n'_k)[n_l < n'_l]$ to the total number of comparisons made in step L3. Thus the mean is $\frac{1}{N} \sum_{1 \leq k < l \leq t} w(f_{kl}(z))$, where

$$\begin{aligned} f_{kl}(z) &= \left(\prod_{\substack{1 \leq m \leq t \\ m \neq k, m \neq l}} e_{n_m}(z) \right) \left(\sum_{r=0}^{n_k} (n_k - r) \frac{z^r}{r!} \right) e_{n_l-1}(z) \\ &= n_k \left(\prod_{\substack{1 \leq m \leq t \\ m \neq l}} e_{n_m}(z) \right) e_{n_l-1}(z) - z \left(\prod_{\substack{1 \leq m \leq t \\ m \neq k, m \neq l}} e_{n_m}(z) \right) e_{n_l-1}(z) e_{n_k-1}(z). \end{aligned}$$

In the two-valued case this formula reduces to $\frac{1}{N} w((m e_m(z) - z e_{m-1}(z)) e_{n-m-1}(z)) = \frac{m}{N} ((\binom{n+1}{m+1} - 1) - \frac{1}{N} ((\binom{n+1}{m+1})(m - \frac{m+1}{n-m+1}) + 1) = \frac{1}{N} (-m - 1 + \binom{n+1}{m}) = \frac{n+1}{n-m+1} - \frac{m+1}{N}$.

7. In the notation of the previous answer, the quantity $\frac{1}{N} w(e_{n_1}(z) \dots e_{n_t}(z)) - 1$ is $\frac{n_1 + \dots + n_t}{n} + \frac{(n_1 n_2 + n_1 n_3 + \dots + n_{t-1} n_t) + n_1(n_1-1) + \dots + n_t(n_t-1)}{n(n-1)} + \dots - 1$.

One can show using Eq. 1.2.9–(38) that the limit is $-1 + \exp \sum_{k \geq 1} r_k/k$, where $r_k = \lim_{t \rightarrow \infty} (n_1^k + \dots + n_t^k)/(n_1 + \dots + n_t)^k$. 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 $-1 + \exp \sum_{k \geq 1} 1/(k(2^k - 1)) \approx 2.46275$.

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

L1'. [Visit.] Visit the variation $a_1 \dots a_j$. 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 $w(e_{n_1}(z) \dots e_{n_t}(z))$ in the notation of answer 6. This counting problem was first treated by Jakob Bernoulli in *Ars Conjectandi* (1713), Part 2, Chapter 9.

9. V1. [Visit.] Visit the variation $a_1 \dots a_r$. (At this point $a_{r+1} \leq \dots \leq a_n$.)

V2. [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 V1.

V3. [Reverse.] Set $(a_{r+1}, \dots, a_n) \leftarrow (a_n, \dots, a_{r+1})$ as in step L4.

V4. [Find j .] Set $j \leftarrow r - 1$. If $a_j \geq a_{j+1}$, decrease j by 1 repeatedly until $a_j < a_{j+1}$. Terminate if $j = 0$.

V5. [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 $a_j \leftrightarrow a_l$.

V6. [Reverse again.] Set $(a_{j+1}, \dots, a_n) \leftarrow (a_n, \dots, a_{j+1})$ as in step L4, and return to V1. ■

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

10. $a_1 a_2 \dots a_n = 213 \dots n$, $c_1 c_2 \dots c_n = 010 \dots 0$, $d_1 d_2 \dots d_n = 1(-1)1 \dots 1$, if $n \geq 2$.

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

12. We want the permutation of rank 999999. The answers are (a) 2783915460, by exercise 3; (b) 8750426319, because the reflected mixed-radix number corresponding to $[0, 0, 1, 2, 3, 0, 2, 7, 0, 9]$ is $[0, 0, 1, 3-2, 3, 5-0, 2, 7, 8-0, 9-9]$ by 7.2.1.1-(50); (c) the product $(01 \dots 9)^9 (01 \dots 8)^0 (01 \dots 7)^7 (01 \dots 6)^2 \dots (01 2)^1$, namely 9703156248.

13. The first statement is true for all $n \geq 2$. But when 2 crosses 1, namely when c_2 changes from 0 to 1, we have $c_3 = 2$, $c_4 = 3$, $c_5 = \dots = c_n = 0$, and the next permutation when $n \geq 5$ is 432156... 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 $\{x_1, \dots, x_{j-1}\}$ and s from $\{x_{j+1}, \dots, x_n\}$. (In a sense, this formula is the main point of Algorithm P.)

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

16. P1'. [Initialize.] Set $c_j \leftarrow j$ and $d_j \leftarrow -1$ for $1 \leq j < n$; also set $z \leftarrow a_n$.

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

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

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

P4'. [Ready to change?] Set $q \leftarrow c_j + d_j$. 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 $d_j \leftarrow -d_j$, $j \leftarrow j - 1$, and go back to P4'. ■

17. Initially $a_j \leftarrow a'_j \leftarrow j$ for $1 \leq j \leq n$. Step P5 should now set $t \leftarrow j - c_j + s$, $u \leftarrow j - q + s$, $v \leftarrow a_u$, $a_t \leftarrow v$, $d_v \leftarrow t$, $a_u \leftarrow j$, $a'_j \leftarrow u$, $c_j \leftarrow q$. (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 $c_1 \dots c_n$ 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 $1 \leq j \leq n$. Also set $c_0 = -1$.

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

Q3. [Find k .] Set $k \leftarrow n$. Then if $c_k = 0$, set $c_k \leftarrow k - 1$, $d_k \leftarrow -d_k$, $k \leftarrow k - 1$, 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 $i = j + d_k$. Then set $t \leftarrow a_i$, $a_i \leftarrow k$, $a_j \leftarrow t$, $a'_t \leftarrow j$, $a'_k \leftarrow i$, and return to Q2. ▀

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

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

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

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

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

21. Clearly $M = 1$, hence O must be 0 and S must be $b-1$. Then $N = E+1$, $R = b-2$, and $D+E = b+Y$. This leaves exactly $\max(0, b-7-k)$ choices for E when $Y = k \geq 2$, 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.]

22. $(XY)_b + (XX)_b = (XYX)_b$ 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 $X + (XX)_3 + (YY)_3 + (XZ)_3 = (XYX)_3$.)

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), 155; 5 (1972), 296; 10 (1977), 41; 10 (1978), 274; 12 (1979), 133–134; 9 (1977), 207.]

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

25. We may assume that $s_1 \leq \dots \leq s_{10}$. Let i be the least index $\notin F$, and set $a_i \leftarrow 0$; then set the remaining elements a_j in order of increasing j . A proof like that

of Theorem 6.1S shows that this procedure maximizes $a \cdot s$. A similar procedure yields the minimum, because $\min(a \cdot s) = -\max(a \cdot (-s))$.

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

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

28. (a) $11 = 3 + 3 + 2 + 2 + 1$, $20 = 11 + 3 + 3 + 3$, $20 = 11 + 3 + 3 + 2 + 1$, $20 = 11 + 3 + 3 + 1 + 1 + 1$, $20 = 8 + 8 + 2 + 1 + 1$, $20 = 7 + 7 + 6$, $20 = 7 + 7 + 2 + 2 + 2$, $20 = 7 + 7 + 2 + 1 + 1 + 1 + 1$, $20 = 7 + 5 + 5 + 2 + 1$, $20 = 7 + 5 + 2 + 2 + 2 + 1 + 1$, $20 = 7 + 5 + 2 + 2 + 1 + 1 + 1 + 1$, $20 = 5 + 3 + 3 + 3 + 3 + 3$. [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 $50 = 2 + 2 + \dots + 2$), 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 arbitrary large integers is not standard. Is there an example that exceeds NINETYNINENONILLIONNINETYNINETRILLIONNNINETYONE = NINETYNINENONILLIONNINETYNINETRILLIONFORTYFIVE + NINETEEN + ELEVEN + TEN + FIVE + ONE?

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

30. (a) $567^2 = 321489$, $807^2 = 651249$, or $854^2 = 729316$. (b) $958^2 = 917764$. (c) $96 \times 7^2 = 4704$. (d) $51304/61904 = 7260/8760$. [*Strand* **78** (1929), 91, 208; *J. Recr. Math* **3** (1970), 43; **13** (1981), 212; **27** (1995), 137. The solutions to (b), (c), and (d) are unique. With a right-to-left approach based on Algorithm X, the answers are found in (14, 13, 11, 3423) kilomems, respectively.]

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.

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 $a_k \dots a_{l-1}/a_l \dots a_0$ is an integer, then to record solutions for all permutations of $a_1 \dots a_{k-1}$. 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 $1 \leq k < n$, there are $s_1 \dots s_{k-1}$ ways to choose each of the nonidentity elements of S_k . Hence the answer is $\prod_{k=1}^{n-1} (\prod_{j=1}^{k-1} s_j^{s_k-1})$, which in this case is $2^2 \cdot 6^3 \cdot 24^{15} = 436196692474023836123136$.

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

- 36.** Since each of $\{0, 3, 5, 6, 9, a, c, f\}$ lies on three lines, but every other element lies on only two, it is clear that we may let $S_f = \{(), \sigma, \sigma^2, \sigma^3, \alpha, \alpha\sigma, \alpha\sigma^2, \alpha\sigma^3\}$, where $\sigma = (03fc)(17e4)(2bd4)(56a9)$ is a 90° rotation and $\alpha = (05)(14)(27)(36)(8d)(9c)(af)(be)$ is an inside-out twist. Also $S_e = \{(), \beta, \gamma, \beta\gamma\}$, where $\beta = (14)(28)(3c)(69)(be)$ is a transposition and $\gamma = (12)(48)(5a)(69)(7b)(de)$ is another twist; $S_a = \dots = S_1 = \{()\}$. (There are $4^7 - 1$ alternative answers.)

- 37.** The set S_k can be chosen in $k!^{k-1}$ ways (see exercise 35), and its nonidentity elements can be assigned to $\sigma(k, 1), \dots, \sigma(k, k)$ in $k!$ further ways. So the answer is $A_n = \prod_{k=1}^{n-1} k!^k = n!^{(n)} / \prod_{k=1}^n k^{\binom{k}{2}}$. 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 $1 \leq k < n$; 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 $\sum_{k=1}^{n-1} (1/k! - 1/(k+1)!) \lceil k/2 \rceil = \sum_{k=1}^{n-1} (\lceil k/2 \rceil - \lceil (k-1)/2 \rceil)/k! - \lceil (n-1)/2 \rceil/n! = \sum_{k \text{ odd}} 1/k! + O(1/(n-1)!)$.

- 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, 0312, 3012, 1032, 0132, 0231, 2031, 3021, 0321, 2301, 3201, 3210, 2310, 1320, 3120, 2130, 1230.

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

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

also $\omega(k) = (0 \ k)$ when k is even, $\omega(k) = (0 \ k-2 \ \dots \ 1 \ k-1 \ k)$ when $k \geq 3$ is odd. Thus when $k \geq 3$ is odd, $\sigma(k, 1) = (k \ k-1 \ 0)$ and $\sigma(k, j)$ takes $k \mapsto j-1$ for $1 < j < k$; when $k \geq 4$ is even, $\sigma(k, j) = (0 \ k \ k-3 \ \dots \ 1 \ k-2 \ k-1)^j$ for $1 \leq j \leq k$.

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

- 41.** We may assume that $r < n$. Algorithm G will generate r -variations for any Sims table if we simply change ‘ $k \leftarrow 1$ ’ to ‘ $k \leftarrow n - r$ ’ in step G3, provided that we redefine $\omega(k)$ to be $\sigma(n - r, n - r) \dots \sigma(k, k)$ 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 $\sigma(k, j) = (k \ j-1 \ \dots \ 1 \ 0)$ and $\omega(k) = (k \ \dots \ 1 \ 0)$ when $k = n - r$; $\sigma(k, j) = (k \ \dots \ 1 \ 0)^j$ when $k = n - r + 1$.

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

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

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

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

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

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

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

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

- 48.** Steps (X2, X3, X4, X5, X6) are performed respectively $A_n + (1, n!, 0, 0, 1)$ times, where $A_n = \sum_{k=1}^{n-1} n^k = n! \sum_{k=1}^{n-1} 1/k! \approx n!(e-1)$. 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 $|s_0| \geq \dots \geq |s_9|$; also prepare tables $w_0 \dots w_9, x_0 \dots x_9, y_0 \dots y_9$, so that the signatures $\{s_k, \dots, s_9\}$ are $w_{x_k} \leq \dots \leq w_{y_k}$. For example, when SEND + MORE = MONEY we have $(s_0, \dots, s_9) = (-9000, 1000, -900, 91, -90, 10, 1, -1, 0, 0)$ for the respective letters (M, S, O, E, N, R, D, Y, A, B); also $(w_0, \dots, w_9) = (-9000, -900, -90, -1, 0, 0, 1, 10, 91, 1000)$ and $x_0 \dots x_9 = 01122333344, y_0 \dots y_9 = 9988776554$. Yet another table $f_0 \dots f_9$ has $f_j = 1$ if the digit corresponding to w_j cannot be zero; in this case $f_0 \dots f_9 = 1000000001$. These tables make it easy to compute the largest and smallest values of

$$s_k a_k + \dots + s_9 a_9$$

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

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

50. If α is in a permutation group, so are all its powers $\alpha^2, \alpha^3, \dots$, 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 $\alpha^- = \sigma_1 \sigma_2 \dots \sigma_{n-1}$.

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

$$\omega(k) = (n-1 \ \dots \ 0)(n-2 \ \dots \ 0) \dots (k \ \dots \ 0) = \phi(n-1)\phi(k-1)$$

is a k -flip followed by an n -flip. The permutation corresponding to control table $c_0 \dots c_{n-1}$ 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 $c_1 \dots c_n$ 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 ‘(3c)’ 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 $1 \leq i \leq t$ 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:

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

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 $\{1, \dots, k\}$.

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 $\beta_{n!+m} = \sigma_{\rho_!(n!+m)} \dots \sigma_{\rho_!(n!+1)} \beta_{n!} = \sigma_{\rho_!(m)} \dots \sigma_{\rho_!(1)} \beta_{n!} = \beta_m \beta_{n!}$ for $0 \leq m < n \cdot n!$, 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 $0 \leq m < n \cdot n!$.

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

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

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

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

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 $\alpha_{n!-1} = (0 \ n) \beta_n^- (n \ \dots \ 0)$, and this takes $0 \mapsto n-1$ because $\beta_{n!}$ fixes n . (Consequently Algorithm E will define a Hamiltonian *circuit* on the graph of exercise 66 if and only if $\beta_{n!} = (n-1 \ \dots \ 2 \ 1)$, 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 $n \leq 250,000$.)

59. The Cayley graph with generators $(\alpha_1, \dots, \alpha_k)$ in the text's definition is isomorphic to the Cayley graph with generators $(\alpha_1^-, \dots, \alpha_k^-)$ in the alternative definition, since $\pi \rightarrow \alpha_j \pi$ in the former if and only if $\pi^- \rightarrow \pi^- \alpha_j^-$ 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 paths leading from $12\dots n$ to any odd permutation $a_1a_2\dots a_n$. 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 $(\alpha_0, \alpha_1, \dots, \alpha_n)$ are any odd permutations such that $\alpha_j \in A(x_j, x_{j+1})$, then $(12)\alpha_j$ is an even permutation in $A(x_{j+1}, x_j)$. Consequently, if $x_1x_2\dots x_n$ is a permutation of $\{1, 2, \dots, n\}$, there is at least one Hamiltonian path of the form

$$(12)\alpha_0 \longrightarrow \dots \longrightarrow \alpha_1 \longrightarrow (12)\alpha_1 \longrightarrow \dots \longrightarrow \alpha_2 \longrightarrow \dots \longrightarrow (12)\alpha_{n-1} \longrightarrow \dots \longrightarrow \alpha_n;$$

the subpath from $(12)\alpha_{j-1}$ 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 $\alpha_0 = 21\dots n$ and $\alpha_n = a_1a_2\dots a_n$; there are $(n-2)!$ ways to choose $x_2\dots x_{n-1}$, and $(n-2)!/2$ ways to choose each of $\alpha_1, \dots, \alpha_{n-1}$.

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

$$\alpha \longrightarrow (12)\alpha_1 \longrightarrow \dots \longrightarrow \alpha_2 \longrightarrow \dots \longrightarrow (12)\alpha_{n-1} \longrightarrow \dots \longrightarrow \alpha_n \longrightarrow \alpha',$$

using a construction like the Hamiltonian path above but now with $\alpha_1 = \alpha$, $\alpha_n = (12)\alpha'$, $x_1 = 1$, $x_2 = j$, $x_n = j'$, and $x_{n+1} = 1$. (In this case the permutations $\alpha_1, \dots, \alpha_n$ might all be even.)

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

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

$$\alpha = 12343234321232121232321232121234343212123432123432121232321,$$

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

$$\alpha = \beta 2\beta 4\beta 4\beta 4, \quad \beta = 32121232123.$$

65. Such a path exists for any given $N \leq n!$: Let the N th permutation be $\alpha = a_1 \dots a_n$, and let $j = a_1$. Also let Π_k be the set of all permutations $\beta = b_1 \dots b_n$ 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 $\Pi_j \cup \Pi_1 \cup \dots \cup \Pi_{k-1}$ for $2 \leq k \leq j$, successively combining P_{k-1} with a Gray code for Π_{k-1} . (See the “absorption” construction of answer 62. In fact, P_j will be a Gray code 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 $(xyzyzyxzyzy)^2$. (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 circuits, 240 of which lead to delta sequences of the form α^5 in which α uses each transposition 6 times (for example, $\alpha = 354232534234532454352452$). The total number of solutions to this problem is probably about 10^{16} .

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

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

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

1234	1243	1423	4123	4132	1432	1342	1324	3124	3142	3412	4312
↓	↑	↓	↑	↓	↑	↓	↑	↓	↑	↓	↑
54321	24351	24153	54123	14523	14325	24315	24513	54213	14253	14352	54312
12345	15342	35142	32145	32541	52341	51342	31542	31245	35241	25341	21345
15342	12435	32415	35412 ← 31452	51432	52431	32451 ← 35421	31425	21435	21435	25431	
23451	53421	51423	21453 → 25413	23415	13425	15423 → 12453	52413	53412	13452		
21543	51243	53241	23541	23145	25143	15243	13245	13542	53142	52143	12543
34512	34215	14235	14532	54132	34152	34251	54231	24531	24135	34125	34521
32154 → 35124	15324 → 12354	52314	32514 ← 31524	51324	21354 → 25314	35214 → 31254					
45123 ← 42153	42351 ← 45321	41325	41523 → 42513	42315	45312 ← 41352	41253 ← 45213					
43215	43512 ← 41532	41235	45231 → 43251	43152 → 45132	42135	42531 ← 43521	43125				
51234	21534 → 23514	53214	13254 ← 15234	25134 ← 23154	53124	13524 → 12534	52134				
↓	↑	↓	↑	↓	↑	↓	↑	↓	↑	↓	↑

Here the columns represent sets of permutations that are cyclically rotated and/or reflected in all $2n$ ways; therefore each column contains exactly one “rosary permutation” (exercise 18). We can use Algorithm P to run through the rosary permutations systematically, knowing that the pair xy will occur before yx in its column, at which time τ' instead of ρ' will move us to the right or to the left. Step Z2 omits the interchange $a_1 \leftrightarrow a_2$, thereby causing the permutations $a_1 \dots a_{n-1}$ 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 $1 \dots n$ by $24 \dots 31$ and change $A_1 \dots A_n$ to $A_1 A_n A_2 A_{n-1} \dots$, 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 circuit when $\rho\tau = \tau\rho$ [see I. Pak and R. Radoičić, “Hamiltonian expanders,” 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

$$\begin{aligned}\alpha &= \tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^5\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^1\tau\sigma^1; \\ \beta &= \sigma^3\tau\sigma^5\tau\sigma^2\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^5\tau\sigma^1\tau\sigma^3\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^1\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^4; \\ \gamma &= \sigma\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^5\tau\sigma^1\tau\sigma^3\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^1\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^3\tau\sigma^2 \\ &\quad \tau\sigma^5\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^5\tau\sigma^2\tau\sigma^1\tau\sigma^5\tau\sigma^1\tau\sigma^3\tau\sigma^5\tau\sigma^1\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^2.\end{aligned}$$

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. 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 $x \mapsto x$.

72. (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 circuit 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.]

73. The representation $\beta^j\gamma^k$ is unique if we require $0 \leq j < g/c$ and $0 \leq k < c$. 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 circuit, 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 $j \geq 0$ there is a $k \leq j$ 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 .

74. 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 path 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 circuit if and only if m is a multiple of n . It is natural to conjecture (falsely) that at least one Hamiltonian circuit 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 $C_m \times C_n$.

75. 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

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

76.

01	*	Permutation generator \`a la Heap		
02	N	IS	10	The value of n (3 or more, not large)
03	t	IS	\$255	
04	j	IS	\$0	$8j$
05	k	IS	\$1	$8k$
06	ak	IS	\$2	
07	aj	IS	\$3	
08		LOC	Data_Segment	
09	a	GREG	0	Base address for $a_0 \dots a_{n-1}$
10	A0	IS	0	
11	A1	IS	0+8	
12	A2	IS	0+16	
13		LOC	0+8*N	Space for $a_0 \dots a_{n-1}$
14	c	GREG	0-8*3	Location of $8c_0$
15		LOC	0-8*3+8*N	$8c_3 \dots 8c_{n-1}$, initially zero
16		OCTA	-1	$8c_n = -1$, a convenient sentinel
17	u	GREG	0	Contents of a_0 , except in inner loop
18	v	GREG	0	Contents of a_1 , except in inner loop
19	w	GREG	0	Contents of a_2 , except in inner loop
20		LOC	#100	
21	1H	STCO	0,c,k	$B - A \quad c_k \leftarrow 0$.
22		INCL	k,8	$B - A \quad k \leftarrow k + 1$.
23	OH	LDO	j,c,k	$B \quad j \leftarrow c_k$.
24		CMP	t,j,k	B
25		BZ	t,1B	$B \quad$ Loop if $c_k = k$.
26		BN	j,Done	$A \quad$ Terminate if $c_k < 0$ ($k = n$).
27		LDO	ak,a,k	$A - 1 \quad$ Fetch a_k .
28		ADD	t,j,8	$A - 1$
29		STO	t,c,k	$A - 1 \quad c_k \leftarrow j + 1$.
30		AND	t,k,#8	$A - 1$
31		CSZ	j,t,0	$A - 1 \quad$ Set $j \leftarrow 0$ if k is even.
32		LDO	aj,a,j	$A - 1 \quad$ Fetch a_j .
33		STO	ak,a,j	$A - 1 \quad$ Replace it by a_k .
34		CSZ	u,j,ak	$A - 1 \quad$ Set $u \leftarrow a_k$ if $j = 0$.
35		SUB	j,j,8	$A - 1 \quad j \leftarrow j - 1$.
36		CSZ	v,j,ak	$A - 1 \quad$ Set $v \leftarrow a_k$ if $j = 0$.
37		SUB	j,j,8	$A - 1 \quad j \leftarrow j - 1$.
38		CSZ	w,j,ak	$A - 1 \quad$ Set $w \leftarrow a_k$ if $j = 0$.
39		STO	aj,a,k	$A - 1 \quad$ 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 $u \leftrightarrow w$.
57	SET u,w	A	
58	SET w,t	A	
59	SET k,8*3	A	$k \leftarrow 3$.
60	JMP OB	A	
61	Main LDO u,A0	1	
62	LDO v,A1	1	
63	LDO w,A2	1	
64	JMP Inner	1	■

77. 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 $((2r! + 2)A + 2B + r - 5)\mu + ((2r! + 2r + 7 + (r - 2)[r \text{ even}])A + 7B - r - 4)v$, where $A = n!/r!$ and $B = n!(1/r! + \dots + 1/n!)$.

78. 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'.

79. 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 c_1, \dots, c_5 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.

80. t IS \$255 ;k IS \$0 ;kk IS \$1 ;c IS \$2 ;d IS \$3
 SET k,1 $k \leftarrow 1$.
 3H SRU d,a,60 $d \leftarrow$ leftmost nybble.
 SLU a,a,4 $a \leftarrow 16a \bmod 16^{16}$.
 CMP c,d,k
 SLU kk,k,2
 SLU d,d,kk
 OR t,t,d $t \leftarrow t + 16^k d$.
 PBNZ c,1B Return to main loop if $d \neq k$.
 INCL k,1 $k \leftarrow k + 1$.
 PBNZ a,3B Return to second loop if $k < n$. ■

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

82. 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,OB; INCL p,8
3H    LDOU c,p,0; PBNZ c,OB

```

■

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

84. 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 $1 \leq j \leq n$ (the inverse permutation). Then set $k \leftarrow 0$, and for $j = n, n-1, \dots, 2$ (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 $j = 2, \dots, n-1, n$ (in this order) set $t \leftarrow (k \bmod j) + 1$, $a_j \leftarrow a_t$, $a_t \leftarrow j$, $k \leftarrow [k/j]$. [See S. Pleszczyński, *Inf. Proc. Letters* **3** (1975), 180–183; W. Myrvold and F. Ruskey, *Inf. Proc. Letters* **79** (2001), 281–284.]

85. 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 .

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

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

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

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

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

(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 j and k are both maximal, they both are output before any element x with $x \prec j$ or $x \prec k$, because steps T5 and T7 keep maximal elements in a queue (not a stack). Thus if k is nontrivial and output first, the next nontrivial element will not be output before j ; and k is unrelated to j .

(e) Let the nontrivial t 's be $s_1 < s_2 < \dots < s_r = N$. Then we have $s_j \geq 2s_{j-2}$, by (c). Consequently $M = s_1 + \dots + s_r \leq s_r(1 + \frac{1}{2} + \frac{1}{4} + \dots) + s_{r-1}(1 + \frac{1}{2} + \frac{1}{4} + \dots) < 4s_r$. (A sharper estimate is probably possible, but exercise 89 shows that M can be larger than $2.6N$.)

89. The number N of such permutations is F_{n+1} by exercise 5.2.1–25. Therefore $M = F_{n+1} + \dots + F_1 = F_{n+3} - 1 \approx \phi^2 N$. Notice incidentally that all such permutations satisfy $a_1 \dots a_n = a'_1 \dots a'_n$. They can be arranged in a Gray path (exercise 7.2.1.1–89).

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

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

91. 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 $1 \leq n \leq 12$ 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 $n \rightarrow \infty$ have been deduced by Brightwell, Prömel, and Steger [*J. Combinatorial Theory A* **73** (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 $n \leq 5$ by S. P. Avann [*Aequationes Math.* **8** (1972), 95–102].

92. The basic idea is to introduce dummy elements $n + 1$ and $n + 2$ with $j \prec n + 1$ and $j \prec n + 2$ for $1 \leq j \leq n$, 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 $a_1 \dots a_{n-2}$ 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.]

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

Matchings can also be regarded as involutions of $\{1, \dots, 2n\}$ 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, $C_1 = C_3 = \dots = C_{2n-1} = 0$ and the n -tuples $(C_2, C_4, \dots, C_{2n})$ run through a reflected Gray code on the radices $(2n - 1, 2n - 3, \dots, 1)$. 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.]

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

95. For example, we can start with $a_1 \dots a_{n-1} a_n = 2 \dots n1$ and $b_1 b_2 \dots b_n b_{n+1} = 12 \dots n1$, and use Algorithm P to generate the $(n - 1)!$ permutations $b_2 \dots b_n$ of $\{2, \dots, n\}$. 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 $a_1 \dots a_n$.

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

97. Using the notation of exercise 47, we have $N_k = \sum \binom{k}{j} (-1)^j (n - j)^{k-j}$ 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.]

98. Suppose L_n generates $D_n \cup D_{n-1}$, beginning with $(1 \ 2 \ \dots \ n)$, then $(2 \ 1 \ \dots \ n)$, and ending with $(1 \ \dots \ n-1)$; for example, $L_3 = (1 \ 2 \ 3), (2 \ 1 \ 3), (1 \ 2)$. Then we can generate

D_{n+1} as $K_{nn}, \dots, K_{n2}, K_{n1}$, where $K_{nk} = (1\ 2\ \dots\ n)^{-k}(n\ n+1)L_n(1\ 2\ \dots\ n)^k$; 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 $(k+1\ \dots\ n\ 1\ \dots\ k\ n+1)$ and ends with $(k+1\ \dots\ n\ 1\ \dots\ k-1)(k\ n+1)$; 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 $(2\ 1\ 3\ \dots\ n)$, from which we can return to $(1\ 2\ \dots\ n)$ by following the cycle for D_n backwards, thereby completing the list L_{n+1} as desired.

99. Use Algorithm X, with $t_k(a_1, \dots, a_k) = 'p > 0 \text{ or } l[q] \neq k+1'$.

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. Paris* **A275** (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.

100. 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 $a_1 \dots a_n$ 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 $k \leq n$. 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.)

101. 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} , \dots , K_{nn}^R , where $K_{nk} = (k\ \dots\ n)^-L_{n-1}(k\ \dots\ n)(k\ n+1)$. 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 , $K_{n(n-1)}$, $K_{n(n-2)}^R$, \dots , 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 93.

102. 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 $n \geq 4$; 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 k th ρ_{n-1} of A_n by $\rho_n A'_n \rho_n$, where $k = 1, 2, 4, \dots, 2n - 2$ if n is even and $k = 1, 3, \dots, 2n - 3, 2n - 2$ if n is

odd, and where A'_n is A_n with its first or last element deleted. Then, if we begin with $a_1 \dots a_n = 1 \dots n$, the operations ρ_{n-1} of A_n will cause position a_n to run through the successive values $n \rightarrow p_1 \rightarrow n \rightarrow p_2 \rightarrow \dots \rightarrow p_{n-1} \rightarrow n$, where $p_1 \dots p_{n-1} = (n-1 - [n \text{ even}]) \dots 4213 \dots (n-1 - [n \text{ odd}])$; the final permutation will again be $1 \dots n$.

103. (a) A well-balanced permutation has $\sum_{k=1}^n k a_k = n(n+1)^2/4$, 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 $\sum k a_k$ 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 $1 \leq n \leq 15$ are 0, 0, 0, 2, 6, 0, 184, 936, 6688, 0, 420480, 4298664, 44405142, 0, 6732621476.

104. (a) For each permutation $a_1 \dots a_n$, 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.)

(b) Start with $a_0 a_1 \dots a_n a_{n+1} = 01 \dots 11$. Perform Algorithm L until it stops with $j = 0$. Find k such that $a_1 > \dots > a_k = a_{k+1}$, and terminate if $k = n$. Otherwise set $a_l \leftarrow a_{k+1} + 1$ for $1 \leq l \leq k$ and go to step L4. [See M. Mor and A. S. Fraenkel, *Discrete Math.* **48** (1984), 101–112.]

105. 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 $1 \leq j < n$.)

106. 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 $1* \dots *$ 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 $a_1 \dots a_n$, 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 $21* \dots *$, $3*1* \dots *$, \dots , $n* \dots *$. 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 $* \dots *$ and turning over the top card when it is face down, running through all $(n-1)!$ permutations of $\{2, \dots, n\}$ as cards are turned. If the bottom $n-m$ cards are known to be equal to $(m+1)(m+2) \dots n$, 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, ..., 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.

107. This result holds for any game in which

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

when $a_1 = k$, where $p(k,2) \dots p(k,k-1)$ is an arbitrary permutation of $\{2, \dots, k-1\}$. Suppose a_1 takes on exactly m distinct values $d(1) < \dots < d(m)$ 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 $r + 1 \leq F_m + 1 \leq F_{m+1}$. Otherwise $r \leq F_{m-1}$, and at most F_m further permutations follow step r . [SIAM Review 19 (1977), 239–241.]

The values of $f(n)$ for $1 \leq n \leq 16$ 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, 4, 6, 1) ways. The unique longest-winded permutation for $n = 16$ is

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

108. The forward method of answer 106 suggests that $f(n)$ probably grows at least as fast as $n \log n$ (by comparison with coupon collecting).

109. For $0 \leq j \leq 9$ construct the bit vectors $A_j = [a_j \in S_1] \dots [a_j \in S_m]$ and $B_j = [j \in S_1] \dots [j \in S_m]$. 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

$$(A_0, \dots, A_9) = (9, 6, 8, b, 5, 4, 0, a, 2, 0), \quad (B_0, \dots, B_9) = (5, 0, 8, 6, 2, a, 4, b, 9, 0),$$

in hexadecimal notation; hence $a_0 \dots a_9 = 8327061549$ 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.

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

Mark Cooke has pointed out that a universal cycle of permutations is also equivalent to a *Hamiltonian* circuit on the Cayley graph with generators $\sigma = (1 \ 2 \ \dots \ n)$ and $\rho = (1 \ 2 \ \dots \ n-1)$. For example, the cycle just given for $n = 4$ corresponds to $\sigma^3 \rho^2 \sigma \rho \sigma^2 \rho^2 \sigma^3 \rho \sigma^2 \rho^2 \sigma \rho \sigma^2 \rho$.

111. By exercise 2.3.4.2–22 it suffices to count the oriented trees rooted at $12\dots(n-2)$, in the digraph of the preceding answer. For $n = 2, 3, 4, 5, 6$, the numbers can be calculated by exercise 2.3.4.2–19, and they turn out to be tantalizingly simple: 1, 3, $2^7 \cdot 3$, $2^{33} \cdot 3^8 \cdot 5^3$, $2^{190} \cdot 3^{49} \cdot 5^{33}$. (Here we consider (121323) to be the same cycle as (213231), but different from (131232).) The graph has symmetries related to Young tableaux in beautiful ways, so the answer to this problem should be quite instructive.

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.

- 0-origin indexing, 8.
4-cube, 9–10.
 $K\mu$: *see* Kilomems.
 $M\mu$: *see* Megamems.
 π (circle ratio), 27, 30, 36.
 σ – τ path, 20–21, 33.
 $\phi(k)$ permutation, 12–13, 31.
- Additive alphametics, 6–7, 14–15, 30.
Adjacent interchanges, 2–7, 31, 35, 54, 55.
Akl, Selim George (سلیم جورج عقل), 53.
Alphametics, 6.
 additive, 6–7, 14–15, 30.
 doubly true, 29.
 multiplicative, 29.
 pure, 7, 28–29.
Alternating group, 5, 36.
Analysis of algorithms, 26–31, 34–35.
Applying a permutation, 8–10.
Arisawa, Makoto (有澤誠), 43.
Artificial intelligence, 28.
Ascents, 55.
Assignment problem, 26.
Automorphisms, 9–10, 28, 29.
Avann, Sherwin Parker, 53.
- Balanced permutation, 36.
Barwell, Brian R., 29.
Beidler, John Anthony, 6.
Bell ringing, 1, 4–5, 21.
Bernoulli, Jacques (= Jakob = James), 39.
Breisch, Richard L., 42.
Brightwell, Graham Richard, 53.
Bruijn, Nicolaas Govert de, cycle, 37.
Bubble sort, 3.
Buckley, Michael R. W., 28.
Bypassing blocks of permutations, 13–16, 30, 53.
- Cambridge Forty-Eight, 4, 5.
Canfield, Earl Rodney, 53.
Casting out nines, 43.
Cayley, Arthur, 20.
 graphs, 20, 31–34, 48, 56.
Cesare, Giulio (pen name of Dani Ferrari, Luigi Rafaiani, Luigi Morelli, and Dario Uri), 42.
Change ringing, 1, 4–5, 21.
Childs, Roy Sydney, 42.
Colex order, *see* Reverse colex order.
Complete relation, 36.
Compton, Robert Christopher, 21, 32.
Comtet, Louis, 54.
- Conjugate permutation, 12.
Conway, John Horton, 36.
Cooke, Raymond Mark, 56.
Coroutine, 33.
Coupon collecting, 56.
Cryptarithms, 6.
Cycle structure of a permutation, 8, 12.
Cycle, undirected, 28.
Cyclic permutation, 35.
Cyclic shift, 18, 20, 23, 30.
- de Bruijn, Nicolaas Govert, cycle, 37.
Delta sequence, 31.
Derangements, 35.
Dijkstra, Edsger Wijbe, 4.
Directed torus, 34.
Doubly Gray code, 32.
Doubly true alphabetic, 29.
Dual permutation generation, 17–19, 30.
Duckworth, Richard, iii, 4.
Dudeney, Henry Ernest, 6, 29, 43.
Ehrlich, Gideon (הדריך אהרליץ), 19, 32, 40, 41.
 swap method, 19–20, 31–32.
Enggren, Willy, 28, 29.
Er, Meng Chiau (余明劭), 16.
Erdős, Pál (= Paul), 49.
Euler, Leonhard (Эйлер, Леонард),
 summation formula, 43.
Eulerian circuit in directed graph, 56.
Even permutation, 5, 36.
Exclusive or, 51.
Exponential series, partial sums of, 39.
Extending a partial order, 24.
Factorial number system, 38.
Factorial ruler function, 30.
Ferrari, Dani, 56.
Fibonacci, Leonardo, of Pisa, numbers, 36, 52.
First-element swaps, 19–20, 32.
Fischer, Ludwig Joseph, 39.
Five-letter words, 28.
Flip operation, 12–13, 31, 33, 36, 45.
Fraenkel, Aviezri S (אביעזרי פרנקל), 55.
- Gallian, Joseph Anthony, 50.
Galois, Évariste, 9.
Gardner, Martin, 19, 27.
General permutation generator, 10–13,
 22–23, 29–30.
Generating functions, techniques for
 using, 27, 39–40, 54.
Goldstein, Alan Jay, 23.
Grandsire Doubles, 5.

- Gray binary code, 3.
 Gray code for mixed radices, 3, 40, 45, 49.
 Gray code for permutations, 31–32.
 Gray path for permutations, 32.
 Group of permutations, 9–10, 20, 45.
- h*-ordered permutation, 35.
 Hamilton, William Rowan, circuit, 3, 20–21, 31–34, 48, 56.
 path, 3, 20–21, 32–33, 47–48.
 Hawaii, 28.
 Heap, Brian Richard, 13, 15, 21, 29, 30, 34, 41.
 Hexadecimal digits, 9, 56.
 Hindenburg, Carl Friedrich, 2.
 Hunter, James Alston Hope, 6.
- Identity permutation, 9.
 Image of an element, 8.
 Inclusion and exclusion, 53.
 Indecomposable permutation, 35.
 Internet, ii, iii.
 Inverse permutation, 24–25, 28, 52.
 Inversion tables, 3, 38, 53.
 Inversions of a permutation, 3, 5.
 Involutions, 35–36, 48, 53.
 Ives, Frederick Malcolm, 30.
- Jackson, Bradley Warren, 37.
 Jiang, Ming (姜明), 21.
 Johnson, Allan William, Jr., 42.
 Johnson, Selmer Martin, 28.
- Kahan, Steven Jay, 42.
 Kemp, Rainer, 38.
 Kilomem: One thousand memory accesses.
 Knight's tour, northeasterly, 34.
 Knuth, Donald Ervin (高德纳), i, iv.
 Kompel'makher, Vladimir Leont'evich
 (Компельмакхер, Владимир Леонтьевич), 32.
 Krause, Karl Christian Friedrich, 39.
- Langdon, Glen George, Jr., 19, 23, 34.
 Lehmer, Derrick Henry, 1.
 Lexicographic order, 1, 8.
 Lexicographic permutation generation, 12, 15, 26–27.
 for involutions, 54.
 Lexicographic successor, 2.
 Linear embedding, 24.
 Linked lists, 15–16, 54.
 Lipski, Witold, Jr., 44.
 Liskovets, Valery Anisimovich (Лисковец,
 Валерий Анисимович), 32.
 Loopless generation, 28, 41, 53.
- Macdonald, Peter, 28.
 Matchings, 25, 35.
 Matrix tree theorem, 57.
 Maximal element, 52.
- McCrary, Edwin Parker, Jr., 28.
 Megamem: One million memory accesses.
 Minimal element, 52.
 Mixed-radix number, 17, 27, 38.
 MMIX computer , ii, iv, 21–23, 34.
 Modular Gray code for mixed radices, 49.
 Monte Carlo estimates, 47.
 Mor, Moshe (מור השם), 55.
 Morelli, Luigi, 56.
 Morris, Ernest, 4.
 Multinomial coefficient, 27.
 Multiplication of permutations, 8.
 Multiplicative alphametics, 29.
 Multisets, 1, 3, 24, 27, 33.
 Mundy, Peter, 4.
 MXOR (multiple exclusive-or), 34.
 Myrvold, Wendy Joanne, 52.
- n*-cube, 9–10, 28.
 Nijenhuis, Albert, 20.
 Nijon, Herman, 28.
 Northeasterly knight's tour, 34.
 Nybble: A 4-bit quantity, 22–23.
- Octahedral group, 41.
 Odd permutation, 5, 47–48.
 Ord-Smith, Richard Albert James
 (= Jimmy), 12, 13, 18, 29, 30, 39.
 Order of a group element, 20, 45.
 Organ pipe order, 48, 55.
- Pak, Igor Markovich (Пак, Игорь
 Маркович), 48.
- Parallel computation, 34, 41.
 Partial ordering, 24, 34, 35.
 Partition of a number, 29.
 Pepperdine, Andrew Howard, 55.
 Permutation generation, 1–56.
 cyclic shift method, 18, 20, 23, 30.
 dual, 17–19, 30.
 Ehrlich swap method, 19–20, 31–32.
 fastest, 21–24.
 general, 10–13, 22–23, 29–30.
 lexicographic, 12, 15, 26–27.
 lexicographic with restricted prefixes,
 16, 30, 53.
 plain changes, 4–7, 17, 23, 25,
 27–28, 33, 55.
 when to use, 26.
- Permutations, 1–56.
 applying, 8–10.
 conjugate, 12.
 cycles of, 8, 12.
 cyclic, 35.
 derangements, 35.
 even, 5, 36.
 groups of, 9–10, 20, 45.
 Gray codes for, 31–32.
 h-ordered, 35.
 indecomposable, 35.
 inverse, 24–25, 28, 52.

- inversions of, 3, 5.
- involutions, 35–36, 53.
- multiplication of, 8.
- notations for, 8.
- odd, 5, 47–48.
- of a multiset, 1–2, 24.
- rank of, 27, 34, 52.
- sign of, 5, 33.
- signed, 28.
- universal cycle of, 37.
- up-down, 35.
- well-balanced, 36.
- Phillips, John Patrick Norman, 38.
- Pi (π), 27, 30, 36.
- Plain changes, 4–7, 17, 23, 25, 27–28, 33, 55.
- Pleszczyński, Stefan, 52.
- Postmultiplication, 9.
- Preferential arrangements, 55.
- Premultiplication, 9, 11–12, 14, 54.
- Preorder in a tree, 11, 14.
- Prömel, Hans Jürgen, 53.
- Pruesse, Gara, 53.
- Pure alphabetic, 7, 28–29.
- Queue, 52.
- Radoičić, Radoš, 48.
- Rafaiani, Luigi, 56.
- Rank of a permutation, 27, 34, 52.
- Rankin, Robert Alexander, 20, 33, 49.
- Rapoport, Elvira Strasser, 48.
- Reflected Gray code for mixed radices, 3, 40, 45, 53.
- Reversal of a string, 31, 36.
- Reverse colex order, 8, 12, 15, 17, 26, 38.
- Reversing, 2, 38, 40, *see also* Flip operation.
- Roman numerals, 42.
- Rosary permutations, 28, 48.
- Rotem, Doron (רותם דורון), 22, 25.
- Rothe, Heinrich August, 38.
- Roy, Mohit Kumar (মোহিত রূপাল রায়), 41.
- Rüdiger, Christian Friedrich, 2.
- Ruskey, Frank, 21, 34, 47, 52, 53.
- Savage, Carla Diane, 47, 48, 54.
- Sayers, Dorothy Leigh, 1.
- Sedgewick, Robert, 21.
- Seitz, R., 19.
- Sign of a permutation, 5, 33.
- Signature of an alphabetic, 6.
- Signed permutation, 28.
- Silver, Alfred Lindsey Leigh, 46.
- Sims, Charles Coffin, 9.
 - tables, 9–15, 17–18, 29–30.
- Skipping blocks of permutations, 13–16, 30, 53.
- Stanford GraphBase, ii, iii.
- Star graph, 32.
- Star transpositions, 19–20, 32.
- Stedman, Fabian, 4.
 - Doubles, 5.
- Steger, Angelika, 53.
- Swapping with the first element, 19–20, 32.
- Symmetries, 9–10, 28, 29.
- Tchuente, Maurice, 47.
- Tic-tac-toe board, 29.
- Tompkins, Charles Brown, 19.
- Topological sorting, 24–26, 34–35.
- Topswops, 36.
- Torus, 9.
 - directed, 34.
 - twisted, 32.
- Total ordering, 24.
- Transitive relation, 34, 36.
- Traveling salesrep problem, 26.
- Trotter, Hale Freeman, 5.
- Trotter, William Thomas, 49.
- Twisted torus, 32.
- Two-line form of permutation, 8.
- Undirected cycle, 28.
- Undoing, 16, 54.
- Universal cycle of permutations, 37.
- Unranking, 34.
- Up-down permutation, 35.
- Uri, Dario, 56.
- Variations, 27, 30, 42.
- Varol, Yaakov Leon (וַאֲקָב לָאוֹן), 22, 25.
- Vatriquant, Simon, 6.
- Vinnicombe, Robert Ian James, 28.
- Walsh, Timothy Robert Stephen, 54, 55.
- Wayne, Alan, 29.
- Weak order, 36.
- Well-balanced permutation, 36.
- Wells, Mark Brimhall, 43.
- Weston, Andrew, 21.
- White, Arthur Thomas, II, 5.
- Wilf, Herbert Saul, 20, 55.
- Williamson, Stanley Gill, 21, 32, 53.
- Wilson, Wilfrid George, 5.
- XOR (bitwise exclusive-or), 51.
- Yoshigahara, Nobuyuki (= Nob) (吉川伸之), 29.
- Young, Alfred, tableaux, 24–25, 57.

THE ART OF COMPUTER PROGRAMMING

PRE-FASCICLE 2C

A DRAFT OF SECTION 7.2.1.3: GENERATING ALL COMBINATIONS

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.

Copyright © 2002 by Addison-Wesley

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

Zeroth printing (revision 1), 16 June 2002

PREFACE

*[The Art of Combinations] has a relation
to almost every species of useful knowledge
that the mind of man can be employed upon.*

— JACQUES BERNOULLI, *Ars Conjectandi* (1713)

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. Those 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 7.2.1.3 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,” and Section 7.2.1.2, “Generating all permutations.” (Readers of the present booklet should have already looked at those sections, drafts of which are available as Prefascicles 2A and 2B.) The stage is now set for the main contents of this booklet, Section 7.2.1.3: “Generating all combinations.” Then will come Section 7.2.1.4 (about partitions), 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 combination generation turns out to be surprisingly rich, with ties to Sections 1.2.1, 1.2.4, 1.2.6, 2.3.2, 2.3.4.2, 4.3.2, 4.6.1, 4.6.2, 5.1.2, 5.4.1, 5.4.2, 6.1, and 6.3 of the first three volumes. I strongly believe in building up a firm foundation, so I have discussed this topic much more thoroughly than I will be able to do with material that is newer or less basic. To my surprise, I came up with 109 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 53, 55, 66, and 82; I’ve also implicitly posed additional unsolved questions in the answers to exercises 62, 100, 104, and 108. 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 9, 18, 19, 20, 26, 27, 28, 30, 31, 32, 33, 34, 35, 36, 37, 41, 42, 43, 44, 45, 48, 49, 51, 58, 61, 62, 63, 64, 65, 68, 78, 81(b–f), 84, 85, 86, 92, and/or 109.

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

Happy reading!

*Stanford, California
13 June 2002*

D. E. K.

7.2.1.3. Generating all combinations. Combinatorial mathematics is often described as “the study of permutations, combinations, etc.,” so we turn our attention now to combinations. A *combination of n things, taken t at a time*, often called simply a t -combination of n things, is a way to select a subset of size t from a given set of size n . We know from Eq. 1.2.6–(2) that there are exactly $\binom{n}{t}$ ways to do this; and we learned in Section 3.4.2 how to choose t -combinations at random.

Selecting t of n objects is equivalent to choosing the $n - t$ elements not selected. We will emphasize this symmetry by letting

$$n = s + t \quad (1)$$

throughout our discussion, and we will often refer to a t -combination of n things as an “ (s, t) -combination.” Thus, an (s, t) -combination is a way to subdivide $s + t$ objects into two collections of sizes s and t .

*If I ask how many combinations of 21 can be taken out of 25,
I do in effect ask how many combinations of 4 may be taken.
For there are just as many ways of taking 21 as there are of leaving 4.*
— AUGUSTUS DE MORGAN, *An Essay on Probabilities* (1838)

There are two main ways to represent (s, t) -combinations: We can list the elements $c_t \dots c_2 c_1$ that have been selected, or we can work with binary strings $a_{n-1} \dots a_1 a_0$ for which

$$a_{n-1} + \dots + a_1 + a_0 = t. \quad (2)$$

The latter representation has s 0s and t 1s, corresponding to elements that are unselected or selected. The list representation $c_t \dots c_2 c_1$ tends to work out best if we represent the objects as subsets of the set $\{0, 1, \dots, n - 1\}$ and if we list them in *decreasing* order:

$$n > c_t > \dots > c_2 > c_1 \geq 0. \quad (3)$$

Binary notation connects these two representations nicely, because the item list $c_t \dots c_2 c_1$ corresponds to the sum

$$2^{c_t} + \dots + 2^{c_2} + 2^{c_1} = \sum_{k=0}^{n-1} a_k 2^k = (a_{n-1} \dots a_1 a_0)_2. \quad (4)$$

Of course we could also list the positions $b_s \dots b_2 b_1$ of the 0s in $a_{n-1} \dots a_1 a_0$, where

$$n > b_s > \dots > b_2 > b_1 \geq 0. \quad (5)$$

Combinations are important not only because subsets are omnipresent in mathematics but also because they are equivalent to many other configurations. For example, every (s, t) -combination corresponds to a combination of $s + 1$ things taken t at a time *with repetitions permitted*, also called a *multicombination*, namely a sequence of integers $d_t \dots d_2 d_1$ with

$$s \geq d_t \geq \dots \geq d_2 \geq d_1 \geq 0. \quad (6)$$

One reason is that $d_t \dots d_2 d_1$ solves (6) if and only if $c_t \dots c_2 c_1$ solves (3), where

$$c_t = d_t + t - 1, \dots, c_2 = d_2 + 1, c_1 = d_1 \quad (7)$$

(see exercise 1.2.6–60). And there is another useful way to relate combinations with repetition to ordinary combinations, suggested by Solomon Golomb [AMM 75 (1968), 530–531], namely to define

$$e_j = \begin{cases} c_j, & \text{if } c_j \leq s; \\ e_{c_j-s}, & \text{if } c_j > s. \end{cases} \quad (8)$$

In this form the numbers $e_t \dots e_1$ don't necessarily appear in descending order, but the multiset $\{e_1, e_2, \dots, e_t\}$ is equal to $\{c_1, c_2, \dots, c_t\}$ if and only if $\{e_1, e_2, \dots, e_t\}$ is a set. (See Table 1 and exercise 1.)

An (s, t) -combination is also equivalent to a *composition* of $n + 1$ into $t + 1$ parts, namely an ordered sum

$$n + 1 = p_t + \dots + p_1 + p_0, \quad \text{where } p_t, \dots, p_1, p_0 \geq 1. \quad (9)$$

The connection with (3) is now

$$p_t = n - c_t, \quad p_{t-1} = c_t - c_{t-1}, \dots, \quad p_1 = c_2 - c_1, \quad p_0 = c_1 + 1. \quad (10)$$

Equivalently, if $q_j = p_j - 1$, we have

$$s = q_t + \dots + q_1 + q_0, \quad \text{where } q_t, \dots, q_1, q_0 \geq 0, \quad (11)$$

a composition of s into $t + 1$ *nonnegative* parts, related to (6) by setting

$$q_t = s - d_t, \quad q_{t-1} = d_t - d_{t-1}, \dots, \quad q_1 = d_2 - d_1, \quad q_0 = d_1. \quad (12)$$

Furthermore it is easy to see that an (s, t) -combination is equivalent to a path of length $s + t$ from corner to corner of an $s \times t$ grid, because such a path contains s vertical steps and t horizontal steps. Thus, combinations can be studied in at least eight different guises. Table 1 illustrates all $\binom{6}{3} = 20$ possibilities in the case $s = t = 3$.

These cousins of combinations might seem rather bewildering at first glance, but most of them can be understood directly from the binary representation $a_{n-1} \dots a_1 a_0$. Consider, for example, the “random” bit string

$$a_{23} \dots a_1 a_0 = 011001001000011111101101, \quad (13)$$

Table 1
THE (3,3)-COMBINATIONS AND THEIR EQUIVALENTS

$a_5a_4a_3a_2a_1a_0$	$b_3b_2b_1$	$c_3c_2c_1$	$d_3d_2d_1$	$e_3e_2e_1$	$p_3p_2p_1p_0$	$q_3q_2q_1q_0$	path
000111	543	210	000	210	4111	3000	■■■■
001011	542	310	100	310	3211	2100	■■■■
001101	541	320	110	320	3121	2010	■■■■
001110	540	321	111	321	3112	2001	■■■■
010011	532	410	200	010	2311	1200	■■■■
010101	531	420	210	020	2221	1110	■■■■
010110	530	421	211	121	2212	1101	■■■■
011001	521	430	220	030	2131	1020	■■■■
011010	520	431	221	131	2122	1011	■■■■
011100	510	432	222	232	2113	1002	■■■■
100011	432	510	300	110	1411	0300	■■■■
100101	431	520	310	220	1321	0210	■■■■
100110	430	521	311	221	1312	0201	■■■■
101001	421	530	320	330	1231	0120	■■■■
101010	420	531	321	331	1222	0111	■■■■
101100	410	532	322	332	1213	0102	■■■■
110001	321	540	330	000	1141	0030	■■■■
110010	320	541	331	111	1132	0021	■■■■
110100	310	542	332	222	1123	0012	■■■■
111000	210	543	333	333	1114	0003	■■■■

which has $s = 11$ zeros and $t = 13$ ones, hence $n = 24$. The dual combination $b_s \dots b_1$ lists the positions of the zeros, namely

$$23\ 20\ 19\ 17\ 16\ 14\ 13\ 12\ 11\ 4\ 1,$$

because the leftmost position is $n - 1$ and the rightmost is 0. The primal combination $c_t \dots c_1$ lists the positions of the ones, namely

$$22\ 21\ 18\ 15\ 10\ 9\ 8\ 7\ 6\ 5\ 3\ 2\ 0.$$

The corresponding multicomposition $d_t \dots d_1$ lists the number of 0s to the right of each 1:

$$10\ 10\ 8\ 6\ 2\ 2\ 2\ 2\ 2\ 1\ 1\ 0.$$

The composition $p_t \dots p_0$ lists the distances between consecutive 1s, if we imagine additional 1s at the left and the right:

$$2\ 1\ 3\ 3\ 5\ 1\ 1\ 1\ 1\ 1\ 2\ 1\ 2\ 1.$$

And the nonnegative composition $q_t \dots q_0$ counts how many 0s appear between “fenceposts” represented by 1s:

$$1\ 0\ 2\ 2\ 4\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0;$$

thus we have

$$a_{n-1} \dots a_1 a_0 = 0^{q_t} 1 0^{q_{t-1}} \dots 1 0^{q_1} 1 0^{q_0}. \quad (14)$$

The paths in Table 1 also have a simple interpretation (see exercise 2).

Lexicographic generation. Table 1 shows combinations $a_{n-1} \dots a_1 a_0$ and $c_t \dots c_1$ in lexicographic order, which is also the lexicographic order of $d_t \dots d_1$. Notice that the dual combinations $b_s \dots b_1$ and the corresponding compositions $p_t \dots p_0, q_t \dots q_0$ then appear in *reverse* lexicographic order.

Lexicographic order usually suggests the most convenient way to generate combinatorial configurations. Indeed, Algorithm 7.2.1.2L already solves the problem for combinations in the form $a_{n-1} \dots a_1 a_0$, since (s, t) -combinations in bitstring form are the same as permutations of the multiset $\{s \cdot 0, t \cdot 1\}$. That general-purpose algorithm can be streamlined in obvious ways when it is applied to this special case. (See also exercise 7.1–00, which presents a remarkable sequence of seven bitwise operations that will convert any given binary number $(a_{n-1} \dots a_1 a_0)_2$ to the lexicographically next t -combination, assuming that n does not exceed the computer’s word length.)

Let’s focus, however, on generating combinations in the other principal form $c_t \dots c_2 c_1$, which is more directly relevant to the ways in which combinations are often needed, and which is more compact than the bit strings when t is small compared to n . In the first place we should keep in mind that a simple sequence of nested loops will do the job nicely when t is very small. For example, when $t = 3$ the following instructions suffice:

For $c_3 = 2, 3, \dots, n - 1$ (in this order) do the following:
 For $c_2 = 1, 2, \dots, c_3 - 1$ (in this order) do the following:
 For $c_1 = 0, 1, \dots, c_2 - 1$ (in this order) do the following:
 Visit the combination $c_3 c_2 c_1$.

(See the analogous situation in 7.2.1.1–(3).)

On the other hand when t is variable or not so small, we can generate combinations lexicographically by following the general recipe discussed after Algorithm 7.2.1.2L, namely to find the rightmost element c_j that can be increased and then to set the subsequent elements $c_{j-1} \dots c_1$ to their smallest possible values:

Algorithm L (*Lexicographic combinations*). This algorithm generates all t -combinations $c_t \dots c_2 c_1$ of the n numbers $\{0, 1, \dots, n - 1\}$, given $n \geq t \geq 0$. Additional variables c_{t+1} and c_{t+2} are used as sentinels.

- L1.** [Initialize.] Set $c_j \leftarrow j - 1$ for $1 \leq j \leq t$; also set $c_{t+1} \leftarrow n$ and $c_{t+2} \leftarrow 0$.
- L2.** [Visit.] Visit the combination $c_t \dots c_2 c_1$.
- L3.** [Find j .] Set $j \leftarrow 1$. Then, while $c_j + 1 = c_{j+1}$, set $c_j \leftarrow j - 1$ and $j \leftarrow j + 1$; repeat until $c_j + 1 \neq c_{j+1}$.
- L4.** [Done?] Terminate the algorithm if $j > t$.
- L5.** [Increase c_j .] Set $c_j \leftarrow c_j + 1$ and return to L2. ▀

The running time of this algorithm is not difficult to analyze. Step L3 sets $c_j \leftarrow j - 1$ just after visiting a combination for which $c_{j+1} = c_1 + j$, and the number of such combinations is the number of solutions to the inequalities

$$n > c_t > \dots > c_{j+1} \geq j; \quad (16)$$

but this formula is equivalent to a $(t - j)$ -combination of the $n - j$ objects $\{n-1, \dots, j\}$, so the assignment $c_j \leftarrow j - 1$ occurs exactly $\binom{n-j}{t-j}$ times. Summing for $1 \leq j \leq t$ tells us that the loop in step L3 is performed

$$\binom{n-1}{t-1} + \binom{n-2}{t-2} + \dots + \binom{n-t}{0} = \binom{n-1}{s} + \binom{n-2}{s} + \dots + \binom{s}{s} = \binom{n}{s+1} \quad (17)$$

times altogether, or an average of

$$\binom{n}{s+1} / \binom{n}{t} = \frac{n!}{(s+1)!(t-1)!} / \frac{n!}{s!t!} = \frac{t}{s+1} \quad (18)$$

times per visit. This ratio is less than 1 when $t \leq s$, so Algorithm L is quite efficient in such cases.

But the quantity $t/(s+1)$ can be embarrassingly large when t is near n and s is small. Indeed, Algorithm L occasionally sets $c_j \leftarrow j - 1$ needlessly, at times when c_j already equals $j - 1$. Further scrutiny reveals that we need not always search for the index j that is needed in steps L4 and L5, since the correct value of j can often be predicted from the actions just taken. For example, after we have increased c_4 and reset $c_3c_2c_1$ to their starting values 210, the next combination will inevitably increase c_3 . These observations lead to a tuned-up version of the algorithm:

Algorithm T (*Lexicographic combinations*). This algorithm is like Algorithm L, but faster. It also assumes, for convenience, that $t < n$.

- T1.** [Initialize.] Set $c_j \leftarrow j - 1$ for $1 \leq j \leq t$; then set $c_{t+1} \leftarrow n$, $c_{t+2} \leftarrow 0$, and $j \leftarrow t$.
- T2.** [Visit.] (At this point j is the smallest index such that $c_{j+1} > j$.) Visit the combination $c_t \dots c_2c_1$. Then, if $j > 0$, set $x \leftarrow j$ and go to step T6.
- T3.** [Easy case?] If $c_1 + 1 < c_2$, set $c_1 \leftarrow c_1 + 1$ and return to T2. Otherwise set $j \leftarrow 2$.
- T4.** [Find j .] Set $c_{j-1} \leftarrow j - 1$ and $x \leftarrow c_j + 1$. If $x = c_{j+1}$, set $j \leftarrow j + 1$ and repeat this step until $x < c_{j+1}$.
- T5.** [Done?] Terminate the algorithm if $j > t$.
- T6.** [Increase c_j .] Set $c_j \leftarrow x$, $j \leftarrow j - 1$, and return to T2. ▀

Now $j = 0$ in step T2 if and only if $c_1 > 0$, so the assignments in step T4 are never redundant. Exercise 6 carries out a complete analysis of Algorithm T.

Notice that the parameter n appears only in the initialization steps L1 and T1, not in the principal parts of Algorithms L and T. Thus we can think of the process as generating the first $\binom{n}{t}$ combinations of an *infinite* list, which depends only on t . This simplification arises because the list of t -combinations for $n + 1$ things begins with the list for n things, under our conventions; we have been using lexicographic order on the decreasing sequences $c_t \dots c_1$ for this very reason, instead of working with the increasing sequences $c_1 \dots c_t$.

Derrick Lehmer noticed another pleasant property of Algorithms L and T [*Applied Combinatorial Mathematics*, edited by E. F. Beckenbach (1964), 27–30]:

Theorem L. *The combination $c_t \dots c_2 c_1$ is visited after exactly*

$$\binom{c_t}{t} + \dots + \binom{c_2}{2} + \binom{c_1}{1} \quad (19)$$

other combinations have been visited.

Proof. There are $\binom{c_k}{k}$ combinations $c'_t \dots c'_2 c'_1$ with $c'_j = c_j$ for $t \geq j > k$ and $c'_k < c_k$, namely $c_t \dots c_{k+1}$ followed by the k -combinations of $\{0, \dots, c_k - 1\}$. \blacksquare

When $t = 3$, for example, the numbers

$$\binom{2}{3} + \binom{1}{2} + \binom{0}{1}, \quad \binom{3}{3} + \binom{1}{2} + \binom{0}{1}, \quad \binom{3}{3} + \binom{2}{2} + \binom{0}{1}, \quad \dots, \quad \binom{5}{3} + \binom{4}{2} + \binom{3}{1}$$

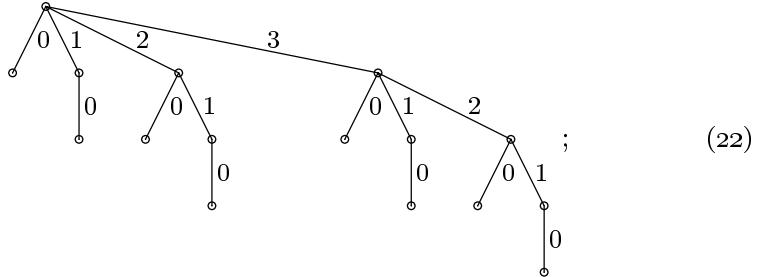
that correspond to the combinations $c_3 c_2 c_1$ in Table 1 simply run through the sequence 0, 1, 2, ..., 19. Theorem L gives us a nice way to understand the *combinatorial number system* of degree t , which represents every nonnegative integer N uniquely in the form

$$N = \binom{n_t}{t} + \dots + \binom{n_2}{2} + \binom{n_1}{1}, \quad n_t > \dots > n_2 > n_1 \geq 0. \quad (20)$$

Binomial trees. The family of trees T_n defined by

$$T_0 = \circ, \quad T_n = \begin{array}{ccccc} & \text{---} & \text{---} & \text{---} & \\ & 0 & 1 & n-1 & \\ T_0 & & T_1 & \dots & T_{n-1} \end{array} \quad \text{for } n > 0, \quad (21)$$

arises in several important contexts and sheds further light on combination generation. For example, T_4 is



and T_5 , rendered more artistically, appears as the frontispiece to Volume 1 of this series of books.

Notice that T_n is like T_{n-1} , except for an additional copy of T_{n-1} ; therefore T_n has 2^n nodes altogether. Furthermore, the number of nodes on level t is the binomial coefficient $\binom{n}{t}$; this fact accounts for the name “binomial tree.” Indeed, the sequence of labels encountered on the path from the root to each node on level t defines a combination $c_t \dots c_1$, and all combinations occur in lexicographic order from left to right. Thus, Algorithms L and T can be regarded as procedures to traverse the nodes on level t of the binomial tree T_n .

The infinite binomial tree T_∞ is obtained by letting $n \rightarrow \infty$ in (21). The root of this tree has infinitely many branches, but every other node is the root

of a finite binomial subtree. All possible t -combinations appear in lexicographic order on level t of T_∞ .

Let's get more familiar with binomial trees by considering all possible ways to pack a rucksack. More precisely, suppose we have n items that take up respectively w_{n-1}, \dots, w_1, w_0 units of capacity, where

$$w_{n-1} \geq \dots \geq w_1 \geq w_0; \quad (23)$$

we want to generate all binary vectors $a_{n-1} \dots a_1 a_0$ such that

$$a \cdot w = a_{n-1}w_{n-1} + \dots + a_1w_1 + a_0w_0 \leq N, \quad (24)$$

where N is the total capacity of a rucksack. Equivalently, we want to find all subsets C of $\{0, 1, \dots, n-1\}$ such that $w(C) = \sum_{c \in C} w_c \leq N$; such subsets will be called *feasible*. We will write a feasible subset as $c_1 \dots c_t$, where $c_1 > \dots > c_t \geq 0$, numbering the subscripts differently from the convention of (3) above because t is variable in this problem.

Every feasible subset corresponds to a node of T_n , and our goal is to visit each feasible node. Clearly the parent of every feasible node is feasible, and so is the left sibling, if any; therefore a simple tree exploration procedure solves the problem.

Algorithm F (*Filling a rucksack*). This algorithm generates all feasible ways $c_1 \dots c_t$ to fill a rucksack, given w_{n-1}, \dots, w_1, w_0 , and N . We let $\delta_j = w_j - w_{j-1}$ for $1 \leq j < n$.

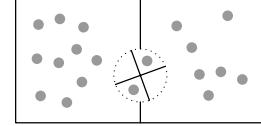
- F1.** [Initialize.] Set $t \leftarrow 0$, $c_0 \leftarrow n$, and $r \leftarrow N$.
- F2.** [Visit.] Visit the combination $c_1 \dots c_t$, which uses $N - r$ units of capacity.
- F3.** [Try to add w_0 .] If $c_t > 0$ and $r \geq w_0$, set $t \leftarrow t + 1$, $c_t \leftarrow 0$, $r \leftarrow r - w_0$, and return to F2.
- F4.** [Try to increase c_t .] Terminate if $t = 0$. Otherwise, if $c_{t-1} > c_t + 1$ and $r \geq \delta_{c_t+1}$, set $c_t \leftarrow c_t + 1$, $r \leftarrow r - \delta_{c_t}$, and return to F2.
- F5.** [Remove c_t .] Set $r \leftarrow r + w_{c_t}$, $t \leftarrow t - 1$, and return to F4. ■

Notice that the algorithm implicitly visits nodes of T_n in preorder, skipping over unfeasible subtrees. An element $c > 0$ is never placed in the rucksack until after the procedure has explored all possibilities using element $c-1$. The running time is proportional to the number of feasible combinations visited (see exercise 20).

Incidentally, the classical “knapsack problem” of operations research is different: It asks for a feasible subset C such that $v(C) = \sum_{c \in C} v(c)$ is maximum, where each item c has been assigned a value $v(c)$. Algorithm F is not a particularly good way to solve that problem, because it often considers cases that could be ruled out. For example, if C and C' are subsets of $\{1, \dots, n-1\}$ with $w(C) \leq w(C') \leq N - w_0$ and $v(C) \geq v(C')$, Algorithm F will examine both $C \cup \{0\}$ and $C' \cup \{0\}$, but the latter subset will never improve the maximum. We will consider methods for the classical knapsack problem later; Algorithm F is intended only for situations when *all* of the feasible possibilities are potentially relevant.

Gray codes for combinations. Instead of merely generating all combinations, we often prefer to visit them in such a way that each one is obtained by making only a small change to its predecessor.

For example, we can ask for what Nijenhuis and Wilf have called a “revolving door algorithm”: Imagine two rooms that contain respectively s and t people, with a revolving door between them. Whenever a person goes into the opposite room, somebody else comes out. Can we devise a sequence of moves so that each (s, t) -combination occurs exactly once?



The answer is yes, and in fact a huge number of such patterns exist. For example, it turns out that if we examine all n -bit strings $a_{n-1} \dots a_1 a_0$ in the well-known order of Gray binary code (Section 7.2.1.1), but select only those that have exactly s 0s and t 1s, the resulting strings form a revolving door code.

Here's the proof: Gray binary code is defined by the recurrence $\Gamma_n = 0\Gamma_{n-1}$, $1\Gamma_{n-1}^R$ of 7.2.1.1–(5), so its (s, t) subset satisfies the recurrence

$$\Gamma_{st} = 0\Gamma_{(s-1)t}, 1\Gamma_{s(t-1)}^R \quad (25)$$

when $st > 0$. We also have $\Gamma_{s0} = 0^s$ and $\Gamma_{0t} = 1^t$. Therefore it is clear by induction that Γ_{st} begins with $0^s 1^t$ and ends with $10^{s-1} 1^{t-1}$ when $st > 0$. The transition at the comma in (25) is from the last element of $0\Gamma_{(s-1)t}$ to the last element of $1\Gamma_{s(t-1)}$, namely from $010^{s-1} 1^{t-1} = 010^{s-1} 11^{t-2}$ to $110^{s-1} 1^{t-2} = 110^{s-1} 01^{t-2}$ when $t \geq 2$, and this satisfies the revolving-door constraint. The case $t = 1$ also checks out. For example, Γ_{33} is given by the columns of

$$\begin{array}{cccc} 000111 & 011010 & 110001 & 101010 \\ 001101 & 011100 & 110010 & 101100 \\ 001110 & 010101 & 110100 & 100101 \\ 001011 & 010110 & 111000 & 100110 \\ 011001 & 010011 & 101001 & 100011 \end{array} \quad (26)$$

and Γ_{23} can be found in the first two columns of this array. One more turn of the door takes the last element into the first. [These properties of Γ_{st} were discovered by D. T. Tang and C. N. Liu, *IEEE Trans. C-22* (1973), 176–180; a loopless implementation was presented by J. R. Bitner, G. Ehrlich, and E. M. Reingold, *CACM* **19** (1976), 517–521.]

When we convert the bitstrings $a_5 a_4 a_3 a_2 a_1 a_0$ in (26) to the corresponding index-list forms $c_3 c_2 c_1$, a striking pattern becomes evident:

$$\begin{array}{cccc} 210 & 431 & 540 & 531 \\ 320 & 432 & 541 & 532 \\ 321 & 420 & 542 & 520 \\ 310 & 421 & 543 & 521 \\ 430 & 410 & 530 & 510 \end{array} \quad (27)$$

The first components c_3 occur in increasing order; but for each fixed value of c_3 , the values of c_2 occur in *decreasing* order. And for fixed $c_3 c_2$, the values of c_1 are again increasing. The same is true in general: *All combinations $c_t \dots c_2 c_1$*

appear in lexicographic order of

$$(c_t, -c_{t-1}, c_{t-2}, \dots, (-1)^t c_1) \quad (28)$$

in the revolving-door Gray code Γ_{st} . This property follows by induction, because (25) becomes

$$\Gamma_{st} = \Gamma_{(s-1)t}, (s+t-1)\Gamma_{s(t-1)}^R \quad (29)$$

for $st > 0$ when we use index-list notation instead of bitstring notation. Consequently the sequence can be generated efficiently by the following algorithm due to W. H. Payne [see ACM Trans. Math. Software 5 (1979), 163–172]:

Algorithm R (*Revolving-door combinations*). This algorithm generates all t -combinations $c_t \dots c_2 c_1$ of $\{0, 1, \dots, n - 1\}$ in lexicographic order of the alternating sequence (28), assuming that $n > t > 1$. Step R3 has two variants, depending on whether t is even or odd.

R1. [Initialize.] Set $c_j \leftarrow j - 1$ for $t \geq j \geq 1$, and $c_{t+1} \leftarrow n$.

R2. [Visit.] Visit the combination $c_t \dots c_2 c_1$.

R3. [Easy case?] If t is odd: If $c_1 + 1 < c_2$, increase c_1 by 1 and return to R2, otherwise set $j \leftarrow 2$ and go to R4. If t is even: If $c_1 > 0$, decrease c_1 by 1 and return to R2, otherwise set $j \leftarrow 2$ and go to R5.

R4. [Try to decrease c_j .] (At this point $c_j = c_{j-1} + 1$.) If $c_j \geq j$, set $c_j \leftarrow c_{j-1}$, $c_{j-1} \leftarrow j - 2$, and return to R2. Otherwise increase j by 1.

R5. [Try to increase c_j .] (At this point $c_{j-1} = j - 2$.) If $c_j + 1 < c_{j+1}$, set $c_{j-1} \leftarrow c_j$, $c_j \leftarrow c_j + 1$, and return to R2. Otherwise increase j by 1, and go to R4 if $j \leq t$. ■

Exercises 21–25 explore further properties of this interesting sequence. One of them is a nice companion to Theorem L: *The combination $c_t c_{t-1} \dots c_2 c_1$ is visited by Algorithm R after exactly*

$$N = \binom{c_t+1}{t} - \binom{c_{t-1}+1}{t-1} + \dots + (-1)^t \binom{c_2+1}{2} - (-1)^t \binom{c_1+1}{1} - [t \text{ odd}] \quad (30)$$

other combinations have been visited. We may call this the representation of N in the “alternating combinatorial number system” of degree t ; one consequence, for example, is that every positive integer has a unique representation of the form $N = \binom{a}{3} - \binom{b}{2} + \binom{c}{1}$ with $a > b > c > 0$. Algorithm R tells us how to add 1 to N in this system.

Although the strings of (26) and (27) are not in lexicographic order, they are examples of a more general concept called *genlex order*, a name coined by Timothy Walsh. A sequence of strings $\alpha_1, \dots, \alpha_N$ is said to be in genlex order when all strings with a common prefix occur consecutively. For example, all 3-combinations that begin with 53 appear together in (27).

Genlex order means that the strings can be arranged in a trie structure, as in Fig. 31 of Section 6.3, but with the children of each node ordered arbitrarily. When a trie is traversed in any order such that each node is visited just before or just after its descendants, all nodes with a common prefix—that is, all nodes of

a subtrie — appear consecutively. This principle makes genlex order convenient, because it corresponds to recursive generation schemes. Many of the algorithms we have seen for generating n -tuples have therefore produced their results in some version of genlex order; similarly, the method of “plain changes” (Algorithm 7.2.1.2P) visits permutations in a genlex order of the corresponding inversion tables.

The revolving-door method of Algorithm R is a genlex routine that changes only one element of the combination at each step. But it isn’t totally satisfactory, because it frequently must change two of the indices c_j simultaneously, in order to preserve the condition $c_t > \dots > c_2 > c_1$. For example, Algorithm R changes 210 into 320, and (27) includes nine such “crossing” moves.

The source of this defect can be traced to our proof that (25) satisfies the revolving-door property: We observed that the string $010^{s-1}11^{t-2}$ is followed by $110^{s-1}01^{t-2}$ when $t \geq 2$. Hence the recursive construction Γ_{st} involves transitions of the form $110^a 0 \leftrightarrow 010^a 1$, when a substring like 11000 is changed to 01001 or vice versa; the two 1s cross each other.

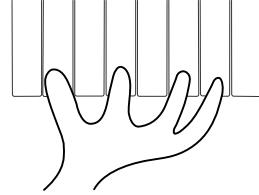
A Gray path for combinations is said to be *homogeneous* if it changes only one of the indices c_j at each step. A homogeneous scheme is characterized in bitstring form by having only transitions of the forms $10^a \leftrightarrow 0^a 1$ within strings, for $a \geq 1$, when we pass from one string to the next. With a homogeneous scheme we can, for example, play all t -note chords on an n -note keyboard by moving only one finger at a time.

A slight modification of (25) yields a genlex scheme for (s, t) -combinations that is pleasantly homogeneous. The basic idea is to construct a sequence that begins with $0^s 1^t$ and ends with $1^t 0^s$, and the following recursion suggests itself almost immediately: Let $K_{s0} = 0^s$, $K_{0t} = 1^t$, $K_{s(-1)} = \emptyset$, and

$$K_{st} = 0K_{(s-1)t}, 10K_{(s-1)(t-1)}^R, 11K_{s(t-2)} \quad \text{for } st > 0. \quad (31)$$

At the commas of this sequence we have $01^t 0^{s-1}$ followed by $101^{t-1} 0^{s-1}$, and $10^s 1^{t-1}$ followed by $110^s 1^{t-2}$; both of these transitions are homogeneous, although the second one requires the 1 to jump across s 0s. The combinations K_{33} for $s = t = 3$ are

$$\begin{array}{cccc} 000111 & 010101 & 101100 & 100011 \\ 001011 & 010011 & 101001 & 110001 \\ 001101 & 011001 & 101010 & 110010 \\ 001110 & 011010 & 100110 & 110100 \\ 010110 & 011100 & 100101 & 111000 \end{array} \quad (32)$$



in bitstring form, and the corresponding “finger patterns” are

$$\begin{array}{cccc} 210 & 420 & 532 & 510 \\ 310 & 410 & 530 & 540 \\ 320 & 430 & 531 & 541 \\ 321 & 431 & 521 & 542 \\ 421 & 432 & 520 & 543 \end{array} \quad (33)$$

When a homogeneous scheme for ordinary combinations $c_t \dots c_1$ is converted to the corresponding scheme (6) for combinations with repetitions $d_t \dots d_1$, it retains the property that only one of the indices d_j changes at each step. And when it is converted to the corresponding schemes (9) or (11) for compositions $p_t \dots p_0$ or $q_t \dots q_0$, only two (adjacent) parts change when c_j changes.

Near-perfect schemes. But we can do even better! All (s, t) -combinations can be generated by a sequence of strongly homogeneous transitions that are either $01 \leftrightarrow 10$ or $001 \leftrightarrow 100$. In other words, we can insist that each step causes a single index c_j to change by at most 2. Let's call such generation schemes *near-perfect*.

Imposing such strong conditions actually makes it easier to discover near-perfect schemes, because comparatively few choices are available. Indeed, if we restrict ourselves to genlex methods that are near-perfect on n -bit strings, T. A. Jenkyns and D. McCarthy observed that all such methods can be easily characterized [*Ars Combinatoria* **40** (1995), 153–159]:

Theorem N. *If $st > 0$, there are exactly $2s$ near-perfect ways to list all (s, t) -combinations in a genlex order. In fact, when $1 \leq a \leq s$, there is exactly one such listing, N_{sta} , that begins with $1^t 0^s$ and ends with $0^a 1^t 0^{s-a}$; the other s possibilities are the reverse lists, N_{sta}^R .*

Proof. The result certainly holds when $s = t = 1$; otherwise we use induction on $s + t$. The listing N_{sta} , if it exists, must have the form $1X_{s(t-1)}, 0Y_{(s-1)t}$ for some near-perfect genlex listings $X_{s(t-1)}$ and $Y_{(s-1)t}$. If $t = 1$, $X_{s(t-1)}$ is the single string 0^s ; hence $Y_{(s-1)t}$ must be $N_{(s-1)1(a-1)}$ if $a > 1$, $N_{(s-1)11}$ if $a = 1$. On the other hand if $t > 1$, the near-perfect condition implies that the last string of $X_{s(t-1)}$ cannot begin with 1; hence $X_{s(t-1)} = N_{s(t-1)b}$ for some b . If $a > 1$, $Y_{(s-1)t}$ must be $N_{(s-1)t(a-1)}$, hence b must be 1; similarly, b must be 1 if $s = 1$. Otherwise we have $a = 1 < s$, and this forces $Y_{(s-1)t} = N_{(s-1)tc}^R$ for some c . The transition from $10^b 1^{t-1} 0^{s-b}$ to $0^{c+1} 1^t 0^{s-1-c}$ is near-perfect only if $c = 1$ and $b = 2$. ■

The proof of Theorem N yields the following recursive formulas when $st > 0$:

$$N_{sta} = \begin{cases} 1N_{s(t-1)1}, 0N_{(s-1)t(a-1)}, & \text{if } 1 < a \leq s; \\ 1N_{s(t-1)2}, 0N_{(s-1)t1}^R, & \text{if } 1 = a < s; \\ 1N_{1(t-1)1}, 01^t, & \text{if } 1 = a = s. \end{cases} \quad (34)$$

Also, of course, $N_{s0a} = 0^s$.

Let us set $A_{st} = N_{st1}$ and $B_{st} = N_{st2}$. These near-perfect listings, discovered by Phillip J. Chase in 1976, have the net effect of shifting a leftmost block of 1s to the right by one or two positions, respectively, and they satisfy the following mutual recursions:

$$A_{st} = 1B_{s(t-1)}, 0A_{(s-1)t}^R; \quad B_{st} = 1A_{s(t-1)}, 0A_{(s-1)t}. \quad (35)$$

“To take one step forward, take two steps forward, then one step backward; to take two steps forward, take one step forward, then another.” These equations

Table 2
CHASE'S SEQUENCES FOR (3,3)-COMBINATIONS

$A_{33} = \hat{C}_{33}^R$				$B_{33} = C_{33}$			
543	531	321	420	543	520	432	410
541	530	320	421	542	510	430	210
540	510	310	431	540	530	431	310
542	520	210	430	541	531	421	320
532	521	410	432	521	532	420	321

hold for all integer values of s and t , if we define A_{st} and B_{st} to be \emptyset when s or t is negative, except that $A_{00} = B_{00} = \epsilon$ (the empty string). Thus A_{st} actually takes $\min(s, 1)$ forward steps, and B_{st} actually takes $\min(s, 2)$. For example, Table 2 shows the relevant listings for $s = t = 3$, using an equivalent index-list form $c_3c_2c_1$ instead of the bit strings $a_5a_4a_3a_2a_1a_0$.

Chase noticed that a computer implementation of these sequences becomes simpler if we define

$$C_{st} = \begin{cases} A_{st}, & \text{if } s+t \text{ is odd;} \\ B_{st}, & \text{if } s+t \text{ is even;} \end{cases} \quad \hat{C}_{st} = \begin{cases} A_{st}^R, & \text{if } s+t \text{ is even;} \\ B_{st}^R, & \text{if } s+t \text{ is odd.} \end{cases} \quad (36)$$

[See *Congressus Numerantium* **69** (1989), 215–242.] Then we have

$$C_{st} = \begin{cases} 1C_{s(t-1)}, 0\hat{C}_{(s-1)t}, & \text{if } s+t \text{ is odd;} \\ 1C_{s(t-1)}, 0C_{(s-1)t}, & \text{if } s+t \text{ is even;} \end{cases} \quad (37)$$

$$\hat{C}_{st} = \begin{cases} 0C_{(s-1)t}, 1\hat{C}_{s(t-1)}, & \text{if } s+t \text{ is even;} \\ 0\hat{C}_{(s-1)t}, 1\hat{C}_{s(t-1)}, & \text{if } s+t \text{ is odd.} \end{cases} \quad (38)$$

When bit a_j is ready to change, we can tell where we are in the recursion by testing whether j is even or odd.

Indeed, the sequence C_{st} can be generated by a surprisingly simple algorithm, based on general ideas that apply to *any* genlex scheme. Let us say that bit a_j is *active* in a genlex algorithm if it is supposed to change before anything to its left is altered. (The node for an active bit in the corresponding trie is not the rightmost child of its parent.) Suppose we have an auxiliary table $w_n \dots w_1 w_0$, where $w_j = 1$ if and only if either a_j is active or $j < r$, where r is the least subscript such that $a_r \neq a_0$; we also let $w_n = 1$. Then the following method will find the successor of $a_{n-1} \dots a_1 a_0$:

Set $j \leftarrow r$. If $w_j = 0$, set $w_j \leftarrow 1$, $j \leftarrow j + 1$, and repeat until $w_j = 1$. Terminate if $j = n$; otherwise set $w_j \leftarrow 0$. Change a_j to $1 - a_j$, and make any other changes to $a_{j-1} \dots a_0$ and r that apply to the particular genlex scheme being used. (39)

The beauty of this approach comes from the fact that the loop is guaranteed to be efficient: We can prove that the operation $j \leftarrow j + 1$ will be performed less than once per generation step, on the average (see exercise 36).

By analyzing the transitions that occur when bits change in (37) and (38), we can readily flesh out the remaining details:

Algorithm C (*Chase's sequence*). This algorithm visits all (s, t) -combinations $a_{n-1} \dots a_1 a_0$, where $n = s + t$, in the near-perfect order of Chase's sequence C_{st} .

- C1.** [Initialize.] Set $a_j \leftarrow 0$ for $0 \leq j < s$, $a_j \leftarrow 1$ for $s \leq j < n$, and $w_j \leftarrow 1$ for $0 \leq j \leq n$. If $s > 0$, set $r \leftarrow s$; otherwise set $r \leftarrow t$.
- C2.** [Visit.] Visit the combination $a_{n-1} \dots a_1 a_0$.
- C3.** [Find j and branch.] Set $j \leftarrow r$. If $w_j = 0$, set $w_j \leftarrow 1$, $j \leftarrow j + 1$, and repeat until $w_j = 1$. Terminate if $j = n$; otherwise set $w_j \leftarrow 0$ and make a four-way branch: Go to C4 if j is odd and $a_j \neq 0$, to C5 if j is even and $a_j \neq 0$, to C6 if j is even and $a_j = 0$, to C7 if j is odd and $a_j = 0$.
- C4.** [Move right one.] Set $a_{j-1} \leftarrow 1$, $a_j \leftarrow 0$. If $r = j > 1$, set $r \leftarrow j - 1$; otherwise if $r = j - 1$ set $r \leftarrow j$. Return to C2.
- C5.** [Move right two.] If $a_{j-2} \neq 0$, go to C4. Otherwise set $a_{j-2} \leftarrow 1$, $a_j \leftarrow 0$. If $r = j$, set $r \leftarrow \max(j - 2, 1)$; otherwise if $r = j - 2$, set $r \leftarrow j - 1$. Return to C2.
- C6.** [Move left one.] Set $a_j \leftarrow 1$, $a_{j-1} \leftarrow 0$. If $r = j > 1$, set $r \leftarrow j - 1$; otherwise if $r = j - 2$ set $r \leftarrow j - 1$. Return to C2.
- C7.** [Move left two.] If $a_{j-1} \neq 0$, go to C6. Otherwise set $a_j \leftarrow 1$, $a_{j-2} \leftarrow 0$. If $r = j - 2$, set $r \leftarrow j$; otherwise if $r = j - 1$, set $r \leftarrow j - 2$. Return to C2. \blacksquare

***Analysis of Chase's sequence.** The magical properties of Algorithm C cry out for further exploration, and a closer look turns out to be quite instructive. Given a bit string $a_{n-1} \dots a_1 a_0$, let us define $a_n = 1$, $u_n = n \bmod 2$, and

$$u_j = a_{j+1}(1 - u_{j+1}), \quad v_j = (u_j + j) \bmod 2, \quad w_j = (v_j + a_j) \bmod 2, \quad (40)$$

for $n > j \geq 0$. For example, we might have $n = 26$ and

$$\begin{aligned} a_{25} \dots a_1 a_0 &= 11001001000011111101101010, \\ u_{25} \dots u_1 u_0 &= 10100100100001010100100101, \\ v_{25} \dots v_1 v_0 &= 0000111000101111110001111, \\ w_{25} \dots w_1 w_0 &= 11000111001000000011100101. \end{aligned} \quad (41)$$

With these definitions we can prove by induction that $v_j = 0$ if and only if bit a_j is being “controlled” by C rather than by \hat{C} in the recursions (37)–(38) that generate $a_{n-1} \dots a_1 a_0$, except when a_j is part of the final run of 0s or 1s at the right end. Therefore w_j agrees with the value computed by Algorithm C at the moment when $a_{n-1} \dots a_1 a_0$ is visited, for $r \leq j < n$. These formulas can be used to determine exactly where a given combination appears in Chase's sequence (see exercise 39).

If we want to work with the index-list form $c_t \dots c_2 c_1$ instead of the bit strings $a_{n-1} \dots a_1 a_0$, it is convenient to change the notation slightly, writing

$C_t(n)$ for C_{st} and $\hat{C}_t(n)$ for \hat{C}_{st} when $s + t = n$. Then $C_0(n) = \hat{C}_0(n) = \epsilon$, and the recursions for $t \geq 0$ take the form

$$C_{t+1}(n+1) = \begin{cases} nC_t(n), & \hat{C}_{t+1}(n), \text{ if } n \text{ is even;} \\ nC_t(n), & C_{t+1}(n), \text{ if } n \text{ is odd;} \end{cases} \quad (42)$$

$$\hat{C}_{t+1}(n+1) = \begin{cases} C_{t+1}(n), & n\hat{C}_t(n), \text{ if } n \text{ is odd;} \\ \hat{C}_{t+1}(n), & n\hat{C}_t(n), \text{ if } n \text{ is even.} \end{cases} \quad (43)$$

These new equations can be expanded to tell us, for example, that

$$\begin{aligned} C_{t+1}(9) &= 8C_t(8), 6C_t(6), 4C_t(4), \dots, 3\hat{C}_t(3), 5\hat{C}_t(5), 7\hat{C}_t(7); \\ C_{t+1}(8) &= 7C_t(7), 6C_t(6), 4C_t(4), \dots, 3\hat{C}_t(3), 5\hat{C}_t(5); \\ \hat{C}_{t+1}(9) &= 6C_t(6), 4C_t(4), \dots, 3\hat{C}_t(3), 5\hat{C}_t(5), 7\hat{C}_t(7), 8\hat{C}_t(8); \\ \hat{C}_{t+1}(8) &= 6C_t(6), 4C_t(4), \dots, 3\hat{C}_t(3), 5\hat{C}_t(5), 7\hat{C}_t(7); \end{aligned} \quad (44)$$

notice that the same pattern predominates in all four sequences. The meaning of “...” in the middle depends on the value of t : We simply omit all terms $nC_t(n)$ and $n\hat{C}_t(n)$ where $n < t$.

Except for edge effects at the very beginning or end, all of the expansions in (44) are based on the infinite progression

$$\dots, 10, 8, 6, 4, 2, 0, 1, 3, 5, 7, 9, \dots, \quad (45)$$

which is a natural way to arrange the nonnegative integers into a doubly infinite sequence. If we omit all terms of (45) that are $< t$, given any integer $t \geq 0$, the remaining terms retain the property that adjacent elements differ by either 1 or 2. Richard Stanley has suggested the name *endo-order* for this sequence, because we can remember it by thinking “even numbers decreasing, odd ...”. (Notice that if we retain only the terms less than N and complement with respect to N , endo-order becomes organ-pipe order; see exercise 6.1–18.)

We could program the recursions of (42) and (43) directly, but it is interesting to unwind them using (44), thus obtaining an iterative algorithm analogous to Algorithm C. The result needs only $O(t)$ memory locations, and it is especially efficient when t is relatively small compared to n . Exercise 45 contains the details.

***Near-perfect multiset permutations.** Chase’s sequences lead in a natural way to an algorithm that will generate permutations of any multiset $\{s_0 \cdot 0, s_1 \cdot 1, \dots, s_d \cdot d\}$ in a near-perfect manner, meaning that

- i) every transition is either $a_{j+1}a_j \leftrightarrow a_ja_{j+1}$ or $a_{j+2}a_{j+1}a_j \leftrightarrow a_ja_{j+1}a_{j+2}$;
- ii) transitions of the second kind have $a_{j+1} = \min(a_j, a_{j+2})$.

Algorithm C tells us how to do this when $d = 1$, and we can extend it to larger values of d by the following recursive construction [CACM 13 (1970), 368–369, 376]: Suppose

$$\alpha_0, \alpha_1, \dots, \alpha_{N-1}$$

is any near-perfect listing of the permutations of $\{s_1 \cdot 1, \dots, s_d \cdot d\}$. Then Algorithm C, with $s = s_0$ and $t = s_1 + \dots + s_d$, tells us how to generate a listing

$$\Lambda_j = \alpha_j 0^s, \dots, 0^a \alpha_j 0^{s-a} \quad (46)$$

in which all transitions are $0x \leftrightarrow x0$ or $00x \leftrightarrow x00$; the final entry has $a = 1$ or 2 leading zeros, depending on s and t . Therefore all transitions of the sequence

$$\Lambda_0, \Lambda_1^R, \Lambda_2, \dots, (\Lambda_{N-1} \text{ or } \Lambda_{N-1}^R) \quad (47)$$

are near-perfect; and this list clearly contains all the permutations.

For example, the permutations of $\{0, 0, 0, 1, 1, 2\}$ generated in this way are

211000, 210100, 210001, 210010, 200110, 200101, 200011, 201001, 201010, 201100, 021100, 021001, 021010, 020110, 020101, 020011, 000211, 002011, 002101, 002110, 001210, 001201, 001021, 000121, 010021, 010201, 010210, 012010, 012001, 012100, 102100, 102010, 102001, 100021, 100201, 100210, 120010, 120001, 120100, 121000, 112000, 110200, 110002, 110020, 100120, 100102, 100012, 101002, 101020, 101200, 011200, 011002, 011020, 010120, 010102, 010012, 000112, 001012, 001102, 001120.

***Perfect schemes.** Why should we settle for a near-perfect generator like C_{st} , instead of insisting that all transitions have the simplest possible form $01 \leftrightarrow 10$?

One reason is that perfect schemes don't always exist. For example, we observed in 7.2.1.2-(2) that there is no way to generate all six permutations of $\{1, 1, 2, 2\}$ with adjacent interchanges; thus there is no perfect scheme for $(2, 2)$ -combinations. In fact, our chances of achieving perfection are only about 1 in 4:

Theorem P. *The generation of all (s, t) -combinations $a_{s+t-1} \dots a_1 a_0$ by adjacent interchanges $01 \leftrightarrow 10$ is possible if and only if $s \leq 1$ or $t \leq 1$ or st is odd.*

Proof. Consider all permutations of the multiset $\{s \cdot 0, t \cdot 1\}$. We learned in exercise 5.1.2-16 that the number m_k of such permutations having k inversions is the coefficient of z^k in the z -nomial coefficient

$$\binom{s+t}{t}_z = \prod_{k=s+1}^{s+t} (1+z+\dots+z^{k-1}) / \prod_{k=1}^t (1+z+\dots+z^{k-1}). \quad (48)$$

Every adjacent interchange changes the number of inversions by ± 1 , so a perfect generation scheme is possible only if approximately half of all the permutations have an odd number of inversions. More precisely, the value of $\binom{s+t}{t}_{-1} = m_0 - m_1 + m_2 - \dots$ must be 0 or ± 1 . But exercise 49 shows that

$$\binom{s+t}{t}_{-1} = \begin{cases} \lfloor (s+t)/2 \rfloor & [st \text{ is even}], \\ \lceil (s+t)/2 \rceil & [st \text{ is odd}], \end{cases} \quad (49)$$

and this quantity exceeds 1 unless $s \leq 1$ or $t \leq 1$ or st is odd.

Conversely, perfect schemes are easy with $s \leq 1$ or $t \leq 1$, and they turn out to be possible also whenever st is odd. The first nontrivial case occurs for $s = t = 3$, when there are four essentially different solutions; the most symmetrical of these is

$$\begin{array}{ccccccccccccccccc} 210 & - & 310 & - & 410 & - & 510 & - & 520 & - & 521 & - & 531 & - & 532 & - & 432 & - & 431 & - \\ 421 & - & 321 & - & 320 & - & 420 & - & 430 & - & 530 & - & 540 & - & 541 & - & 542 & - & 543 & \end{array} \quad (50)$$

(see exercise 51). Several authors have constructed Hamiltonian paths in the relevant graph for arbitrary odd numbers s and t , most notably Eades, Hickey, and Read [JACM 31 (1984), 19–29], whose method makes an interesting exercise in programming with recursive routines. Unfortunately, however, none of the known constructions are sufficiently simple to describe in a short space, or to implement with reasonable efficiency. Perfect combination generators have therefore not yet proved to be of practical importance. ■

In summary, then, we have seen that the study of (s,t) -combinations leads to many fascinating patterns, some of which are of great practical importance and some of which are merely elegant and/or beautiful. Fig. 26 illustrates the principal options that are available in the case $s = t = 5$, when $\binom{10}{5} = 252$ combinations arise. Lexicographic order (Algorithm L), the revolving-door Gray code (Algorithm R), the homogeneous scheme K_{55} of (31), and Chase’s near-perfect scheme (Algorithm C) are shown in parts (a), (b), (c), and (d) of the illustration. Part (e) shows the near-perfect scheme that is as close to perfection as possible while still being in genlex order (see exercise 34), while part (f) is the perfect scheme of Eades, Hickey, and Read. Finally, Fig. 26(g) is a listing that proceeds by swapping $a_j \leftrightarrow a_0$, akin to Algorithm 7.2.1.2E (see exercise 55).

***Combinations of a multiset.** If multisets can have permutations, they can have combinations too. For example, consider the multiset $\{b, b, b, b, g, g, g, r, r, r, w, w\}$, representing a sack that contains four blue balls and three that are green, three red, two white. There are 37 ways to choose five balls from this sack; in lexicographic order (but descending in each combination) they are

$$\begin{aligned} &gbbb, ggbbb, gggbb, rbbbb, rgbbb, rggbb, rrbbb, rrgbb, rrggb, \\ &rrggg, rrrbb, rrrgb, rrrgg, wbbbb, wgbbb, wggbb, wgggb, wrbbb, wrgbb, \\ &wrggb, wrggg, wrrbb, wrrgb, wrrgg, wrrrb, wrrrg, wwbbb, wwgb, wwgg, \\ &wwgg, wwrbb, wwrgb, wwrgg, wwrrb, wwrrg, wwrrr. \end{aligned} \quad (51)$$

This fact might seem frivolous and/or esoteric, yet we will see in Theorem W below that the lexicographic generation of multiset combinations yields optimal solutions to significant combinatorial problems.

James Bernoulli observed in his *Ars Conjectandi* (1713), 119–123, that we can count the number of such combinations by looking at the coefficient of z^5 in $(1+z+z^2)(1+z+z^2+z^3)^2(1+z+z^2+z^3+z^4)$. Indeed, his observation is easy to understand, because we get all possible selections from the sack if we multiply out the polynomials

$$(1+w+ww)(1+r+rr+rrr)(1+g+gg+ggg)(1+b+bb+bbb+bbbb).$$

Multiset combinations are also equivalent to *bounded compositions*, namely to compositions in which the individual parts are bounded. For example, the 37 multicombinations listed in (51) correspond to 37 solutions of

$$5 = r_3 + r_2 + r_1 + r_0, \quad 0 \leq r_3 \leq 2, \quad 0 \leq r_2, r_1 \leq 3, \quad 0 \leq r_0 \leq 4,$$

namely $5 = 0+0+1+4 = 0+0+2+3 = 0+0+3+2 = 0+1+0+4 = \dots = 2+3+0+0$.

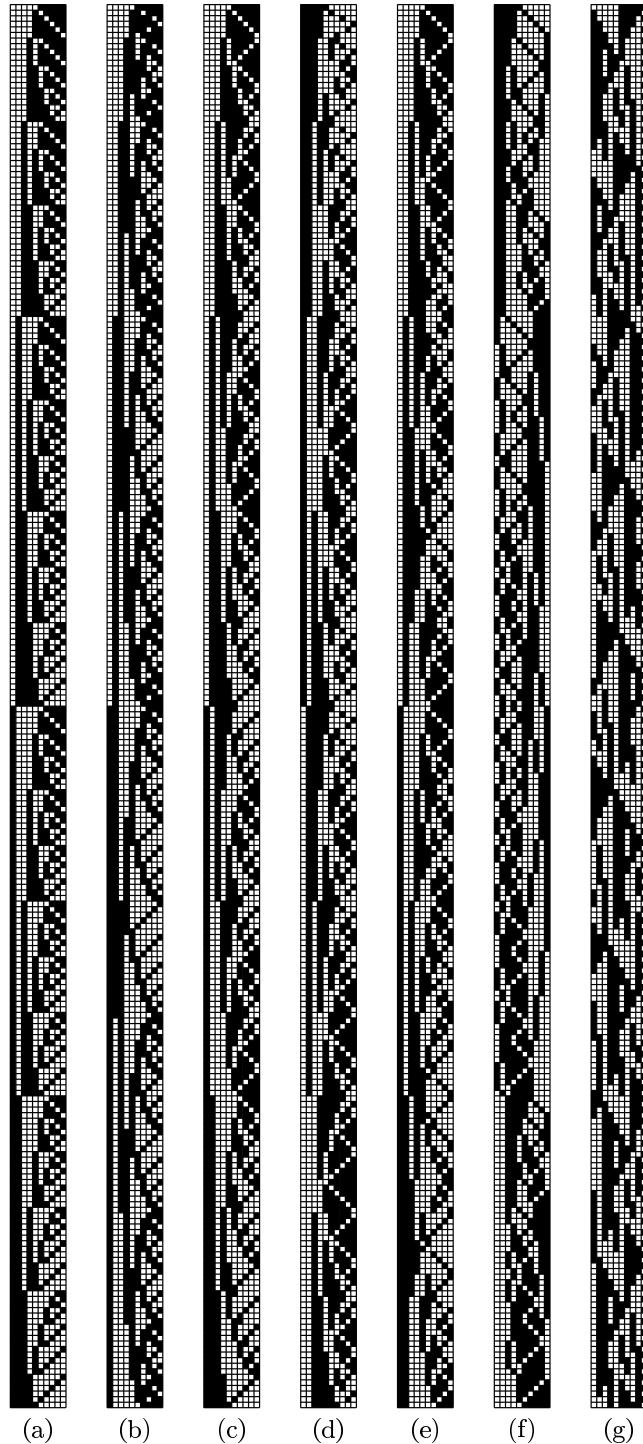


Fig. 26. Examples
of $(5, 5)$ -combinations:

- a) lexicographic;
- b) revolving-door;
- c) homogeneous;
- d) near-perfect;
- e) nearer-perfect;
- f) perfect;
- g) right-swapped.

Bounded compositions, in turn, are special cases of *contingency tables*, which are of great importance in statistics. And all of these combinatorial configurations can be generated with Graylike codes as well as in lexicographic order. Exercises 59–62 explore some of the basic ideas involved.

***Shadows.** Sets of combinations appear frequently in mathematics. For example, a set of 2-combinations (namely a set of pairs) is essentially a graph, and a set of t -combinations for general t is called a uniform hypergraph. If the vertices of a convex polyhedron are perturbed slightly, so that no three are collinear, no four lie in a plane, and in general no $t+1$ lie in a $(t-1)$ -dimensional hyperplane, the resulting $(t-1)$ -dimensional faces are “simplexes” whose vertices have great significance in computer applications. Researchers have learned that such sets of combinations have important properties related to lexicographic generation.

If α is any t -combination $c_t \dots c_2 c_1$, its *shadow* $\partial\alpha$ is the set of all its $(t-1)$ -element subsets $c_{t-1} \dots c_2 c_1, \dots, c_t \dots c_3 c_1, c_t \dots c_3 c_2$. For example, $\partial 5310 = \{310, 510, 530, 531\}$. We can also represent a t -combination as a bit string $a_{n-1} \dots a_1 a_0$, in which case $\partial\alpha$ is the set of all strings obtained by changing a 1 to a 0: $\partial 101011 = \{001011, 100011, 101001, 101010\}$. If A is any set of t -combinations, we define its shadow

$$\partial A = \{\partial\alpha \mid \alpha \in A\} \quad (52)$$

to be the set of all $(t-1)$ -combinations in the shadows of its members. For example, $\partial\partial 5310 = \{10, 30, 31, 50, 51, 53\}$.

These definitions apply also to combinations with repetitions, namely to multicombinations: $\partial 5330 = \{330, 530, 533\}$ and $\partial\partial 5330 = \{30, 33, 50, 53\}$. In general, when A is a set of t -element multisets, ∂A is a set of $(t-1)$ -element multisets. Notice, however, that ∂A never has repeated elements itself.

The *upper shadow* $\ell\alpha$ with respect to a universe U is defined similarly, but it goes from t -combinations to $(t+1)$ -combinations:

$$\ell\alpha = \{\beta \in U \mid \alpha \in \partial\beta\}, \quad \text{for } \alpha \in U; \quad (53)$$

$$\ell A = \{\ell\alpha \mid \alpha \in A\}, \quad \text{for } A \subseteq U. \quad (54)$$

If, for example, $U = \{0, 1, 2, 3, 4, 5, 6\}$, we have $\ell 5310 = \{53210, 54310, 65310\}$; on the other hand, if $U = \{\infty \cdot 0, \infty \cdot 1, \dots, \infty \cdot 6\}$, we have $\ell 5310 = \{53100, 53110, 53210, 53310, 54310, 55310, 65310\}$.

The following fundamental theorems, which have many applications in various branches of mathematics and computer science, tell us how small a set’s shadows can be:

Theorem K. *If A is a set of N t -combinations contained in $U = \{0, 1, \dots, n-1\}$, then*

$$\|\partial A\| \geq \|\partial P_{Nt}\| \quad \text{and} \quad \|\ell A\| \geq \|\ell Q_{Nnt}\|, \quad (55)$$

where P_{Nt} denotes the first N combinations generated by Algorithm L, namely the N lexicographically smallest combinations $c_t \dots c_2 c_1$ that satisfy (3), and Q_{Nnt} denotes the N lexicographically largest. ■

Theorem M. *If A is a set of N t -multicombinations contained in the multiset $U = \{\infty \cdot 0, \infty \cdot 1, \dots, \infty \cdot s\}$, then*

$$\|\partial A\| \geq \|\partial \hat{P}_{Nt}\| \quad \text{and} \quad \|\ell A\| \geq \|\ell \hat{Q}_{Nst}\|, \quad (56)$$

where \hat{P}_{Nt} denotes the N lexicographically smallest multicombinations $d_t \dots d_2 d_1$ that satisfy (6), and \hat{Q}_{Nst} denotes the N lexicographically largest. ■

Both of these theorems are consequences of a stronger result that we shall prove later. Theorem K is generally called the Kruskal–Katona theorem, because it was discovered by J. B. Kruskal [Math. Optimization Techniques, edited by R. Bellman (1963), 251–278] and rediscovered by G. Katona [Theory of Graphs, Tihany 1966, edited by Erdős and Katona, (Academic Press, 1968), 187–207]; M. P. Schützenberger had previously stated it in a less-well-known publication, with incomplete proof [RLE Quarterly Progress Report 55 (1959), 117–118]. Theorem M goes back to F. S. Macaulay, many years earlier [Proc. London Math. Soc. (2) 26 (1927), 531–555].

Before proving (55) and (56), let's take a closer look at what those formulas mean. We know from Theorem L that the first N combinations visited by Algorithm L are those that precede $n_t \dots n_2 n_1$, where

$$N = \binom{n_t}{t} + \dots + \binom{n_2}{2} + \binom{n_1}{1}, \quad n_t > \dots > n_2 > n_1 \geq 0$$

is the degree- t combinatorial representation of N . Sometimes this representation has fewer than t nonzero terms, because n_j can be equal to $j - 1$; let's suppress the zeros, and write

$$N = \binom{n_t}{t} + \binom{n_{t-1}}{t-1} + \dots + \binom{n_v}{v}, \quad n_t > n_{t-1} > \dots > n_v \geq v \geq 1. \quad (57)$$

Now the first $\binom{n_t}{t}$ combinations $c_t \dots c_1$ are the t -combinations of $\{0, \dots, n_t - 1\}$; the next $\binom{n_{t-1}}{t-1}$ are those in which $c_t = n_t$ and $c_{t-1} \dots c_1$ is a $(t-1)$ -combination of $\{0, \dots, n_{t-1} - 1\}$; and so on. For example, if $t = 5$ and $N = \binom{9}{5} + \binom{7}{4} + \binom{4}{3}$, the first N combinations are

$$P_{N5} = \{43210, \dots, 87654\} \cup \{93210, \dots, 96543\} \cup \{97210, \dots, 97321\}. \quad (58)$$

The shadow of this set P_{N5} is, fortunately, easy to understand: It is

$$\partial P_{N5} = \{3210, \dots, 8765\} \cup \{9210, \dots, 9654\} \cup \{9710, \dots, 9732\}, \quad (59)$$

namely the first $\binom{9}{4} + \binom{7}{3} + \binom{4}{2}$ combinations in lexicographic order when $t = 4$.

In other words, if we define Kruskal's function κ_t by the formula

$$\kappa_t N = \binom{n_t}{t-1} + \binom{n_{t-1}}{t-2} + \dots + \binom{n_v}{v-1} \quad (60)$$

when N has the unique representation (57), we have

$$\partial P_{Nt} = P_{(\kappa_t N)(t-1)}. \quad (61)$$

Theorem K tells us, for example, that a graph with a million edges can contain at most

$$\binom{1414}{3} + \binom{1009}{2} = 470,700,300$$

triangles, that is, at most 470,700,300 sets of vertices $\{u, v, w\}$ with $u — v — w — u$. The reason is that $1000000 = \binom{1414}{2} + \binom{1009}{1}$ by exercise 17, and the edges $P_{(1000000)_2}$ do support $\binom{1414}{3} + \binom{1009}{2}$ triangles; but if there were more, the graph would necessarily have at least $\kappa_3 470700301 = \binom{1414}{2} + \binom{1009}{1} + \binom{1}{0} = 1000001$ edges in their shadow.

Kruskal defined the companion function

$$\lambda_t N = \binom{n_t}{t+1} + \binom{n_{t-1}}{t} + \cdots + \binom{n_v}{v+1} \quad (62)$$

to deal with questions such as this. The κ and λ functions are related by an interesting law proved in exercise 71:

$$M + N = \binom{s+t}{t} \text{ implies } \kappa_s M + \lambda_t N = \binom{s+t}{t+1}, \text{ if } st > 0. \quad (63)$$

Turning to Theorem M, the sizes of $\partial \hat{P}_{Nt}$ and $\partial \hat{Q}_{Nst}$ turn out to be

$$\|\partial \hat{P}_{Nt}\| = \mu_t N \quad \text{and} \quad \|\partial \hat{Q}_{Nst}\| = N + \kappa_s N \quad (64)$$

(see exercise 80), where the function μ_t satisfies

$$\mu_t N = \binom{n_t - 1}{t - 1} + \binom{n_{t-1} - 1}{t - 2} + \cdots + \binom{n_v - 1}{v - 1} \quad (65)$$

when N has the combinatorial representation (57).

Table 3 shows how these functions $\kappa_t N$, $\lambda_t N$, and $\mu_t N$ behave for small values of t and N . When t and N are large, they can be well approximated in terms of a remarkable function $\tau(x)$ introduced by Teiji Takagi in 1903; see Fig. 27 and exercises 81–84.

Theorems K and M are corollaries of a much more general theorem of discrete geometry, discovered by Da-Lun Wang and Ping Wang [SIAM J. Applied Math. **33** (1977), 55–59], which we shall now proceed to investigate. Consider the *discrete n-dimensional torus* $T(m_1, \dots, m_n)$ whose elements are integer vectors $x = (x_1, \dots, x_n)$ with $0 \leq x_1 < m_1, \dots, 0 \leq x_n < m_n$. We define the sum and difference of two such vectors x and y as in Eqs. 4.3.2–(2) and 4.3.2–(3):

$$x + y = ((x_1 + y_1) \bmod m_1, \dots, (x_n + y_n) \bmod m_n), \quad (66)$$

$$x - y = ((x_1 - y_1) \bmod m_1, \dots, (x_n - y_n) \bmod m_n). \quad (67)$$

We also define the so-called *cross order* on such vectors by saying that $x \preceq y$ if and only if

$$\nu x < \nu y \quad \text{or} \quad (\nu x = \nu y \text{ and } x \geq y \text{ lexicographically}); \quad (68)$$

here, as usual, $\nu(x_1, \dots, x_n) = x_1 + \dots + x_n$. For example, when $m_1 = m_2 = 2$ and $m_3 = 3$, the 12 vectors $x_1 x_2 x_3$ in cross order are

$$000, 100, 010, 001, 110, 101, 011, 002, 111, 102, 012, 112, \quad (69)$$

Table 3
EXAMPLES OF THE KRUSKAL-MACAULAY FUNCTIONS κ , λ , AND μ

$N =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\kappa_1 N =$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
$\kappa_2 N =$	0	2	3	3	4	4	4	5	5	5	5	6	6	6	6	7	7	7	7	7	
$\kappa_3 N =$	0	3	5	6	6	8	9	9	10	10	10	12	13	13	14	14	14	15	15	15	
$\kappa_4 N =$	0	4	7	9	10	10	13	15	16	16	18	19	19	20	20	20	23	25	26	26	28
$\kappa_5 N =$	0	5	9	12	14	15	15	19	22	24	25	25	28	30	31	31	33	34	35	35	35
$\lambda_1 N =$	0	0	1	3	6	10	15	21	28	36	45	55	66	78	91	105	120	136	153	171	190
$\lambda_2 N =$	0	0	0	1	1	2	4	4	5	7	10	10	11	13	16	20	20	21	23	26	30
$\lambda_3 N =$	0	0	0	0	1	1	1	2	2	3	5	5	5	6	6	7	9	9	10	12	15
$\lambda_4 N =$	0	0	0	0	0	1	1	1	1	2	2	2	3	3	4	6	6	6	6	7	7
$\lambda_5 N =$	0	0	0	0	0	1	1	1	1	1	2	2	2	3	3	3	4	4	4	5	
$\mu_1 N =$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\mu_2 N =$	0	1	2	2	3	3	3	4	4	4	4	5	5	5	5	5	6	6	6	6	6
$\mu_3 N =$	0	1	2	3	3	4	5	5	6	6	6	7	8	8	9	9	9	10	10	10	10
$\mu_4 N =$	0	1	2	3	4	4	5	6	7	7	8	9	9	10	10	10	11	12	13	13	14
$\mu_5 N =$	0	1	2	3	4	5	5	6	7	8	9	9	10	11	12	12	13	14	14	15	15

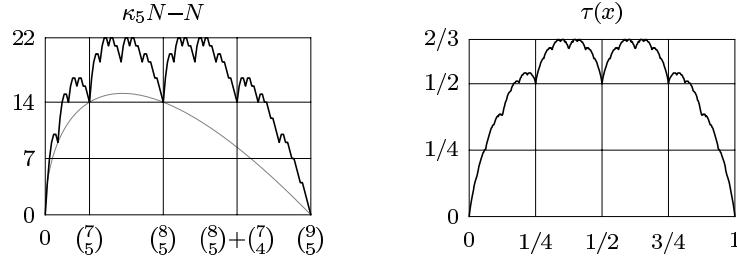


Fig. 27. Approximating a Kruskal function with the Takagi function. (The smooth curve in the left-hand graph is the lower bound $\underline{\kappa}_5 N - N$ of exercise 79.)

omitting parentheses and commas for convenience. The *complement* of a vector in $T(m_1, \dots, m_n)$ is

$$\bar{x} = (m_1 - 1 - x_1, \dots, m_n - 1 - x_n). \quad (70)$$

Notice that $x \preceq y$ holds if and only if $\bar{x} \succeq \bar{y}$. Therefore we have

$$\text{rank}(x) + \text{rank}(\bar{x}) = T - 1, \quad \text{where } T = m_1 \dots m_n, \quad (71)$$

if $\text{rank}(x)$ denotes the number of vectors that precede x in cross order.

We will find it convenient to call the vectors “points” and to name the points e_0, e_1, \dots, e_{T-1} in increasing cross order. Thus we have $e_7 = 002$ in (69), and $\bar{e}_r = e_{T-1-r}$ in general. Notice that

$$e_1 = 100\dots00, \quad e_2 = 010\dots00, \quad \dots, \quad e_n = 000\dots01; \quad (72)$$

these are the so-called *unit vectors*. The set

$$S_N = \{e_0, e_1, \dots, e_{N-1}\} \quad (73)$$

consisting of the smallest N points is called a *standard set*, and in the special case $N = n + 1$ we write

$$E = \{e_0, e_1, \dots, e_n\} = \{000\dots00, 100\dots00, 010\dots00, \dots, 000\dots01\}. \quad (74)$$

Any set of points X has a *spread* X^+ , a *core* X° , and a *dual* X^\sim , defined by the rules

$$X^+ = \{x \in S_T \mid x \in X \text{ or } x - e_1 \in X \text{ or } \dots \text{ or } x - e_n \in X\}; \quad (75)$$

$$X^\circ = \{x \in S_T \mid x \in X \text{ and } x + e_1 \in X \text{ and } \dots \text{ and } x + e_n \in X\}; \quad (76)$$

$$X^\sim = \{x \in S_T \mid \bar{x} \notin X\}. \quad (77)$$

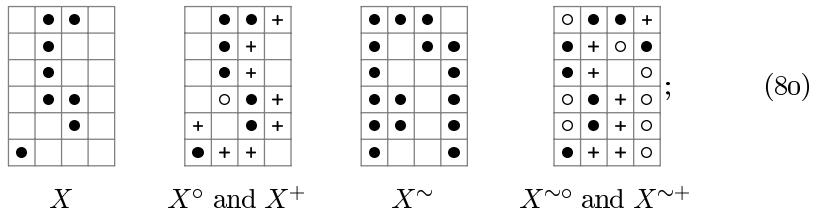
We can also define the spread of X algebraically, writing

$$X^+ = X + E, \quad (78)$$

where $X + Y$ denotes $\{x + y \mid x \in X \text{ and } y \in Y\}$. Clearly

$$X^+ \subseteq Y \quad \text{if and only if} \quad X \subseteq Y^\circ. \quad (79)$$

These notions can be illustrated in the two-dimensional case $m_1 = 4$, $m_2 = 6$, by the more-or-less random toroidal arrangements



here X consists of points marked \bullet or \circ , X° comprises just the \circ s, and X^+ consists of $+$ s plus \bullet s plus \circ s. Notice that if we rotate the diagram for $X^{\sim\circ}$ and $X^{\sim+}$ by 180° , we obtain the diagram for X° and X^+ , but with $(\bullet, \circ, +,)$ respectively changed to $(+, \bullet, \circ)$; and in fact the identities

$$X^\circ = X^{\sim\circ}, \quad X^+ = X^{\sim+} \quad (81)$$

hold in general (see exercise 85).

Now we are ready to state the theorem of Wang and Wang:

Theorem W. *Let X be any set of N points in the discrete torus $T(m_1, \dots, m_n)$, where $m_1 \leq \dots \leq m_n$. Then $\|X^+\| \geq \|S_N^+\|$ and $\|X^\circ\| \leq \|S_N^\circ\|$.*

In other words, the standard sets S_N have the smallest spread and largest core, among all N -point sets. We will prove this result by following a general approach first used by F. W. J. Whipple to prove Theorem M [Proc. London Math. Soc. (2) **28** (1928), 431–437]. The first step is to prove that the spread and the core of standard sets are standard:

Lemma S. There are functions α and β such that $S_N^+ = S_{\alpha N}$ and $S_N^\circ = S_{\beta N}$.

Proof. We may assume that $N > 0$. Let r be maximum with $e_r \in S_N^+$, and let $\alpha N = r + 1$; we must prove that $e_q \in S_N^+$ for $0 \leq q < r$. Suppose $e_q = x = (x_1, \dots, x_n)$ and $e_r = y = (y_1, \dots, y_n)$, and let k be the largest subscript with $x_k > 0$. Since $y \in S_N^+$, there is a subscript j such that $y - e_j \in S_N$. It suffices to prove that $x - e_k \preceq y - e_j$, and exercise 87 does this.

The second part follows from (81), with $\beta N = T - \alpha(T - N)$, because $S_N^\sim = S_{T-N}$. ■

Theorem W is obviously true when $n = 1$, so we assume by induction that it has been proved in $n - 1$ dimensions. The next step is to *compress* the given set X in the k th coordinate position, by partitioning it into disjoint sets

$$X_k(a) = \{x \in X \mid x_k = a\} \quad (82)$$

for $0 \leq a < m_k$ and replacing each $X_k(a)$ by

$$X'_k(a) = \{(s_1, \dots, s_{k-1}, a, s_k, \dots, s_{n-1}) \mid (s_1, \dots, s_{n-1}) \in S_{\|X_k(a)\|}\}, \quad (83)$$

a set with the same number of elements. The sets S used in (83) are standard in the $(n - 1)$ -dimensional torus $T(m_1, \dots, m_{k-1}, m_{k+1}, \dots, m_n)$. Notice that we have $(x_1, \dots, x_{k-1}, a, x_{k+1}, \dots, x_n) \preceq (y_1, \dots, y_{k-1}, a, y_{k+1}, \dots, y_n)$ if and only if $(x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_n) \preceq (y_1, \dots, y_{k-1}, y_{k+1}, \dots, y_n)$; therefore $X'_k(a) = X_k(a)$ if and only if the $(n - 1)$ -dimensional points $(x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_n)$ with $(x_1, \dots, x_{k-1}, a, x_{k+1}, \dots, x_n) \in X$ are as small as possible when projected onto the $(n - 1)$ -dimensional torus. We let

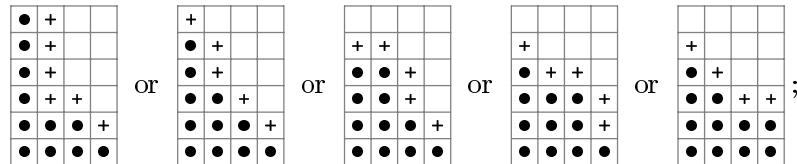
$$C_k X = X'_k(0) \cup X'_k(1) \cup \dots \cup X'_k(m_k - 1) \quad (84)$$

be the compression of X in position k . Exercise 89 proves the basic fact that compression does not increase the size of the spread:

$$\|X^+\| \geq \|(C_k X)^+\|, \quad \text{for } 1 \leq k \leq n. \quad (85)$$

Furthermore, if compression changes X , it replaces some of the elements by other elements of lower rank. Therefore we need only prove Theorem W for sets X that are totally compressed, having $X = C_k X$ for all k .

Consider, for example, the case $n = 2$. A totally compressed set in two dimensions has all points moved to the left of their rows and the bottom of their columns, as in



the rightmost of these is standard, and has the smallest spread. Exercise 90 completes the proof of Theorem W in two dimensions.

When $n > 2$, suppose $x = (x_1, \dots, x_n) \in X$ and $x_j > 0$. The condition $C_k X = X$ implies that, if $0 \leq i < j$ and $i \neq k \neq j$, we have $x + e_i - e_j \in X$. Applying this fact for three values of k tells us that $x + e_i - e_j \in X$ whenever $0 \leq i < j$. Consequently

$$X_n(a) + E_n(0) \subseteq X_n(a-1) + e_n \quad \text{for } 0 < a < m, \quad (86)$$

where $m = m_n$ and $E_n(0)$ is a clever abbreviation for the set $\{e_0, \dots, e_{n-1}\}$.

Let $X_n(a)$ have N_a elements, so that $N = \|X\| = N_0 + N_1 + \dots + N_{m-1}$, and let $Y = X^+$. Then

$$Y_n(a) = (X_n((a-1) \bmod m) + e_n) \cup (X_n(a) + E_n(0))$$

is standard in $n-1$ dimensions, and (86) tells us that

$$N_{m-1} \leq \beta N_{m-2} \leq N_{m-3} \leq \dots \leq N_1 \leq \beta N_0 \leq N_0 \leq \alpha N_0,$$

where α and β refer to coordinates 1 through $n-1$. Therefore

$$\begin{aligned} \|Y\| &= \|Y_n(0)\| + \|Y_n(1)\| + \|Y_n(2)\| + \dots + \|Y_n(m-1)\| \\ &= \alpha N_0 + N_0 + N_1 + \dots + N_{m-2} = \alpha N_0 + N - N_{m-1}. \end{aligned}$$

The proof of Theorem W now has a beautiful conclusion. Let $Z = S_N$, and suppose $\|Z_n(a)\| = M_a$. We want to prove that $\|Y\| \geq \|Z^+\|$, namely that

$$\alpha N_0 + N - N_{m-1} \geq \alpha M_0 + N - M_{m-1}, \quad (87)$$

because the arguments of the previous paragraph apply to Z as well as to X . We will prove (87) by showing that $N_{m-1} \leq M_{m-1}$ and $N_0 \geq M_0$.

Using the $(n-1)$ -dimensional α and β functions, let us define

$$N'_{m-1} = N_{m-1}, \quad N'_{m-2} = \alpha N'_{m-1}, \quad \dots, \quad N'_1 = \alpha N'_2, \quad N'_0 = \alpha N'_1; \quad (88)$$

$$N''_0 = N_0, \quad N''_1 = \beta N''_0, \quad N''_2 = \beta N''_1, \quad \dots, \quad N''_{m-1} = \beta N''_{m-2}. \quad (89)$$

Then we have $N'_a \leq N_a \leq N''_a$ for $0 \leq a < m$, and it follows that

$$N' = N'_0 + N'_1 + \dots + N'_{m-1} \leq N \leq N'' = N''_0 + N''_1 + \dots + N''_{m-1}. \quad (89)$$

Exercise 91 proves that the standard set $Z'' = S_{N''}$ has exactly N''_a elements with n th coordinate equal to a , for each a ; and by the duality between α and β , the standard set $Z' = S_{N'}$ likewise has exactly N'_a elements with n th coordinate a . Finally, therefore,

$$\begin{aligned} M_{m-1} &= \|Z_n(m-1)\| \geq \|Z'_n(m-1)\| = N_{m-1}, \\ M_0 &= \|Z_n(0)\| \leq \|Z''_n(0)\| = N_0, \end{aligned}$$

because $Z' \subseteq Z \subseteq Z''$ by (89). ■

Now we are ready to prove the claimed results about shadows, which are in fact special cases of a more general theorem of Clements and Lindström that applies to arbitrary multisets [*J. Combinatorial Theory* 7 (1969), 230–238]:

Corollary C. If A is a set of N t -multicombinations contained in the multiset $U = \{s_0 \cdot 0, s_1 \cdot 1, \dots, s_d \cdot d\}$, where $s_0 \geq s_1 \geq \dots \geq s_d$, then

$$\|\partial A\| \geq \|\partial P_{Nt}\| \quad \text{and} \quad \|\mathcal{C}A\| \geq \|\mathcal{C}Q_{Nt}\|, \quad (90)$$

where P_{Nt} denotes the N lexicographically smallest multicombinations $c_t \dots c_2 c_1$ of U , and Q_{Nt} denotes the N lexicographically largest.

Proof. Multicombinations of U can be represented as points $x_1 \dots x_n$ of the torus $T(m_1, \dots, m_n)$, where $n = d + 1$ and $m_j = s_{n-j} + 1$; we let x_j be the number of occurrences of $n - j$. This correspondence preserves lexicographic order. For example, if $U = \{0, 0, 0, 1, 1, 2, 3\}$, its 3-multicombinations are

$$000, 100, 110, 200, 210, 211, 300, 310, 311, 321, \quad (91)$$

in lexicographic order, and the corresponding points $x_1 x_2 x_3 x_4$ are

$$0003, 0012, 0021, 0102, 0111, 0120, 1002, 1011, 1020, 1110. \quad (92)$$

Let T_w be the points of the torus that have weight $x_1 + \dots + x_n = w$. Then every allowable set A of t -multicombinations is a subset of T_t . Furthermore—and this is the main point—the spread of $T_0 \cup T_1 \cup \dots \cup T_{t-1} \cup A$ is

$$\begin{aligned} (T_0 \cup T_1 \cup \dots \cup T_{t-1} \cup A)^+ &= T_0^+ \cup T_1^+ \cup \dots \cup T_{t-1}^+ \cup A^+ \\ &= T_0 \cup T_1 \cup \dots \cup T_t \cup \mathcal{C}A. \end{aligned} \quad (93)$$

Thus the upper shadow $\mathcal{C}A$ is simply $(T_0 \cup T_1 \cup \dots \cup T_{t-1} \cup A)^+ \cap T_{t+1}$, and Theorem W tells us that $\|A\| = N$ implies $\|\mathcal{C}A\| \geq \|\mathcal{C}(S_{M+N} \cap T_t)\|$, where $M = \|T_0 \cup \dots \cup T_{t-1}\|$. Hence, by the definition of cross order, $S_{M+N} \cup T_t$ consists of the lexicographically largest N t -multicombinations, namely Q_{Nt} .

The proof that $\|\partial A\| \geq \|\partial P_{Nt}\|$ follows by complementation (see exercise 93). ■

EXERCISES

1. [M23] Explain why Golomb's rule (8) makes all sets $\{c_1, \dots, c_t\} \subseteq \{0, \dots, n-1\}$ correspond uniquely to multisets $\{e_1, \dots, e_t\} \subseteq \{\infty \cdot 0, \dots, \infty \cdot n-t\}$.
2. [16] What path in an 11×13 grid corresponds to the bit string (13)?
3. [21] (R. R. Fenichel, 1968.) Show that the compositions $q_t + \dots + q_1 + q_0$ of s into $t+1$ nonnegative parts can be generated in lexicographic order by a simple loopless algorithm.
4. [16] Show that every composition $q_t \dots q_0$ of s into $t+1$ nonnegative parts corresponds to a composition $r_s \dots r_0$ of t into $s+1$ nonnegative parts. What composition corresponds to 10224000001010 under this correspondence?
5. [20] What is a good way to generate all of the integer solutions to the following systems of inequalities?
 - a) $n > x_t \geq x_{t-1} > x_{t-2} \geq x_{t-3} > \dots > x_1 \geq 0$, when t is odd.
 - b) $n \gg x_t \gg x_{t-1} \gg \dots \gg x_2 \gg x_1 \gg 0$, where $a \gg b$ means $a \geq b+2$.
6. [M22] How often is each step of Algorithm T performed?

7. [22] Design an algorithm that runs through the “dual” combinations $b_s \dots b_2 b_1$ in *decreasing* lexicographic order (see (5) and Table 1). Like Algorithm T, your algorithm should avoid redundant assignments and unnecessary searching.

8. [M23] Design an algorithm that generates all (s, t) -combinations $a_{n-1} \dots a_1 a_0$ lexicographically in bitstring form. The total running time should be $O(\binom{n}{t})$, assuming that $st > 0$.

9. [M26] When all (s, t) -combinations $a_{n-1} \dots a_1 a_0$ are listed in lexicographic order, let $2A_{st}$ be the total number of bit changes between adjacent strings. For example, $A_{33} = 25$ because there are respectively

$$2 + 2 + 2 + 4 + 2 + 2 + 4 + 2 + 2 + 6 + 2 + 2 + 4 + 2 + 2 + 4 + 2 + 2 + 2 = 50$$

bit changes between the 20 strings in Table 1.

- a) Show that $A_{st} = \min(s, t) + A_{(s-1)t} + A_{s(t-1)}$ when $st > 0$; $A_{st} = 0$ when $st = 0$.
- b) Prove that $A_{st} < 2\binom{s+t}{t}$.

► **10.** [21] The “World Series” of baseball is traditionally a competition in which the American League champion (A) plays the National League champion (N) until one of them has beaten the other four times. What is a good way to list all possible scenarios AAAA, AAANA, AAANNA, ..., NNNN? What is a simple way to assign consecutive integers to those scenarios?

11. [19] Which of the scenarios in exercise 10 occurred most often during the 1900s? Which of them never occurred? [Hint: World Series scores are easily found on the Internet.]

12. [HM32] A set V of n -bit vectors that is closed under addition modulo 2 is called a *binary vector space*.

- a) Prove that every such V contains 2^t elements, for some integer t , and can be represented as the set $\{x_1\alpha_1 \oplus \dots \oplus x_t\alpha_t \mid 0 \leq x_1, \dots, x_t \leq 1\}$ where the vectors $\alpha_1, \dots, \alpha_t$ form a “canonical basis” with the following property: There is a t -combination $c_t \dots c_2 c_1$ of $\{0, 1, \dots, n-1\}$ such that, if α_k is the binary vector $a_{k(n-1)} \dots a_{k1} a_{k0}$, we have

$$a_{kc_j} = [j=k] \quad \text{for } 1 \leq j, k \leq t; \quad a_{kl} = 0 \quad \text{for } 0 \leq l < c_k, 1 \leq k \leq t.$$

For example, the canonical bases with $n = 9$, $t = 4$, and $c_4 c_3 c_2 c_1 = 7641$ have the general form

$$\begin{aligned} \alpha_1 &= *00*0**10, \\ \alpha_2 &= *00*10000, \\ \alpha_3 &= *01000000, \\ \alpha_4 &= *10000000; \end{aligned}$$

there are 2^8 ways to replace the eight asterisks by 0s and/or 1s, and each of these defines a canonical basis. We call t the dimension of V .

- b) How many t -dimensional spaces are possible with n -bit vectors?
- c) Design an algorithm to generate all canonical bases $(\alpha_1, \dots, \alpha_t)$ of dimension t . Hint: Let the associated combinations $c_t \dots c_1$ increase lexicographically as in Algorithm L.
- d) What is the 1000000th basis visited by your algorithm when $n = 9$ and $t = 4$?

13. [25] A one-dimensional *Ising configuration* of length n , weight t , and energy r , is a binary string $a_{n-1} \dots a_0$ such that $\sum_{j=0}^{n-1} a_j = t$ and $\sum_{j=1}^{n-1} b_j = r$, where $b_j =$

$a_j \oplus a_{j-1}$. For example, $a_{12} \dots a_0 = 1100100100011$ has weight 6 and energy 6, since $b_{12} \dots b_1 = 010110110010$.

Design an algorithm to generate all such configurations, given n , t , and r .

- 14.** [26] When the binary strings $a_{n-1} \dots a_1 a_0$ of (s, t) -combinations are generated in lexicographic order, we sometimes need to change $2\min(s, t)$ bits to get from one combination to the next. For example, 011100 is followed by 100011 in Table 1. Therefore we apparently cannot hope to generate all combinations with a loopless algorithm unless we visit them in some other order.

Show, however, that there actually is a way to compute the lexicographic successor of a given combination in $O(1)$ steps, if each combination is represented indirectly in a doubly linked list as follows: There are arrays $l[0], \dots, l[n]$ and $r[0], \dots, r[n]$ such that $l[r[j]] = j$ for $0 \leq j \leq n$. If $x_0 = l[0]$ and $x_j = l[x_{j-1}]$ for $0 < j < n$, then $a_j = [x_j > s]$ for $0 \leq j < n$.

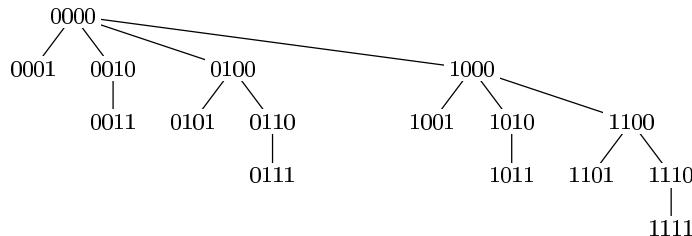
- 15.** [M22] Use the fact that dual combinations $b_s \dots b_2 b_1$ occur in reverse lexicographic order to prove that the sum $\binom{b_s}{s} + \dots + \binom{b_2}{2} + \binom{b_1}{1}$ has a simple relation to the sum $\binom{c_t}{t} + \dots + \binom{c_2}{2} + \binom{c_1}{1}$.

- 16.** [M21] What is the millionth combination generated by Algorithm L when t is (a) 2? (b) 3? (c) 4? (d) 5? (e) 1000000?

- 17.** [HM25] Given N and t , what is a good way to compute the combinatorial representation (20)?

- **18.** [20] What binary tree do we get when the binomial tree T_n is represented by “right child” and “left sibling” pointers as in exercise 2.3.2–5?

- 19.** [21] Instead of labeling the branches of the binomial tree T_4 as shown in (22), we could label each node with the bit string of its corresponding combination:



If T_∞ has been labeled in this way, suppressing leading zeros, preorder is the same as the ordinary increasing order of binary notation; so the millionth node turns out to be 11110100001000111111. But what is the millionth node of T_∞ in *postorder*?

- 20.** [M20] Find generating functions g and h such that Algorithm F finds exactly $[z^N]g(z)$ feasible combinations and sets $t \leftarrow t + 1$ exactly $[z^N]h(z)$ times.

- 21.** [M22] Prove the alternating combination law (30).

- 22.** [M23] What is the millionth revolving-door combination visited by Algorithm R when t is (a) 2? (b) 3? (c) 4? (d) 5? (e) 1000000?

- 23.** [M23] Suppose we augment Algorithm R by setting $j \leftarrow t + 1$ in step R1, and $j \leftarrow 1$ if R3 goes directly to R2. Find the probability distribution of j , and its average value. What does this imply about the running time of the algorithm?

- 24. [M25] (W. H. Payne, 1974.) Continuing the previous exercise, let j_k be the value of j on the k th visit by Algorithm R. Show that $|j_{k+1} - j_k| \leq 2$, and explain how to make the algorithm loopless by exploiting this property.
25. [M35] Let $c_t \dots c_2 c_1$ and $c'_t \dots c'_2 c'_1$ be the N th and N' th combinations generated by the revolving-door method, Algorithm R. If the set $C = \{c_t, \dots, c_2, c_1\}$ has m elements not in $C' = \{c'_t, \dots, c'_2, c'_1\}$, prove that $|N - N'| > \sum_{k=1}^{m-1} \binom{2^k}{k-1}$.
26. [26] Do elements of the *ternary* reflected Gray path have properties similar to the revolving-door Gray code Γ_{st} , if we extract only the n -tuples $a_{n-1} \dots a_1 a_0$ such that
(a) $a_{n-1} + \dots + a_1 + a_0 = t$? (b) $\{a_{n-1}, \dots, a_1, a_0\} = \{r \cdot 0, s \cdot 1, t \cdot 2\}$?
- 27. [25] Show that there is a simple way to generate all combinations of *at most* t elements of $\{0, 1, \dots, n-1\}$, using only Gray-code-like transitions $0 \leftrightarrow 1$ and $01 \leftrightarrow 10$. (In other words, each step should either insert a new element, delete an element, or shift an element by ± 1 .) For example,

0000, 0001, 0011, 0010, 0110, 0101, 0100, 1100, 1010, 1001, 1000

is one such sequence when $n = 4$ and $t = 2$. *Hint:* Think of Chinese rings.

28. [M21] True or false: A listing of (s, t) -combinations $a_{n-1} \dots a_1 a_0$ in bitstring form is in genlex order if and only if the corresponding index-form listings $b_s \dots b_2 b_1$ (for the 0s) and $c_t \dots c_2 c_1$ (for the 1s) are both in genlex order.

- 29. [M28] (P. J. Chase.) Given a string on the symbols $+$, $-$, and 0 , say that an *R-block* is a substring of the form $-^{k+1}$ that is preceded by 0 and not followed by $-$; an *L-block* is a substring of the form $+-^k$ that is followed by 0 ; in both cases $k \geq 0$. For example, the string $\boxed{+}00++-\boxed{-}000$ has two L-blocks and one R-block, shown in gray. Notice that blocks cannot overlap.

We form the *successor* of such a string as follows, whenever at least one block is present: Replace the rightmost $0-^{k+1}$ by $-+^k 0$, if the rightmost block is an R-block; otherwise replace the rightmost $+-^k 0$ by $0+^{k+1}$. Also negate the first sign, if any, that appears to the right of the block that has been changed. For example,

$$\boxed{+}00++- \rightarrow -0\boxed{+}0\boxed{+-} \rightarrow -0\boxed{+}0\boxed{-} \rightarrow -0+-\boxed{+}0 \rightarrow -0\boxed{+}\boxed{-}0+ \rightarrow -00++-,$$

where the notation $\alpha \rightarrow \beta$ means that β is the successor of α .

- a) What strings have no blocks (and therefore no successor)?
- b) Can there be a cycle of strings with $\alpha_0 \rightarrow \alpha_1 \rightarrow \dots \rightarrow \alpha_{k-1} \rightarrow \alpha_0$?
- c) Prove that if $\alpha \rightarrow \beta$ then $-\beta \rightarrow -\alpha$, where “ $-$ ” means “negate all the signs.” (Therefore every string has at most one predecessor.)
- d) Show that if $\alpha_0 \rightarrow \alpha_1 \rightarrow \dots \rightarrow \alpha_k$ and $k > 0$, the strings α_0 and α_k do not have all their 0s in the same positions. (Therefore, if α_0 has s signs and t zeros, k must be less than $\binom{s+t}{t}$.)
- e) Prove that every string α with s signs and t zeros belongs to exactly one chain $\alpha_0 \rightarrow \alpha_1 \rightarrow \dots \rightarrow \alpha_{\binom{s+t}{t}-1}$.

30. [M32] The previous exercise defines 2^s ways to generate all combinations of s 0s and t 1s, via the mapping $+$ $\mapsto 0$, $-$ $\mapsto 0$, and 0 $\mapsto 1$. Show that each of these ways is a homogeneous genlex sequence, definable by an appropriate recurrence. Is Chase’s sequence (37) a special case of this general construction?

31. [M23] How many genlex listings of (s, t) -combinations are possible in (a) bitstring form $a_{n-1} \dots a_1 a_0$? (b) index-list form $c_t \dots c_2 c_1$?

- ▶ **32.** [M30] How many of the genlex listings of (s, t) -combination strings $a_{n-1} \dots a_1 a_0$ (a) have the revolving-door property? (b) are homogeneous?
- 33.** [HM31] How many of the genlex listings in exercise 31(b) are near-perfect?
- 34.** [M30] Continuing exercise 33, explain how to find such schemes that are as near as possible to perfection, in the sense that the number of “imperfect” transitions $c_j \leftarrow c_j \pm 2$ is minimized, when s and t are not too large.
- 35.** [M26] How many steps of Chase’s sequence C_{st} use an imperfect transition?
- ▶ **36.** [M21] Prove that method (39) performs the operation $j \leftarrow j + 1$ a total of exactly $\binom{s+t}{t} - 1$ times as it generates all (s, t) -combinations $a_{n-1} \dots a_1 a_0$, given any genlex scheme for combinations in bitstring form.
- ▶ **37.** [27] What algorithm results when the general genlex method (39) is used to produce (s, t) -combinations $a_{n-1} \dots a_1 a_0$ in (a) lexicographic order? (b) the revolving-door order of Algorithm R? (c) the homogeneous order of (31)?
- 38.** [26] Design a genlex algorithm like Algorithm C for the reverse sequence C_{st}^R .
- 39.** [M21] When $s = 12$ and $t = 14$, how many combinations precede the bit string 1100100100001111101101010 in Chase’s sequence C_{st} ? (See (41).)
- 40.** [M22] What is the millionth combination in Chase’s sequence C_{st} , when $s = 12$ and $t = 14$?
- 41.** [M27] Show that there is a permutation $c(0), c(1), c(2), \dots$ of the nonnegative integers such that the elements of Chase’s sequence C_{st} are obtained by complementing the least significant $s + t$ bits of the elements $c(k)$ for $0 \leq k < 2^{s+t}$ that have weight $\nu(c(k)) = s$. (Thus the sequence $\bar{c}(0), \dots, \bar{c}(2^n - 1)$ contains, as subsequences, all of the C_{st} for which $s + t = n$, just as Gray binary code $g(0), \dots, g(2^n - 1)$ contains all the revolving-door sequences Γ_{st} .) Explain how to compute the binary representation $c(k) = (\dots a_2 a_1 a_0)_2$ from the binary representation $k = (\dots b_2 b_1 b_0)_2$.
- 42.** [HM34] Use generating functions of the form $\sum_{s,t} g_{st} w^s z^t$ to analyze each step of Algorithm C.
- 43.** [20] Prove or disprove: If $s(x)$ and $p(x)$ denote respectively the successor and predecessor of x in endo-order, then $s(x+1) = p(x) + 1$.
- ▶ **44.** [M21] Let $C_t(n) - 1$ denote the sequence obtained from $C_t(n)$ by striking out all combinations with $c_1 = 0$, then replacing $c_t \dots c_1$ by $(c_t - 1) \dots (c_1 - 1)$ in the combinations that remain. Show that $C_t(n) - 1$ is near-perfect.
- 45.** [32] Exploit endo-order and the expansions sketched in (44) to generate the combinations $c_t \dots c_2 c_1$ of Chase’s sequence $C_t(n)$ with a nonrecursive procedure.
- ▶ **46.** [33] Construct a nonrecursive algorithm for the dual combinations $b_s \dots b_2 b_1$ of Chase’s sequence C_{st} , namely the positions of the zeros in $a_{n-1} \dots a_1 a_0$.
- 47.** [26] Implement the near-perfect multiset permutation method of (46) and (47).
- 48.** [M21] Suppose $\alpha_0, \alpha_1, \dots, \alpha_{N-1}$ is any listing of the permutations of the multiset $\{s_1 \cdot 1, \dots, s_d \cdot d\}$, where α_k differs from α_{k+1} by the interchange of two elements. Let $\beta_0, \dots, \beta_{M-1}$ be any revolving-door listing for (s, t) -combinations, where $s = s_0, t = s_1 + \dots + s_d$, and $M = \binom{s+t}{t}$. Then let Λ_j be the list of M elements obtained by starting with $\alpha_j \uparrow \beta_0$ and applying the revolving-door exchanges; here $\alpha \uparrow \beta$ denotes the string obtained by substituting the elements of α for the 1s in β , preserving left-right order. For example, if $\beta_0, \dots, \beta_{M-1}$ is 0110, 0101, 1100, 1001, 0011, 1010, and if $\alpha_j = 12$,

then Λ_j is 0120, 0102, 1200, 1002, 0012, 1020. (The revolving-door listing need *not* be homogeneous.)

Prove that the list (47) contains all permutations of $\{s_0 \cdot 0, s_1 \cdot 1, \dots, s_d \cdot d\}$, and that adjacent permutations differ from each other by the interchange of two elements.

- 49.** [HM23] If q is a primitive m th root of unity, such as $e^{2\pi i/m}$, show that

$$\binom{n}{k}_q = \left(\begin{matrix} \lfloor n/m \rfloor \\ \lfloor k/m \rfloor \end{matrix} \right) \binom{n \bmod m}{k \bmod m}_q.$$

- **50.** [HM25] Extend the formula of the previous exercise to *q-multinomial* coefficients

$$\binom{n_1 + \dots + n_t}{n_1, \dots, n_t}_q.$$

- 51.** [25] Find all Hamiltonian paths in the graph whose vertices are permutations of $\{0, 0, 0, 1, 1, 1\}$ related by adjacent transposition. Which of those paths are equivalent under the operations of interchanging 0s with 1s and/or left-right reflection?

- 52.** [M37] Generalizing Theorem P, find a necessary and sufficient condition that all permutations of the multiset $\{s_0 \cdot 0, \dots, s_d \cdot d\}$ can be generated by adjacent transpositions $a_j a_{j-1} \leftrightarrow a_{j-1} a_j$.

- 53.** [M46] (D. H. Lehmer, 1965.) Suppose the N permutations of $\{s_0 \cdot 0, \dots, s_d \cdot d\}$ cannot be generated by a perfect scheme, because $(N+x)/2$ of them have an even number of inversions, where $x \geq 2$. Is it possible to generate them all with a sequence of $N+x-2$ adjacent interchanges $a_{\delta_k} \leftrightarrow a_{\delta_{k-1}}$ for $1 \leq k < N+x-1$, where $x-1$ cases are “spurs” with $\delta_k = \delta_{k-1}$ that take us back to the permutation we’ve just seen? For example, a suitable sequence $\delta_1 \dots \delta_{94}$ for the 90 permutations of $\{0, 0, 1, 1, 2, 2\}$, where $x = \binom{2+2+2}{2,2,2} - 1 = 6$, is 234535432523451 α 42 α^R 51 α 42 α^R 51 α 4, where $\alpha = 45352542345355$, if we start with $a_5 a_4 a_3 a_2 a_1 a_0 = 221100$.

- 54.** [M40] For what values of s and t can all (s, t) -combinations be generated if we allow end-around swaps $a_{n-1} \leftrightarrow a_0$ in addition to adjacent interchanges $a_j \leftrightarrow a_{j-1}$?

- 55.** [M49] (Buck and Wiedemann, 1984.) Can all (t, t) -combinations $a_{2t-1} \dots a_1 a_0$ be generated by repeatedly swapping a_0 with some other element?

- **56.** [22] (Frank Ruskey.) Can a piano player run through all possible 4-note chords that span at most one octave, changing only one finger at a time? This is the problem of generating all combinations $c_t \dots c_1$ such that $n > c_t > \dots > c_1 \geq 0$ and $c_t - c_1 < m$, where $t = 4$ and (a) $m = 8$, $n = 52$ if we consider only the white notes of a piano keyboard; (b) $m = 13$, $n = 88$ if we consider also the black notes.

- 57.** [20] Consider the piano player’s problem of exercise 56 with the additional condition that the chords don’t involve adjacent notes. (In other words, $c_{j+1} > c_j + 1$ for $t > j \geq 1$.)

- 58.** [M25] Is there a *perfect* solution to the piano player’s problem, in which each step moves a finger to an *adjacent* key?

- 59.** [23] Design an algorithm to generate all *bounded* compositions

$$t = r_s + \dots + r_1 + r_0, \quad \text{where } 0 \leq r_j \leq m_j \text{ for } s \geq j \geq 0.$$

- 60.** [32] Show that all bounded compositions can be generated by changing only two of the parts at each step.

► **61.** [M27] A *contingency table* is an $m \times n$ matrix of nonnegative integers (a_{ij}) having given row sums $r_i = \sum_{j=1}^n a_{ij}$ and column sums $c_j = \sum_{i=1}^m a_{ij}$, where $r_1 + \dots + r_m = c_1 + \dots + c_n$.

- Show that $2 \times n$ contingency tables are equivalent to bounded compositions.
- What is the lexicographically largest contingency table for $(r_1, \dots, r_m; c_1, \dots, c_n)$, when matrix entries are read row-wise from left to right and top to bottom, namely in the order $(a_{11}, a_{12}, \dots, a_{1n}, a_{21}, \dots, a_{mn})$?
- What is the lexicographically largest contingency table for $(r_1, \dots, r_m; c_1, \dots, c_n)$, when matrix entries are read column-wise from top to bottom and left to right, namely in the order $(a_{11}, a_{21}, \dots, a_{m1}, a_{12}, \dots, a_{mn})$?
- What is the lexicographically smallest contingency table for $(r_1, \dots, r_m; c_1, \dots, c_n)$, in the row-wise and column-wise senses?
- Explain how to generate all contingency tables for $(r_1, \dots, r_m; c_1, \dots, c_n)$ in lexicographic order.

62. [M41] Show that all contingency tables for $(r_1, \dots, r_m; c_1, \dots, c_n)$ can be generated by changing exactly four entries of the matrix at each step.

► **63.** [M30] Construct a genlex Gray code for all of the $2^s \binom{s+t}{t}$ subcubes that have s digits and t asterisks, using only the transformations $*0 \leftrightarrow 0*$, $*1 \leftrightarrow 1*$, $0 \leftrightarrow 1$. For example, one such cycle when $s = t = 2$ is

$$(00**, 01**, 0*1*, 0**1, 0**0, 0*0*, *00*, *01*, *0*1, *0*0, **00, **01, \\ **11, **10, *1*0, *1*1, *11*, *10*, 1*0*, 1**0, 1**1, 1*1*, 11**, 10**).$$

64. [M40] Enumerate the total number of genlex Gray paths on subcubes that use only the transformations allowed in exercise 63. How many of those paths are cycles?

► **65.** [22] Given $n \geq t \geq 0$, show that there is a Gray path through all of the canonical bases $(\alpha_1, \dots, \alpha_t)$ of exercise 12, changing just one bit at each step. For example, one such path when $n = 3$ and $t = 2$ is

$$\begin{array}{ccccccccc} 001 & 101 & 101 & 001 & 001 & 011 & 010 \\ 010' & 010' & 110' & 110' & 100' & 100' & 100' \end{array}.$$

66. [46] Consider the Ising configurations of exercise 13 for which $a_0 = 0$. Given n , t , and r , is there a Gray code for these configurations in which all transitions have the forms $0^k 1 \leftrightarrow 10^k$ or $01^k \leftrightarrow 1^k 0$? For example, in the case $n = 9$, $t = 5$, $r = 6$, there is a unique cycle

$$(010101110, 010110110, 011010110, 011011010, 011101010, 010111010).$$

67. [M01] If α is a t -combination, what is (a) $\partial^t \alpha$? (b) $\partial^{t+1} \alpha$?

► **68.** [M22] How large is the smallest set A of t -combinations for which $\|\partial A\| < \|A\|$?

69. [M25] What is the maximum value of $\kappa_t N - N$, for $N \geq 0$?

70. [M20] How many t -cliques can a million-edge graph have?

► **71.** [M22] Show that if N has the degree- t combinatorial representation (57), there is an easy way to find the degree- s combinatorial representation of the complementary number $M = \binom{s+t}{t} - N$, whenever $N < \binom{s+t}{t}$. Derive (63) as a consequence.

72. [M23] (A. J. W. Hilton, 1976.) Let A be a set of s -combinations and B a set of t -combinations, both contained in $U = \{0, \dots, n-1\}$ where $n \geq s+t$. Show that if A and B are *cross-intersecting*, in the sense that $\alpha \cap \beta \neq \emptyset$ for all $\alpha \in A$ and $\beta \in B$, then so are the sets Q_{Mns} and Q_{Nnt} defined in Theorem K, where $M = \|A\|$ and $N = \|B\|$.

73. [M21] What are $\|\mathcal{Q}P_{Nt}\|$ and $\|\mathcal{Q}Q_{Nnt}\|$ in Theorem K?

74. [M20] The right-hand side of (60) is not always the degree-($t - 1$) combinatorial representation of $\kappa_t N$, because $v - 1$ might be zero. Show, however, that a positive integer N has at most two representations if we allow $v = 0$ in (57), and both of them yield the same value $\kappa_t N$ according to (60). Therefore

$$\kappa_k \kappa_{k+1} \dots \kappa_t N = \binom{n_t}{k-1} + \binom{n_{t-1}}{k-2} + \dots + \binom{n_v}{k-1+v-t} \quad \text{for } 1 \leq k \leq t.$$

75. [M20] Find a simple formula for $\kappa_t(N+1) - \kappa_t N$.

► **76.** [M26] Prove the following properties of the κ functions by manipulating binomial coefficients, without assuming Theorem K:

a) $\kappa_t(M+N) \leq \kappa_t M + \kappa_t N$.

b) $\kappa_t(M+N) \leq \max(\kappa_t M, N) + \kappa_{t-1} N$.

Hint: $\binom{m_t}{t} + \dots + \binom{m_1}{1} + \binom{n_t}{t} + \dots + \binom{n_1}{1}$ is equal to $\binom{m_t \vee n_t}{t} + \dots + \binom{m_1 \vee n_1}{1} + \binom{m_t \wedge n_t}{t} + \dots + \binom{m_1 \wedge n_1}{1}$, where \vee and \wedge denote max and min.

77. [M22] Show that Theorem K follows easily from inequality (b) in the previous exercise. Conversely, both inequalities are simple consequences of Theorem K. *Hint:* Any set A of t -combinations can be written $A = A_1 + A_0 0$, where $A_1 = \{\alpha \in A \mid 0 \notin \alpha\}$.

78. [M23] Prove that if $t \geq 2$, we have $M \geq \mu_t N$ if and only if $M + \lambda_{t-1} M \geq N$.

79. [HM26] (L. Lovász, 1979.) The function $\binom{x}{t}$ increases monotonically from 0 to ∞ as x increases from $t - 1$ to ∞ ; hence we can define

$$\underline{\kappa}_t N = \binom{x}{t-1}, \quad \text{if } N = \binom{x}{t} \text{ and } x \geq t - 1.$$

Prove that $\underline{\kappa}_t N \geq \kappa_t N$ for all integers $t \geq 1$ and $N \geq 0$. *Hint:* Equality holds when x is an integer.

► **80.** [M27] Show that the minimum shadow sizes in Theorem M are given by (64).

81. [HM31] The Takagi function of Fig. 27 is defined for $0 \leq x \leq 1$ by the formula

$$\tau(x) = \sum_{k=1}^{\infty} \int_0^x r_k(t) dt,$$

where $r_k(t) = (-1)^{\lfloor 2^k t \rfloor}$ is the Rademacher function of Eq. 7.2.1.1–(16).

a) Prove that $\tau(x)$ is continuous in the interval $[0..1]$, but its derivative does not exist at any point.

b) Show that $\tau(x)$ is the only continuous function that satisfies

$$\tau(\frac{1}{2}x) = \tau(1 - \frac{1}{2}x) = \frac{1}{2}x + \frac{1}{2}\tau(x) \quad \text{for } 0 \leq x \leq 1.$$

c) What is the asymptotic value of $\tau(\epsilon)$ when ϵ is small?

d) Prove that $\tau(x)$ is rational when x is rational.

e) Find all roots of the equation $\tau(x) = 1/2$.

f) Find all roots of the equation $\tau(x) = \max_{0 \leq x \leq 1} \tau(x)$.

82. [HM46] Determine the set R of all rational numbers r such that the equation $\tau(x) = r$ has uncountably many solutions. If $\tau(x)$ is rational and x is irrational, is it true that $\tau(x) \in R$?

- 83.** [HM27] If $T = \binom{2t-1}{t}$, prove the asymptotic formula

$$\kappa_t N - N = \frac{T}{t} \left(\tau \left(\frac{N}{T} \right) + O \left(\frac{(\log t)^3}{t} \right) \right) \quad \text{for } 0 \leq N \leq T.$$

- 84.** [HM21] Relate the functions $\lambda_t N$ and $\mu_t N$ to the Takagi function $\tau(x)$.

- 85.** [M20] Prove the law of spread/core duality, $X^{\sim+} = X^{\circ\circ}$.

- 86.** [M21] True or false: (a) $X \subseteq Y^\circ$ if and only if $Y^\sim \subseteq X^{\sim\circ}$; (b) $X^{\circ+\circ} = X^\circ$; (c) $\alpha M \leq N$ if and only if $M \leq \beta N$.

- 87.** [M20] Explain why cross order is useful, by completing the proof of Lemma S.

- 88.** [16] Compute the α and β functions for the $2 \times 2 \times 3$ torus (69).

- 89.** [M22] Prove the basic compression lemma, (85).

- 90.** [M24] Prove Theorem W for two-dimensional toruses $T(l, m)$, $l \leq m$.

- 91.** [M28] Let $x = x_1 \dots x_{n-1}$ be the N th element of the torus $T(m_1, \dots, m_{n-1})$, and let S be the set of all elements of $T(m_1, \dots, m_{n-1}, m)$ that are $\preceq x_1 \dots x_{n-1}(m-1)$ in cross order. If N_a elements of S have final component a , for $0 \leq a < m$, prove that $N_{m-1} = N$ and $N_{a-1} = \alpha N_a$ for $1 \leq a < m$, where α is the spread function for standard sets in $T(m_1, \dots, m_{n-1})$.

- 92.** [M25] (a) Find an N for which the conclusion of Theorem W is false when the parameters m_1, m_2, \dots, m_n have not been sorted into nondecreasing order. (b) Where does the proof of that theorem use the hypothesis that $m_1 \leq m_2 \leq \dots \leq m_n$?

- 93.** [M20] Show that the ∂ half of Corollary C follows from the ℓ half. Hint: The complements of the multicombinations (91) with respect to U are 3211, 3210, 3200, 3110, 3100, 3000, 2110, 3100, 2000, 1000.

- 94.** [15] Explain why Theorems K and M follow from Corollary C.

- **95.** [M22] If S is an infinite sequence (s_0, s_1, s_2, \dots) of positive integers, let

$$\binom{S(n)}{k} = [z^k] \prod_{j=0}^{n-1} (1 + z + \dots + z^{s_j});$$

thus $\binom{S(n)}{k}$ is the ordinary binomial coefficient $\binom{n}{k}$ if $s_0 = s_1 = s_2 = \dots = 1$.

Generalizing the combinatorial number system, show that every nonnegative integer N has a unique representation

$$N = \binom{S(n_t)}{t} + \binom{S(n_{t-1})}{t-1} + \dots + \binom{S(n_1)}{1}$$

where $n_t \geq n_{t-1} \geq \dots \geq n_1 \geq 0$ and $\{n_t, n_{t-1}, \dots, n_1\} \subseteq \{s_0 \cdot 0, s_1 \cdot 1, s_2 \cdot 2, \dots\}$. Use this representation to give a simple formula for the numbers $\|\partial P_{Nt}\|$ in Corollary C.

- **96.** [M26] The text remarked that the vertices of a convex polyhedron can be perturbed slightly so that all of its faces are simplexes. In general, any set of combinations that contains the shadows of all its elements is called a *simplicial complex*; thus C is a simplicial complex if and only if $\alpha \subseteq \beta$ and $\beta \in C$ implies that $\alpha \in C$, if and only if C is an order ideal with respect to set inclusion.

The *size vector* of a simplicial complex C on n vertices is (N_0, N_1, \dots, N_n) when C contains exactly N_t combinations of size t .

- a) What are the size vectors of the five regular solids (the tetrahedron, cube, octahedron, dodecahedron, and icosahedron), when their vertices are slightly tweaked?

- b) Construct a simplicial complex with size vector $(1, 4, 5, 2, 0)$.
- c) Find a necessary and sufficient condition that a given size vector (N_0, N_1, \dots, N_n) is feasible.
- d) Prove that (N_0, \dots, N_n) is feasible if and only its “dual” vector $(\bar{N}_0, \dots, \bar{N}_n)$ is feasible, where we define $\bar{N}_t = \binom{n}{t} - N_{n-t}$.
- e) List all feasible size vectors $(N_0, N_1, N_2, N_3, N_4)$ and their duals. Which of them are self-dual?

97. [30] Continuing exercise 96, find an efficient way to count the number of feasible size vectors (N_0, N_1, \dots, N_n) when $n \leq 100$.

98. [M25] A *clutter* is a set C of combinations that are incomparable, in the sense that $\alpha \subseteq \beta$ and $\alpha, \beta \in C$ implies $\alpha = \beta$. The size vector of a clutter is defined as in exercise 96.

- a) Find a necessary and sufficient condition that (M_0, M_1, \dots, M_n) is the size vector of a clutter.
- b) List all such size vectors in the case $n = 4$.

► **99.** [M30] (Clements and Lindström.) Let A be a “simplicial multicomplex,” a set of submultisets of the multiset U in Corollary C with the property that $\partial A \subseteq A$. How large can the total weight $\nu A = \sum\{\|\alpha\| \mid \alpha \in A\}$ be when $\|A\| = N$?

100. [M25] If $f(x_1, \dots, x_n)$ is a Boolean formula, let $F(p)$ be the probability that $f(x_1, \dots, x_n) = 1$ when each variable x_j independently is 1 with probability p .

- a) Calculate $G(p)$ and $H(p)$ for the Boolean formulas $g(w, x, y, z) = wxz \vee wyz \vee xy\bar{z}$, $h(w, x, y, z) = \bar{w}yz \vee xyz$.
- b) Show that there is a *monotone* Boolean function $f(w, x, y, z)$ such that $F(p) = G(p)$, but there is no such function with $F(p) = H(p)$. Explain how to test this condition in general.

101. [HM35] (F. S. Macaulay, 1927.) A *polynomial ideal* I in the variables $\{x_1, \dots, x_s\}$ is a set of polynomials closed under the operations of addition, multiplication by a constant, and multiplication by any of the variables. It is called *homogeneous* if it consists of all linear combinations of a set of homogeneous polynomials, namely of polynomials like $xy + z^2$ whose terms all have the same degree. Let N_t be the maximum number of linearly independent elements of degree t in I . For example, if $s = 2$, the set of all $\alpha(x_0, x_1, x_2)(x_0x_1^2 - 2x_1x_2^2) + \beta(x_0, x_1, x_2)x_0x_1x_2^2$, where α and β run through all possible polynomials in $\{x_0, x_1, x_2\}$, is a homogeneous polynomial ideal with $N_0 = N_1 = N_2 = 0$, $N_3 = 1$, $N_4 = 4$, $N_5 = 9$, $N_6 = 15$,

- a) Prove that for any such ideal I there is another ideal I' in which all homogeneous polynomials of degree t are linear combinations of N_t independent *monomials*. (A monomial is a product of variables, like $x_1^3x_2x_5^4$.)
- b) Use Theorem M and (64) to prove that $N_{t+1} \geq N_t + \kappa_s N_t$ for all $t \geq 0$.
- c) Show that $N_{t+1} > N_t + \kappa_s N_t$ occurs for only finitely many t . (This statement is equivalent to “Hilbert’s basis theorem,” proved by David Hilbert in *Göttinger Nachrichten* (1888), 450–457; *Math. Annalen* **36** (1890), 473–534.)

► **102.** [M38] The shadow of a subcube $a_1 \dots a_n$, where each a_j is either 0 or 1 or *, is obtained by replacing some * by 0 or 1. For example,

$$\partial 0*11*0 = \{0011*0, 011*0, 0*1100, 0*1110\}.$$

Find a set P_{Nst} such that, if A is any set of N subcubes $a_1 \dots a_n$ having s digits and t asterisks, $\|\partial A\| \geq \|P_{Nst}\|$.

103. [M41] The shadow of a binary string $a_1 \dots a_n$ is obtained by deleting one of its bits. For example,

$$\partial 110010010 = \{10010010, 11010010, 11000010, 11001000, 11001001\}.$$

Find a set P_{Nn} such that, if A is any set of N binary strings $a_1 \dots a_n$, $\|\partial A\| \geq \|P_{Nn}\|$.

104. [M20] A *universal cycle* of t -combinations for $\{0, 1, \dots, n - 1\}$ is a cycle of $\binom{n}{t}$ numbers whose blocks of t consecutive elements run through every t -combination $\{c_1, \dots, c_t\}$. For example,

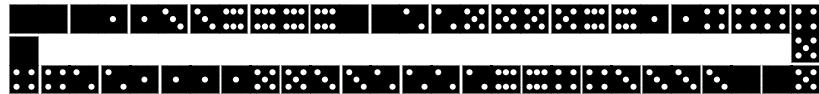
$$(02145061320516243152630425364103546)$$

is a universal cycle when $t = 3$ and $n = 7$.

Prove that no such cycle is possible unless $\binom{n}{t}$ is a multiple of n .

105. [M21] (L. Poinsot, 1809.) Find a “nice” universal cycle of 2-combinations for $\{0, 1, \dots, 2m\}$. Hint: Consider the differences of consecutive elements, mod $(2m + 1)$.

106. [22] (O. Terquem, 1849.) Poinsot’s theorem implies that all 28 dominoes of a traditional “double-six” set can be arranged in a cycle so that the spots of adjacent dominoes match each other:



How many such cycles are possible?

107. [M31] Find universal cycles of 3-combinations for the sets $\{0, \dots, n - 1\}$ when $n \bmod 3 \neq 0$.

108. [M31] Find universal cycles of 3-*multicombinations* for $\{0, 1, \dots, n - 1\}$ when $n \bmod 3 \neq 0$ (namely for combinations $d_1 d_2 d_3$ with repetitions permitted).

► **109. [26]** *Cribbage* is a game played with 52 cards, where each card has a suit (\clubsuit , \diamondsuit , \heartsuit , or \spadesuit) and a face value (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, or K). One feature of the game is to compute the score of a 5-card combination $C = \{c_1, c_2, c_3, c_4, c_5\}$, where one card c_k is called the *starter*. The score is the sum of points computed as follows, for each subset S of C and each choice of k : Let $\|S\| = s$.

- i) Fifteens: If $\sum\{v(c) \mid c \in S\} = 15$, where $(v(\text{A}), v(2), v(3), \dots, v(9), v(10), v(\text{J}), v(\text{Q}), v(\text{K})) = (1, 2, 3, \dots, 9, 10, 10, 10, 10)$, score two points.
- ii) Pairs: If $s = 2$ and both cards have the same face value, score two points.
- iii) Runs: If $s \geq 3$ and the face values are consecutive, and if C does not contain a run of length $s + 1$, score s points.
- iv) Flushes: If $s = 4$ and all cards of S have the same suit, and if $c_k \notin S$, score $4 + [c_k \text{ has the same suit as the others}]$.
- v) Nobs: If $s = 1$ and $c_k \notin S$, score 1 if the card is J of the same suit as c_k .

For example, if you hold $\{\text{J}\clubsuit, 5\clubsuit, 5\diamondsuit, 6\heartsuit\}$ and if $4\clubsuit$ is the starter, you score 4×2 for fifteens, 2 for a pair, 2×3 for runs, plus 1 for nobs, totalling 17.

Exactly how many combinations and starter choices lead to a score of x points, for $x = 0, 1, 2, \dots$?

SECTION 7.2.1.3

1. Given a multiset, form the sequence $e_t \dots e_2 e_1$ from right to left by listing the distinct elements first, then those that appear twice, then those that appear thrice, etc. Let us set $e_{-j} \leftarrow s - j$ for $0 \leq j \leq s$, so that every element e_j for $1 \leq j \leq t$ is equal to some element to its right in the sequence $e_t \dots e_1 e_0 \dots e_{-s}$. If the first such element is e_{c_j-s} , we obtain a solution to (3). Conversely, every solution to (3) yields a unique multiset $\{e_1, \dots, e_t\}$, because $c_j < s + j$ for $1 \leq j \leq t$.

2. Start at the bottom left corner; then go up for each 0, go right for each 1. The result is

3. In this algorithm, variable r is the least positive index such that $q_r > 0$.

F1. [Initialize.] Set $q_j \leftarrow 0$ for $1 \leq j \leq t$, and $q_0 \leftarrow s$. (We assume that $st > 0$.)

F2. [Visit.] Visit the composition $q_t \dots q_0$. Go to F4 if $q_0 = 0$.

F3. [Easy case.] Set $q_0 \leftarrow q_0 - 1$, $r \leftarrow 1$, and go to F5.

F4. [Tricky case.] Terminate if $r = t$. Otherwise set $q_0 \leftarrow q_r - 1$, $q_r \leftarrow 0$, $r \leftarrow r + 1$.

F5. [Increase q_r .] Set $q_r \leftarrow q_r + 1$ and return to F2. ▀

[See CACM 11 (1968), 430; 12 (1969), 187. The task of generating such compositions in *decreasing* lexicographic order is more difficult.]

4. We can reverse the roles of 0 and 1 in (14), so that $0^{q_t} 1^{q_{t-1}} \dots 10^{q_1} 10^{q_0} = 1^{r_s} 0 1^{r_{s-1}} 0 \dots 0 1^{r_1} 0 1^{r_0}$. This gives $0^1 1^{0^1} 10^2 10^2 10^4 10^0 10^0 10^0 10^0 10^0 10^1 10^0 = 1^0 01^2 01^0 01^1 01^0 01^1 01^0 01^0 01^6 01^2 01^1$. Lexicographic order of $a_{n-1} \dots a_1 a_0$ corresponds to lexicographic order of $r_s \dots r_1 r_0$.

Incidentally, there's also a multiset connection: $\{d_t, \dots, d_1\} = \{r_s \cdot s, \dots, r_0 \cdot 0\}$. For example, $\{10, 10, 8, 6, 2, 2, 2, 2, 2, 2, 1, 1, 0\} = \{0 \cdot 11, 2 \cdot 10, 0 \cdot 9, 1 \cdot 8, 0 \cdot 7, 1 \cdot 6, 0 \cdot 5, 0 \cdot 4, 0 \cdot 3, 6 \cdot 2, 2 \cdot 1, 1 \cdot 0\}$.

5. (a) Set $x_j = c_j - \lfloor (j-1)/2 \rfloor$ in each t -combination of $n + \lfloor t/2 \rfloor$. (b) Set $x_j = c_j + j + 1$ in each t -combination of $n - t - 2$.

(A similar approach finds all solutions (x_t, \dots, x_1) to the inequalities $x_{j+1} \geq x_j + \delta_j$ for $0 \leq j \leq t$, given the values of x_{t+1} , $(\delta_t, \dots, \delta_1)$, and x_0 .)

6. Assume that $t > 0$. We get to T3 when $c_1 > 0$; to T5 when $c_2 = c_1 + 1 > 1$; to T4 for $2 \leq j \leq t+1$ when $c_j = c_1 + j - 1 \geq j$. So the counts are: T1, 1; T2, $\binom{n}{t}$; T3, $\binom{n-1}{t}$; T4, $\binom{n-2}{t-1} + \binom{n-2}{t-2} + \dots + \binom{n-t-1}{0} = \binom{n-1}{t-1}$; T5, $\binom{n-2}{t-1}$; T6, $\binom{n-1}{t-1} + \binom{n-2}{t-1} - 1$.

7. A procedure slightly simpler than Algorithm T suffices: Assume that $s < n$.

S1. [Initialize.] Set $b_j \leftarrow j + n - s - 1$ for $1 \leq j \leq s$; then set $j \leftarrow 1$.

S2. [Visit.] Visit the combination $b_s \dots b_2 b_1$. Terminate if $j > s$.

S3. [Decrease b_j .] Set $b_j \leftarrow b_j - 1$. If $b_j < j$, set $j \leftarrow j + 1$ and return to S2.

S4. [Reset $b_{j-1} \dots b_1$.] While $j > 1$, set $b_{j-1} \leftarrow b_j - 1$, $j \leftarrow j - 1$, and repeat until $j = 1$. Go to S2. ▀

(See S. Dvořák, *Comp. J.* 33 (1990), 188. Notice that if $x_k = n - b_k$ for $1 \leq k \leq s$, this algorithm runs through all combinations $x_s \dots x_2 x_1$ of $\{1, 2, \dots, n\}$ with $1 \leq x_s < \dots < x_2 < x_1 \leq n$, in *increasing* lexicographic order.)

8. A1. [Initialize.] Set $a_n \dots a_0 \leftarrow 0^{s+1}1^t$, $q \leftarrow t$, $r \leftarrow 0$. (We assume that $0 < t < n$.)

A2. [Visit.] Visit the combination $a_{n-1} \dots a_1 a_0$. Go to A4 if $q = 0$.

A3. [Replace $\dots 01^q$ by $\dots 101^{q-1}$.] Set $a_q \leftarrow 1$, $a_{q-1} \leftarrow 0$, $q \leftarrow q - 1$; then if $q = 0$, set $r \leftarrow 1$. Return to A2.

A4. [Shift block of 1s.] Set $a_r \leftarrow 0$ and $r \leftarrow r + 1$. Then if $a_r = 1$, set $a_q \leftarrow 1$, $q \leftarrow q + 1$, and repeat step A4.

A5. [Carry to left.] Terminate if $r = n$; otherwise set $a_r \leftarrow 1$.

A6. [Odd?] If $q > 0$, set $r \leftarrow 0$. Return to A2. ▀

In step A2, q and r point respectively to the rightmost 0 and 1 in $a_{n-1} \dots a_0$. Steps A1, ..., A6 are executed with frequency 1, $\binom{n}{t}$, $\binom{n-1}{t-1}$, $\binom{n}{t} - 1$, $\binom{n-1}{t}$, $\binom{n-1}{t} - 1$.

9. (a) The first $\binom{n-1}{t}$ strings begin with 0 and have $2A_{(s-1)t}$ bit changes; the other $\binom{n-1}{t-1}$ begin with 1 and have $2A_{s(t-1)}$. And $\nu(01^t 0^{s-1} \oplus 10^s 1^{t-1}) = 2 \min(s, t)$.

(b) Solution 1 (direct): Let $B_{st} = A_{st} + \min(s, t) + 1$. Then

$$B_{st} = B_{(s-1)t} + B_{s(t-1)} + [s=t] \quad \text{when } st > 0; \quad B_{st} = 1 \quad \text{when } st = 0.$$

Consequently $B_{st} = \sum_{k=0}^{\min(s,t)} \binom{s+t-2k}{s-k}$. If $s \leq t$ this is $\leq \sum_{k=0}^s \binom{s+t-k}{s-k} = \binom{s+t+1}{s} = \binom{s+t}{s} \frac{s+t+1}{t+1} < 2 \binom{s+t}{t}$.

Solution 2 (indirect): The algorithm in answer 8 makes $2(x+y)$ bit changes when steps (A3, A4) are executed (x, y) times. Thus $A_{st} \leq \binom{n-1}{t-1} + \binom{n}{t} - 1 < 2 \binom{n}{t}$.

[The comment in answer 7.2.1.1–3 therefore applies to combinations as well.]

10. Each scenario corresponds to a $(4,4)$ -combination $b_4 b_3 b_2 b_1$ or $c_4 c_3 c_2 c_1$ in which A wins games $\{8-b_4, 8-b_3, 8-b_2, 8-b_1\}$ and N wins games $\{8-c_4, 8-c_3, 8-c_2, 8-c_1\}$, because we can assume that the losing team wins the remaining games in a series of 8. (Equivalently, we can generate all permutations of $\{A, A, A, A, N, N, N, N\}$ and omit the trailing run of As or Ns.) The American League wins if and only if $b_1 \neq 0$, if and only if $c_1 = 0$. The formula $\binom{c_4}{4} + \binom{c_3}{3} + \binom{c_2}{2} + \binom{c_1}{1}$ assigns a unique integer between 0 and 69 to each scenario.

For example, ANANAA $\iff a_7 \dots a_1 a_0 = 01010011 \iff b_4 b_3 b_2 b_1 = 7532 \iff c_4 c_3 c_2 c_1 = 6410$, and this is the scenario of rank $\binom{6}{4} + \binom{4}{3} + \binom{1}{2} + \binom{0}{1} = 19$ in lexicographic order. (Notice that the term $\binom{c_j}{j}$ will be zero if and only if it corresponds to a trailing N.)

11. AAAA (9 times), NNNN (8), and ANAAA (7) were most common. Exactly 27 of the 70 failed to occur, including all four beginning with NNNN. (We disregard the games that were tied because of darkness, in 1907, 1912, and 1922. The case ANNAAA should perhaps be excluded too, because it occurred only in 1920 as part of ANNAAAA in a best-of-nine series. The scenario NNAAANN occurred for the first time in 2001.)

12. (a) Let V_j be the subspace $\{a_{n-1} \dots a_0 \in V \mid a_k = 0 \text{ for } 0 \leq k < j\}$, so that $\{0 \dots 0\} = V_n \subseteq V_{n-1} \subseteq \dots \subseteq V_0 = V$. Then $\{c_1, \dots, c_t\} = \{c \mid V_c \neq V_{c+1}\}$, and α_k is the unique element $a_{n-1} \dots a_0$ of V with $a_{c_j} = [j=k]$ for $1 \leq j \leq t$.

Incidentally, the $t \times n$ matrix corresponding to a canonical basis is said to be in *reduced row-echelon form*. It can be found by a standard “triangulation” algorithm (see exercise 4.6.1–19 and Algorithm 4.6.2N).

(b) The 2-nomial coefficient $\binom{n}{t}_2 = 2^t \binom{n-1}{t}_2 + \binom{n-1}{t-1}_2$ of exercise 1.2.6–58 has the right properties, because $2^t \binom{n-1}{t}_2$ binary vector spaces have $c_t < n-1$ and $\binom{n-1}{t-1}_2$ have $c_t = n-1$. [In general the number of canonical bases with r asterisks is the number of

partitions of r into at most t parts, with no part exceeding $n - t$, and this is $[z^r] \binom{n}{t}_z$ by Eq. 7.2.1.4–(oo). See D. E. Knuth, *J. Combinatorial Theory* **10** (1971), 178–180.]

(c) The following algorithm assumes that $n > t > 0$ and that $a_{(t+1)j} = 0$ for $t \leq j \leq n$.

V1. [Initialize.] Set $a_{kj} \leftarrow [j=k-1]$ for $1 \leq k \leq t$ and $0 \leq j < n$. Also set $q \leftarrow t$, $r \leftarrow 0$.

V2. [Visit.] (At this point we have $a_{k(k-1)} = 1$ for $1 \leq k \leq q$, $a_{(q+1)q} = 0$, and $a_{1r} = 1$.) Visit the canonical basis $(a_{1(n-1)} \dots a_{11}a_{10}, \dots, a_{t(n-1)} \dots a_{t1}a_{t0})$. Go to V4 if $q > 0$.

V3. [Find block of 1s.] Set $q \leftarrow 1, 2, \dots$, until $a_{(q+1)(q+r)} = 0$. Terminate if $q + r = n$.

V4. [Add 1 to column $q+r$.] Set $k \leftarrow 1$. If $a_{k(q+r)} = 1$, set $a_{k(q+r)} \leftarrow 0$, $k \leftarrow k+1$, and repeat until $a_{k(q+r)} = 0$. Then if $k \leq q$, set $a_{k(q+r)} \leftarrow 1$; otherwise set $a_{q(q+r)} \leftarrow 1$, $a_{q(q+r-1)} \leftarrow 0$, $q \leftarrow q-1$.

V5. [Shift block right.] If $q = 0$, set $r \leftarrow r+1$. Otherwise, if $r > 0$, set $a_{k(k-1)} \leftarrow 1$ and $a_{k(r+k-1)} \leftarrow 0$ for $1 \leq k \leq q$, then set $r \leftarrow 0$. Go to V2. ■

Step V2 finds $q > 0$ with probability $1 - (2^{n-t} - 1)/(2^n - 1) \approx 1 - 2^{-t}$, so we could save time by treating this case separately.

(d) Since $999999 = 4\binom{8}{4}_2 + 16\binom{7}{4}_2 + 5\binom{6}{3}_2 + 5\binom{5}{3}_2 + 8\binom{4}{3}_2 + 0\binom{3}{2}_2 + 4\binom{2}{2}_2 + 1\binom{1}{1}_2 + 2\binom{0}{1}_2$, the millionth output has binary columns 4, 16/2, 5, 5, 8/2, 0, 4/2, 1, 2/2, namely

$$\begin{aligned}\alpha_1 &= 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1, \\ \alpha_2 &= 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0, \\ \alpha_3 &= 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0, \\ \alpha_4 &= 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0.\end{aligned}$$

[Reference: E. Calabi and H. S. Wilf, *J. Combinatorial Theory* **A22** (1977), 107–109.]

13. Let $n = s + t$. There are $\binom{s-1}{\lceil(r-1)/2\rceil} \binom{t-1}{\lfloor(r-1)/2\rfloor}$ configurations beginning with 0 and $\binom{s-1}{\lfloor(r-1)/2\rfloor} \binom{t-1}{\lceil(r-1)/2\rceil}$ beginning with 1, because an Ising configuration that begins with 0 corresponds to a composition of s 0s into $\lceil(r+1)/2\rceil$ parts and a composition of t 1s into $\lfloor(r+1)/2\rfloor$ parts. We can generate all such pairs of compositions and weave them into configurations. [See E. Ising, *Zeitschrift für Physik* **31** (1925), 253–258; J. M. S. Simões Pereira, *CACM* **12** (1969), 562.]

14. Start with $l[j] \leftarrow j-1$ and $r[j-1] \leftarrow j$ for $1 \leq j \leq n$; $l[0] \leftarrow n$, $r[n] \leftarrow 0$. To get the next combination, assuming that $t > 0$, set $p \leftarrow s$ if $l[0] > s$, otherwise $p \leftarrow r[n]-1$. Terminate if $p \leq 0$; otherwise set $q \leftarrow r[p]$, $l[q] \leftarrow l[p]$, and $r[l[p]] \leftarrow q$. Then if $r[q] > s$ and $p < s$, set $r[p] \leftarrow r[n]$, $l[r[n]] \leftarrow p$, $r[s] \leftarrow r[q]$, $l[r[q]] \leftarrow s$, $r[n] \leftarrow 0$, $l[0] \leftarrow n$; otherwise set $r[p] \leftarrow r[q]$, $l[r[q]] \leftarrow p$. Finally set $r[q] \leftarrow p$ and $l[p] \leftarrow q$.

[See Korsh and Lipschutz, *J. Algorithms* **25** (1997), 321–335, where the idea is extended to a loopless algorithm for multiset permutations. Caution: This exercise, like exercise 7.2.1.1–16, is more academic than practical, because the routine that visits the linked list might need a loop that nullifies any advantage of loopless generation.]

15. (The stated fact is true because lexicographic order of $c_t \dots c_1$ corresponds to lexicographic order of $a_{n-1} \dots a_0$, which is reverse lexicographic order of the complementary sequence $1 \dots 1 \oplus a_{n-1} \dots a_0$.) By Theorem L, the combination $c_t \dots c_1$ is

visited before exactly $\binom{b_s}{s} + \cdots + \binom{b_2}{2} + \binom{b_1}{1}$ others have been visited, and we must have

$$\binom{b_s}{s} + \cdots + \binom{b_1}{1} + \binom{c_t}{t} + \cdots + \binom{c_1}{1} = \binom{s+t}{t} - 1.$$

This general identity can be written

$$\sum_{j=0}^{n-1} x_j \binom{j}{x_0 + \cdots + x_j} + \sum_{j=0}^{n-1} \bar{x}_j \binom{j}{\bar{x}_0 + \cdots + \bar{x}_j} = \binom{n}{x_0 + \cdots + x_{n-1}} - 1$$

when each x_j is 0 or 1, and $\bar{x}_j = 1 - x_j$; it follows also from the equation

$$x_n \binom{n}{x_0 + \cdots + x_n} + \bar{x}_n \binom{n}{\bar{x}_0 + \cdots + \bar{x}_n} = \binom{n+1}{x_0 + \cdots + x_n} - \binom{n}{x_0 + \cdots + x_{n-1}}.$$

- 16.** Since $999999 = \binom{1414}{2} + \binom{1008}{1} = \binom{182}{3} + \binom{153}{2} + \binom{111}{1} = \binom{71}{4} + \binom{56}{3} + \binom{36}{2} + \binom{14}{1} = \binom{43}{5} + \binom{32}{4} + \binom{21}{3} + \binom{15}{2} + \binom{6}{1}$, the answers are (a) 1414 1008; (b) 182 153 111; (c) 71 56 36 14; (d) 43 32 21 15 6; (e) 1000000 999999 ... 2 0.

- 17.** By Theorem L, n_t is the largest integer such that $N \geq \binom{n_t}{t}$; the remaining terms are the degree- $(t-1)$ representation of $N - \binom{n_t}{t}$.

A simple sequential method for $t > 1$ starts with $x = 1$, $c = t$, and sets $c \leftarrow c + 1$, $x \leftarrow xc/(c-t)$ zero or more times until $x > N$; then we complete the first phase by setting $x \leftarrow x(c-t)/c$, $c \leftarrow c - 1$, at which point we have $x = \binom{c}{t} \leq N < \binom{c+1}{t}$. Set $n_t \leftarrow c$, $N \leftarrow N - x$; terminate with $n_1 \leftarrow N$ if $t = 2$; otherwise set $x \leftarrow xt/c$, $t \leftarrow t-1$, $c \leftarrow c - 1$; while $x > N$ set $x \leftarrow x(c-t)/c$, $c \leftarrow c - 1$; repeat. This method requires $O(n)$ arithmetic operations if $N < \binom{n}{t}$, so it is suitable unless t is small and N is large.

When $t = 2$, exercise 1.2.4–41 tells us that $n_2 = \lfloor \sqrt{2N+2} + \frac{1}{2} \rfloor$. In general, n_t is $\lfloor x \rfloor$ where x is the largest root of $x^t = t!N$; this root can be approximated by reverting the series $y = (x^t)^{1/t} = x - \frac{1}{2}(t-1) + \frac{1}{24}(t^2-1)x^{-1} + \cdots$ to get $x = y + \frac{1}{2}(t-1) + \frac{1}{24}(t^2-1)/y + O(y^{-3})$. Setting $y = (t!N)^{1/t}$ in this formula gives a good approximation, after which we can check that $\binom{\lfloor x \rfloor}{t} \leq N < \binom{\lfloor x \rfloor + 1}{t}$ or make a final adjustment. [See A. S. Fraenkel and M. Mor, *Comp. J.* **26** (1983), 336–343.]

- 18.** A complete binary tree of $2^n - 1$ nodes is obtained, with an extra node at the top, like the “tree of losers” in replacement selection sorting (Fig. 63 in Section 5.4.1). Therefore explicit links aren’t necessary; the right child of node k is node $2k + 1$, and the left sibling is node $2k$, for $1 \leq k < 2^{n-1}$.

This representation of a binomial tree has the curious property that node $k = (0^a 1 \alpha)_2$ corresponds to the combination whose binary string is $0^a 1 \alpha^R$.

- 19.** It is $\text{post}(1000000)$, where $\text{post}(n) = 2^k + \text{post}(n - 2^k + 1)$ if $2^k \leq n < 2^{k+1}$, and $\text{post}(0) = 0$. So it is 11110100001001000100.

- 20.** $f(z) = (1 + z^{w_{n-1}}) \cdots (1 + z^{w_1}) / (1 - z)$, $g(z) = (1 + z^{w_0})f(z)$, $h(z) = z^{w_0}f(z)$.

- 21.** The rank of $c_t \dots c_2 c_1$ is $\binom{ct+1}{t} - 1$ minus the rank of $c_{t-1} \dots c_2 c_1$. [See H. Lüneburg, *Abh. Math. Sem. Hamburg* **52** (1982), 208–227.]

- 22.** Since $999999 = \binom{1415}{2} - \binom{406}{1} = \binom{183}{3} - \binom{98}{2} + \binom{21}{1} = \binom{72}{4} - \binom{57}{3} + \binom{32}{2} - \binom{27}{1} = \binom{44}{5} - \binom{40}{4} + \binom{33}{3} - \binom{13}{2} + \binom{3}{1}$, the answers are (a) 1414 405; (b) 182 97 21; (c) 71 56 31 26; (d) 43 39 32 12 3; (e) 1000000 999999 999998 999996 ... 0.

- 23.** There are $\binom{n-r}{t-r}$ combinations with $j > r$, for $r = 1, 2, \dots, t$. (If $r = 1$ we have $c_2 = c_1 + 1$; if $r = 2$ we have $c_1 = 0, c_2 = 1$; if $r = 3$ we have $c_1 = 0, c_2 = 1, c_4 = c_3 + 1$; etc.) Thus the mean is $((\binom{n}{t} + \binom{n-1}{t-1} + \cdots + \binom{n-t}{0}) / \binom{n}{t}) = (\binom{n+1}{t} / \binom{n}{t}) = (n+1)/(n+1-t)$.

The average running time per step is approximately proportional to this quantity; thus the algorithm is quite fast when t is small, but slow if t is near n .

- 24.** In fact $j_k - 2 \leq j_{k+1} \leq j_k + 1$ when $j_k \equiv t$ (modulo 2) and $j_k - 1 \leq j_{k+1} \leq j_k + 2$ when $j_k \not\equiv t$, because R5 is performed only when $c_i = i - 1$ for $1 \leq i < j$.

Thus we could say, “If $j \geq 4$, set $j \leftarrow j - 1 - [j \text{ odd}]$ and go to R5” at the end of R2, if t is odd; “If $j \geq 3$, set $j \leftarrow j - 1 - [j \text{ even}]$ and go to R5” if t is even. The algorithm will then be loopless, since R4 and R5 will be performed at most twice per visit.

- 25.** Assume that $N > N'$ and $N - N'$ is minimum; furthermore let c_t be minimum, subject to those assumptions. Then $c_t > c'_t$.

If there is an element $x \notin C \cup C'$ with $0 \leq x < c_t$, map each t -combination of $C \cup C'$ by changing $j \mapsto j - 1$ for $j > x$; or, if there is an element $x \in C \cap C'$, map each t -combination that contains x into a $(t - 1)$ -combination by omitting x and changing $j \mapsto x - j$ for $j < x$. In either case the mapping preserves alternating lexicographic order; hence $N - N'$ must exceed the number of combinations between the images of C and C' . But c_t is minimum, so no such x can exist. Consequently $t = m$ and $c_t = 2m - 1$.

Now if $c'_m < c_m - 1$, we could decrease $N - N'$ by increasing c'_m . Therefore $c'_m = 2m - 2$, and the problem has been reduced to finding the *maximum* of $\text{rank}(c_{m-1} \dots c_1) - \text{rank}(c'_{m-1} \dots c'_1)$, where rank refers to Gray binary code.

Let $f(s, t) = \max(\text{rank}(b_s \dots b_1) - \text{rank}(c_t \dots c_1))$ over all $\{b_s, \dots, b_1, c_t, \dots, c_1\} = \{0, \dots, s+t-1\}$. Then $f(s, t)$ satisfies the curious recurrence

$$\begin{aligned} f(s, 0) &= f(0, t) = 0; & f(1, t) &= t; \\ f(s, t) &= \binom{s+t-1}{s} + \max(f(t-1, s-1), f(s-2, t)) & \text{if } st > 0. \end{aligned}$$

When $s + t = 2u + 2$ the solution turns out to be

$$f(s, t) = \binom{2u+1}{t-1} + \sum_{j=1}^{u-r} \binom{2u+1-2j}{r} + \sum_{j=0}^{r-1} \binom{2j+1}{j}, \quad r = \min(s-2, t-1),$$

with the maximum occurring at $f(t-1, s-1)$ when $s \leq t$ and at $f(s-2, t)$ when $s \geq t+2$.

Therefore the minimum $N - N'$ occurs for

$$C = \{2m - 1\} \cup \{2m - 2 - x \mid 1 \leq x \leq 2m - 2, x \bmod 4 \leq 1\},$$

$$C' = \{2m - 2\} \cup \{2m - 2 - x \mid 1 \leq x \leq 2m - 2, x \bmod 4 \geq 2\};$$

and it equals $\binom{2m-1}{m-1} - \sum_{k=0}^{m-2} \binom{2k+1}{k} = 1 + \sum_{k=1}^{m-1} \binom{2k}{k-1}$. [A. J. van Zanten, *IEEE Trans. IT-37* (1991), 1229–1233.]

- 26.** (a) Yes: The first is $0^{n-\lceil t/2 \rceil} 1^{t \bmod 2} 2^{\lfloor t/2 \rfloor}$ and the last is $2^{\lfloor t/2 \rfloor} 1^{t \bmod 2} 0^{n-\lceil t/2 \rceil}$; transitions are substrings of the forms $02^a 1 \leftrightarrow 12^a 0$, $02^a 2 \leftrightarrow 12^a 1$, $10^a 1 \leftrightarrow 20^a 0$, $10^a 2 \leftrightarrow 20^a 1$.

(b) No: If $s = 0$ there is a big jump from $02^t 0^{r-1}$ to $20^r 2^{t-1}$.

- 27.** The following procedure extracts all combinations $c_1 \dots c_n$ of Γ_n that have weight $\leq t$: Begin with $k \leftarrow 0$ and $c_0 \leftarrow n$. Visit $c_1 \dots c_k$. If k is even and $c_k = 0$, set $k \leftarrow k - 1$; if k is even and $c_k > 0$, set $c_k \leftarrow c_k - 1$ if $k = t$, otherwise $k \leftarrow k + 1$ and $c_k \leftarrow 0$. On the other hand if k is odd and $c_k + 1 = c_{k-1}$, set $k \leftarrow k - 1$ and $c_k \leftarrow c_{k+1}$ (but terminate if $k = 0$); if k is odd and $c_k + 1 < c_{k-1}$, set $c_k \leftarrow c_k + 1$ if $k = t$, otherwise $k \leftarrow k + 1$, $c_k \leftarrow c_{k-1}$, $c_{k-1} \leftarrow c_k + 1$. Repeat.

(This loopless algorithm reduces to that of exercise 7.2.1.12(b) when $t = n$, with slight changes of notation.)

28. True. Bit strings $a_{n-1} \dots a_0 = \alpha\beta$ and $a'_{n-1} \dots a'_0 = \alpha\beta'$ correspond to index lists $(b_s \dots b_1 = \theta\chi, c_t \dots c_1 = \phi\psi)$ and $(b'_s \dots b'_1 = \theta\chi', c'_t \dots c'_1 = \phi\psi')$ such that everything between $\alpha\beta$ and $\alpha\beta'$ begins with α if and only if everything between $\theta\chi$ and $\theta\chi'$ begins with θ and everything between $\phi\psi$ and $\phi\psi'$ begins with ϕ . For example, if $n = 10$, the prefix $\alpha = 01101$ corresponds to prefixes $\theta = 96$ and $\phi = 875$.

(But just having $c_t \dots c_1$ in genlex order is a much weaker condition. For example, *every* such sequence is genlex when $t = 1$.)

29. (a) $-k0^{l+1}$ or $-k0^{l+1}\pm m$ or $\pm k$, for $k, l, m \geq 0$.

(b) No; the successor is always smaller in balanced ternary notation.

(c) For all α and all $k, l, m \geq 0$ we have $\alpha 0^{-k+1}0^l\pm m \rightarrow \alpha\pm k0^{l+1}\pm m$ and $\alpha\pm k0^{l+1}\pm m \rightarrow \alpha 0^{l+1}0^l\pm m$; also $\alpha 0^{-k+1}0^l \rightarrow \alpha\pm k0^{l+1}$ and $\alpha\pm k0^{l+1} \rightarrow \alpha 0^{l+1}0^l$.

(d) Let the j th sign of α_i be $(-1)^{a_{ij}}$, and let it be in position b_{ij} . Then we have $(-1)^{a_{ij}+b_{i(j-1)}} = (-1)^{a_{(i+1)j}+b_{(i+1)(j-1)}}$ for $0 \leq i < k$ and $1 \leq j \leq t$, if we let $b_{i0} = 0$.

(e) By parts (a), (b), and (c), α belongs to some chain $\alpha_0 \rightarrow \dots \rightarrow \alpha_k$, where α_k is final (has no successor) and α_0 is initial (has no predecessor). By part (d), every such chain has at most $\binom{s+t}{t}$ elements. But there are 2^s final strings, by (a), and there are $2^s \binom{s+t}{t}$ strings with s signs and t zeros; so k must be $\binom{s+t}{t} - 1$.

Reference: SICOMP 2 (1973), 128–133.

30. Assume that $t > 0$. Initial strings are the negatives of final strings. Let σ_j be the initial string $0^t - \tau_j$ for $0 \leq j < 2^{s-1}$, where the k th character of τ_j for $1 \leq k < s$ is $(-1)^{a_{jk}}$ when j is the binary number $(a_{s-1} \dots a_1)_2$; thus $\sigma_0 = 0^t - ++ \dots +$, $\sigma_1 = 0^t --- \dots +$, \dots , $\sigma_{2^{s-1}-1} = 0^t ---- \dots -$. Let ρ_j be the final string obtained by inserting -0^t after the first run of minus signs in τ_j ; thus $\rho_0 = -0^t ++ \dots +$, $\rho_1 = --0^t + \dots +$, \dots , $\rho_{2^{s-1}-1} = -- \dots -0^t$. We also let $\sigma_{2^{s-1}} = \sigma_0$ and $\rho_{2^{s-1}} = \rho_0$. Then we can prove by induction that the chain beginning with σ_j ends with ρ_j when t is even, with ρ_{j-1} when t is odd, for $1 \leq j \leq 2^{s-1}$. Therefore the chain beginning with $-\rho_j$ ends with $-\sigma_j$ or $-\sigma_{j+1}$.

Let $A_j(s, t)$ be the sequence of (s, t) -combinations derived by mapping the chain that starts with σ_j , and let $B_j(s, t)$ be the analogous sequence derived from $-\rho_j$. Then, for $1 \leq j \leq 2^{s-1}$, the reverse sequence $A_j(s, t)^R$ is $B_j(s, t)$ when t is even, $B_{j-1}(s, t)$ when t is odd. The corresponding recurrences when $st > 0$ are

$$A_j(s, t) = \begin{cases} 1A_j(s, t-1), 0A_{\lfloor (2^{s-1}-1-j)/2 \rfloor}(s-1, t)^R, & \text{if } t \text{ is even;} \\ 1A_j(s, t-1), 0A_{\lfloor j/2 \rfloor}(s-1, t), & \text{if } t \text{ is even;} \end{cases}$$

and when $st > 0$ all 2^{s-1} of these sequences are distinct.

Chase's sequence C_{st} is $A_{\lfloor 2^s/3 \rfloor}(s, t)$, and \widehat{C}_{st} is $A_{\lfloor 2^{s-1}/3 \rfloor}(s, t)$. Incidentally, the homogeneous sequence K_{st} of (31) is $A_{2^{s-1}-[t \text{ even}]}(s, t)^R$.

31. (a) $2^{\binom{s+t}{t}-1}$ solves the recurrence $f(s, t) = 2f(s-1, t)f(s, t-1)$ when $f(s, 0) = f(0, t) = 1$. (b) Now $f(s, t) = (s+1)!f(s, t-1) \dots f(0, t-1)$ has the solution

$$(s+1)!^t s!^{\binom{t}{2}} (s-1)!^{\binom{t+1}{3}} \dots 2!^{\binom{s+t-2}{s}} = \prod_{r=1}^s (r+1)!^{\binom{s+t-1-r}{t-2} + [r=s]}.$$

32. (a) No simple formula seems to exist, but the listings can be counted for small s and t by systematically computing the number of genlex paths that run through all weight- t strings from a given starting point to a given ending point via revolving-door

moves. The totals for $s + t \leq 6$ are

$$\begin{array}{ccccccc}
 & & & & 1 & & \\
 & & & & 1 & 1 & \\
 & & & & 1 & 2 & 1 \\
 & & & & 1 & 4 & 4 \\
 & & & & 1 & 8 & 20 \\
 & & & & 1 & 16 & 160 \\
 & & & & 1 & 32 & 2264 \\
 & & & & 1 & 160 & 17152 \\
 & & & & 1 & 2264 & 32 \\
 & & & & 1 & 32 & 1
 \end{array}$$

and $f(4,4) = 95,304,112,865,280$; $f(5,5) \approx 5.92646 \times 10^{48}$. [This class of combination generators was first studied by G. Ehrlich, JACM 20 (1973), 500–513, but he did not attempt to enumerate them.]

(b) By extending the proof of Theorem N, one can show that all such listings or their reversals must run from $1^t 0^s$ to $0^a 1^t 0^{s-a}$ for some a , $1 \leq a \leq s$. Moreover, the number n_{sta} of possibilities, given s , t , and a , satisfies

$$n_{sta} = \begin{cases} n_{s(t-1)} n_{(s-1)t(a-1)}, & \text{if } a > 1, \\ n_{s(t-1)2} n_{(s-1)t1} + \cdots + n_{s(t-1)s} n_{(s-1)t(s-1)}, & \text{if } a = 1 < s, \end{cases}$$

when $st > 0$. This recurrence has the remarkable solution $n_{sta} = 2^{m(s,t,a)}$, where

$$m(s,t,a) = \begin{cases} \binom{s+t-3}{t} + \binom{s+t-5}{t-2} + \cdots + \binom{s-1}{2}, & \text{if } t \text{ is even;} \\ \binom{s+t-3}{t} + \binom{s+t-5}{t-2} + \cdots + \binom{s}{3} + s - a - [a < s], & \text{if } t \text{ is odd.} \end{cases}$$

33. Consider first the case $t = 1$: The number of near-perfect paths from i to $j > i$ is $f(j-i-[i>0]-[j<n-1])$, where $\sum_j f(j)z^j = 1/(1-z-z^3)$. (By coincidence, the same sequence $f(j)$ arises in Caron's polyphase merge on 6 tapes, Table 5.4.2–2.) The sum over $0 \leq i < j < n$ is $3f(n) + f(n-1) + f(n-2) + 2 - n$; and we must double this, to cover cases with $j > i$.

When $t > 1$ we can construct $\binom{n}{t} \times \binom{n}{t}$ matrices that tell how many genlex listings begin and end with particular combinations. The entries of these matrices are sums of products of matrices for the case $t = 1$, summed over all paths of the type considered for $t = 1$. The totals for $s + t \leq 6$ turn out to be

$$\begin{array}{ccccccc}
 & & & & 1 & & \\
 & & & & 1 & 1 & \\
 & & & & 1 & 2 & 1 \\
 & & & & 1 & 6 & 2 \\
 & & & & 1 & 12 & 10 \\
 & & & & 1 & 20 & 44 \\
 & & & & 1 & 34 & 238
 \end{array}
 \quad
 \begin{array}{ccccccc}
 & & & & 1 & & \\
 & & & & 1 & 1 & \\
 & & & & 1 & 2 & 1 \\
 & & & & 1 & 2 & 0 \\
 & & & & 1 & 2 & 2 \\
 & & & & 1 & 2 & 0 \\
 & & & & 1 & 2 & 6
 \end{array}$$

where the right-hand triangle shows the number of cycles, $g(s,t)$. Further values include $f(4,4) = 17736$; $f(5,5) = 9,900,888,879,984$; $g(4,4) = 96$; $g(5,5) = 30,961,456,320$.

There are exactly 10 such schemes when $s = 2$ and $n \geq 4$. For example, when $n = 7$ they run from 43210 to 65431 or 65432, or from 54321 to 65420 or 65430 or 65432, or the reverse.

34. The minimum can be computed as in the previous answer, but using min-plus matrix multiplication $c_{ij} = \min_k (a_{ik} + b_{kj})$ instead of ordinary matrix multiplication $c_{ij} = \sum_k a_{ik}b_{kj}$. (When $s = t = 5$, the genlex path in Fig. 26(e) with only 49 imperfect transitions is essentially unique. There is a genlex cycle for $s = t = 5$ that has only 55 imperfections.)

35. From the recurrences (35) we have $a_{st} = b_{s(t-1)} + [s > 1][t > 0] + a_{(s-1)t}$, $b_{st} = a_{s(t-1)} + a_{(s-1)t}$; consequently $a_{st} = b_{st} + [s > 1][t \text{ odd}]$ and $a_{st} = a_{s(t-1)} + a_{(s-1)t} + [s > 1][t \text{ odd}]$. The solution is

$$a_{st} = \sum_{k=0}^{t/2} \binom{s+t-2-2k}{s-2} - [s > 1][t \text{ even}];$$

this sum is approximately $s/(s+2t)$ times $\binom{s+t}{t}$.

36. Consider the binary tree with root node (s, t) and with recursively defined subtrees rooted at $(s-1, t)$ and $(s, t-1)$ whenever $st > 0$; the node (s, t) is a leaf if $st = 0$. Then the subtree rooted at (s, t) has $\binom{s+t}{t}$ leaves, corresponding to all (s, t) -combinations $a_{n-1} \dots a_1 a_0$. Nodes on level l correspond to prefixes $a_{n-1} \dots a_{n-l}$, and leaves on level l are combinations with $r = n - l$.

Any genlex algorithm for combinations $a_{n-1} \dots a_1 a_0$ corresponds to preorder traversal of such a tree, after the children of the $\binom{s+t}{t} - 1$ branch nodes have been ordered in any desired way; that, in fact, is why there are $2^{\binom{s+t}{t}-1}$ such genlex schemes (exercise 31(a)). And the operation $j \leftarrow j + 1$ is performed exactly once per branch node, namely after both children have been processed.

Incidentally, exercise 7.2.1.2–6(a) implies that the average value of r is $s/(t+1) + t/(s+1)$, which can be $\Omega(n)$; thus the extra time needed to keep track of r is worthwhile.

37. (a) In the lexicographic case we needn't maintain the w_j table, since a_j is active for $j \geq r$ if and only if $a_j = 0$. After setting $a_j \leftarrow 1$ and $a_{j-1} \leftarrow 0$ there are two cases to consider if $j > 1$: If $r = j$, set $r \leftarrow j - 1$; otherwise set $a_{j-2} \dots a_0 \leftarrow 0^r 1^{j-1-r}$ and $r \leftarrow j - 1 - r$ (or $r \leftarrow j$ if r was $j - 1$).

(b) Now the transitions to be handled when $j > 1$ are to change $a_j \dots a_0$ as follows: $01^r \rightarrow 1101^{r-2}$, $010^r \rightarrow 10^{r+1}$, $010^a 1^r \rightarrow 110^{a+1} 1^{r-1}$, $10^r \rightarrow 010^{r-1}$, $110^r \rightarrow 010^{r-1} 1$, $10^a 1^r \rightarrow 0^a 1^{r+1}$; these six cases are easily distinguished. The value of r should change appropriately.

(c) Again the case $j = 1$ is trivial. Otherwise $01^a 0^r \rightarrow 101^{a-1} 0^r$; $0^a 1^r \rightarrow 10^a 1^{r-1}$; $101^a 0^r \rightarrow 01^{a+1} 0^r$; $10^a 1^r \rightarrow 0^a 1^{r+1}$; and there is also an ambiguous case, which can occur only if $a_{n-1} \dots a_{j+1}$ contains at least one 0: Let $k > j$ be minimal with $a_k = 0$. Then $10^r \rightarrow 010^{r-1}$ if k is odd, $10^r \rightarrow 0^r 1$ if k is even.

38. The same algorithm works, except that (i) step C1 sets $a_{n-1} \dots a_0 \leftarrow 01^t 0^{s-1}$ if n is odd or $s = 1$, $a_{n-1} \dots a_0 \leftarrow 001^t 0^{s-2}$ if n is even and $s > 1$, with an appropriate value of r ; (ii) step C3 interchanges the roles of even and odd; (iii) step C5 goes to C4 also if $j = 1$.

39. In general, start with $r \leftarrow 0$, $j \leftarrow s + t - 1$, and repeat the following steps until $st = 0$:

$$r \leftarrow r + [w_j = 0] \binom{s+t-a_j}{s-a_j}, \quad s \leftarrow s - [a_j = 0], \quad t \leftarrow t - [a_j = 1], \quad j \leftarrow j - 1.$$

Then r is the rank of $a_{n-1} \dots a_1 a_0$. So the rank of $1100100100001111101101010$ is $\binom{25}{11} + \binom{24}{11} + \binom{20}{10} + \binom{19}{9} + \binom{18}{7} + \binom{15}{6} + \binom{7}{4} + \binom{6}{2} + \binom{5}{2} + \binom{2}{2} = 2390131$.

40. We start with $N \leftarrow 999999$, $v \leftarrow 0$, and repeat the following steps until $st = 0$: If $v = 0$, set $t \leftarrow t - 1$ and $a_{s+t} \leftarrow 1$ if $N < \binom{s+t-1}{s}$, otherwise set $N \leftarrow N - \binom{s+t-1}{s}$, $v \leftarrow (s+t) \bmod 2$, $s \leftarrow s - 1$, $a_{s+t} \leftarrow 0$. If $v = 1$, set $v \leftarrow (s+t) \bmod 2$, $s \leftarrow s - 1$, and $a_{s+t} \leftarrow 0$ if $N < \binom{s+t-1}{t}$, otherwise set $N \leftarrow N - \binom{s+t-1}{t}$, $t \leftarrow t - 1$, $a_{s+t} \leftarrow 1$.

Finally if $s = 0$, set $a_{t-1} \dots a_0 \leftarrow 1^t$; if $t = 0$, set $a_{s-1} \dots a_0 \leftarrow 0^s$. The answer is $a_{25} \dots a_0 = 1110100111110101001000001$.

41. Let $c(0), \dots, c(2^n - 1) = C_n$ where $C_{2n} = 0C_{2n-1}, 1C_{2n-1}; C_{2n+1} = 0C_{2n}, 1\widehat{C}_{2n}; \widehat{C}_{2n} = 1C_{2n-1}, 0\widehat{C}_{2n-1}; \widehat{C}_{2n+1} = 1\widehat{C}_{2n}, 0\widehat{C}_{2n}; C_0 = \widehat{C}_0 = \epsilon$. Then $a_j \oplus b_j = b_{j+1} \wedge (b_{j+2} \vee (b_{j+3} \wedge (b_{j+4} \vee \dots)))$ if j is even, $b_{j+1} \vee (b_{j+2} \wedge (b_{j+3} \vee (b_{j+4} \wedge \dots)))$ if j is odd. Curiously we also have the inverse relation $c((\dots \bar{a}_5 a_4 \bar{a}_3 a_2 \bar{a}_1 a_0)_2) = (\dots \bar{b}_5 b_4 \bar{b}_3 b_2 \bar{b}_1 b_0)$.

42. Equation (40) shows that the left context $a_{n-1} \dots a_{l+1}$ does not affect the behavior of the algorithm on $a_{l-1} \dots a_0$ if $a_l = 0$ and $l > r$. Therefore we can analyze Algorithm C by counting combinations that end with certain bit patterns, and it follows that the number of times each operation is performed can be represented as $[w^s z^t] p(w, z) / (1 - w^2)^2 (1 - z^2)^2 (1 - w - z)$ for an appropriate polynomial $p(w, z)$.

For example, the algorithm goes from C5 to C4 once for each combination that ends with $01^{2a+1}01^{2b+1}$ or has the form $1^{a+1}01^{2b+1}$, for integers $a, b \geq 0$; the corresponding generating functions are $w^2 z^2 / (1 - z^2)^2 (1 - w - z)$ and $w(z^2 + z^3) / (1 - z^2)^2$.

Here are the polynomials $p(w, z)$ for key operations. Let $W = 1 - w^2$, $Z = 1 - z^2$.

$$\begin{array}{llll}
 \text{C3} \rightarrow \text{C4}: & wzW(1+wz)(1-w-z^2): & \text{C5}(r \leftarrow 1): & w^2zW^2Z(1-wz-z^2): \\
 \text{C3} \rightarrow \text{C5}: & wzW(w+z)(1-wz-z^2): & \text{C5}(r \leftarrow j-1): & w^2z^3W^2(1-wz-z^2): \\
 \text{C3} \rightarrow \text{C6}: & w^2z^2W(w+z): & \text{C6}(j=1): & w^2zW^2Z: \\
 \text{C3} \rightarrow \text{C7}: & w^2zW(1+wz): & \text{C6}(r \leftarrow j-1): & w^2z^3W^2: \\
 \text{C4}(j=1): & wzW^2Z(1-w-z^2): & \text{C6}(r \leftarrow j): & w^3z^2WZ: \\
 \text{C4}(r \leftarrow j-1): & w^3zWZ(1-w-z^2): & \text{C7} \rightarrow \text{C6}: & w^2zW^2: \\
 \text{C4}(r \leftarrow j): & wz^2W^2(1+z-2wz-z^2-z^3): & \text{C7}(r \leftarrow j): & w^4zWZ: \\
 \text{C5} \rightarrow \text{C4}: & wz^2W^2(1-wz-z^2): & \text{C7}(r \leftarrow j-2): & w^3z^2W^2: \\
 \text{C5}(r \leftarrow j-2): & w^4zWZ(1-wz-z^2): & &
 \end{array}$$

The asymptotic value is $\binom{s+t}{t} (p(1-x, x) / (2x - x^2)^2 (1 - x^2)^2 + O(n^{-1}))$, for fixed $0 < x < 1$, if $t = xn + O(1)$ as $n \rightarrow \infty$. Thus we find, for example, that the four-way branching in step C3 takes place with relative frequencies $x + x^2 - x^3 : 1 : x : 1 + x - x^2$.

Incidentally, the number of cases with j odd exceeds the number of cases with j even by

$$\sum_{k,l \geq 1} \binom{s+t-2k-2l}{s-2k} [2k+2l \leq s+t] + [s \text{ odd}] [t \text{ odd}],$$

in *any* genlex scheme that uses (39). This quantity has the interesting generating function $wz / (1+w)(1+z)(1-w-z)$.

43. The identity is true for all nonnegative integers x , except when $x = 1$.

44. In fact, $C_t(n) - 1 = \widehat{C}_t(n-1)^R$, and $\widehat{C}_t(n) - 1 = C_t(n-1)^R$. (Hence $C_t(n) - 2 = C_t(n-2)$, etc.)

45. In the following algorithm, r is the least subscript with $c_r \geq r$.

CC1. [Initialize.] Set $c_j \leftarrow n - t - 1 + j$ and $z_j \leftarrow 0$ for $1 \leq j \leq t+1$. Also set $r \leftarrow 1$. (We assume that $0 < t < n$.)

CC2. [Visit.] Visit the combination $c_t \dots c_2 c_1$. Then set $j \leftarrow r$.

CC3. [Branch.] Go to CC5 if $z_j \neq 0$.

CC4. [Try to decrease c_j .] Set $x \leftarrow c_j + (c_j \bmod 2) - 2$. If $x \geq j$, set $c_j \leftarrow x$, $r \leftarrow 1$; otherwise if $c_j = j$, set $c_j \leftarrow j-1$, $z_j \leftarrow c_{j+1} - ((c_{j+1} + 1) \bmod 2)$, $r \leftarrow j$; otherwise if $c_j < j$, set $c_j \leftarrow j$, $z_j \leftarrow c_{j+1} - ((c_{j+1} + 1) \bmod 2)$, $r \leftarrow \max(1, j-1)$; otherwise set $c_j \leftarrow x$, $r \leftarrow j$. Return to CC2.

CC5. [Try to increase c_j .] Set $x \leftarrow c_j + 2$. If $x < z_j$, set $c_j \leftarrow x$; otherwise if $x = z_j$ and $z_j \neq 0$, set $c_j \leftarrow x - (c_{j+1} \bmod 2)$; otherwise set $z_j \leftarrow 0$, $j \leftarrow j + 1$, and go to CC3 (but terminate if $j > t$). If $c_1 > 0$, set $r \leftarrow 1$; otherwise set $r \leftarrow j - 1$. Return to CC2. ■

46. Eq. (40) implies that $u_k = (b_j + k + 1) \bmod 2$ when j is minimal with $b_j > k$. Then (37) and (38) yield the following algorithm, where we assume for convenience that $3 \leq s < n$.

CB1. [Initialize.] Set $b_j \leftarrow j - 1$ for $1 \leq j \leq s$; also set $z \leftarrow s + 1$, $b_z \leftarrow 1$. (When subsequent steps examine the value of z , it is the smallest index such that $b_z \neq z - 1$.)

CB2. [Visit.] Visit the dual combination $b_s \dots b_2 b_1$.

CB3. [Branch.] If b_2 is odd: Go to CB4 if $b_2 \neq b_1 + 1$, otherwise to CB5 if $b_1 > 0$, otherwise to CB6 if b_z is odd. Go to CB9 if b_2 is even and $b_1 > 0$. Otherwise go to CB8 if $b_{z+1} = b_z + 1$, otherwise to CB7.

CB4. [Increase b_1 .] Set $b_1 \leftarrow b_1 + 1$ and return to CB2.

CB5. [Slide b_1 and b_2 .] If b_3 is odd, set $b_1 \leftarrow b_1 + 1$ and $b_2 \leftarrow b_2 + 1$; otherwise set $b_1 \leftarrow b_1 - 1$, $b_2 \leftarrow b_2 - 1$, $z \leftarrow 3$. Go to CB2.

CB6. [Slide left.] If z is odd, set $z \leftarrow z - 2$, $b_{z+1} \leftarrow z + 1$, $b_z \leftarrow z$; otherwise set $z \leftarrow z - 1$, $b_z \leftarrow z$. Go to CB2.

CB7. [Slide b_z .] If b_{z+1} is odd, set $b_z \leftarrow b_z + 1$ and terminate if $b_z \geq n$; otherwise set $b_z \leftarrow b_z - 1$, then if $b_z < z$ set $z \leftarrow z + 1$. To CB2.

CB8. [Slide b_z and b_{z+1} .] If b_{z+2} is odd, set $b_z \leftarrow b_{z+1}$, $b_{z+1} \leftarrow b_z + 1$, and terminate if $b_{z+1} \geq n$. Otherwise set $b_{z+1} \leftarrow b_z$, $b_z \leftarrow b_z - 1$, then if $b_z < z$ set $z \leftarrow z + 2$. To CB2.

CB9. [Decrease b_1 .] Set $b_1 \leftarrow b_1 - 1$, $z \leftarrow 2$, and return to CB2. ■

Notice that this algorithm is *loopless*. Chase gave a similar procedure for the sequence \widehat{C}_{st}^R in *Cong. Num.* **69** (1989), 233–237. It is truly amazing that this algorithm defines precisely the complements of the indices $c_t \dots c_1$ produced by the algorithm in the previous exercise.

47. We can, for example, use Algorithm C and its reverse (exercise 38), with w_j replaced by a d -bit number whose bits represent activity at different levels of the recursion. Separate pointers r_0, r_1, \dots, r_{d-1} are needed to keep track of the r -values on each level. (Many other solutions are possible.)

48. There are permutations π_1, \dots, π_M such that the k th element of Λ_j is $\pi_k \alpha_j \uparrow \beta_{k-1}$. And $\pi_k \alpha_j$ runs through all permutations of $\{s_1 \cdot 1, \dots, s_d \cdot d\}$ as j varies from 0 to $N-1$.

Historical note: The first publication of a homogeneous revolving-door scheme for (s, t) -combinations was by Éva Török, *Matematikai Lapok* **19** (1968), 143–146, who was motivated by the generation of multiset permutations. Many authors have subsequently relied on the homogeneity condition for similar constructions, but this exercise shows that homogeneity is not necessary.

49. We have $\lim_{z \rightarrow q} (z^{km+r} - 1)/(z^{lm+r} - 1) = 1$ when $0 < r < m$, and the limit is $\lim_{z \rightarrow q} (kmz^{km-1})/(lmz^{lm-1}) = k/l$ when $r = 0$. So we can pair up factors of the numerator $\prod_{n-k < a \leq n} (z^a - 1)$ with factors of the denominator $\prod_{0 < b \leq k} (z^b - 1)$ when $a \equiv b$ (modulo m).

Notes: In the special case $m = 2$, $q = -1$, the second factor vanishes only when n is even and k is odd. The formula $\binom{n}{k}_q = \binom{n}{n-k}_q$ holds for all $n \geq 0$, but $\binom{\lfloor n/m \rfloor}{\lfloor k/m \rfloor}$ is not

always equal to $\binom{\lfloor n/m \rfloor}{\lfloor (n-k)/m \rfloor}$. We do, however, have $\lfloor k/m \rfloor + \lfloor (n-k)/m \rfloor = \lfloor n/m \rfloor$ in the case when $n \bmod m \geq k \bmod m$; otherwise the second factor is zero.

- 50.** The stated coefficient is zero when $n_1 \bmod m + \dots + n_t \bmod m \geq m$. Otherwise it equals

$$\binom{\lfloor (n_1 + \dots + n_t)/m \rfloor}{\lfloor n_1/m \rfloor, \dots, \lfloor n_t/m \rfloor} \binom{(n_1 + \dots + n_t) \bmod m}{n_1 \bmod m, \dots, n_t \bmod m},$$

by Eq. 1.2.6–(43); here each upper index is the sum of the lower indices.

- 51.** All paths clearly run between 000111 and 111000, since those vertices have degree 1. Fourteen total paths reduce to four under the stated equivalences. The path in (50), which is equivalent to itself under reflection-and-reversal, can be described by the delta sequence $A = 3452132523414354123$; the other three classes are $B = 3452541453414512543$, $C = 3452541453252154123$, $D = 3452134145341432543$. D. H. Lehmer found path C [AMM 72 (1965), part II, 36–46]; D is essentially the path constructed by Eades, Hickey, and Read.

(Incidentally, perfect schemes aren't really rare, although they seem to be difficult to construct systematically. The case $s = 3$, $t = 5$ has 4,050,046 of them.)

- 52.** We may assume that each s_j is nonzero and that $d > 1$. Then the difference between permutations with an even and odd number of inversions is $\binom{\lfloor (s_0 + \dots + s_d)/2 \rfloor}{\lfloor s_0/2 \rfloor, \dots, \lfloor s_d/2 \rfloor} \geq 2$, by exercise 50, unless at least two of the multiplicities s_j are odd.

Conversely, if at least two multiplicities are odd, a general construction by G. Stachowiak [SIAM J. Discrete Math. 5 (1992), 199–206] shows that a perfect scheme exists. Indeed, his construction applies to a variety of topological sorting problems; in the special case of multisets it gives a Hamiltonian circuit in all cases with $d > 1$ and $s_0 s_1$ odd, except when $d = 2$, $s_0 = s_1 = 1$, and s_2 is even.

- 53.** See AMM 72 (1965), Part II, 36–46.

- 54.** Assuming that $st \neq 0$, a Hamiltonian path exists if and only if s and t are not both even; a Hamiltonian circuit exists if and only if, in addition, $(s \neq 2 \text{ and } t \neq 2)$ or $n = 5$. [T. C. Enns, Discrete Math. 122 (1993), 153–165.]

- 55.** [Discrete Math. 48 (1984), 163–171.] This problem is equivalent to the “middle levels conjecture,” which states that there is a Gray path through all binary strings of length $2t - 1$ and weights $\{t - 1, t\}$. In fact, such strings can almost certainly be generated by a delta sequence of the special form $\alpha_0 \alpha_1 \dots \alpha_{2t-2}$ where the elements of α_k are those of α_0 shifted by k , modulo $2t - 1$. For example, when $t = 3$ we can start with $a_5 a_4 a_3 a_2 a_1 a_0 = 000111$ and repeatedly swap $a_0 \leftrightarrow a_\delta$, where δ runs through the cycle (4134 5245 1351 2412 3523). The middle levels conjecture is known to be true for $t \leq 15$ [see I. Shields and C. D. Savage, Cong. Num. 140 (1999), 161–178].

- 56.** Yes; there is a near-perfect genlex solution for all m , n , and t when $n \geq m > t$. One such scheme, in bitstring notation, is $1A_{(m-t)(t-1)}0^{n-m}, 01A_{(m-t)(t-1)}0^{n-m-1}, \dots, 0^{n-m}1A_{(m-t)(t-1)}, 0^{n-m+1}1A_{(m-1-t)(t-1)}, \dots, 0^{n-t}1A_{0(t-1)}$, using the sequences A_{st} of (35).

- 57.** Solve the previous problem with m and n reduced by $t - 1$, then add $j - 1$ to each c_j . (Case (a), which is particularly simple, was probably known to Czerny.)

- 58.** The generating function $G_{mnt}(q) = \sum g_{mntk} q^k$ for the number g_{mntk} of chords reachable in k steps from $0^{n-t}1^t$ satisfies $G_{mnt}(q) = \binom{m}{t}_q$ and $G_{m(n+1)t}(q) = G_{mnt}(q) + q^{tn-(t-1)m} \binom{m-1}{t-1}_q$, because the latter term accounts for cases with $c_t = n$ and $c_1 > n - m$. A perfect scheme is possible only if $|G_{mnt}(-1)| \leq 1$. But if $n \geq m > t \geq 2$, this

condition holds only when $m = t + 1$ or $(n - t)t$ is odd, by exercise 49. So there is no perfect solution when $t = 4$ and $m > 5$. (Many chords have only two neighbors when $n = t + 2$, so one can easily rule out that case. All cases with $n \geq m > 5$ and $t = 3$ apparently do have perfect paths when n is even.)

59. The following solution uses lexicographic order, taking care to ensure that the average amount of computation per step is bounded. We may assume that $stm_s \dots m_0 \neq 0$ and $t \leq m_s + \dots + m_1 + m_0$.

Q1. [Initialize.] Set $q_j \leftarrow 0$ for $1 \leq j \leq s$, and $x = t$.

Q2. [Distribute.] Set $j \leftarrow 0$. Then while $x > m_j$, set $q_j \leftarrow m_j$, $x \leftarrow x - m_j$, $j \leftarrow j + 1$, and repeat until $x \leq m_j$. Finally set $q_j \leftarrow x$.

Q3. [Visit.] Visit the bounded composition $q_s \dots q_1 q_0$.

Q4. [Pick up the rightmost units.] If $j = 0$, set $x \leftarrow q_0 - 1$, $j \leftarrow 1$. Otherwise if $q_0 = 0$, set $x \leftarrow q_j - 1$, $q_j \leftarrow 0$, and $j \leftarrow j + 1$. Otherwise go to Q7.

Q5. [Full?] Terminate if $j > s$. Otherwise if $q_j = m_j$, set $x \leftarrow x + m_j$, $q_j \leftarrow 0$, $j \leftarrow j + 1$, and repeat this step.

Q6. [Increase q_j .] Set $q_j \leftarrow q_j + 1$. Then if $x = 0$, set $q_0 \leftarrow 0$ and return to Q3. Otherwise go to Q2.

Q7. [Increase and decrease.] While $q_j = m_j$, set $j \leftarrow j + 1$ and repeat until $q_j < m_j$ (but terminate if $j > s$). Then set $q_j \leftarrow q_j + 1$, $j \leftarrow j - 1$, $q_j \leftarrow q_j - 1$. If $q_0 = 1$, set $j \leftarrow 1$. Return to Q3. ■

For example, if $m_s = \dots = m_0 = 9$, the successors of the composition $3+9+9+7+0+0$ are $4+0+0+6+9+9$, $4+0+0+7+8+9$, $4+0+0+7+9+8$, $4+0+0+8+7+9$,

60. Let $F_s(t) = \emptyset$ if $t < 0$ or $t > m_s + \dots + m_0$; otherwise let $F_0(t) = t$, and

$$F_s(t) = 0 + F_{s-1}(t), 1 + F_{s-1}(t-1)^R, 2 + F_{s-1}(t-2), \dots, m_s + F_{s-1}(t-m_s)^{R^{m_s}}$$

when $s > 0$. This sequence can be shown to have the required properties; it is, in fact, equivalent to the compositions defined by the homogeneous sequence K_{st} of (31) under the correspondence of exercise 4, when restricted to the subsequence defined by the bounds m_s, \dots, m_0 . [See T. Walsh, *J. Combinatorial Math. and Combinatorial Computing* **33** (2000), 323–345, who has implemented it looplessly.]

61. (a) A $2 \times n$ contingency table with row sums r and $c_1 + \dots + c_n - r$ is equivalent to solving $r = a_1 + \dots + a_n$ with $0 \leq a_1 \leq c_1, \dots, 0 \leq a_n \leq c_n$.

(b) We can compute it sequentially by setting $a_{ij} \leftarrow \min(r_i - a_{i1} - \dots - a_{i(j-1)}, c_j - a_{1j} - \dots - a_{(i-1)j})$ for $j = 1, \dots, n$, for $i = 1, \dots, m$. Alternatively, if $r_1 \leq c_1$, set $a_{11} \leftarrow r_1$, $a_{12} \leftarrow \dots \leftarrow a_{1n} \leftarrow 0$, and do the remaining rows with c_1 decreased by r_1 ; if $r_1 > c_1$, set $a_{11} \leftarrow c_1$, $a_{21} \leftarrow \dots \leftarrow a_{m1} \leftarrow 0$, and do the remaining columns with r_1 decreased by c_1 . The second approach shows that at most $m + n - 1$ of the entries are nonzero. We can also write down the explicit formula

$$a_{ij} = \max(0, \min(r_i, c_j, r_1 + \dots + r_i - c_1 - \dots - c_{j-1}, c_1 + \dots + c_j - r_1 - \dots - r_{i-1})).$$

(c) The same matrix is obtained as in (b).

(d) Reverse left and right in (b) and (c), obtaining

$$a_{ij} = \max(0, \min(r_i, c_j, r_{i+1} + \dots + r_m - c_1 - \dots - c_{j-1}, c_1 + \dots + c_j - r_i - \dots - r_m))$$

in both cases.

(e) Here we choose, say, row-wise order: Generate the first row just as for bounded compositions of r_1 , with bounds (c_1, \dots, c_n) ; and for each row (a_{11}, \dots, a_{1n}) , generate the remaining rows recursively in the same way, but with the column sums $(c_1 - a_{11}, \dots, c_n - a_{1n})$. Most of the action takes place on the bottom two rows, but when a change is made to an earlier row the later rows must be re-initialized.

62. If a_{ij} and a_{kl} are positive, we obtain another contingency table by setting $a_{ij} \leftarrow a_{ij} - 1$, $a_{il} \leftarrow a_{il} + 1$, $a_{kj} \leftarrow a_{kj} + 1$, $a_{kl} \leftarrow a_{kl} - 1$. We want to show that the graph G whose vertices are the contingency tables for $(r_1, \dots, r_m; c_1, \dots, c_n)$, adjacent if they can be obtained from each other by such a transformation, has a Hamiltonian path.

When $m = n = 2$, G is a simple path. When $m = 2$ and $n = 3$, G has a two-dimensional structure from which we can see that every vertex is the starting point of at least two Hamiltonian paths, having distinct endpoints. When $m = 2$ and $n \geq 4$ we can show, inductively, that G actually has Hamiltonian paths from any vertex to any other.

When $m \geq 3$ and $n \geq 3$, we can reduce the problem from m to $m - 1$ as in answer 61(e), if we are careful not to “paint ourselves into a corner.” Namely, we must avoid reaching a state where the nonzero entries of the bottom two rows have the form $\begin{pmatrix} 1 & a & 0 \\ 0 & b & c \end{pmatrix}$ for some $a, b, c > 0$ and a change to row $m - 2$ forces this to become $\begin{pmatrix} 0 & a & 1 \\ 0 & b & c \end{pmatrix}$. The previous round of changes to rows $m - 1$ and m can avoid such a trap unless $c = 1$ and it begins with $\begin{pmatrix} 0 & a+1 & 0 \\ 1 & b-1 & 1 \end{pmatrix}$ or $\begin{pmatrix} 1 & a-1 & 1 \\ 0 & b+1 & 0 \end{pmatrix}$. But that situation can be avoided too.

(A genlex method based on exercise 60 would be considerably simpler, and it almost always would make only four changes per step. But it would occasionally need to update $2 \min(m, n)$ entries at a time.)

63. When $x_1 \dots x_s$ is a binary string and A is a list of subcubes, let $A \oplus x_1 \dots x_s$ denote replacing the digits (a_1, \dots, a_s) in each subcube of A by $(a_1 \oplus x_1, \dots, a_s \oplus x_s)$, from left to right. For example, $0*1**10 \oplus 1010 = 1*1**00$. Then the following mutual recursions define a Gray code, because A_{st} gives a Gray path from $0^s *^t$ to $10^{s01} *^t$ and B_{st} gives a Gray path from $0^s *^t$ to $*01^{s-1} *^{t-1}$, when $st > 0$:

$$\begin{aligned} A_{st} &= 0B_{(s-1)t}, *A_{s(t-1)} \oplus 001^{s-2}, 1B_{(s-1)t}^R; \\ B_{st} &= 0A_{(s-1)t}, 1B_{(s-1)t} \oplus 010^{s-2}, 1A_{s(t-1)} \oplus 1^s. \end{aligned}$$

The strings 001^{s-2} and 010^{s-2} are simply 0^s when $s < 2$; A_{s0} is Gray binary code; $A_{0t} = B_{0t} = *^t$.

Incidentally, the somewhat simpler construction

$$G_{st} = *G_{s(t-1)}, a_t G_{(s-1)t}, a_{t-1} G_{(s-1)t}^R, \quad a_t = t \bmod 2,$$

defines a pleasant Gray path from $*^t 0^s$ to $a_{t-1} *^{t-1} 0^{s-1}$.

64. If a path P is considered equivalent to P^R and to $P \oplus x_1 \dots x_s$, the total number can be computed systematically as in exercise 33, with the following results for $s+t \leq 6$:

paths	cycles
1	1
1 1	1 1
1 2 1	1 1 1
1 3 3 1	1 1 1 1
1 5 10 4 1	1 2 1 1 1
1 6 36 35 5 1	1 2 3 1 1 1
1 9 310 4630 218 6 1	1 3 46 4 1 1 1

In general there are $t+1$ paths when $s=1$ and $\binom{\lceil s/2 \rceil + 2}{2} - (s \bmod 2)$ when $t=1$. The cycles for $s \leq 2$ are unique. When $s=t=5$ there are approximately 6.869×10^{170} paths and 2.495×10^{70} cycles.

65. Let $G(n, 0) = \epsilon$; $G(n, t) = \emptyset$ when $n < t$; and for $1 \leq t \leq n$, let $G(n, t)$ be

$$\hat{g}(0)G(n-1, t), \hat{g}(1)G(n-1, t)^R, \dots, \hat{g}(2^t-1)G(n-1, t)^R, \hat{g}(2^t-1)G(n-1, t-1),$$

where $\hat{g}(k)$ is a t -bit column containing the Gray binary number $g(k)$ with its least significant bit at the top. In this general formula we implicitly add a row of zeros below the bases of $G(n-1, t-1)$.

This remarkable rule gives ordinary Gray binary code when $t=1$, omitting $0\dots00$. A cyclic Gray code is impossible because $\binom{n}{t}_2$ is odd.

66. A Gray path for compositions corresponding to Algorithm C implies that there is a path in which all transitions are $0^k 1^l \leftrightarrow 1^l 0^k$ with $\min(k, l) \leq 2$. Perhaps there is, in fact, a cycle with $\min(k, l) = 1$ in each transition.

67. (a) $\{\emptyset\}$; (b) \emptyset .

68. The least N with $\kappa_t N < N$ is $\binom{2t-1}{t} + \binom{2t-3}{t-1} + \dots + \binom{1}{1} + 1 = \frac{1}{2}(\binom{2t}{t} + \binom{2t-2}{t-1}) + \dots + \binom{0}{0} + 1$, because $\binom{n}{t-1} \leq \binom{n}{t}$ if and only if $n \geq 2t-1$.

69. From the identity

$$\kappa_t(\binom{2t-3}{t} + N') - (\binom{2t-3}{t} + N') = \kappa_t(\binom{2t-2}{t} + N') - (\binom{2t-2}{t} + N') = \binom{2t-2}{t} \frac{1}{t-1} + \kappa_{t-1} N' - N'$$

when $N' < \binom{2t-3}{t}$, we conclude that the maximum is $\binom{2t-2}{t} \frac{1}{t} + \binom{2t-4}{t-1} \frac{1}{t-2} + \dots + \binom{2}{1} \frac{1}{1}$, and it occurs at 2^{t-1} values of N when $t > 1$.

70. Let C_t be the t -cliques. The first $\binom{1414}{t} + \binom{1009}{t-1}$ t -combinations visited by Algorithm L define a graph on 1415 vertices with 1000000 edges. If $\|C_t\|$ were larger, $\|\partial^{t-2}C_t\|$ would exceed 1000000. Thus the single graph defined by $P_{(1000000)_2}$ has the maximum number of t -cliques for all $t \geq 2$.

71. $M = \binom{m_s}{s} + \dots + \binom{m_u}{u}$ for $m_s > \dots > m_u \geq u \geq 1$, where $\{m_s, \dots, m_u\} = \{s+t-1, \dots, n_v\} \setminus \{n_t, \dots, n_{v+1}\}$. (Compare with exercise 15, which gives $\binom{s+t}{t} - 1 - N$.)

If $\alpha = a_{n-1} \dots a_0$ is the bit string corresponding to the combination $n_t \dots n_1$, then v is 1 plus the number of trailing 1s in α , and u is the length of the rightmost run of 0s. For example, when $\alpha = 1010001111$ we have $s=4, t=6, M = \binom{8}{4} + \binom{6}{3}, u=3, N = \binom{9}{6} + \binom{7}{5}, v=5$.

72. A and B are cross-intersecting $\iff \alpha \not\subseteq U \setminus \beta$ for all $\alpha \in A$ and $\beta \in B \iff A \cap \partial^{n-s-t}B^- = \emptyset$, where $B^- = \{U \setminus \beta \mid \beta \in B\}$ is a set of $(n-t)$ -combinations. Since $Q_{Nnt}^- = P_{N(n-t)}$, we have $\|\partial^{n-s-t}B^-\| \geq \|\partial^{n-s-t}P_{N(n-t)}\|$, and $\partial^{n-s-t}P_{N(n-t)} = P_{N's}$ where $N' = \kappa_{s+1} \dots \kappa_{n-t} N$. Thus if A and B are cross-intersecting we have $M + N' \leq \|A\| + \|\partial^{n-s-t}B^-\| \leq \binom{n}{s}$, and $Q_{Mns} \cap P_{N's} = \emptyset$.

Conversely, if $Q_{Mns} \cap P_{N's} \neq \emptyset$ we have $\binom{n}{s} < M + N' \leq \|A\| + \|\partial^{n-s-t}B^-\|$, so A and B cannot be cross-intersecting.

73. $\|\partial Q_{Nnt}\| = \kappa_{n-t} N$ (see exercise 93). Also, arguing as in (58) and (59), we find $\partial P_{N5} = (n-1)P_{N5} \cup \dots \cup 10P_{N5} \cup \{543210, \dots, 987654\}$ in that particular case; and $\|\partial P_{Nt}\| = (n+1-n_t)N + \binom{n+t+1}{t+1}$ in general.

74. The identity $\binom{n+1}{k} = \binom{n}{k} + \binom{n-1}{k-1} + \dots + \binom{n-k}{0}$, Eq. 1.2.6–(10), gives another representation if $n_v > v$. But (6o) is unaffected, since we have $\binom{n+1}{k-1} = \binom{n}{k-1} + \binom{n-1}{k-2} + \dots + \binom{n-k+1}{0}$.

75. Represent $N+1$ by adding $\binom{v-1}{v-1}$ to (57); then use the previous exercise to deduce that $\kappa_t(N+1) - \kappa_t N = \binom{v-1}{v-2} = v-1$.

76. [D. E. Daykin, *Nanta Math.* **8**, 2 (1975), 78–83.] We work with extended representations $M = \binom{m_t}{t} + \cdots + \binom{m_u}{u}$ and $N = \binom{n_t}{t} + \cdots + \binom{n_v}{v}$ as in exercise 74, calling them *improper* if the final index u or v is zero. Call N *flexible* if it has both proper and improper representations, that is, if $n_v > v > 0$.

(a) Given an integer S , find $M+N$ such that $M+N=S$ and $\kappa_t M + \kappa_t N$ is minimum, with M as large as possible. If $N=0$, we're done. Otherwise the max-min operation preserves both $M+N$ and $\kappa_t M + \kappa_t N$, so we can assume that $v \geq u \geq 1$ in the proper representations of M and N . If N is inflexible, $\kappa_t(M+1) + \kappa_t(N-1) = (\kappa_t M + u - 1) + (\kappa_t N - v) < \kappa_t M + \kappa_t N$, by exercise 75; therefore N must be flexible. But then we can apply the max-min operation to M and the improper representation of N , increasing M : Contradiction.

This proof shows that equality holds if and only if $MN=0$, a fact that was noted in 1927 by F. S. Macaulay.

(b) Now we try to minimize $\max(\kappa_t M, N) + \kappa_{t-1} N$ when $M+N=S$, this time representing N as $\binom{n_{t-1}}{t-1} + \cdots + \binom{n_v}{v}$. The max-min operation can still be used if $n_{t-1} < m_t$; leaving m_t unchanged, it preserves $M+N$ and $\kappa_t M + \kappa_{t-1} N$ as well as the relation $\kappa_t M > N$. We arrive at a contradiction as in (a) if $N \neq 0$, so we can assume that $n_{t-1} \geq m_t$.

If $n_{t-1} > m_t$ we have $N > \kappa_t M$ and also $\lambda_t N > M$; hence $M+N < \lambda_t N + N = \binom{n_{t-1}+1}{t} + \cdots + \binom{n_v+1}{v}$, and we have $\kappa_t(M+N) \leq \kappa_t(\lambda_t N + N) = N + \kappa_{t-1} N$.

Finally if $n_{t-1} = m_t = a$, let $M = \binom{a}{t} + M'$ and $N = \binom{a}{t-1} + N'$. Then $\kappa_t(M+N) = \binom{a+1}{t-1} + \kappa_{t-1}(M' + N')$, $\kappa_t M = \binom{a}{t-1} + \kappa_{t-1} M'$, and $\kappa_{t-1} N = \binom{a}{t-2} + \kappa_{t-2} N'$; the result follows by induction on t .

77. [A. J. W. Hilton, *Periodica Math. Hung.* **10** (1979), 25–30.] Let $M = \|A_1\|$ and $N = \|A_0\|$; we can assume that $t > 0$ and $N > 0$. Then $\|\partial A\| = \|\partial A_1 \cup A_0\| + \|\partial A_0\| \geq \max(\|\partial A_1\|, \|A_0\|) + \|\partial A_0\| \geq \max(\kappa_t M, N) + \kappa_{t-1} N \geq \kappa_t(M+N) = \|P_{\|A\||t}\|$, by induction on $m+n+t$.

Conversely, let $A_1 = P_{Mt} + 1$ and $A_0 = P_{N(t-1)} + 1$; this notation means, for example, that $\{210, 320\} + 1 = \{321, 431\}$. Then $\kappa_t(M+N) \leq \|\partial A\| = \|\partial A_1 \cup A_0\| + \|\partial(A_0)0\| = \max(\kappa_t M, N) + \kappa_{t-1} N$, because $\partial A_1 = P_{(\kappa_t M)(t-1)} + 1$. [Schützenberger observed in 1959 that $\kappa_t(M+N) \leq \kappa_t M + \kappa_{t-1} N$ if and only if $\kappa_t M \geq N$.]

For the first inequality, let A and B be disjoint sets of t -combinations with $\|A\| = M$, $\|\partial A\| = \kappa_t M$, $\|B\| = N$, $\|\partial B\| = \kappa_t N$. Then $\kappa_t(M+N) = \kappa_t\|A \cup B\| \leq \|\partial(A \cup B)\| = \|\partial A \cup \partial B\| = \|\partial A\| + \|\partial B\| = \kappa_t M + \kappa_t N$.

78. In fact, $\mu_t(M + \lambda_{t-1} M) = M$, and $\mu_t N + \lambda_{t-1} \mu_t N = N + (n_2 - n_1)[v=1]$ when N is given by (57).

79. If $N > 0$ and $t > 1$, represent N as in (57) and let $N = N_0 + N_1$, where

$$N_0 = \binom{n_t-1}{t} + \cdots + \binom{n_v-1}{v}, \quad N_1 = \binom{n_t-1}{t-1} + \cdots + \binom{n_v-1}{v-1}.$$

Let $N_0 = \binom{y}{t}$ and $N_1 = \binom{z}{t-1}$. Then, by induction on t and $\lfloor x \rfloor$, we have $\binom{x}{t} = N_0 + \kappa_t N_0 \geq \binom{y}{t} + \binom{y}{t-1} = \binom{y+1}{t}$; $N_1 = \binom{x}{t} - \binom{y}{t} \geq \binom{x}{t} - \binom{x-1}{t} = \binom{x-1}{t-1}$; and $\kappa_t N = N_1 + \kappa_{t-1} N_1 \geq \binom{z}{t-1} + \binom{z}{t-2} = \binom{z+1}{t-1} \geq \binom{x}{t-1}$.

Lovász actually proved a stronger result [*Combinatorial Problems and Exercises*, (Akadémiai Kiadó, 1979), problem 13.31(a)], which was strengthened further by R. M. Redheffer [*AMM* **103** (1996), 62–64]: Suppose $\binom{u}{t} = \binom{v}{t} + \binom{w}{t-1}$ where $u \geq t-1$,

$v \geq t - 1$, and $w \geq t - 2$. Then for $1 \leq k < t$ we have $\binom{u}{k} \leq \binom{v}{k} + \binom{w}{k-1}$ if and only if $v \geq w$; for $t < k \leq \min(v+1, w+2)$ we have $\binom{u}{k} \leq \binom{v}{k} + \binom{w}{k-1}$ if and only if $v \leq w$.

80. For example, if the largest element of \widehat{P}_{N5} is 66433, we have

$$\widehat{P}_{N5} = \{00000, \dots, 55555\} \cup \{60000, \dots, 65555\} \cup \{66000, \dots, 66333\} \cup \{66400, \dots, 66433\}$$

so $N = \binom{10}{5} + \binom{9}{4} + \binom{6}{3} + \binom{5}{2}$. Its lower shadow is

$$\partial \widehat{P}_{N5} = \{0000, \dots, 5555\} \cup \{6000, \dots, 6555\} \cup \{6600, \dots, 6633\} \cup \{6640, \dots, 6643\},$$

of size $\binom{9}{4} + \binom{8}{3} + \binom{5}{2} + \binom{4}{1}$.

If the smallest element of Q_{N95} is 66433, we have

$$\widehat{Q}_{N95} = \{99999, \dots, 70000\} \cup \{66666, \dots, 66500\} \cup \{66444, \dots, 66440\} \cup \{66433\}$$

so $N = (\binom{13}{9} + \binom{12}{8} + \binom{11}{7}) + (\binom{8}{6} + \binom{7}{5}) + \binom{5}{4} + \binom{3}{3}$. Its upper shadow is

$$\begin{aligned} \partial \widehat{Q}_{N95} = & \{999999, \dots, 700000\} \cup \{666666, \dots, 665000\} \\ & \cup \{664444, \dots, 664400\} \cup \{664333, \dots, 664330\}, \end{aligned}$$

of size $(\binom{14}{9} + \binom{13}{8} + \binom{12}{7}) + (\binom{9}{6} + \binom{8}{5}) + \binom{6}{4} + \binom{4}{3} = N + \kappa_9 N$. The size, t , of each combination is essentially irrelevant, as long as $N \leq \binom{s+t}{t}$; for example, the smallest element of \widehat{Q}_{N98} is 99966433 in the case we have considered.

81. (a) The derivative would have to be $\sum_{k>0} r_k(x)$, but that series diverges.

[Informally, the graph of $\tau(x)$ shows “pits” of relative magnitude 2^{-k} at all odd multiples of 2^{-k} . Takagi’s original publication, in *Proc. Physico-Math. Soc. Japan (2)* **1** (1903), 176–177, has been translated into English in his *Collected Papers* (Iwanami Shoten, 1973).]

(b) Since $r_k(1-t) = (1)^{\lceil 2^k t \rceil}$ when $k > 0$, we have $\int_0^{1-x} r_k(t) dt = \int_x^1 r_k(1-u) du = - \int_x^1 r_k(u) du = \int_0^x r_k(u) du$. The second equation follows from the fact that $r_k(\frac{1}{2}t) = r_{k-1}(t)$. Part (d) shows that these two equations suffice to define $\tau(x)$ when x is rational.

(c) Since $\tau(2^{-a}x) = a2^{-a}x + 2^{-a}\tau(x)$ for $0 \leq x \leq 1$, we have $\tau(\epsilon) = a\epsilon + O(\epsilon)$ when $2^{-a-1} \leq \epsilon \leq 2^{-a}$. Therefore $\tau(\epsilon) = \epsilon \lg \frac{1}{\epsilon} + O(\epsilon)$ for $0 < \epsilon \leq 1$.

(d) Suppose $0 \leq p/q \leq 1$. If $p/q \leq 1/2$ we have $\tau(p/q) = p/q + \tau(2p/q)/2$; otherwise $\tau(p/q) = (q-p)/q + \tau(2(q-p)/q)/2$. Therefore we can assume that q is odd. When q is odd, let $p' = p/2$ when p is even, $p' = (q-p)/2$ when p is odd. Then $\tau(p/q) = 2\tau(p'/q) - 2p'/q$ for $0 < p < q$; this system of $q-1$ equations has a unique solution. For example, the values for $q = 3, 4, 5, 6, 7$ are $2/3, 2/3; 1/2, 1/2, 1/2; 8/15, 2/3, 2/3, 8/15; 1/2, 2/3, 1/2, 2/3, 1/2; 22/49, 30/49, 32/49, 32/49, 30/49, 22/49$.

(e) The solutions $< \frac{1}{2}$ are $x = \frac{1}{4}, \frac{1}{4} - \frac{1}{16}, \frac{1}{4} - \frac{1}{16} - \frac{1}{64}, \frac{1}{4} - \frac{1}{16} - \frac{1}{64} - \frac{1}{256}, \dots, \frac{1}{3}$.

(f) The value $\frac{2}{3}$ is achieved for $x = \frac{1}{2} \pm \frac{1}{8} \pm \frac{1}{32} \pm \frac{1}{128} \pm \dots$, an uncountable set.

82. Consider paths starting from 0 in the digraph

$$\begin{array}{ccccccccccc} 0 & \leftarrow & 1 & \leftarrow & 2 & \leftarrow & 3 & \leftarrow & 4 & \leftarrow & 5 & \leftarrow & \cdots \\ \downarrow & \uparrow & \\ 1 & \rightarrow & 2 & \rightarrow & 3 & \rightarrow & 4 & \rightarrow & 5 & \rightarrow & 6 & \rightarrow & \cdots \end{array}$$

Compute an associated value v , starting with $v \leftarrow -p$; horizontal moves change $v \leftarrow 2v$, vertical moves from node a change $v \leftarrow 2(qa - v)$. The path stops if we reach a node twice with the same value v . Transitions are not allowed to upper nodes with $-v \leq -q$

or $v \geq qa$; they are not allowed to lower nodes with $v \leq 0$ or $v \geq q(a+1)$. These restrictions force most steps of the path. (Node a in the upper row means, “Solve $\tau(x) = ax - v/q$ ”; in the lower row it means, “Solve $\tau(x) = v/q - ax$.”) Empirical tests suggest that, for all integers $q > p > 0$, at least one such path exists, and that all such paths are finite. The equation $\tau(x) = p/q$ then has solutions $x = x_0$ defined by the sequence x_0, x_1, x_2, \dots where $x_{k+1} = \frac{1}{2}x_k$ on a horizontal step and $x_{k+1} = 1 - \frac{1}{2}x_k$ on a vertical step; these are all the solutions to $\tau(x) = p/q$ when $x < \frac{1}{2}$, when q is not a power of 2.

For example, this procedure establishes that $\tau(x) = \frac{1}{5}$ only when x or $1-x$ is $3459/87040$; there are, similarly, just two solutions to $\tau(x) = 3/5$, having denominator $2^{46}(2^{56}-1)/3$.

Moreover, it appears that all cycles in the digraph that pass through node 0 define values of p and q such that $\tau(x) = p/q$ has uncountably many solutions. Such values are, for example, $2/3, 8/15, 8/21$, corresponding to the cycles $(01), (0121), (012321)$. The value $32/63$ corresponds to (012121) and also to (0121012345454321) , as well as to two other paths that do not return to 0.

83. [Frankl, Matsumoto, Ruzsa, and Tokushige, *J. Combinatorial Theory A* **69** (1995), 125–148.] If $a \leq b$ we have

$$\binom{2t-1-b}{t-a}/T = t^a(t-1)^{b-a}/(2t-1)^b = 2^{-b}(1 + f(a,b)t^{-1} + O(b^4/t^2)),$$

where $f(a,b) = a(1+b) - a^2 - b(1+b)/4 = f(a+1,b) - b + 2a$. Therefore if N has the combinatorial representation (57), and if we set $n_j = 2t-1-b_j$, we have

$$\frac{t}{T} \left(\kappa_t N - N \right) = \frac{b_t}{2^{b_t}} + \frac{b_{t-1}-2}{2^{b_{t-1}}} + \frac{b_{t-2}-4}{2^{b_{t-2}}} + \cdots + \frac{O(\log t)^3}{t},$$

the terms being negligible when b_j exceeds $2 \lg t$. And one can show that

$$\tau \left(\sum_{j=0}^l 2^{-e_j} \right) = \sum_{j=0}^l (e_j - 2j) 2^{-e_j}.$$

84. $N - \lambda_{t-1} N$ has the same asymptotic form as $\kappa_t N - N$, by (63), since $\tau(x) = \tau(1-x)$. So does $2\mu_t N - N$, up to $O(T(\log t)^3/t^2)$, because $\binom{2t-1-b}{t-a} = 2 \binom{2t-2-b}{t-a} (1 + O(\log t)/t)$ when $b < 2 \lg t$.

85. $x \in X^{\circ\sim} \iff \bar{x} \notin X^\circ \iff \bar{x} \notin X \text{ or } \bar{x} \notin X + e_1 \text{ or } \dots \text{ or } \bar{x} \notin X + e_n \iff x \in X^\sim \text{ or } x \in X^\sim - e_1 \text{ or } \dots \text{ or } x \in X^\sim - e_n \iff x \in X^{\sim+}$.

86. All three are true, using the fact that $X \subseteq Y^\circ$ if and only if $X^+ \subseteq Y$: (a) $X \subseteq Y^\circ \iff X^\sim \supseteq Y^{\circ\sim} = Y^{\sim+} \iff Y^\sim \subseteq X^{\sim\circ}$. (b) $X^+ \subseteq X^+ \implies X \subseteq X^{+\circ}$; hence $X^\circ \subseteq X^{\circ+\circ}$. Also $X^\circ \subseteq X^\circ \implies X^{\circ+\circ} \subseteq X$; hence $X^{\circ+\circ} \subseteq X^\circ$. (c) $\alpha M \leq N \iff S_M^+ \subseteq S_N \iff S_M \subseteq S_N^\circ \iff M \leq \beta N$.

87. If $\nu x < \nu y$ then $\nu(x - e_k) < \nu(y - e_j)$, so we can assume that $\nu x = \nu y$ and $x < y$. We must have $y_j > 0$; otherwise $\nu(y - e_j)$ would exceed $\nu(x - e_k)$. If $x_i = y_i$ for $1 \leq i \leq j$, clearly $k > j$ and $x - e_k \prec y - e_j$. Otherwise $x_i > y_i$ for some $i \leq j$; again we have $x - e_k \prec y - e_j$, unless $x - e_k = y - e_j$.

88. From the table

$j =$	0	1	2	3	4	5	6	7	8	9	10	11
$e_j + e_1 =$	e_1	e_0	e_4	e_5	e_2	e_3	e_8	e_9	e_6	e_7	e_{11}	e_{10}
$e_j + e_2 =$	e_2	e_4	e_0	e_6	e_1	e_8	e_3	e_{10}	e_5	e_{11}	e_7	e_9
$e_j + e_3 =$	e_3	e_5	e_6	e_7	e_8	e_9	e_{10}	e_0	e_{11}	e_1	e_2	e_4

we find $(\alpha 0, \alpha 1, \dots, \alpha 12) = (0, 4, 6, 7, 8, 9, 10, 11, 11, 12, 12, 12, 12)$; $(\beta 0, \beta 1, \dots, \beta 12) = (0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 8, 12)$.

89. Let $Y = X^+$ and $Z = C_k X$, and let $N_a = \|X_k(a)\|$ for $0 \leq a < m_k$. Then

$$\begin{aligned} \|Y\| &= \sum_{a=0}^{m_k-1} \|Y_k(a)\| = \sum_{a=0}^{m_k-1} \|(X_k(a-1) + e_k) \cup (X_k(a) + E_k(0))\| \\ &\geq \sum_{a=0}^{m_k-1} \max(N_{a-1}, \alpha N_a), \end{aligned}$$

where $a - 1$ stands for $(a - 1) \bmod m_k$ and the α function comes from the $(n - 1)$ -dimensional torus, because $\|X_k(a) + E_k(0)\| \geq \alpha N_a$ by induction. Also

$$\begin{aligned} \|Z^+\| &= \sum_{a=0}^{m_k-1} \|Z_k^+(a)\| = \sum_{a=0}^{m_k-1} \|(Z_k(a-1) + e_k) \cup (Z_k(a) + E_k(0))\| \\ &= \sum_{a=0}^{m_k-1} \max(N_{a-1}, \alpha N_a), \end{aligned}$$

because both $Z_k(a-1) + e_k$ and $Z_k(a) + E_k(0)$ are standard in $n - 1$ dimensions.

90. Let there be N_a points in row a of a totally compressed array, where row 0 is at the bottom; thus $l = N_{-1} \geq N_0 \geq \dots \geq N_{m-1} \geq N_m = 0$. We show first that there is an optimum X for which the “bad” condition $N_a = N_{a+1}$ never occurs except when $N_a = 0$ or $N_a = l$. For if a is the smallest bad subscript, suppose $N_{a-1} > N_a = N_{a+1} = \dots = N_{a+k} > N_{a+k+1}$. Then we can always decrease N_{a+k} by 1 and add 1 to some N_b for $b \leq a$ without increasing $\|X^+\|$, except in cases where $k = 1$ and $N_{a+2} = N_{a+1} - 1$ and $N_b = N_a + a - b$ for $0 \leq b \leq a$. Exploring such cases further, we can find a subscript d such that $N_c = N_{a+1} + a + 1 - c > 0$ for $a < c < d$, and either $N_d = 0$ or $N_d < N_{d+1} - 1$. Then it is OK to decrease N_c by 1 for $a < c < d$ and increase N_b by 1 for $0 \leq b < d - a - 1$. (It is important to note that if $N_d = 0$ we have $N_0 \geq d - 1$; hence $d = m$ implies $l = m$.)

Repeating such transformations until $N_a > N_{a+1}$ whenever $N_a \neq l$ and $N_{a+1} \neq 0$, we reach situation (86), and the proof can be completed as in the text.

91. Let $x + k$ denote the lexicographically smallest element of $T(m_1, \dots, m_{n-1})$ that exceeds x and has weight $\nu x + k$, if any such element exists. For example, if $m_1 = m_2 = m_3 = 4$ and $x = 211$, we have $x+1 = 212$, $x+2 = 213$, $x+3 = 223$, $x+4 = 233$, $x+5 = 333$, and $x+6$ does not exist; in general, $x+k+1$ is obtained from $x+k$ by increasing the rightmost component that can be increased. If $x+k = (m_1 - 1, \dots, m_{n-1} - 1)$, let us set $x+k+1 = x+k$. Then if $S(k)$ is the set of all elements of $T(m_1, \dots, m_{n-1})$ that are $\leq x+k$, we have $S(k+1) = S(k)^+$. Furthermore, the elements of S that end in a are those whose first $n-1$ components are in $S(m-1-a)$.

The result of this exercise can be stated more intuitively: As we generate n -dimensional standard sets S_1, S_2, \dots , the $(n-1)$ -dimensional standard sets on each layer become spreads of each other just after each point is added to layer $m-1$. Similarly, they become cores of each other just before each point is added to layer 0.

92. (a) Suppose the parameters are $2 \leq m'_1 \leq m'_2 \leq \dots \leq m'_n$ when sorted properly, and let k be minimal with $m_k \neq m'_k$. Then take $N = 1 + \text{rank}(0, \dots, 0, m'_k - 1, 0, \dots, 0)$. (We must assume that $\min(m_1, \dots, m_n) \geq 2$, since parameters equal to 1 can be placed anywhere.)

(b) Only in the proof for $n = 2$, buried inside the answer to exercise 90. That proof is incorporated by induction when n is larger.

93. Complementation reverses lexicographic order and changes \mathcal{O} to ∂ .

94. For Theorem K, let $d = n - 1$ and $s_0 = \dots = s_d = 1$. For Theorem M, let $d = s$ and $s_0 = \dots = s_d = t + 1$.

95. In such a representation, N is the number of t -multicombinations of $\{s_0 \cdot 0, s_1 \cdot 1, s_2 \cdot 2, \dots\}$ that precede $n_t n_{t-1} \dots n_1$ in lexicographic order, because the generalized coefficient $\binom{S(n)}{t}$ counts the multicombinations whose leftmost component is $< n$.

If we truncate the representation by stopping at the rightmost nonzero term $\binom{S(n_v)}{v}$, we obtain a nice generalization of (6o):

$$\|\partial P_{Nt}\| = \binom{S(n_t)}{t-1} + \binom{S(n_{t-1})}{t-2} + \dots + \binom{S(n_v)}{v-1}.$$

[See G. F. Clements, *J. Combinatorial Theory A* **37** (1984), 91–97. The inequalities $s_0 \geq s_1 \geq \dots \geq s_d$ are needed for the validity of Corollary C, but not for the calculation of $\|\partial P_{Nt}\|$. Some terms $\binom{S(n_k)}{k}$ for $t \geq k > v$ may be zero. For example, when $N = 1$, $t = 4$, $s_0 = 3$, and $s_1 = 2$, we have $N = \binom{S(1)}{4} + \binom{S(1)}{3} = 0 + 1$.]

96. (a) The tetrahedron has four vertices, six edges, four faces: $(N_0, \dots, N_4) = (1, 4, 6, 4, 1)$. The octahedron, similarly, has $(N_0, \dots, N_6) = (1, 6, 8, 8, 0, 0, 0)$, and the icosahedron has $(N_0, \dots, N_{12}) = (1, 12, 30, 20, 0, \dots, 0)$. The hexahedron, aka the 3-cube, has eight vertices, 12 edges, and six square faces; perturbation breaks each square face into two triangles and introduces new edges, so we have $(N_0, \dots, N_8) = (1, 8, 18, 12, 0, \dots, 0)$. Finally, the perturbed pentagonal faces of the dodecahedron lead to $(N_0, \dots, N_{20}) = (1, 20, 54, 36, 0, \dots, 0)$.

(b) $\{210, 310\} \cup \{10, 20, 21, 30, 31\} \cup \{0, 1, 2, 3\} \cup \{\epsilon\}$.

(c) $0 \leq N_t \leq \binom{n}{t}$ for $0 \leq t \leq n$ and $N_{t-1} \geq \kappa_t N_t$ for $1 \leq t \leq n$. The second condition is equivalent to $\lambda_{t-1} N_{t-1} \geq N_t$ for $1 \leq t \leq n$, if we define $\lambda_0 1 = \infty$. These conditions are necessary for Theorem K, and sufficient if $A = \bigcup P_{Ntt}$.

(d) The complements of the elements not in a simplicial complex, namely the sets $\{\{0, \dots, n-1\} \setminus \alpha \mid \alpha \notin C\}$, form a simplicial complex. (We can also verify that the necessary and sufficient condition holds: $N_{t-1} \geq \kappa_t N_t \iff \lambda_{t-1} N_{t-1} \geq N_t \iff \kappa_{n-t+1} \bar{N}_{n-t+1} \leq \bar{N}_{n-t}$, because $\kappa_{n-t} \bar{N}_{n-t+1} = \binom{n}{t} - \lambda_{t-1} N_{t-1}$ by exercise 93.)

(e) $00000 \leftrightarrow 14641$; $10000 \leftrightarrow 14640$; $11000 \leftrightarrow 14630$; $12000 \leftrightarrow 14620$; $13000 \leftrightarrow 14610$; $14000 \leftrightarrow 14600$; $12100 \leftrightarrow 14520$; $13100 \leftrightarrow 14510$; $14100 \leftrightarrow 14500$; $13200 \leftrightarrow 14410$; $14200 \leftrightarrow 14400$; $13300 \leftrightarrow 14400$; and the self-dual cases $14300, 13310$.

97. The following procedure by S. Linusson [*Combinatorica* **19** (1999), 255–266], who considered also the more general problem for multisets, is considerably faster than a more obvious approach. Let $L(n, h, l)$ count feasible vectors with $N_t = \binom{n}{t}$ for $0 \leq t \leq l$, $N_{t+1} < \binom{n}{t+1}$, and $N_t = 0$ for $t > h$. Then $L(n, h, l) = 0$ unless $-1 \leq l \leq h \leq n$; also $L(n, h, h) = L(n, h, -1) = 1$, and $L(n, n, l) = L(n, n-1, l)$ for $l < n$. When $n > h \geq l \geq 0$ we can compute $L(n, h, l) = \sum_{j=l}^h L(n-1, h, j) L(n-1, j-1, l-1)$, a recurrence that follows from Theorem K. (Each size vector corresponds to the complex $\bigcup P_{Ntt}$, with $L(n-1, h, j)$ representing combinations that do not contain the maximum element $n-1$ and $L(n-1, j-1, l-1)$ representing those that do.) Finally the grand total is $L(n) = \sum_{l=1}^n L(n, n, l)$.

We have $L(0), L(1), L(2), \dots = 2, 3, 5, 10, 26, 96, 553, 5461, 100709, 3718354, 289725509, \dots$; $L(100) \approx 3.2299 \times 10^{1842}$.

98. The maximal elements of a simplicial complex form a clutter; conversely, the combinations contained in elements of a clutter form a simplicial complex. Thus the two concepts are essentially equivalent.

(a) If (M_0, M_1, \dots, M_n) is the size vector of a clutter, then (N_0, N_1, \dots, N_n) is the size vector of a simplicial complex if $N_n = M_n$ and $N_t = M_t + \kappa_{t+1}N_{t+1}$ for $0 \leq t < n$. Conversely, every such (N_0, \dots, N_n) yields an (M_0, \dots, M_n) if we use the lexicographically first N_t t -combinations. [G. F. Clements extended this result to general multisets in *Discrete Math.* 4 (1973), 123–128.]

(b) In the order of answer 96(e) they are 00000, 00001, 10000, 00040, 01000, 00030, 02000, 00120, 03000, 00310, 04000, 00600, 00100, 00020, 01100, 00210, 02100, 00500, 00200, 00110, 01200, 00400, 00300, 01010, 01300, 00010. Notice that (M_0, \dots, M_n) is feasible if and only if (M_n, \dots, M_0) is feasible, so we have a different sort of duality in this interpretation.

99. Represent A as a subset of $T(m_1, \dots, m_n)$ as in the proof of Corollary C. Then the maximum value of νA is obtained when A consists of the N lexicographically smallest points $x_1 \dots x_n$.

The proof starts by reducing to the case that A is compressed, in the sense that its t -multicombinations are $P_{||A \cap T_t||^t}$ for each t . Then if y is the largest element $\in A$ and if x is the smallest element $\notin A$, we prove that $x < y$ implies $\nu x > \nu y$, hence $\nu(A \setminus \{y\} \cup \{x\}) > \nu A$. For if $\nu x = \nu y - k$ we could find an element of $\partial^k y$ that is greater than x , contradicting the assumption that A is compressed.

100. In general, $F(p) = N_0 p^n + N_1 p^{n-1}(1-p) + \dots + N_n(1-p)^n$ when $f(x_1, \dots, x_n)$ is satisfied by exactly N_t binary strings $x_1 \dots x_n$ of weight t . Thus we find $G(p) = p^4 + 3p^3(1-p) + p^2(1-p)^2$; $H(p) = p^4 + p^3(1-p) + p^2(1-p)^2$.

(b) A monotone formula f is equivalent to a simplicial complex C under the correspondence $f(x_1, \dots, x_n) = 1 \iff \{j-1 \mid x_j = 0\} \in C$. Therefore the functions $f(p)$ of monotone Boolean functions are those that satisfy the condition of exercise 96(c), and we obtain a suitable function by choosing the lexicographically last N_{n-t} t -combinations (which are complements of the first N_s s -combinations): $\{3210\}, \{321, 320, 310\}, \{32\}$ gives $f(w, x, y, z) = wxyz \vee xyz \vee wyz \vee wxz \vee yz = wxz \vee yz$.

M. P. Schützenberger observed that we can find the parameters N_t values easily from $f(p)$ by noting that $f(1/(1+u)) = (N_0 + N_1u + \dots + N_nu^n)/(1+u)^n$. One can show that $H(p)$ is not equivalent to a monotone formula in any number of variables, because $(1+u+u^2)/(1+u)^4 = (N_0 + N_1u + \dots + N_nu^n)/(1+u)^n$ implies that $N_1 = n-3$, $N_2 = \binom{n-3}{2} + 1$, and $\kappa_2 N_2 = n-2$.

But the task of deciding this question is not so simple in general. For example, the function $(1+5u+5u^2+5u^3)/(1+u)^5$ does not match any monotone formula in five variables, because $\kappa_{35} = 7$; but it equals $(1+6u+10u^2+10u^3+5u^4)/(1+u)^6$, which works fine with six.

101. (a) Choose N_t linearly independent polynomials of degree t in I ; order their terms lexicographically, and take linear combinations so that the lexicographically smallest terms are distinct monomials. Let I' consist of all multiples of those monomials.

(b) Each monomial of degree t in I' is essentially a t -multicomination; for example, $x_1^3 x_2 x_5^4$ corresponds to 55552111. If M_t is the set of independent monomials for degree t , the ideal property is equivalent to saying that $M_{t+1} \supseteq \partial M_t$.

In the given example, $M_3 = \{x_0 x_1^2\}$; $M_4 = \partial M_3 \cup \{x_0 x_1 x_2^2\}$; $M_5 = \partial M_4 \cup \{x_1 x_2^4\}$, since $x_2^2(x_0 x_1^2 - 2x_1 x_2^2) - x_1(x_0 x_1 x_2^2) = -2x_1 x_2^4$; and $M_{t+1} = \partial M_t$ thereafter.

(c) By Theorem M we can assume that $M_t = \hat{Q}_{Mst}$. Let $N_t = \binom{n_{ts}}{s} + \cdots + \binom{n_{t2}}{2} + \binom{n_{t1}}{1}$, where $s+t \geq n_{ts} > \cdots > n_{t2} > n_{t1} \geq 0$; then $n_{ts} = s+t$ if and only if $n_{t(s-1)} = s-2, \dots, n_{t1} = 0$. Furthermore we have

$$N_{t+1} \geq N_t + \kappa_s N_t = \binom{n_{ts} + [n_{ts} \geq s]}{s} + \cdots + \binom{n_{t2} + [n_{t2} \geq 2]}{2} + \binom{n_{t1} + [n_{t1} \geq 1]}{1}.$$

Therefore the sequence $(n_{ts} - t - \infty[n_{ts} < s], \dots, n_{t2} - t - \infty[n_{t2} < 2], n_{t1} - t - \infty[n_{t1} < 1])$ is lexicographically nondecreasing as t increases, where we insert ‘ $-\infty$ ’ in components that have $n_{tj} = j-1$. Such a sequence cannot increase infinitely many times without exceeding the maximum value $(s, -\infty, \dots, -\infty)$, by exercise 1.2.1-15(d).

102. Let P_{Nst} be the first N elements of a sequence determined as follows: For each binary string $x = x_{s+t-1} \dots x_0$, in lexicographic order, write down $\binom{\nu x}{t}$ subcubes by changing t of the 1s to *s in all possible ways, in lexicographic order (considering $1 < *$). For example, if $x = 0101101$ and $t = 2$, we generate the subcubes 0101*0*, 010*10*, 010**01, 0*0110*, 0*01*01, 0*0*101.

[See B. Lindström, *Arkiv för Mat.* **8** (1971), 245–257; a generalization analogous to Corollary C appears in K. Engel, *Sperner Theory* (Cambridge Univ. Press, 1997), Theorem 8.1.1.]

103. The first N strings in cross order have the desired property. [T. N. Danh and D. E. Daykin, *J. London Math. Soc.* (2) **55** (1997), 417–426.]

Notes: Beginning with the observation that the “1-shadow” of the N lexicographically first strings of weight t (namely the strings obtained by deleting 1 bits only) consists of the first $\mu_t N$ strings of weight t , R. Ahlswede and N. Cai extended the Danh-Daykin theorem to allow insertion, deletion, and/or transposition of bits [*Combinatorica* **17** (1997), 11–29; *Applied Math. Letters* **11**, 5 (1998), 121–126]. Uwe Leck has proved that no total ordering of *ternary strings* has the analogous minimum-shadow property [Preprint 98/6 (Univ. Rostock, 1998), 6 pages].

104. Every number must occur the same number of times in the cycle. Equivalently, $\binom{n-1}{t-1}$ must be a multiple of t . This necessary condition appears to be sufficient as well, provided that n is not too small with respect to t ; but such a result may well be true yet impossible to prove. [See Chung, Graham, and Diaconis, *Discrete Math.* **110** (1992), 55–57.]

The next few exercises consider the cases $t = 2$ and $t = 3$, when elegant results are known. Similar but more complicated results have been derived for $t = 4$ and $t = 5$, and the case $t = 6$ has been partially resolved. The case $(n, t) = (12, 6)$ is currently the smallest for which the existence of a universal cycle is unknown.

105. Let the differences mod $(2m+1)$ be $1, 2, \dots, m, 1, 2, \dots, m, \dots$, repeated $2m+1$ times; for example, the cycle for $m = 3$ is (013602561450346235124). This works because $1 + \cdots + m = \binom{m+1}{2}$ is relatively prime to $2m+1$. [*J. École Polytechnique* **4**, Cahier 10 (1810), 16–48.]

106. The seven doubles ■■, ■●, ..., ●■ can be inserted in 3^7 ways into any universal cycle of 3-combinations for $\{0, 1, 2, 3, 4, 5, 6\}$. The number of such universal cycles is the number of Eulerian circuits of the complete graph K_7 , which can be shown to be 129,976,320 if we regard $(a_0a_1 \dots a_{20})$ as equivalent to $(a_1 \dots a_{20}a_0)$ but not to the reverse-order cycle $(a_{20} \dots a_1a_0)$. So the answer is 284,258,211,840.

[This problem was first solved in 1859 by M. Reiss, whose method was so complicated that people doubted the result; see *Nouvelles Annales de Mathématiques* **8**

(1849), 74; **11** (1852), 115; *Annali di Matematica Pura ed Applicata* (2) **5** (1871–1873), 63–120. A considerably simpler solution, confirming Reiss's claim, was found by P. Jolivald and G. Tarry, who also enumerated the Eulerian circuits of K_9 ; see *Comptes Rendus Association Française pour l'Avancement des Sciences* **15**, part 2 (1886), 49–53; É. Lucas, *Récréations Mathématiques* **4** (1894), 123–151. Brendan D. McKay and Robert W. Robinson found an approach that is better still, enabling them to continue the enumeration through K_{21} by using the fact that the number of circuits is

$$(m-1)!^{2m+1} [z_0^{2m} z_1^{2m-2} \dots z_{2m}^{2m-2}] \det(a_{jk}) \prod_{1 \leq j < k \leq 2m} (z_j^2 + z_k^2),$$

where $a_{jk} = -1/(z_j^2 + z_k^2)$ when $j \neq k$; $a_{jj} = -1/(2z_j^2) + \sum_{0 \leq k \leq 2m} 1/(z_j^2 + z_k^2)$; see *Combinatorics, Probability, and Computing* **7** (1998), 437–449.]

C. Flye Sainte-Marie, in *L'Intermédiaire des Mathématiciens* **1** (1894), 164–165, noted that the Eulerian circuits of K_7 include 2×720 that have 7-fold symmetry under permutation of $\{0, 1, \dots, 6\}$ (namely Poinsot's cycle and its reverse), plus 32×1680 with 3-fold symmetry, plus 25778×5040 cycles that are asymmetric.

107. No solution is possible for $n < 7$, except in the trivial case $n = 4$. When $n = 7$ there are $12,255,208 \times 7!$ universal cycles, not considering $(a_0a_1\dots a_{34})$ to be the same as $(a_1\dots a_{34}a_0)$, including cases with 5-fold symmetry like the example cycle in exercise 104.

When $n \geq 8$ we can proceed systematically as suggested by B. Jackson in *Discrete Math.* **117** (1993), 141–150; see also G. Hurlbert, *SIAM J. Disc. Math.* **7** (1994), 598–604: Put each 3-combination into the “standard cyclic order” $c_1c_2c_3$ where $c_2 = (c_1 + \delta) \bmod n$, $c_3 = (c_2 + \delta') \bmod n$, $0 < \delta, \delta' < n/2$, and either $\delta = \delta'$ or $\max(\delta, \delta') < n - \delta - \delta' \neq (n-1)/2$ or $(1 < \delta < n/4 \text{ and } \delta' = (n-1)/2)$ or $(\delta = (n-1)/2 \text{ and } 1 < \delta' < n/4)$. For example, when $n = 8$ the allowable values of (δ, δ') are $(1, 1)$, $(1, 2)$, $(1, 3)$, $(2, 1)$, $(2, 2)$, $(3, 1)$, $(3, 3)$; when $n = 11$ they are $(1, 1)$, $(1, 2)$, $(1, 3)$, $(1, 4)$, $(2, 1)$, $(2, 2)$, $(2, 3)$, $(2, 5)$, $(3, 1)$, $(3, 2)$, $(3, 3)$, $(4, 1)$, $(4, 4)$, $(5, 2)$, $(5, 5)$. Then construct the digraph with vertices (c, δ) for $0 \leq c < n$ and $1 < \delta < n/2$, and with arcs $(c_1, \delta) \rightarrow (c_2, \delta')$ for every combination $c_1c_2c_3$ in standard cyclic order. This digraph is connected and balanced, so it has an Eulerian circuit by Theorem 2.3.4.2D. (The peculiar rules about $(n-1)/2$ make the digraph connected when n is odd. The Eulerian circuit can be chosen to have n -fold symmetry when $n = 8$, but not when $n = 12$.)

108. When $n = 1$ the cycle (000) is trivial; when $n = 2$ there is no cycle; and there are essentially only two when $n = 4$, namely (00011122233302021313) and (000111202023332221313) . When $n \geq 5$, let the multicombo $d_1d_2d_3$ be in standard cyclic order if $d_2 = (d_1 + \delta - 1) \bmod n$, $d_3 = (d_2 + \delta' - 1) \bmod n$, and (δ, δ') is allowable for $n+3$ in the previous answer. Construct the digraph with vertices (d, δ) for $0 \leq d < n$ and $1 < \delta < (n+3)/2$, and with arcs $(d_1, \delta) \rightarrow (d_2, \delta')$ for every multicombo $d_1d_2d_3$ in standard cyclic order; then find an Eulerian circuit.

Perhaps a universal cycle of t -multicombinations exists for $\{0, 1, \dots, n-1\}$ if and only if a universal cycle of t -combinations exists for $\{0, 1, \dots, n+t-1\}$.

109. A nice way to check for runs is to compute the numbers $b(S) = \sum\{2^{p(c)} \mid c \in S\}$ where $(p(A), \dots, p(K)) = (1, \dots, 13)$; then set $l \leftarrow b(S) \wedge -b(S)$ and check that $b(S) + l = l \ll s$, and also that $((l \ll s) \vee (l \gg 1)) \wedge a = 0$, where $a = 2^{p(c_1)} \vee \dots \vee 2^{p(c_5)}$. The values of $b(S)$ and $\sum\{v(c) \mid c \in S\}$ are easily maintained as S runs through all 31 nonempty subsets in Gray-code order. The answers are $(1009008, 99792, 2813796, 505008, 2855676, 697508, 1800268, 751324, 1137236, 361224, 388740, 51680, 317340,$

19656, 90100, 9168, 58248, 11196, 2708, 0, 8068, 2496, 444, 356, 3680, 0, 0, 0, 76, 4)
for $x = (0, \dots, 29)$; thus the mean score is ≈ 4.769 and the variance is ≈ 9.768 .

Note: A four-card flush is not allowed in the “crib.” Then the distribution is a bit easier to compute, and it turns out to be (1022208, 99792, 2839800, 508908, 2868960, 703496, 1787176, 755320, 1118336, 358368, 378240, 43880, 310956, 16548, 88132, 9072, 57288, 11196, 2264, 0, 7828, 2472, 444, 356, 3680, 0, 0, 0, 76, 4); the mean and variance decrease to approximately 4.735 and 9.667.

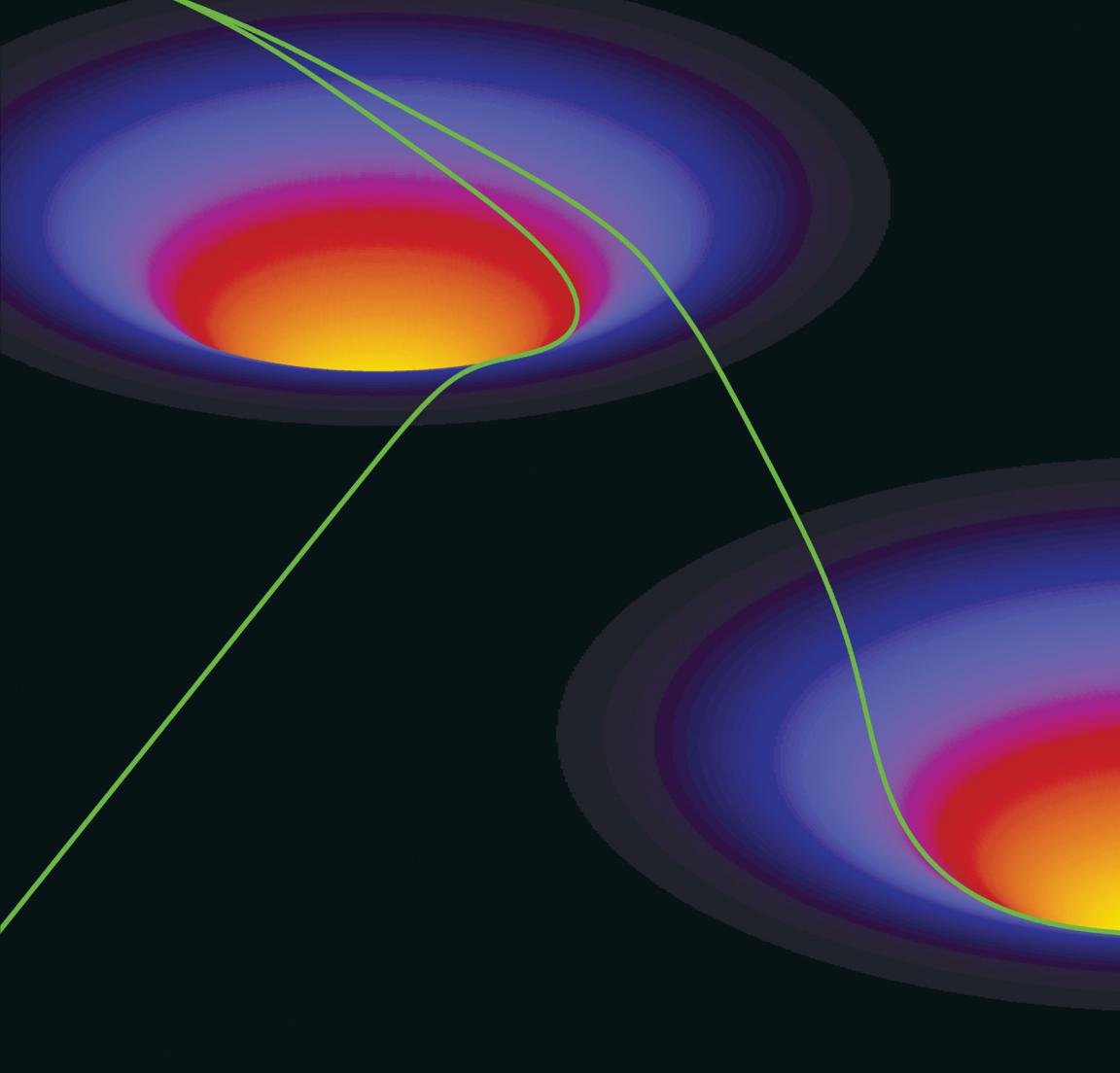
INDEX AND GLOSSARY

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.

- 2-nomial coefficients, 37.
 κ_t (Kruskal function), 19–21, 31–33.
 λ_t (Kruskal function), 20–21, 32–33.
 μ_t (Macaulay function), 20–21, 32–33.
 ν (sideways sum), 20.
 π (circle ratio), 2, 13, 27, 28, 35.
 τ (Takagi function), 20–21, 32–33.
 ∂ (shadow), 18.
 θ (upper shadow), 18.
- Active bits, 12.
Adjacent transpositions, 15–17, 30.
Ahlswede, Rudolph, 56.
Alternating combinatorial number system, 9, 27.
Analysis of algorithms, 4–5, 25, 27, 29.
- Balanced ternary notation, 41.
Baseball, 26.
Basis of vector space, 26, 31.
Basis theorem, 34.
Beckenbach, Edwin Ford, 5.
Bellman, Richard Ernest, 19.
Bernoulli, Jacques (= Jakob = James), iii, 16.
Binary tree representation of tree, 27.
Binary vector spaces, 26, 31.
Binomial coefficients, 1, 32.
generalized, 33.
inequalities for, 50.
Binomial number system, *see* Combinatorial number system.
Binomial trees, 6–7, 27.
Bitner, James Richard, 8.
Bitwise manipulation, 57.
Boolean formulas, 34.
Bounded compositions, 16, 30.
Buck, Marshall Wilbert, 30.
- Cai, Ning (蔡宁), 56.
Calabi, Eugenio, 38.
Canonical bases, 26, 31.
Caron, Jacques, 42.
Chase, Phillip John, 11–12, 28, 45.
Chinese rings, 28.
Chords, 10, 30.
Chung Graham, Fan Rong King (鍾金芳蓉), 56.
Clements, George Francis, 24–25, 34, 54, 55.
Cliques, 31.
Colex order, 5.
- Combination generation, 1–18, 25–31, 35.
Gray codes for, 8–18.
homogeneous, 10–11, 16–17, 28–29, 41, 45, 47.
near-perfect, 11–17, 29.
perfect, 15–17, 30.
Combinations, 1–36.
of a multiset, 25.
with repetitions, 2–3, 11.
Combinatorial number system, 6, 27, 31–32, 37.
alternating, 9, 27.
generalized, 33.
Complement in a torus, 21.
Complete binary tree, 39.
Complete graph, 56.
Compositions, 2–3, 11, 25, 38.
bounded, 16, 30.
Compression of a set, 23, 33, 55.
Contingency tables, 18, 31.
Core set in a torus, 22–23, 33.
Cribbage, 35.
Cross-intersecting sets, 32.
Cross order, 20–25, 33, 56.
Cycle, universal, of combinations, 35.
Czerny, Carl, 46.
- Danh, Tran-Ngoc, 56.
Daykin, David Edward, 50, 56.
De Morgan, Augustus, 1.
Delta sequence, 46.
Derivative, 32.
Diaconis, Persi Warren, 56.
Dimension of a vector space, 26.
Dominoes, 35.
Dual combinations, 2–3, 26–27, 29.
Dual set in a torus, 22–23.
Dual size vector, 34.
Duality, 33, 55.
Dvořák, Stanislav, 36.
- Eades, Peter Dennis, 16, 46.
Ehrlich, Gideon (הילך אידען), 8, 42.
End-around swaps, 30.
Endo-order, 14, 29.
Engel, Konrad Wolfgang, 56.
Enns, Theodore Christian, 46.
Erdős, Pál (= Paul), 19.
Euler, Leonhard (Эйлеръ, Леонардъ = Эйлер, Леонард), circuits, 56, 57.

- Fenichel, Robert Ross, 25.
 First-element swaps, 16–17, 30.
 Flye Sainte-Marie, Camille, 57.
 Fraenkel, Aviezri S (פרנקל אביער), 39.
 Frankl, Péter, 52.
- Generating functions, 29, 46.
 Genlex order, 9–13, 16–17, 28–29, 44.
 Gray paths, 31.
 Golomb, Solomon Wolf, 2, 25.
 Graham, Ronald Lewis (葛立恒), 56.
 Gray, Frank, binary code, 8, 40, 49, 57.
 codes for combinations, 8–18.
 Grid paths, 2–3, 25.
- Hamilton, William Rowan, circuits, 8, 46.
 paths, 30, 48.
- Hickey, Thomas Butler, 16, 46.
- Hilbert, David, basis theorem, 34.
- Hilton, Anthony John William, 31, 50.
- Homogeneous generation, 10–11, 28–29, 45.
 scheme K_{st} , 10, 16–17, 29, 41, 47.
- Homogeneous polynomials, 34.
- Hurlbert, Glenn Howland, 57.
- Hypergraphs, 18.
- Internet, ii, iii, 26.
- Ising, Ernst, configurations, 26, 31, 38.
- Iteration versus recursion, 12–14, 29.
- Jackson, Bradley Warren, 57.
- Jenkyns, Thomas Arnold, 11.
- Jolivald, Philippe (= Paul de Hijo), 57.
- Katona, Gyula (Optimális Halmaz), 19.
- Keyboard, 10.
- Knapsack problem, 7.
- Knuth, Donald Ervin (高德纳), i, iv, 38.
- Korsh, James F., 38.
- Kruskal, Joseph Bernard, 19–20.
 function κ_t 19–21, 31–33.
 function λ_t 20–21, 32–33.
 –Katona theorem, 19.
- Leck, Uwe, 56.
- Lehmer, Derrick Henry, 5, 30, 46.
- Lexicographic generation, 4–7, 16–19,
 25–27, 29, 31, 47.
- Lindström, Bernt Lennart Daniel,
 24–25, 34, 56.
- Linked lists, 27, 39.
- Linusson, Hans Svante, 54.
- Lipschutz, Seymour Saul, 38.
- Liu, Chao-Ning (劉兆寧), 8.
- Loopless generation, 8, 25, 27, 28, 40, 45.
- Lovász, László, 32, 50.
- Lucas, François Édouard Anatole, 57.
- Lüneburg, Heinz, 39.
- Macaulay, Francis Sowerby, 19, 34, 50.
 function μ_t , 20–21, 32–33.
- Matrix multiplication, 42.
- Matsumoto, Makoto (松本眞), 52.
- McCarthy, David, 11.
- McKay, Brendan Damien, 57.
- Middle levels conjecture, 46.
- Min-plus matrix multiplication, 42.
- MMIX, ii.
- Monomials, 34.
- Monotone Boolean functions, 34.
- Mor, Moshe (מור משה), 39.
- Multicombinations, 2–3, 16–17, 25, 33.
- Multisets, 2, 36.
 combinations of, 2–3, 16–17, 25, 33.
 permutations of, 4, 14–15, 29, 30, 38.
- Near-perfect combination generation,
 11–17, 29.
- Near-perfect permutation generation, 15, 29.
- Nijenhuis, Albert, 8.
- Nowhere differentiable function, 32.
- Order ideal, 33.
- Organ-pipe order, 14.
- Partitions, 38.
- Paths on a grid, 2–3, 25.
- Payne, William Harris, 9, 28.
- Perfect combination generation, 15–17, 30.
- Permutations of multisets, 4, 14–15,
 29, 30, 38.
- Pi (π), 2, 13, 27, 28, 35.
- Piano, 10, 30.
- Plain changes, 10.
- Playing cards, 35.
- Poincaré, Henri, 35, 57.
- Polyhedron, 18, 33.
- Polynomial ideal, 34.
- Postorder traversal, 27.
- Preorder traversal, 27, 43.
- q -multinomial coefficients, 30.
- q -nomial coefficients, 15, 37.
- Rademacher, Hans, functions, 32.
- Rank, 39, 43.
- Read, Ronald Cedric, 16, 46.
- Recurrences, 26, 40–42.
- Recursion, 10.
 versus iteration, 12–14, 29.
- Recursive coroutines, 16.
- Redheffer, Raymond Moos, 50.
- Reflected Gray path, 28.
- Regular solids, 33.
- Reingold, Edward Martin (רינולד מילטן), 8.
- Reiss, Michel, 56–57.
- Replacement selection sorting, 39.
- Reversion of power series, 39.

- Revolving door property, 8, 29.
 scheme Γ_{st} , 8–10, 16–17, 27–29.
- Robinson, Robert William, 57.
- Root of unity, 30.
- Row-echelon form, 37.
- Rucksack filling, 7, 27.
- Ruskey, Frank, 30.
- Ruzsa, Imre Zoltán, 52.
- Savage, Carla Diane, 46.
- Schützenberger, Marcel Paul, 19, 50, 55.
- Shadows, 18–25, 31–34.
 of binary strings, 35.
 of subcubes, 34.
- Shields, Ian Beaumont, 46.
- Sibling links, 27.
- Sideways sum, 20.
- Simões Pereira, José Manuel dos Santos, 38.
- Simplexes, 18.
- Simplicial complexes, 33, 55.
- Simplicial multicomplexes, 34.
- Size vectors, 33, 34.
- Spread set in a torus, 22–24, 33.
- Stachowiak, Grzegorz, 46.
- Standard set in a torus, 22–24, 33.
- Stanford GraphBase, ii, iii.
- Stanley, Richard Peter, 14.
- Subcubes, 31, 34.
- Swapping with the first element, 16–17, 30.
- Takagi, Teiji (高木貞治), 20, 51.
 function, 20–21, 32–33.
- Tang, Donald Tao-Nan (唐道南), 8.
- Tarry, Gaston, 57.
- Ternary strings, 28, 56.
- Terquem, Olry, 35.
- Tokushige, Norihide (徳重典英), 52.
- Topological sorting, 46.
- Török, Éva, 45.
- Torus, n -dimensional, 20–24, 33.
- Tree of losers, 39.
- Triangles, 20.
- Triangulation, 37.
- Trie, 9.
- Unit vectors, 22.
- Universal cycles of combinations, 35.
- Upper shadow, 18.
- van Zanten, Arend Jan, 40.
- Vector spaces, 26, 31.
- Walsh, Timothy Robert Stephen, 9, 47.
- Wang, Da-Lun (王大倫), 20, 22.
- Wang, Ping Yang (王平, née 楊平), 20, 22.
- Whipple, Francis John Welsh, 22.
- Wiedemann, Douglas Henry, 30.
- Wilf, Herbert Saul, 8, 38.
- z -nomial coefficient, 15, 37.
- Zanten, Arend Jan van, 40.



FUNCTIONAL DIFFERENTIAL GEOMETRY

Gerald Jay Sussman and Jack Wisdom
with Will Farr

Functional Differential Geometry

Functional Differential Geometry

Gerald Jay Sussman and Jack Wisdom
with Will Farr

The MIT Press
Cambridge, Massachusetts
London, England

© 2013 Massachusetts Institute of Technology

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit creativecommons.org.



Other than as provided by this license, no part of this book may be reproduced, transmitted, or displayed by any electronic or mechanical means without permission from the MIT Press or as permitted by law.

MIT Press books may be purchased at special quantity discounts for business or sales promotional use. For information, please email special_sales@mitpress.mit.edu or write to Special Sales Department, The MIT Press, 55 Hayward Street, Cambridge, MA 02142.

This book was set in Computer Modern by the authors with the L^AT_EX typesetting system and was printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Sussman, Gerald Jay.

Functional Differential Geometry / Gerald Jay Sussman and Jack Wisdom; with Will Farr.

p. cm.

Includes bibliographical references and index.

ISBN 978-0-262-01934-7 (hardcover : alk. paper)

1. Geometry, Differential. 2. Functional Differential Equations.

3. Mathematical Physics.

I. Wisdom, Jack. II. Farr, Will. III. Title.

QC20.7.D52S87 2013

516.3'6—dc23

2012042107

10 9 8 7 6 5 4 3 2 1

“The author has spared himself no pains in his endeavour to present the main ideas in the simplest and most intelligible form, and on the whole, in the sequence and connection in which they actually originated. In the interest of clearness, it appeared to me inevitable that I should repeat myself frequently, without paying the slightest attention to the elegance of the presentation. I adhered scrupulously to the precept of that brilliant theoretical physicist L. Boltzmann, according to whom matters of elegance ought be left to the tailor and to the cobbler.”

Albert Einstein, in *Relativity, the Special and General Theory*, (1961), p. v

Contents

Preface	xi
Prologue	xv
1 Introduction	1
2 Manifolds	11
2.1 Coordinate Functions	12
2.2 Manifold Functions	14
3 Vector Fields and One-Form Fields	21
3.1 Vector Fields	21
3.2 Coordinate-Basis Vector Fields	26
3.3 Integral Curves	29
3.4 One-Form Fields	32
3.5 Coordinate-Basis One-Form Fields	34
4 Basis Fields	41
4.1 Change of Basis	44
4.2 Rotation Basis	47
4.3 Commutators	48
5 Integration	55
5.1 Higher Dimensions	57
5.2 Exterior Derivative	62
5.3 Stokes's Theorem	65

5.4	Vector Integral Theorems	67
6	Over a Map	71
6.1	Vector Fields Over a Map	71
6.2	One-Form Fields Over a Map	73
6.3	Basis Fields Over a Map	74
6.4	Pullbacks and Pushforwards	76
7	Directional Derivatives	83
7.1	Lie Derivative	85
7.2	Covariant Derivative	93
7.3	Parallel Transport	104
7.4	Geodesic Motion	111
8	Curvature	115
8.1	Explicit Transport	116
8.2	Torsion	124
8.3	Geodesic Deviation	125
8.4	Bianchi Identities	129
9	Metrics	133
9.1	Metric Compatibility	135
9.2	Metrics and Lagrange Equations	137
9.3	General Relativity	144
10	Hodge Star and Electrodynamics	153
10.1	The Wave Equation	159
10.2	Electrodynamics	160
11	Special Relativity	167
11.1	Lorentz Transformations	172
11.2	Special Relativity Frames	179

11.3	Twin Paradox	181
A	Scheme	185
B	Our Notation	195
C	Tensors	211
	References	217
	Index	219

Preface

Learning physics is hard. Part of the problem is that physics is naturally expressed in mathematical language. When we teach we use the language of mathematics in the same way that we use our natural language. We depend upon a vast amount of shared knowledge and culture, and we only sketch an idea using mathematical idioms. We are insufficiently precise to convey an idea to a person who does not share our culture. Our problem is that since we share the culture we find it difficult to notice that what we say is too imprecise to be clearly understood by a student new to the subject. A student must simultaneously learn the mathematical language and the content that is expressed in that language. This is like trying to read *Les Misérables* while struggling with French grammar.

This book is an effort to ameliorate this problem for learning the differential geometry needed as a foundation for a deep understanding of general relativity or quantum field theory. Our approach differs from the traditional one in several ways. Our coverage is unusual. We do not prove the general Stokes's Theorem—this is well covered in many other books—instead, we show how it works in two dimensions. Because our target is relativity, we put lots of emphasis on the development of the covariant derivative, and we erect a common context for understanding both the Lie derivative and the covariant derivative. Most treatments of differential geometry aimed at relativity assume that there is a metric (or pseudometric). By contrast, we develop as much material as possible independent of the assumption of a metric. This allows us to see what results depend on the metric when we introduce it. We also try to avoid the use of traditional index notation for tensors. Although one can become very adept at “index gymnastics,” that leads to much mindless (though useful) manipulation without much thought to meaning. Instead, we use a semantically richer language of vector fields and differential forms.

But the single biggest difference between our treatment and others is that we integrate computer programming into our explanations. By programming a computer to interpret our formulas we soon learn whether or not a formula is correct. If a formula is not clear, it will not be interpretable. If it is wrong, we will get a wrong answer. In either case we are led to improve our

program and as a result improve our understanding. We have been teaching advanced classical mechanics at MIT for many years using this strategy. We use precise functional notation and we have students program in a functional language. The students enjoy this approach and we have learned a lot ourselves. It is the experience of writing software for expressing the mathematical content and the insights that we gain from doing it that we feel is revolutionary. We want others to have a similar experience.

Acknowledgments

We thank the people who helped us develop this material, and especially the students who have over the years worked through the material with us. In particular, Mark Tobenkin, William Throwe, Leo Stein, Peter Iannucci, and Micah Brodsky have suffered through bad explanations and have contributed better ones.

Edmund Bertschinger, Norman Margolus, Tom Knight, Rebecca Frankel, Alexey Radul, Edwin Taylor, Joel Moses, Kenneth Yip, and Hal Abelson helped us with many thoughtful discussions and advice about physics and its relation to mathematics.

We also thank Chris Hanson, Taylor Campbell, and the community of Scheme programmers for providing support and advice for the elegant language that we use. In particular, Gerald Jay Sussman wants to thank Guy Lewis Steele and Alexey Radul for many fun days of programming together—we learned much from each other’s style.

Matthew Halfant started us on the development of the Scmutils system. He encouraged us to get into scientific computation, using Scheme and functional style as an active way to explain the ideas, without the distractions of imperative languages such as C. In the 1980s he wrote some of the early Scheme procedures for numerical computation that we still use.

Dan Zuras helped us with the invention of the unique organization of the Scmutils system. It is because of his insight that the system is organized around a generic extension of the chain rule for taking derivatives. He also helped in the heavy lifting that was required to make a really good polynomial GCD algorithm, based on ideas we learned from Richard Zippel.

A special contribution that cannot be sufficiently acknowledged is from Seymour Papert and Marvin Minsky, who taught us that

the practice of programming is a powerful way to develop a deeper understanding of any subject. Indeed, by the act of debugging we learn about our misconceptions, and by reflecting on our bugs and their resolutions we learn ways to learn more effectively. Indeed, *Turtle Geometry* [2], a beautiful book about discrete differential geometry at a more elementary level, was inspired by Papert's work on education. [13]

We acknowledge the generous support of the Computer Science and Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. The laboratory provides a stimulating environment for efforts to formalize knowledge with computational methods. We also acknowledge the Panasonic Corporation (formerly the Matsushita Electric Industrial Corporation) for support of Gerald Jay Sussman through an endowed chair.

Jack Wisdom thanks his wife, Cecile, for her love and support. Julie Sussman, PPA, provided careful reading and serious criticism that inspired us to reorganize and rewrite major parts of the text. She has also developed and maintained Gerald Jay Sussman over these many years.

Gerald Jay Sussman & Jack Wisdom
Cambridge, Massachusetts, USA
August 2012

Prologue

Programming and Understanding

One way to become aware of the precision required to unambiguously communicate a mathematical idea is to program it for a computer. Rather than using canned programs purely as an aid to visualization or numerical computation, we use computer programming in a functional style to encourage clear thinking. Programming forces us to be precise and unambiguous, without forcing us to be excessively rigorous. The computer does not tolerate vague descriptions or incomplete constructions. Thus the act of programming makes us keenly aware of our errors of reasoning or unsupported conclusions.¹

Although this book is about differential geometry, we can show how thinking about programming can help in understanding in a more elementary context. The traditional use of Leibniz's notation and Newton's notation is convenient in simple situations, but in more complicated situations it can be a serious handicap to clear reasoning.

A mechanical system is described by a Lagrangian function of the system state (time, coordinates, and velocities). A motion of the system is described by a path that gives the coordinates for each moment of time. A path is allowed if and only if it satisfies the Lagrange equations. Traditionally, the Lagrange equations are written

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = 0.$$

What could this expression possibly mean?

Let's try to write a program that implements Lagrange equations. What are Lagrange equations for? Our program must take a proposed path and give a result that allows us to decide if the path is allowed. This is already a problem; the equation shown above does not have a slot for a path to be tested.

¹The idea of using computer programming to develop skills of clear thinking was originally advocated by Seymour Papert. An extensive discussion of this idea, applied to the education of young children, can be found in Papert [13].

So we have to figure out how to insert the path to be tested. The partial derivatives do not depend on the path; they are derivatives of the Lagrangian function and thus they are functions with the same arguments as the Lagrangian. But the time derivative d/dt makes sense only for a function of time. Thus we must be intending to substitute the path (a function of time) and its derivative (also a function of time) into the coordinate and velocity arguments of the partial derivative functions.

So probably we meant something like the following (assume that w is a path through the coordinate configuration space, and so $w(t)$ specifies the configuration coordinates at time t):

$$\frac{d}{dt} \left(\frac{\partial L(t, q, \dot{q})}{\partial \dot{q}} \Bigg| \begin{array}{l} q = w(t) \\ \dot{q} = \frac{dw(t)}{dt} \end{array} \right) - \frac{\partial L(t, q, \dot{q})}{\partial q} \Bigg| \begin{array}{l} q = w(t) \\ \dot{q} = \frac{dw(t)}{dt} \end{array} = 0.$$

In this equation we see that the partial derivatives of the Lagrangian function are taken, then the path and its derivative are substituted for the position and velocity arguments of the Lagrangian, resulting in an expression in terms of the time.

This equation is complete. It has meaning independent of the context and there is nothing left to the imagination. The earlier equations require the reader to fill in lots of detail that is implicit in the context. They do not have a clear meaning independent of the context.

By thinking computationally we have reformulated the Lagrange equations into a form that is explicit enough to specify a computation. We could convert it into a program for any symbolic manipulation program because it tells us *how* to manipulate expressions to compute the residuals of Lagrange's equations for a purported solution path.²

²The *residuals* of equations are the expressions whose value must be zero if the equations are satisfied. For example, if we know that for an unknown x , $x^3 - x = 0$ then the residual is $x^3 - x$. We can try $x = -1$ and find a residual of 0, indicating that our purported solution satisfies the equation. A residual may provide information. For example, if we have the differential equation $df(x)/dx - af(x) = 0$ and we plug in a test solution $f(x) = Ae^{bx}$ we obtain the residual $(b - a)Ae^{bx}$, which can be zero only if $b = a$.

Functional Abstraction

But this corrected use of Leibniz notation is ugly. We had to introduce extraneous symbols (q and \dot{q}) in order to indicate the argument position specifying the partial derivative. Nothing would change here if we replaced q and \dot{q} by a and b .³ We can simplify the notation by admitting that the partial derivatives of the Lagrangian are themselves new functions, and by specifying the particular partial derivative by the position of the argument that is varied

$$\frac{d}{dt}((\partial_2 L)(t, w(t), \frac{d}{dt}w(t))) - (\partial_1 L)(t, w(t), \frac{d}{dt}w(t)) = 0,$$

where $\partial_i L$ is the function which is the partial derivative of the function L with respect to the i th argument.⁴

Two different notions of derivative appear in this expression. The functions $\partial_2 L$ and $\partial_1 L$, constructed from the Lagrangian L , have the same arguments as L . The derivative d/dt is an expression derivative. It applies to an expression that involves the variable t and it gives the rate of change of the value of the expression as the value of the variable t is varied.

These are both useful interpretations of the idea of a derivative. But functions give us more power. There are many equivalent ways to write expressions that compute the same value. For example $1/(1/r_1 + 1/r_2) = (r_1 r_2)/(r_1 + r_2)$. These expressions compute the same function of the two variables r_1 and r_2 . The first expression fails if $r_1 = 0$ but the second one gives the right value of the function. If we abstract the function, say as $\Pi(r_1, r_2)$, we can ignore the details of how it is computed. The ideas become clearer because they do not depend on the detailed shape of the expressions.

³That the symbols q and \dot{q} can be replaced by other arbitrarily chosen non-conflicting symbols without changing the meaning of the expression tells us that the partial derivative symbol is a logical quantifier, like forall and exists (\forall and \exists).

⁴The argument positions of the Lagrangian are indicated by indices starting with zero for the time argument.

So let's get rid of the expression derivative d/dt and replace it with an appropriate functional derivative. If f is a function then we will write Df as the new function that is the derivative of f :⁵

$$(Df)(t) = \frac{d}{dx} f(x) \Big|_{x=t}.$$

To do this for the Lagrange equation we need to construct a function to take the derivative of.

Given a configuration-space path w , there is a standard way to make the state-space path. We can abstract this method as a mathematical function Γ :

$$\Gamma[w](t) = (t, w(t), \frac{d}{dt}w(t)).$$

Using Γ we can write:

$$\frac{d}{dt}((\partial_2 L)(\Gamma[w](t))) - (\partial_1 L)(\Gamma[w](t)) = 0.$$

If we now define composition of functions $(f \circ g)(x) = f(g(x))$, we can express the Lagrange equations entirely in terms of functions:

$$D((\partial_2 L) \circ (\Gamma[w])) - (\partial_1 L) \circ (\Gamma[w]) = 0.$$

The functions $\partial_1 L$ and $\partial_2 L$ are partial derivatives of the function L . Composition with $\Gamma[w]$ evaluates these partials with coordinates and velocities appropriate for the path w , making functions of time. Applying D takes the time derivative. The Lagrange equation states that the difference of the resulting functions of time must be zero. This statement of the Lagrange equation is complete, unambiguous, and functional. It is not encumbered with the particular choices made in expressing the Lagrangian. For example, it doesn't matter if the time is named t or τ , and it has an explicit place for the path to be tested.

This expression is equivalent to a computer program:⁶

⁵An explanation of functional derivatives is in Appendix B, page 202.

⁶The programs in this book are written in Scheme, a dialect of Lisp. The details of the language are not germane to the points being made. What is important is that it is mechanically interpretable, and thus unambiguous. In this book we require that the mathematical expressions be explicit enough

```
(define ((Lagrange-equations Lagrangian) w)
  (- (D (compose ((partial 2) Lagrangian) (Gamma w)))
      (compose ((partial 1) Lagrangian) (Gamma w))))
```

In the Lagrange equations procedure the parameter `Lagrangian` is a procedure that implements the Lagrangian. The derivatives of the Lagrangian, for example `((partial 2) Lagrangian)`, are also procedures. The state-space path procedure `(Gamma w)` is constructed from the configuration-space path procedure `w` by the procedure `Gamma`:

```
(define ((Gamma w) t)
  (up t (w t) ((D w) t)))
```

where `up` is a constructor for a data structure that represents a state of the dynamical system (time, coordinates, velocities).

The result of applying the `Lagrange-equations` procedure to a procedure `Lagrangian` that implements a Lagrangian function is a procedure that takes a configuration-space path procedure `w` and returns a procedure that gives the residual of the Lagrange equations for that path at a time.

For example, consider the harmonic oscillator, with Lagrangian

$$L(t, q, v) = \frac{1}{2}mv^2 - \frac{1}{2}kq^2,$$

for mass m and spring constant k . This Lagrangian is implemented by

```
(define ((L-harmonic m k) local)
  (let ((q (coordinate local))
        (v (velocity local)))
    (- (* 1/2 m (square v))
       (* 1/2 k (square q)))))
```

We know that the motion of a harmonic oscillator is a sinusoid with a given amplitude a , frequency ω , and phase φ :

$$x(t) = a \cos(\omega t + \varphi).$$

that they can be expressed as computer programs. Scheme is chosen because it is easy to write programs that manipulate representations of mathematical functions. An informal description of Scheme can be found in Appendix A. The use of Scheme to represent mathematical objects can be found in Appendix B. A formal description of Scheme can be obtained in [10]. You can get the software from [21].

Suppose we have forgotten how the constants in the solution relate to the physical parameters of the oscillator. Let's plug in the proposed solution and look at the residual:

```
(define (proposed-solution t)
  (* 'a (cos (+ (* 'omega t) 'phi))))  
  
(show-expression
  (((Lagrange-equations (L-harmonic 'm 'k))
    proposed-solution)
   't))
```

$$\cos(\omega t + \varphi) a (k - m\omega^2)$$

The residual here shows that for nonzero amplitude, the only solutions allowed are ones where $(k - m\omega^2) = 0$ or $\omega = \sqrt{k/m}$.

But, suppose we had no idea what the solution looks like. We could propose a literal function for the path:

```
(show-expression
  (((Lagrange-equations (L-harmonic 'm 'k))
    (literal-function 'x))
   't))
```

$$kx(t) + mD^2x(t)$$

If this residual is zero we have the Lagrange equation for the harmonic oscillator.

Note that we can flexibly manipulate representations of mathematical functions. (See Appendices A and B.)

We started out thinking that the original statement of Lagrange's equations accurately captured the idea. But we really don't know until we try to teach it to a naive student. If the student is sufficiently ignorant, but is willing to ask questions, we are led to clarify the equations in the way that we did. There is no dumber but more insistent student than a computer. A computer will absolutely refuse to accept a partial statement, with missing parameters or a type error. In fact, the original statement of Lagrange's equations contained an obvious type error: the Lagrangian is a function of multiple variables, but the d/dt is applicable only to functions of one variable.

1

Introduction

Philosophy is written in that great book which ever lies before our eyes—I mean the Universe—but we cannot understand it if we do not learn the language and grasp the symbols in which it is written. This book is written in the mathematical language, and the symbols are triangles, circles, and other geometrical figures without whose help it is impossible to comprehend a single word of it, without which one wanders in vain through a dark labyrinth.

Galileo Galilei [8]

Differential geometry is a mathematical language that can be used to express physical concepts. In this introduction we show a typical use of this language. Do not panic! At this point we do not expect you to understand the details of what we are showing. All will be explained as needed in the text. The purpose is to get the flavor of this material.

At the North Pole inscribe a line in the ice perpendicular to the Greenwich Meridian. Hold a stick parallel to that line and walk down the Greenwich Meridian keeping the stick parallel to itself as you walk. (The phrase “parallel to itself” is a way of saying that as you walk you keep its orientation unchanged. The stick will be aligned East-West, perpendicular to your direction of travel.) When you get to the Equator the stick will be parallel to the Equator. Turn East, and walk along the Equator, keeping the stick parallel to the Equator. Continue walking until you get to the 90°E meridian. When you reach the 90°E meridian turn North and walk back to the North Pole keeping the stick parallel to itself. Note that the stick is perpendicular to your direction of travel. When you get to the Pole note that the stick is perpendicular to the line you inscribed in the ice. But you started with that stick parallel to that line and you kept the stick pointing in the same direction on the Earth throughout your walk—how did it change orientation?

The answer is that you walked a closed loop on a curved surface. As seen in three dimensions the stick was actually turning as you walked along the Equator, because you always kept the stick parallel to the curving surface of the Earth. But as a denizen of a 2-dimensional surface, it seemed to you that you kept the stick parallel to itself as you walked, even when making a turn. Even if you had no idea that the surface of the Earth was embedded in a 3-dimensional space you could use this experiment to conclude that the Earth was not flat. This is a small example of intrinsic geometry. It shows that the idea of parallel transport is not simple. For a general surface it is necessary to explicitly define what we mean by parallel.

If you walked a smaller loop, the angle between the starting orientation and the ending orientation of the stick would be smaller. For small loops it would be proportional to the area of the loop you walked. This constant of proportionality is a measure of the curvature. The result does not depend on how fast you walked, so this is not a dynamical phenomenon.

Denizens of the surface may play ball games. The balls are constrained to the surface; otherwise they are free particles. The paths of the balls are governed by dynamical laws. This motion is a solution of the Euler-Lagrange equations¹ for the free-particle Lagrangian with coordinates that incorporate the constraint of living in the surface. There are coefficients of terms in the Euler-Lagrange equations that arise naturally in the description of the behavior of the stick when walking loops on the surface, connecting the static shape of the surface with the dynamical behavior of the balls. It turns out that the dynamical evolution of the balls may be viewed as parallel transport of the ball's velocity vector in the direction of the velocity vector. This motion by parallel transport of the velocity is called *geodesic motion*.

So there are deep connections between the dynamics of particles and the geometry of the space that the particles move in. If we understand this connection we can learn about dynamics by studying geometry and we can learn about geometry by studying dynamics. We enter dynamics with a Lagrangian and the associated Lagrange equations. Although this formulation exposes many important features of the system, such as how symmetries relate to

¹It is customary to shorten “Euler-Lagrange equations” to “Lagrange equations.” We hope Leonhard Euler is not disturbed.

conserved quantities, the geometry is not apparent. But when we express the Lagrangian and the Lagrange equations in differential geometry language, geometric properties become apparent. In the case of systems with no potential energy the Euler-Lagrange equations are equivalent to the geodesic equations on the configuration manifold. In fact, the coefficients of terms in the Lagrange equations are Christoffel coefficients, which define parallel transport on the manifold. Let's look into this a bit.

Lagrange Equations

We write the Lagrange equations in functional notation² as follows:

$$D(\partial_2 L \circ \Gamma[q]) - \partial_1 L \circ \Gamma[q] = 0.$$

In SICM [19], Section 1.6.3, we showed that a Lagrangian describing the free motion of a particle subject to a coordinate-dependent constraint can be obtained by composing a free-particle Lagrangian with a function that describes how dynamical states transform given the coordinate transformation that describes the constraints.

A Lagrangian for a free particle of mass m and velocity v is just its kinetic energy, $mv^2/2$. The procedure `Lfree` implements the free Lagrangian:³

```
(define ((Lfree mass) state)
  (* 1/2 mass (square (velocity state))))
```

For us the dynamical state of a system of particles is a tuple of time, coordinates, and velocities. The free-particle Lagrangian depends only on the velocity part of the state.

For motion of a point constrained to move on the surface of a sphere the configuration space has two dimensions. We can describe the position of the point with the generalized coordinates colatitude and longitude. If the sphere is embedded in 3-dimensional space the position of the point in that space can be

²A short introduction to our functional notation, and why we have chosen it, is given in the prologue: Programming and Understanding. More details can be found in Appendix B.

³An informal description of the Scheme programming language can be found in Appendix A.

given by a coordinate transformation from colatitude and longitude to three rectangular coordinates.

For a sphere of radius R the procedure `sphere->R3` implements the transformation of coordinates from colatitude θ and longitude ϕ on the surface of the sphere to rectangular coordinates in the embedding space. (The \hat{z} axis goes through the North Pole, and the Equator is in the plane $z = 0$.)

```
(define ((sphere->R3 R) state)
  (let ((q (coordinate state)))
    (let ((theta (ref q 0)) (phi (ref q 1)))
      (up (* R (sin theta)) (cos phi)) ; x
      (* R (sin theta)) (sin phi)) ; y
      (* R (cos theta)))) ; z
```

The coordinate transformation maps the generalized coordinates on the sphere to the 3-dimensional rectangular coordinates. Given this coordinate transformation we construct a corresponding transformation of velocities; these make up the state transformation. The procedure `F->C` implements the derivation of a transformation of states from a coordinate transformation:

```
(define ((F->C F) state)
  (up (time state)
       (F state)
       (+ (((partial 0) F) state)
           (* (((partial 1) F) state)
               (velocity state)))))
```

A Lagrangian governing free motion on a sphere of radius R is then the composition of the free Lagrangian with the transformation of states.

```
(define (Lsphere m R)
  (compose (Lfree m) (F->C (sphere->R3 R))))
```

So the value of the Lagrangian at an arbitrary dynamical state is:

```
((Lsphere 'm 'R)
  (up 't (up 'theta 'phi) (up 'thetadot 'phidot)))
  (+ (* 1/2 m (expt R 2) (expt thetadot 2))
      (* 1/2 m (expt R 2) (expt (sin theta) 2) (expt phidot 2))))
```

or, in infix notation:

$$\frac{1}{2}mR^2\dot{\theta}^2 + \frac{1}{2}mR^2(\sin(\theta))^2\dot{\phi}^2. \quad (1.1)$$

The Metric

Let's now take a step into the geometry. A surface has a metric which tells us how to measure sizes and angles at every point on the surface. (Metrics are introduced in Chapter 9.)

The metric is a symmetric function of two vector fields that gives a number for every point on the manifold. (Vector fields are introduced in Chapter 3). Metrics may be used to compute the length of a vector field at each point, or alternatively to compute the inner product of two vector fields at each point. For example, the metric for the sphere of radius R is

$$g(u, v) = R^2 d\theta(u)d\theta(v) + R^2(\sin\theta)^2 d\phi(u)d\phi(v), \quad (1.2)$$

where u and v are vector fields, and $d\theta$ and $d\phi$ are one-form fields that extract the named components of the vector-field argument. (One-form fields are introduced in Chapter 3.) We can think of $d\theta(u)$ as a function of a point that gives the size of the vector field u in the θ direction at the point. Notice that $g(u, u)$ is a weighted sum of the squares of the components of u . In fact, if we identify

$$\begin{aligned} d\theta(v) &= \dot{\theta} \\ d\phi(v) &= \dot{\phi}, \end{aligned}$$

then the coefficients in the metric are the same as the coefficients in the value of the Lagrangian, equation (1.1), apart from a factor of $m/2$.

We can generalize this result and write a Lagrangian for free motion of a particle of mass m on a manifold with metric g :

$$L_2(x, v) = \sum_{ij} \frac{1}{2}mg_{ij}(x)v^iv^j. \quad (1.3)$$

This is written using indexed variables to indicate components of the geometric objects expressed with respect to an unspecified coordinate system. The metric coefficients g_{ij} are, in general, a

function of the position coordinates x , because the properties of the space may vary from place to place.

We can capture this geometric statement as a program:

```
(define ((L2 mass metric) place velocity)
  (* 1/2 mass ((metric velocity velocity) place)))
```

This program gives the Lagrangian in a coordinate-independent, geometric way. It is entirely in terms of geometric objects, such as a place on the configuration manifold, the velocity at that place, and the metric that describes the local shape of the manifold. But to compute we need a coordinate system. We express the dynamical state in terms of coordinates and velocity components in the coordinate system. For each coordinate system there is a natural vector basis and the geometric velocity vectors can be constructed by contracting the basis with the components of the velocity. Thus, we can form a coordinate representation of the Lagrangian.

```
(define ((Lc mass metric coordsys) state)
  (let ((x (coordinates state)) (v (velocities state))
        (e (coordinate-system->vector-basis coordsys)))
    ((L2 mass metric) ((point coordsys) x) (* e v))))
```

The manifold point m represented by the coordinates x is given by `(define m ((point coordsys) x))`. The coordinates of m in a different coordinate system are given by `((chart coordsys2) m)`. The manifold point m is a geometric object that is the same point independent of how it is specified. Similarly, the velocity vector ev is a geometric object, even though it is specified using components v with respect to the basis e . Both v and e have as many components as the dimension of the space so their product is interpreted as a contraction.

Let's make a general metric on a 2-dimensional real manifold:⁴

```
(define the-metric (literal-metric 'g R2-rect))
```

⁴The procedure `literal-metric` provides a metric. It is a general symmetric function of two vector fields, with literal functions of the coordinates of the manifold points for its coefficients in the given coordinate system. The quoted symbol `'g` is used to make the names of the literal coefficient functions. Literal functions are discussed in Appendix B.

The metric is expressed in rectangular coordinates, so the coordinate system is **R2-rect**.⁵ The component functions will be labeled as subscripted gs.

We can now make the Lagrangian for the system:

```
(define L (Lc 'm the-metric R2-rect))
```

And we can apply our Lagrangian to an arbitrary state:

```
(L (up 't (up 'x 'y) (up 'vx 'vy)))
(+ (* 1/2 m (g_00 (up x y)) (expt vx 2))
  (* m (g_01 (up x y)) vx vy)
  (* 1/2 m (g_11 (up x y)) (expt vy 2)))
```

Compare this result with equation (1.3).

Euler-Lagrange Residuals

The Euler-Lagrange equations are satisfied on realizable paths. Let γ be a path on the manifold of configurations. (A path is a map from the 1-dimensional real line to the configuration manifold. We introduce maps between manifolds in Chapter 6.) Consider an arbitrary path:⁶

```
(define gamma (literal-manifold-map 'q R1-rect R2-rect))
```

The values of γ are points on the manifold, not a coordinate representation of the points. We may evaluate **gamma** only on points of the real-line manifold; **gamma** produces points on the \mathbf{R}^2 manifold. So to go from the literal real-number coordinate '**t**' to a point on the real line we use **((point R1-rect) 't)** and to go from a point **m** in \mathbf{R}^2 to its coordinate representation we use **((chart R2-rect) m)**. (The procedures **point** and **chart** are introduced in Chapter 2.) Thus

⁵**R2-rect** is the usual rectangular coordinate system on the 2-dimensional real manifold. (See Section 2.1, page 13.) We supply common coordinate systems for n-dimensional real manifolds. For example, **R2-polar** is a polar coordinate system on the same manifold.

⁶The procedure **literal-manifold-map** makes a map from the manifold implied by its second argument to the manifold implied by the third argument. These arguments must be coordinate systems. The quoted symbol that is the first argument is used to name the literal coordinate functions that define the map.

```
((chart R2-rect) (gamma ((point R1-rect) 't)))
(up (q^0 t) (q^1 t))
```

So, to work with coordinates we write:

```
(define coordinate-path
  (compose (chart R2-rect) gamma (point R1-rect)))

(coordinate-path 't)
(up (q^0 t) (q^1 t))
```

Now we can compute the residuals of the Euler-Lagrange equations, but we get a large messy expression that we will not show.⁷ However, we will save it to compare with the residuals of the geodesic equations.

```
(define Lagrange-residuals
  (((Lagrange-equations L) coordinate-path) 't))
```

Geodesic Equations

Now we get deeper into the geometry. The traditional way to write the geodesic equations is

$$\nabla_v v = 0 \tag{1.4}$$

where ∇ is a covariant derivative operator. Roughly, $\nabla_v w$ is a directional derivative. It gives a measure of the variation of the vector field w as you walk along the manifold in the direction of v . (We will explain this in depth in Chapter 7.) $\nabla_v v = 0$ is intended to convey that the velocity vector is parallel-transported by itself. When you walked East on the Equator you had to hold the stick so that it was parallel to the Equator. But the stick is constrained to the surface of the Earth, so moving it along the Equator required turning it in three dimensions. The ∇ thus must incorporate the 3-dimensional shape of the Earth to provide a notion of “parallel” appropriate for the denizens of the surface of the Earth. This information will appear as the “Christoffel coefficients” in the coordinate representation of the geodesic equations.

The trouble with the traditional way to write the geodesic equations (1.4) is that the arguments to the covariant derivative are

⁷For an explanation of equation residuals see page xvi.

vector fields and the velocity along the path is not a vector field. A more precise way of stating this relation is:

$$\nabla_{\partial/\partial t}^\gamma d\gamma(\partial/\partial t) = 0. \quad (1.5)$$

(We know that this may be unfamiliar notation, but we will explain it in Chapter 7.)

In coordinates, the geodesic equations are expressed

$$D^2 q^i(t) + \sum_{jk} \Gamma_{jk}^i(\gamma(t)) Dq^j(t) Dq^k(t) = 0, \quad (1.6)$$

where $q(t)$ is the coordinate path corresponding to the manifold path γ , and $\Gamma_{jk}^i(m)$ are Christoffel coefficients. The $\Gamma_{jk}^i(m)$ describe the “shape” of the manifold close to the manifold point m . They can be derived from the metric g .

We can get and save the geodesic equation residuals by:

```
(define geodesic-equation-residuals
  (((((covariant-derivative Cartan gamma) d/dt)
    ((differential gamma) d/dt))
    (chart R2-rect))
   (point R1-rect) 't)))
```

where d/dt is a vector field on the real line⁸ and `Cartan` is a way of encapsulating the geometry, as specified by the Christoffel coefficients. The Christoffel coefficients are computed from the metric:

```
(define Cartan
  (Christoffel->Cartan
   (metric->Christoffel-2 the-metric
    (coordinate-system->basis R2-rect))))
```

The two messy residual results that we did not show are related by the metric. If we change the representation of the geodesic equations by “lowering” them using the mass and the metric, we see that the residuals are equal:

⁸We established `t` as a coordinate function on the rectangular coordinates of the real line by

```
(define-coordinates t R1-rect)
```

This had the effect of also defining `d/dt` as a coordinate vector field and `dt` as a one-form field on the real line.

```
(define metric-components
  (metric->components the-metric
    (coordinate-system->basis R2-rect)))

(- Lagrange-residuals
  (* (* 'm (metric-components (gamma ((point R1-rect) 't)))
        geodesic-equation-residuals))
  (down 0 0))
```

This establishes that for a 2-dimensional space the Euler-Lagrange equations are equivalent to the geodesic equations. The Christoffel coefficients that appear in the geodesic equation correspond to coefficients of terms in the Euler-Lagrange equations. This analysis will work for any number of dimensions (but will take your computer longer in higher dimensions, because the complexity increases).

Exercise 1.1: Motion on a Sphere

The metric for a unit sphere, expressed in colatitude θ and longitude ϕ , is

$$g(u, v) = d\theta(u)d\theta(v) + (\sin \theta)^2 d\phi(u)d\phi(v).$$

Compute the Lagrange equations for motion of a free particle on the sphere and convince yourself that they describe great circles. For example, consider motion on the equator ($\theta = \pi/2$) and motion on a line of longitude (ϕ is constant).

2

Manifolds

A *manifold* is a generalization of our idea of a smooth surface embedded in Euclidean space. For an n -dimensional manifold, around every point there is a simply-connected open set, the *coordinate patch*, and a one-to-one continuous function, the *coordinate function* or *chart*, mapping every point in that open set to a tuple of n real numbers, the *coordinates*. In general, several charts are needed to label all points on a manifold. It is required that if a region is in more than one coordinate patch then the coordinates are consistent in that the function mapping one set of coordinates to another is continuous (and perhaps differentiable to some degree). A consistent system of coordinate patches and coordinate functions that covers the entire manifold is called an *atlas*.

An example of a 2-dimensional manifold is the surface of a sphere or of a coffee cup. The space of all configurations of a planar double pendulum is a more abstract example of a 2-dimensional manifold. A manifold that looks locally Euclidean may not look like Euclidean space globally: for example, it may not be simply connected. The surface of the coffee cup is not simply connected, because there is a hole in the handle for your fingers.

An example of a coordinate function is the function that maps points in a simply-connected open neighborhood of the surface of a sphere to the tuple of latitude and longitude.¹ If we want to talk about motion on the Earth, we can identify the space of configurations to a 2-sphere (the surface of a 3-dimensional ball). The map from the 2-sphere to the 3-dimensional coordinates of a point on the surface of the Earth captures the shape of the Earth.

Two angles specify the configuration of the planar double pendulum. The manifold of configurations is a torus, where each point on the torus corresponds to a configuration of the double pendulum. The constraints, such as the lengths of the pendulum rods, are built into the map between the generalized coordi-

¹The open set for a latitude-longitude coordinate system cannot include either pole (because longitude is not defined at the poles) or the 180° meridian (where the longitude is discontinuous). Other coordinate systems are needed to cover these places.

nates of points on the torus and the arrangements of masses in 3-dimensional space.

There are computational objects that we can use to model manifolds. For example, we can make an object that represents the plane²

```
(define R2 (make-manifold R^n 2))
```

and give it the name `R2`. One useful patch of the plane is the one that contains the origin and covers the entire plane.³

```
(define U (patch 'origin R2))
```

2.1 Coordinate Functions

A coordinate function χ maps points in a coordinate patch of a manifold to a coordinate tuple:⁴

$$x = \chi(m), \tag{2.1}$$

where x may have a convenient tuple structure. Usually, the coordinates are arranged as an “up structure”; the coordinates are selected with superscripts:

$$x^i = \chi^i(m). \tag{2.2}$$

The number of independent components of x is the dimension of the manifold.

Assume we have two coordinate functions χ and χ' . The coordinate transformation from χ' coordinates to χ coordinates is just the composition $\chi \circ \chi'^{-1}$, where χ'^{-1} is the functional inverse of χ' (see figure 2.1). We assume that the coordinate transformation is continuous and differentiable to any degree we require.

² The expression `R^n` gives only one kind of manifold. We also have spheres `S^n` and `S03`.

³The word `origin` is an arbitrary symbol here. It labels a predefined patch in `R^n` manifolds.

⁴In the text that follows we will use sans-serif names, such as `f`, `v`, `m`, to refer to objects defined on the manifold. Objects that are defined on coordinates (tuples of real numbers) will be named with symbols like `f`, `v`, `x`.

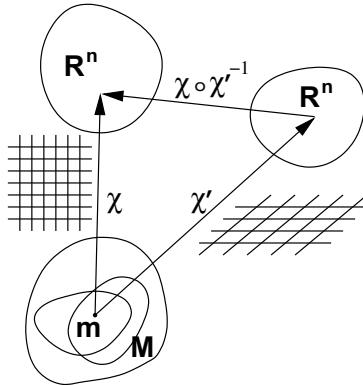


Figure 2.1 Here there are two overlapping coordinate patches that are the domains of the two coordinate functions χ and χ' . It is possible to represent manifold points in the overlap using either coordinate system. The coordinate transformation from χ' coordinates to χ coordinates is just the composition $\chi \circ \chi'^{-1}$.

Given a coordinate system `coordsys` for a patch on a manifold the procedure that implements the function χ that gives coordinates for a point is (`chart coordsys`). The procedure that implements the inverse map that gives a point for coordinates is (`point coordsys`).

We can have both rectangular and polar coordinates on a patch of the plane identified by the origin:^{5,6}

```
;; Some charts on the patch U
(define R2-rect (coordinate-system 'rectangular U))
(define R2-polar (coordinate-system 'polar/cylindrical U))
```

For each of the coordinate systems above we obtain the coordinate functions and their inverses:

⁵The rectangular coordinates are good for the entire plane, but the polar coordinates are singular at the origin because the angle is not defined. Also, the patch for polar coordinates must exclude one ray from the origin, because of the angle variable.

⁶We can avoid explicitly naming the patch:

```
(define R2-rect (coordinate-system-at 'rectangular 'origin R2))
```

```
(define R2-rect-chi (chart R2-rect))
(define R2-rect-chi-inverse (point R2-rect))
(define R2-polar-chi (chart R2-polar))
(define R2-polar-chi-inverse (point R2-polar))
```

The coordinate transformations are then just compositions. The polar coordinates of a rectangular point are:

```
((compose R2-polar-chi R2-rect-chi-inverse)
  (up 'x0 'y0)
  (up (sqrt (+ (expt x0 2) (expt y0 2))) (atan y0 x0)))
```

And the rectangular coordinates of a polar point are:

```
((compose R2-rect-chi R2-polar-chi-inverse)
  (up 'r0 'theta0))
  (up (* r0 (cos theta0)) (* r0 (sin theta0))))
```

And we can obtain the Jacobian of the polar-to-rectangular transformation by taking its derivative:⁷

```
((D (compose R2-rect-chi R2-polar-chi-inverse))
  (up 'r0 'theta0))
  (down (up (cos theta0) (sin theta0))
    (up (* -1 r0 (sin theta0)) (* r0 (cos theta0)))))
```

2.2 Manifold Functions

Let f be a real-valued function on a manifold M : this function maps points m on the manifold to real numbers.

This function has a coordinate representation f_χ with respect to the coordinate function χ (see figure 2.2):

$$f_\chi = f \circ \chi^{-1}. \quad (2.3)$$

Both the coordinate representation f_χ and the tuple x depend on the coordinate system, but the value $f_\chi(x)$ is independent of coordinates:

$$f_\chi(x) = (f \circ \chi^{-1})(\chi(m)) = f(m). \quad (2.4)$$

⁷See Appendix B for an introduction to tuple arithmetic and a discussion of derivatives of functions with structured input or output.

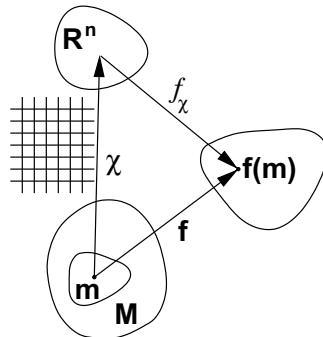


Figure 2.2 The coordinate function χ maps points on the manifold in the coordinate patch to a tuple of coordinates. A function f on the manifold M can be represented in coordinates by a function $f_\chi = f \circ \chi^{-1}$.

The subscript χ may be dropped when it is unambiguous.

For example, in a 2-dimensional real manifold the coordinates of a manifold point m are a pair of real numbers,

$$(x, y) = \chi(m), \quad (2.5)$$

and the manifold function f is represented in coordinates by a function f that takes a pair of real numbers and produces a real number

$$\begin{aligned} f : \mathbb{R}^2 &\rightarrow \mathbb{R} \\ f : (x, y) &\mapsto f(x, y). \end{aligned} \quad (2.6)$$

We define our manifold function

$$\begin{aligned} f : M &\rightarrow \mathbb{R} \\ f : m &\mapsto (f \circ \chi)(m). \end{aligned} \quad (2.7)$$

Manifold Functions Are Coordinate Independent

We can illustrate the coordinate independence with a program. We will show that an arbitrary manifold function f , when defined by its coordinate representation in rectangular coordinates, has the same behavior when applied to a manifold point independent of whether the point is specified in rectangular or polar coordinates.

We define a manifold function by specifying its behavior in rectangular coordinates:⁸

```
(define f
  (compose (literal-function 'f-rect R2->R) R2-rect-chi))
```

where `R2->R` is a signature for functions that map an up structure of two reals to a real:

```
(define R2->R (-> (UP Real Real) Real))
```

We can specify a typical manifold point using its rectangular coordinates:

```
(define R2-rect-point (R2-rect-chi-inverse (up 'x0 'y0)))
```

We can describe the *same point* using its polar coordinates:

```
(define corresponding-polar-point
  (R2-polar-chi-inverse
    (up (sqrt (+ (square 'x0) (square 'y0)))
         (atan 'y0 'x0))))
```

`(f R2-rect-point)` and `(f corresponding-polar-point)` agree, even though the point has been specified in two different coordinate systems:

```
(f R2-rect-point)
(f-rect (up x0 y0))

(f corresponding-polar-point)
(f-rect (up x0 y0))
```

Naming Coordinate Functions

To make things a bit easier, we can give names to the individual coordinate functions associated with a coordinate system. Here we name the coordinate functions for the `R2-rect` coordinate system `x` and `y` and for the `R2-polar` coordinate system `r` and `theta`.

```
(define-coordinates (up x y) R2-rect)
(define-coordinates (up r theta) R2-polar)
```

⁸Alternatively, we can define the same function in a shorthand

```
(define f (literal-manifold-function 'f-rect R2-rect))
```

This allows us to extract the coordinates from a point, independent of the coordinate system used to specify the point.

```
(x (R2-rect-chi-inverse (up 'x0 'y0)))
x0

(x (R2-polar-chi-inverse (up 'r0 'theta0)))
(* r0 (cos theta0))

(r (R2-polar-chi-inverse (up 'r0 'theta0)))
r0

(r (R2-rect-chi-inverse (up 'x0 'y0)))
(sqrt (+ (expt x0 2) (expt y0 2)))

(theta (R2-rect-chi-inverse (up 'x0 'y0)))
(atan y0 x0)
```

We can work with the coordinate functions in a natural manner, defining new manifold functions in terms of them:⁹

```
(define h (+ (* x (square r)) (cube y)))

(h R2-rect-point)
(+ (expt x0 3) (* x0 (expt y0 2))
  (expt y0 3))
```

We can also apply `h` to a point defined in terms of its polar coordinates:

```
(h (R2-polar-chi-inverse (up 'r0 'theta0)))
(+ (* (expt r0 3) (expt (sin theta0) 3))
  (* (expt r0 3) (cos theta0)))
```

Exercise 2.1: Curves

A curve may be specified in different coordinate systems. For example, a cardioid constructed by rolling a circle of radius a around another circle of the same radius is described in polar coordinates by the equation

$$r = 2a(1 + \cos(\theta)).$$

⁹This is actually a nasty, but traditional, abuse of notation. An expression like $\cos(r)$ can either mean the cosine of the angle r (if r is a number), or the composition $\cos \circ r$ (if r is a function). In our system `(cos r)` behaves in this way—either computing the cosine of r or being treated as `(compose cos r)` depending on what r is.

We can convert this to rectangular coordinates by evaluating the residual in rectangular coordinates.

```
(define-coordinates (up r theta) R2-polar)
((- r (* 2 'a (+ 1 (cos theta))))
 ((point R2-rect) (up 'x 'y)))
(/ (+ (* -2 a x)
      (* -2 a (sqrt (+ (expt x 2) (expt y 2))))
      (expt x 2) (expt y 2))
    (sqrt (+ (expt x 2) (expt y 2)))))
```

The numerator of this expression is the equivalent residual in rectangular coordinates. If we rearrange terms and square it we get the traditional formula for the cardioid

$$(x^2 + y^2 - 2ax)^2 = 4a^2(x^2 + y^2).$$

- a.** The rectangular coordinate equation for the Lemniscate of Bernoulli is

$$(x^2 + y^2)^2 = 2a^2(x^2 - y^2).$$

Find the expression in polar coordinates.

- b.** Describe a helix space curve in both rectangular and cylindrical coordinates. Use the computer to show the correspondence. Note that we provide a cylindrical coordinate system on the manifold \mathbf{R}^3 for you to use. It is called **R3-cyl**; with coordinates (**r**, **theta**, **z**).

Exercise 2.2: Stereographic Projection

A stereographic projection is a correspondence between points on the unit sphere and points on the plane cutting the sphere at its equator. (See figure 2.3.)

The coordinate system for points on the sphere in terms of rectangular coordinates of corresponding points on the plane is **S2-Riemann**.¹⁰ The procedure (**chart S2-Riemann**) gives the rectangular coordinates on the plane for every point on the sphere, except for the North Pole. The procedure (**point S2-Riemann**) gives the point on the sphere given rectangular coordinates on the plane. The usual spherical coordinate system on the sphere is **S2-spherical**.

We can compute the colatitude and longitude of a point on the sphere corresponding to a point on the plane with the following incantation:

¹⁰The plane with the addition of a point at infinity is conformally equivalent to the sphere by this correspondence. This correspondence is called the Riemann sphere, in honor of the great mathematician Bernard Riemann (1826–1866), who made major contributions to geometry.

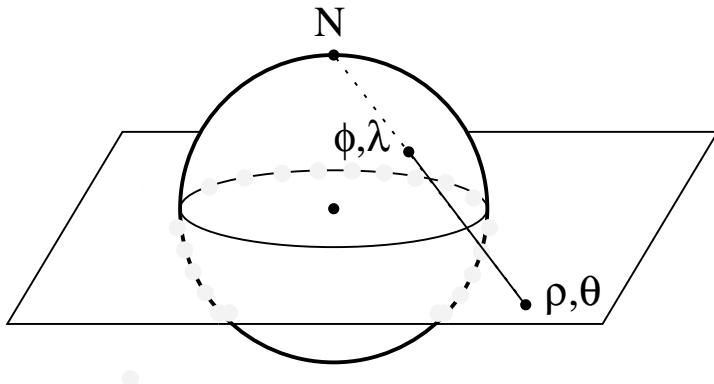


Figure 2.3 For each point on the sphere (except for its north pole) a line is drawn from the north pole through the point and extending to the equatorial plane. The corresponding point on the plane is where the line intersects the plane. The rectangular coordinates of this point on the plane are the Riemann coordinates of the point on the sphere. The points on the plane can also be specified with polar coordinates (ρ, θ) and the points on the sphere are specified both by Riemann coordinates and the traditional colatitude and longitude (ϕ, λ) .

```
((compose
  (chart S2-spherical)
  (point S2-Riemann)
  (chart R2-rect)
  (point R2-polar))
 (up 'rho 'theta))
 (up (/ (+ -1 (expt rho 2))
        (+ +1 (expt rho 2))))
      theta)
```

Perform an analogous computation to get the polar coordinates of the point on the plane corresponding to a point on the sphere given by its colatitude and longitude.

3

Vector Fields and One-Form Fields

We want a way to think about how a function varies on a manifold. Suppose we have some complex linkage, such as a multiple pendulum. The potential energy is an important function on the multi-dimensional configuration manifold of the linkage. To understand the dynamics of the linkage we need to know how the potential energy changes as the configuration changes. The change in potential energy for a step of a certain size in a particular direction in the configuration space is a real physical quantity; it does not depend on how we measure the direction or the step size. What exactly this means is to be determined: What is a step size? What is a direction? We cannot subtract two configurations to determine the distance between them. It is our job here to make sense of this idea.

So we would like something like a derivative, but there are problems. Since we cannot subtract two manifold points, we cannot take the derivative of a manifold function in the way described in elementary calculus. But we can take the derivative of a coordinate representation of a manifold function, because it takes real-number coordinates as its arguments. This is a start, but it is not independent of coordinate system. Let's see what we can build out of this.

3.1 Vector Fields

In multiple dimensions the derivative of a function is the multiplier for the best linear approximation of the function at each argument point:¹

$$f(x + \Delta x) \approx f(x) + (Df(x))\Delta x \quad (3.1)$$

The derivative $Df(x)$ is independent of Δx . Although the derivative depends on the coordinates, the product $(Df(x))\Delta x$ is in-

¹In multiple dimensions the derivative $Df(x)$ is a down tuple structure of the partial derivatives and the increment Δx is an up tuple structure, so the indicated product is to be interpreted as a contraction. (See equation B.8.)

variant under change of coordinates in the following sense. Let $\phi = \chi \circ \chi'^{-1}$ be a coordinate transformation, and $x = \phi(y)$. Then $\Delta x = D\phi(y)\Delta y$ is the linear approximation to the change in x when y changes by Δy . If f and g are the representations of a manifold function in the two coordinate systems, $g(y) = f(\phi(y)) = f(x)$, then the linear approximations to the increments in f and g are equal:

$$Dg(y)\Delta y = Df(\phi(y))(D\phi(y)\Delta y) = Df(x)\Delta x.$$

The invariant product $(Df(x))\Delta x$ is the *directional derivative* of f at x with respect to the vector specified by the tuple of components Δx in the coordinate system. We can generalize this idea to allow the vector at each point to depend on the point, making a *vector field*. Let b be a function of coordinates. We then have a directional derivative of f at each point x , determined by b

$$D_b(f)(x) = (Df(x))b(x). \quad (3.2)$$

Now we bring this back to the manifold and develop a useful generalization of the idea of directional derivative for functions on a manifold, rather than functions on \mathbb{R}^n . A *vector field on a manifold* is an assignment of a vector to each point on the manifold. In elementary geometry, a vector is an arrow anchored at a point on the manifold with a magnitude and a direction. In differential geometry, a vector is an operator that takes directional derivatives of manifold functions at its anchor point. The direction and magnitude of the vector are the direction and scale factor of the directional derivative.

Let m be a point on a manifold, v be a vector field on the manifold, and f be a real-valued function on the manifold. Then $v(f)$ is the directional derivative of the function f and $v(f)(m)$ is the directional derivative of the function f at the point m . The vector field is an operator that takes a real-valued manifold function and a manifold point and produces a number. The order of arguments is chosen to make $v(f)$ be a new manifold function that can be manipulated further. Directional derivative operators, unlike ordinary derivative operators, produce a result of the same type as their argument. Note that there is no mention here of any coordinate system. The vector field specifies a direction and magnitude at each manifold point that is independent of how it is described using any coordinate system.

A useful way to characterize a vector field in a particular coordinate system is by applying it to the coordinate functions. The resulting functions $b_{\chi,v}^i$ are called the *coordinate component functions* or *coefficient functions* of the vector field; they measure how quickly the coordinate functions change in the direction of the vector field, scaled by the magnitude of the vector field:

$$b_{\chi,v}^i = v(\chi^i) \circ \chi^{-1}. \quad (3.3)$$

Note that we have chosen the coordinate components to be functions of the coordinate tuple, not of a manifold point.

A vector with coordinate components $b_{\chi,v}$ applies to a manifold function f via

$$v(f)(m) = ((D(f \circ \chi^{-1}) b_{\chi,v}) \circ \chi)(m) \quad (3.4)$$

$$= D(f \circ \chi^{-1})(\chi(m)) b_{\chi,v}(\chi(m)) \quad (3.5)$$

$$= \sum_i \partial_i(f \circ \chi^{-1})(\chi(m)) b_{\chi,v}^i(\chi(m)). \quad (3.6)$$

In equation (3.4), the quantity $f \circ \chi^{-1}$ is the coordinate representation of the manifold function f . We take its derivative, and weight the components of the derivative with the coordinate components $b_{\chi,v}$ of the vector field that specify its direction and magnitude. Since this product is a function of coordinates we use χ to extract the coordinates from the manifold point m . In equation (3.5), the composition of the product with the coordinate chart χ is replaced by function evaluation. In equation (3.6) the tuple multiplication is expressed explicitly as a sum of products of corresponding components. So the application of the vector is a linear combination of the partial derivatives of f in the coordinate directions weighted by the vector components. This computes the rate of change of f in the direction specified by the vector.

Equations (3.3) and (3.5) are consistent:

$$\begin{aligned} v(\chi)(\chi^{-1}(x)) &= D(\chi \circ \chi^{-1})(x) b_{\chi,v}(x) \\ &= D(I)(x) b_{\chi,v}(x) \\ &= b_{\chi,v}(x). \end{aligned} \quad (3.7)$$

The coefficient tuple $b_{\chi,v}(x)$ is an up structure compatible for addition to the coordinates. Note that for any vector field v the coefficients $b_{\chi,v}(x)$ are different for different coordinate functions χ .

In the text that follows we will usually drop the subscripts on b , understanding that it is dependent on the coordinate system and the vector field.

We implement the definition of a vector field (3.4) as:

```
(define (components->vector-field components coordsys)
  (define (v f)
    (compose (* (D (compose f (point coordsys)))
               components)
            (chart coordsys)))
  (procedure->vector-field v))
```

The vector field is an operator, like derivative.²

Given a coordinate system and coefficient functions that map coordinates to real values, we can make a vector field. For example, a general vector field can be defined by giving components relative to the coordinate system R2-rect by

```
(define v
  (components->vector-field
    (up (literal-function 'b^0 R2->R)
        (literal-function 'b^1 R2->R))
  R2-rect))
```

To make it convenient to define literal vector fields we provide a shorthand: (define v (literal-vector-field 'b R2-rect)) This makes a vector field with component functions named b^0 and b^1 and names the result v . When this vector field is applied to an arbitrary manifold function it gives the directional derivative of that manifold function in the direction specified by the components b^0 and b^1 :

```
((v (literal-manifold-function 'f-rect R2-rect)) R2-rect-point)
(+ (* (((partial 0) f-rect) (up x0 y0)) (b^0 (up x0 y0)))
   (* (((partial 1) f-rect) (up x0 y0)) (b^1 (up x0 y0))))
```

This result is what we expect from equation (3.6).

We can recover the coordinate components of the vector field by applying the vector field to the coordinate chart:

²An operator is just like a procedure except that multiplication is interpreted as composition. For example, the derivative procedure is made into an operator D so that we can say (expt D 2) and expect it to compute the second derivative. The procedure `procedure->vector-field` makes a vector-field operator.

```
((v (chart R2-rect)) R2-rect-point)
(up (b^0 (up x y)) (b^1 (up x y)))
```

Coordinate Representation

The vector field v has a coordinate representation v :

$$\begin{aligned} v(f)(m) &= D(f \circ \chi^{-1})(\chi(m)) b(\chi(m)) \\ &= Df(x) b(x) \\ &= v(f)(x), \end{aligned} \tag{3.8}$$

with the definitions $f = f \circ \chi^{-1}$ and $x = \chi(m)$. The function b is the coefficient function for the vector field v . It provides a scale factor for the component in each coordinate direction. However, v is the coordinate representation of the vector field v in that it takes directional derivatives of coordinate representations of manifold functions.

Given a vector field v and a coordinate system `coordsys` we can construct the coordinate representation of the vector field.³

```
(define (coordinatize v coordsys)
  (define ((coordinatized-v f) x)
    (let ((b (compose (v (chart coordsys))
                      (point coordsys))))
      (* ((D f) x) (b x)))))
  (make-operator coordinatized-v))
```

We can apply a coordinatized vector field to a function of coordinates to get the same answer as before.

```
((coordinatize v R2-rect) (literal-function 'f-rect R2->R))
  (up 'x0 'y0)
  (+ (* (((partial 0) f-rect) (up x0 y0)) (b^0 (up x0 y0)))
     (* (((partial 1) f-rect) (up x0 y0)) (b^1 (up x0 y0))))
```

Vector Field Properties

The vector fields on a manifold form a vector space over the field of real numbers and a module over the ring of real-valued manifold functions. A module is like a vector space except that there is no multiplicative inverse operation on the scalars of a module. Manifold functions that are not the zero function do not necessarily

³The `make-operator` procedure takes a procedure and returns an operator.

have multiplicative inverses, because they can have isolated zeros. So the manifold functions form a ring, not a field, and vector fields must be a module over the ring of manifold functions rather than a vector space.

Vector fields have the following properties. Let u and v be vector fields and let α be a real-valued manifold function. Then

$$(u + v)(f) = u(f) + v(f) \quad (3.9)$$

$$(\alpha u)(f) = \alpha(u(f)). \quad (3.10)$$

Vector fields are linear operators. Assume f and g are functions on the manifold, a and b are real constants.⁴ The constants a and b are not manifold functions, because vector fields take derivatives. See equation (3.13).

$$v(af + bg)(m) = av(f)(m) + bv(g)(m) \quad (3.11)$$

$$v(af)(m) = av(f)(m) \quad (3.12)$$

Vector fields satisfy the product rule (Leibniz rule).

$$v(fg)(m) = v(f)(m)g(m) + f(m)v(g)(m) \quad (3.13)$$

Vector fields satisfy the chain rule. Let F be a function on the range of f .

$$v(F \circ f)(m) = DF(f(m))v(f)(m) \quad (3.14)$$

3.2 Coordinate-Basis Vector Fields

For an n -dimensional manifold any set of n linearly independent vector fields⁵ form a *basis* in that any vector field can be expressed as a linear combination of the basis fields with manifold-function

⁴If f has structured output then $v(f)$ is the structure resulting from v being applied to each component of f .

⁵A set of vector fields, $\{v_i\}$, is linearly independent with respect to manifold functions if we cannot find nonzero manifold functions, $\{a_i\}$, such that

$$\sum_i a_i v_i(f) = 0(f),$$

where 0 is the vector field such that $0(f)(m) = 0$ for all f and m .

coefficients. Given a coordinate system we can construct a basis as follows: we choose the component tuple $b_i(x)$ (see equation 3.5) to be the i th unit tuple $u_i(x)$ —an up tuple with one in the i th position and zeros in all other positions—selecting the partial derivative in that direction. Here u_i is a constant function. Like b , it formally takes coordinates of a point as an argument, but it ignores them. We then define the basis vector field X_i by

$$\begin{aligned} X_i(f)(m) &= D(f \circ \chi^{-1})(\chi(m)) u_i(\chi(m)) \\ &= \partial_i(f \circ \chi^{-1})(\chi(m)). \end{aligned} \quad (3.15)$$

In terms of X_i the vector field of equation (3.6) is

$$v(f)(m) = \sum_i X_i(f)(m) b^i(\chi(m)). \quad (3.16)$$

We can also write

$$v(f)(m) = X(f)(m) b(\chi(m)), \quad (3.17)$$

letting the tuple algebra do its job.

The basis vector field is often written

$$\frac{\partial}{\partial x^i} = X_i, \quad (3.18)$$

to call to mind that it is an operator that computes the directional derivative in the i th coordinate direction.

In addition to making the coordinate functions, the procedure **define-coordinates** also makes the traditional named basis vectors. Using these we can examine the application of a rectangular basis vector to a polar coordinate function:

```
(define-coordinates (up x y) R2-rect)
(define-coordinates (up r theta) R2-polar)

((d/dx (square r)) R2-rect-point)
(* 2 x0)
```

More general functions and vectors can be made as combinations of these simple pieces:

```
(((+ d/dx (* 2 d/dy)) (+ (square r) (* 3 x))) R2-rect-point)
(+ 3 (* 2 x0) (* 4 y0))
```

Coordinate Transformations

Consider a coordinate change from the chart χ to the chart χ' .

$$\begin{aligned} X(f)(m) &= D(f \circ \chi^{-1})(\chi(m)) \\ &= D(f \circ (\chi')^{-1} \circ \chi' \circ \chi^{-1})(\chi(m)) \\ &= D(f \circ (\chi')^{-1})(\chi'(m))(D(\chi' \circ \chi^{-1}))(\chi(m)) \\ &= X'(f)(m)(D(\chi' \circ \chi^{-1}))(\chi(m)). \end{aligned} \quad (3.19)$$

This is the rule for the transformation of basis vector fields. The second factor can be recognized as “ $\partial x'/\partial x$,” the Jacobian.⁶

The vector field does not depend on coordinates. So, from equation (3.17), we have

$$v(f)(m) = X(f)(m) b(\chi(m)) = X'(f)(m) b'(\chi'(m)). \quad (3.20)$$

Using equation (3.19) with $x = \chi(m)$ and $x' = \chi'(m)$, we deduce

$$D(\chi' \circ \chi^{-1})(x) b(x) = b'(x'). \quad (3.21)$$

Because $\chi' \circ \chi^{-1}$ is the inverse function of $\chi \circ (\chi')^{-1}$, their derivatives are multiplicative inverses,

$$D(\chi' \circ \chi^{-1})(x) = (D(\chi \circ (\chi')^{-1})(x'))^{-1}, \quad (3.22)$$

and so

$$b(x) = D(\chi \circ (\chi')^{-1})(x') b'(x'), \quad (3.23)$$

as expected.⁷

It is traditional to express this rule by saying that the basis elements transform *covariantly* and the coefficients of a vector in

⁶This notation helps one remember the transformation rule:

$$\frac{\partial f}{\partial x^i} = \sum_j \frac{\partial f}{\partial x'^j} \frac{\partial x'^j}{\partial x^i},$$

which is the relation in the usual Leibniz notation. As Spivak pointed out in *Calculus on Manifolds*, p.45, f means something different on each side of the equation.

⁷For coordinate paths q and q' related by $q(t) = (\chi \circ (\chi')^{-1})(q'(t))$ the velocities are related by $Dq(t) = D(\chi \circ (\chi')^{-1})(q'(t))Dq'(t)$. Abstracting off paths, we get $v = D(\chi \circ (\chi')^{-1})(x')v'$.

terms of a basis transform *contravariantly*; their product is invariant under the transformation.

3.3 Integral Curves

A vector field gives a direction and rate for every point on a manifold. We can start at any point and go in the direction specified by the vector field, tracing out a parametric curve on the manifold. This curve is an *integral curve* of the vector field.

More formally, let v be a vector field on the manifold M . An integral curve $\gamma_m^v : \mathbb{R} \rightarrow M$ of v is a parametric path on M satisfying

$$D(f \circ \gamma_m^v)(t) = v(f)(\gamma_m^v(t)) = (v(f) \circ \gamma_m^v)(t) \quad (3.24)$$

$$\gamma_m^v(0) = m, \quad (3.25)$$

for arbitrary functions f on the manifold, with real values or structured real values. The rate of change of a function along an integral curve is the vector field applied to the function evaluated at the appropriate place along the curve. Often we will simply write γ , rather than γ_m^v . Another useful variation is $\phi_t^v(m) = \gamma_m^v(t)$.

We can recover the differential equations satisfied by a coordinate representation of the integral curve by letting $f = \chi$, the coordinate function, and letting $\sigma = \chi \circ \gamma$ be the coordinate path corresponding to the curve γ . Then the derivative of the coordinate path σ is

$$\begin{aligned} D\sigma(t) &= D(\chi \circ \gamma)(t) \\ &= (v(\chi) \circ \gamma)(t) \\ &= (v(\chi) \circ \chi^{-1} \circ \chi \circ \gamma)(t) \\ &= (b \circ \sigma)(t), \end{aligned} \quad (3.26)$$

where $b = v(\chi) \circ \chi^{-1}$ is the coefficient function for the vector field v for coordinates χ (see equation 3.7). So the coordinate path σ satisfies the differential equations

$$D\sigma = b \circ \sigma. \quad (3.27)$$

Differential equations for the integral curve can be expressed only in a coordinate representation, because we cannot go from one point on the manifold to another by addition of an increment.

However, we can do this by adding the coordinates to an increment of coordinates and then finding the corresponding point on the manifold.

Iterating the process described by equation (3.24) we can compute higher-order derivatives of functions along the integral curve:

$$\begin{aligned} D(f \circ \gamma) &= v(f) \circ \gamma \\ D^2(f \circ \gamma) &= D(v(f) \circ \gamma) = v(v(f)) \circ \gamma \\ &\dots \\ D^n(f \circ \gamma) &= v^n(f) \circ \gamma \end{aligned} \tag{3.28}$$

Thus, the evolution of $f \circ \gamma$ can be written formally as a Taylor series in the parameter:

$$\begin{aligned} (f \circ \gamma)(t) &= (f \circ \gamma)(0) + t D(f \circ \gamma)(0) + \frac{1}{2} t^2 D^2(f \circ \gamma)(0) + \dots \\ &= (e^{tD}(f \circ \gamma))(0) \\ &= (e^{tv} f)(\gamma(0)). \end{aligned} \tag{3.29}$$

Using ϕ rather than γ

$$(f \circ \gamma_m^v)(t) = (f \circ \phi_t^v)(m), \tag{3.30}$$

so, when the series converges,

$$(e^{tv} f)(m) = (f \circ \phi_t^v)(m). \tag{3.31}$$

In particular, let $f = \chi$, then

$$\sigma(t) = (\chi \circ \gamma)(t) = (e^{tD}(\chi \circ \gamma))(0) = (e^{tv} \chi)(\gamma(0)), \tag{3.32}$$

a Taylor series representation of the solution to the differential equation (3.27).

For example, a vector field `circular` that generates a rotation about the origin is:⁸

⁸In this expression `d/dx` and `d/dy` are vector fields that take directional derivatives of manifold functions and evaluate them at manifold points; `x` and `y` are manifold functions. `define-coordinates` was used to create these operators and functions, see page 27.

Note that `circular` is an operator—a property inherited from `d/dx` and `d/dy`.

```
(define circular (- (* x d/dy) (* y d/dx)))
```

We can exponentiate the `circular` vector field, to generate an evolution in a circle around the origin starting at (1, 0):

```
(series:for-each print-expression
  (((exp (* 't circular)) (chart R2-rect))
    ((point R2-rect) (up 1 0)))
  6)
(up 1 0)
(up 0 t)
(up (* -1/2 (expt t 2)) 0)
(up 0 (* -1/6 (expt t 3)))
(up (* 1/24 (expt t 4)) 0)
(up 0 (* 1/120 (expt t 5)))
```

These are the first six terms of the series expansion of the coordinates of the position for parameter t .

We can define an evolution operator $E_{\Delta t, v}$ using equation (3.31)

$$(E_{\Delta t, v} f)(m) = (e^{\Delta t v} f)(m) = (f \circ \phi_{\Delta t}^v)(m). \quad (3.33)$$

We can approximate the evolution operator by summing the series up to a given order:

```
(define ((((evolution order) delta-t v) f) m)
  (series:sum
    (((exp (* delta-t v)) f) m)
    order))
```

We can evolve `circular` from the initial point up to the parameter t , and accumulate the first six terms as follows:

```
(((evolution 6) 'delta-t circular) (chart R2-rect))
 ((point R2-rect) (up 1 0)))
(up (+ (* -1/720 (expt delta-t 6))
        (* 1/24 (expt delta-t 4)))
      (* -1/2 (expt delta-t 2)))
  1)
(+ (* 1/120 (expt delta-t 5))
  (* -1/6 (expt delta-t 3))
  delta-t))
```

Note that these are just the series for $\cos \Delta t$ and $\sin \Delta t$, so the coordinate tuple of the evolved point is $(\cos \Delta t, \sin \Delta t)$.

For functions whose series expansions have finite radius of convergence, evolution can progress beyond the point at which the Taylor series converges because evolution is well defined whenever the integral curve is defined.

Exercise 3.1: State Derivatives

Newton's equations for the motion of a particle in a plane, subject to a force that depends only on the position in the plane, are a system of second-order differential equations for the rectangular coordinates (X, Y) of the particle:

$$D^2X(t) = A_x(X(t), Y(t)) \quad \text{and} \quad D^2Y(t) = A_y(X(t), Y(t)),$$

where A is the acceleration of the particle.

These are equivalent to a system of first-order equations for the coordinate path $\sigma = \chi \circ \gamma$, where $\chi = (t, x, y, v_x, v_y)$ is a coordinate system on the manifold \mathbf{R}^5 . Then our equations are:

$$\begin{aligned} D(t \circ \gamma) &= 1 \\ D(x \circ \gamma) &= v_x \circ \gamma \\ D(y \circ \gamma) &= v_y \circ \gamma \\ D(v_x \circ \gamma) &= A_x(x \circ \gamma, y \circ \gamma) \\ D(v_y \circ \gamma) &= A_y(x \circ \gamma, y \circ \gamma) \end{aligned}$$

Construct a vector field on \mathbf{R}^5 corresponding to this system of differential equations. Derive the first few terms in the series solution of this problem by exponentiation.

3.4 One-Form Fields

A vector field that gives a velocity for each point on a topographic map of the surface of the Earth can be applied to a function, such as one that gives the height for each point on the topographic map, or a map that gives the temperature for each point. The vector field then provides the rate of change of the height or temperature as one moves in the way described by the vector field. Alternatively, we can think of a topographic map, which gives the height at each point, as measuring a velocity field at each point. For example, we may be interested in the velocity of the wind or the trajectories of migrating birds. The topographic map gives the rate of change of height at each point for each velocity vector field. The rate of change of height can be thought of as the

number of equally-spaced (in height) contours that are pierced by each velocity vector in the vector field.

Differential of a Function

For example, consider the *differential*⁹ df of a manifold function f , defined as follows. If df is applied to a vector field v we obtain

$$df(v) = v(f), \quad (3.34)$$

which is a function of a manifold point.

The differential of the height function on the topographic map is a function that gives the rate of change of height at each point for a velocity vector field. This gives the same answer as the velocity vector field applied to the height function.

The differential of a function is linear in the vector fields. The differential is also a linear operator on functions: if f_1 and f_2 are manifold functions, and if c is a real constant, then

$$d(f_1 + f_2) = df_1 + df_2$$

and

$$d(cf) = cdf.$$

Note that c is not a manifold function.

One-Form Fields

A one-form field is a generalization of this idea; it is something that measures a vector field at each point.

One-form fields are linear functions of vector fields that produce real-valued functions on the manifold. A one-form field is linear in vector fields: if ω is a one-form field, v and w are vector fields, and c is a manifold function, then

$$\omega(v + w) = \omega(v) + \omega(w) \quad (3.35)$$

and

$$\omega(cv) = c\omega(v). \quad (3.36)$$

⁹The differential of a manifold function will turn out to be a special case of the exterior derivative, which will be introduced later.

Sums and scalar products of one-form fields on a manifold have the following properties. If ω and θ are one-form fields, and if f is a real-valued manifold function, then:

$$(\omega + \theta)(v) = \omega(v) + \theta(v), \quad (3.37)$$

$$(f\omega)(v) = f\omega(v). \quad (3.38)$$

3.5 Coordinate-Basis One-Form Fields

Given a coordinate function χ , we define the coordinate-basis one-form fields \tilde{X}^i by

$$\tilde{X}^i(v)(m) = v(\chi^i)(m) \quad (3.39)$$

or collectively

$$\tilde{X}(v)(m) = v(\chi)(m). \quad (3.40)$$

With this definition the coordinate-basis one-form fields are dual to the coordinate-basis vector fields in the following sense (see equation 3.15):¹⁰

$$\tilde{X}^i(X_j)(m) = X_j(\chi^i)(m) = \partial_j(\chi^i \circ \chi^{-1})(\chi(m)) = \delta_j^i. \quad (3.41)$$

The tuple of basis one-form fields $\tilde{X}(v)(m)$ is an up structure like that of χ .

The general one-form field ω is a linear combination of coordinate-basis one-form fields:

$$\omega(v)(m) = a(\chi(m)) \tilde{X}(v)(m) = \sum_i a_i(\chi(m)) \tilde{X}^i(v)(m), \quad (3.42)$$

with coefficient-function tuple $a(x)$, for $x = \chi(m)$. We can write this more simply as

$$\omega(v) = (a \circ \chi) \tilde{X}(v), \quad (3.43)$$

because everything is evaluated at m .

¹⁰The Kronecker delta δ_j^i is one if $i = j$ and zero otherwise.

The coefficient tuple can be recovered from the one-form field:¹¹

$$a_i(x) = \omega(X_i)(\chi^{-1}(x)). \quad (3.44)$$

This follows from the dual relationship (3.41). We can see this as a program:¹²

```
(define omega
  (components->1form-field
    (down (literal-function 'a_0 R2->R)
          (literal-function 'a_1 R2->R))
    R2-rect))

((omega (down d/dx d/dy)) R2-rect-point)
(down (a_0 (up x0 y0)) (a_1 (up x0 y0)))
```

We provide a shortcut for this construction:

```
(define omega (literal-1form-field 'a R2-rect))
```

A differential can be expanded in a coordinate basis:

$$df(v) = \sum_i c_i \tilde{X}^i(v). \quad (3.45)$$

The coefficients $c_i = df(X_i) = X_i(f) = \partial_i(f \circ \chi^{-1}) \circ \chi$ are the partial derivatives of the coordinate representation of f in the coordinate system of the basis:

```
((d (literal-manifold-function 'f-rect R2-rect))
  (coordinate-system->vector-basis R2-rect))
R2-rect-point
(down (((partial 0) f-rect) (up x0 y0))
  (((partial 1) f-rect) (up x0 y0)))
```

However, if the coordinate system of the basis differs from the coordinates of the representation of the function, the result is complicated by the chain rule:

¹¹The analogous recovery of coefficient tuples from vector fields is equation (3.3): $b_{\chi,v}^i = v(\chi^i) \circ \chi^{-1}$.

¹²The procedure `components->1form-field` is analogous to the procedure `components->vector-field` introduced earlier.

```
((d (literal-manifold-function 'f-polar R2-polar))
  (coordinate-system->vector-basis R2-rect))
((point R2-polar) (up 'r 'theta)))
(down (- (* (((partial 0) f-polar) (up r theta)) (cos theta))
            (/ (* (((partial 1) f-polar) (up r theta))
                  (sin theta))
               r))
      (+ (* (((partial 0) f-polar) (up r theta)) (sin theta))
         (/ (* (((partial 1) f-polar) (up r theta))
                (cos theta))
            r)))
```

The coordinate-basis one-form fields can be used to find the coefficients of vector fields in the corresponding coordinate vector-field basis:

$$\tilde{X}^i(v) = v(\chi^i) = b^i \circ \chi \quad (3.46)$$

or collectively,

$$\tilde{X}(v) = v(\chi) = b \circ \chi. \quad (3.47)$$

A coordinate-basis one-form field is often written dx^i . This traditional notation for the coordinate-basis one-form fields is justified by the relation:

$$dx^i = \tilde{X}^i = d(\chi^i). \quad (3.48)$$

The **define-coordinates** procedure also makes the basis one-form fields with these traditional names inherited from the coordinates.

We can illustrate the duality of the coordinate-basis vector fields and the coordinate-basis one-form fields:

```
(define-coordinates (up x y) R2-rect)
((dx d/dy) R2-rect-point)
  0
((dx d/dx) R2-rect-point)
  1
```

We can use the coordinate-basis one-form fields to extract the coefficients of **circular** on the rectangular vector basis:

```
((dx circular) R2-rect-point)
(* -1 y0)

((dy circular) R2-rect-point)
x0
```

But we can also find the coefficients on the polar vector basis:

```
((dr circular) R2-rect-point)
0

((dtheta circular) R2-rect-point)
1
```

So `circular` is the same as `d/dtheta`, as we can see by applying them both to the general function `f`:

```
(define f (literal-manifold-function 'f-rect R2-rect))
(((circular d/dtheta) f) R2-rect-point)
0
```

Not All One-Form Fields Are Differentials

Although all one-form fields can be constructed as linear combinations of basis one-form fields, not all one-form fields are differentials of functions.

The coefficients of a differential are (see equation 3.45):

$$c_i = X_i(f) = df(X_i) \quad (3.49)$$

and partial derivatives of functions commute

$$X_i(X_j(f)) = X_j(X_i(f)). \quad (3.50)$$

As a consequence, the coefficients of a differential are constrained

$$X_i(c_j) = X_j(c_i), \quad (3.51)$$

but a one-form field can be constructed with arbitrary coefficient functions. For example:

$$xdx + xdy \quad (3.52)$$

is not a differential of any function. This is why we started with the basis one-form fields and built the general one-form fields in terms of them.

Coordinate Transformations

Consider a coordinate change from the chart χ to the chart χ' .

$$\begin{aligned}\tilde{\mathbf{X}}(\mathbf{v}) &= \mathbf{v}(\chi) \\ &= \mathbf{v}(\chi \circ (\chi')^{-1} \circ \chi') \\ &= (D(\chi \circ (\chi')^{-1}) \circ \chi') \mathbf{v}(\chi') \\ &= (D(\chi \circ (\chi')^{-1}) \circ \chi') \tilde{\mathbf{X}}'(\mathbf{v}),\end{aligned}\tag{3.53}$$

where the third line follows from the chain rule for vector fields.

One-form fields are independent of coordinates. So,

$$\boldsymbol{\omega}(\mathbf{v}) = (a \circ \chi) \tilde{\mathbf{X}}(\mathbf{v}) = (a' \circ \chi') \tilde{\mathbf{X}}'(\mathbf{v}).\tag{3.54}$$

Eqs. (3.54) and (3.53) require that the coefficients transform under coordinate transformations as follows:

$$a(\chi(\mathbf{m})) D(\chi \circ (\chi')^{-1})(\chi'(\mathbf{m})) = a'(\chi'(\mathbf{m})),\tag{3.55}$$

or

$$a(\chi(\mathbf{m})) = a'(\chi'(\mathbf{m})) (D(\chi \circ (\chi')^{-1})(\chi'(\mathbf{m})))^{-1}.\tag{3.56}$$

The coefficient tuple $a(x)$ is a down structure compatible for contraction with $b(x)$. Let \mathbf{v} be the vector with coefficient tuple $b(x)$, and $\boldsymbol{\omega}$ be the one-form with coefficient tuple $a(x)$. Then, by equation (3.43),

$$\boldsymbol{\omega}(\mathbf{v}) = (a \circ \chi) (b \circ \chi).\tag{3.57}$$

As a program:

```
(define omega (literal-1form-field 'a R2-rect))

(define v (literal-vector-field 'b R2-rect))

((omega v) R2-rect-point)
(+ (* (b^0 (up x y)) (a_0 (up x0 y0)))
  (* (b^1 (up x y)) (a_1 (up x0 y0)))))
```

Comparing equation (3.56) with equation (3.23) we see that one-form components and vector components transform oppositely, so that

$$a(x) b(x) = a'(x') b'(x'),\tag{3.58}$$

as expected because $\boldsymbol{\omega}(\mathbf{v})(\mathbf{m})$ is independent of coordinates.

Exercise 3.2: Verification

Verify that the coefficients of a one-form field transform as described in equation (3.56). You should use equation (3.44) in your derivation.

Exercise 3.3: Hill Climbing

The topography of a region on the Earth can be specified by a manifold function h that gives the altitude at each point on the manifold. Let v be a vector field on the manifold, perhaps specifying a direction and rate of walking at every point on the manifold.

- a. Form an expression that gives the power that must be expended to follow the vector field at each point.
- b. Write this as a computational expression.

4

Basis Fields

A vector field may be written as a linear combination of basis vector fields. If n is the dimension, then any set of n linearly independent vector fields may be used as a basis. The coordinate basis \mathbf{X} is an example of a basis.¹ We will see later that not every basis is a coordinate basis: in order to be a coordinate basis, there must be a coordinate system such that each basis element is the directional derivative operator in a corresponding coordinate direction.

Let \mathbf{e} be a tuple of basis vector fields, such as the coordinate basis \mathbf{X} . The general vector field \mathbf{v} applied to an arbitrary manifold function f can be expressed as a linear combination

$$\mathbf{v}(f)(m) = \mathbf{e}(f)(m) \mathbf{b}(m) = \sum_i \mathbf{e}_i(f)(m) b^i(m), \quad (4.1)$$

where \mathbf{b} is a tuple-valued coefficient function on the manifold. When expressed in a coordinate basis, the coefficients that specify the direction of the vector are naturally expressed as functions b^i of the coordinates of the manifold point. Here, the coefficient function \mathbf{b} is more naturally expressed as a tuple-valued function on the manifold. If b is the coefficient function expressed as a function of coordinates, then $\mathbf{b} = b \circ \chi$ is the coefficient function as a function on the manifold.

The coordinate-basis forms have a simple definition in terms of the coordinate-basis vectors and the coordinates (equation 3.40). With this choice, the dual property, equation (3.41), holds without further fuss. More generally, we can define a basis of one-forms $\tilde{\mathbf{e}}$ that is dual to \mathbf{e} in that the property

$$\tilde{\mathbf{e}}^i(\mathbf{e}_j)(m) = \delta_j^i \quad (4.2)$$

is satisfied, analogous to property (3.41). Figure 4.1 illustrates the duality of basis fields.

¹We cannot say if the basis vectors are orthogonal or normalized until we introduce a metric.

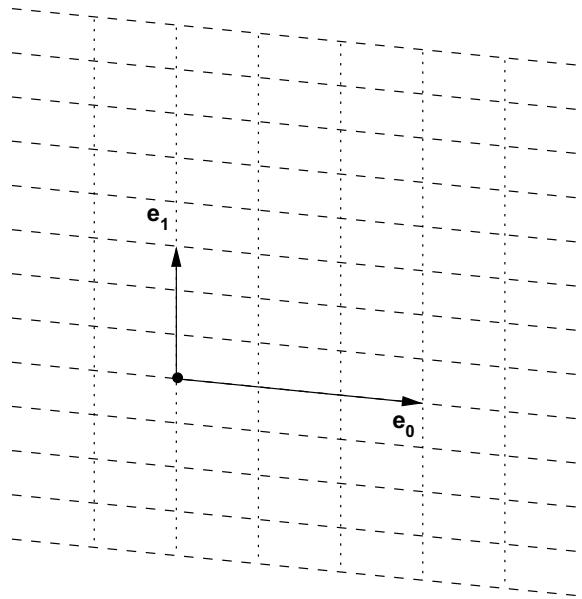


Figure 4.1 Let arrows e_0 and e_1 depict the vectors of a basis vector field at a particular point. Then the foliations shown by the parallel lines depict the dual basis one-form fields at that point. The dotted lines represent the field \tilde{e}^0 and the dashed lines represent the field \tilde{e}^1 . The spacings of the lines are $1/3$ unit. That the vectors pierce three of the lines representing their duals and do not pierce any of the lines representing the other basis elements is one way to see the relationship $\tilde{e}^i(e_j)(m) = \delta_j^i$.

To solve for the dual basis \tilde{e} given the basis e , we express the basis vectors e in terms of a coordinate basis²

$$e_j(f) = \sum_k X_k(f) c_j^k, \quad (4.3)$$

and the dual one-forms \tilde{e} in terms of the dual coordinate one-forms

$$\tilde{e}^i(v) = \sum_l d_l^i \tilde{X}^l(v), \quad (4.4)$$

²We write the vector components on the right and the tuple of basis vectors on the left because if we think of the basis vectors as organized as a row and the components as organized as a column then the formula is just a matrix multiplication.

then

$$\begin{aligned}
 \tilde{\epsilon}^i(\mathbf{e}_j) &= \sum_l \mathbf{d}_l^i \tilde{\mathbf{X}}^l(\mathbf{e}_j) \\
 &= \sum_l \mathbf{d}_l^i \mathbf{e}_j(\chi^l) \\
 &= \sum_l \mathbf{d}_l^i \sum_k \mathbf{X}_k(\chi^l) \mathbf{c}_j^k \\
 &= \sum_{kl} \mathbf{d}_l^i \delta_k^l \mathbf{c}_j^k \\
 &= \sum_k \mathbf{d}_k^i \mathbf{c}_j^k.
 \end{aligned} \tag{4.5}$$

Applying this at \mathbf{m} we get

$$\tilde{\epsilon}^i(\mathbf{e}_j)(\mathbf{m}) = \delta_j^i = \sum_k \mathbf{d}_k^i(\mathbf{m}) \mathbf{c}_j^k(\mathbf{m}). \tag{4.6}$$

So the \mathbf{d} coefficients can be determined from the \mathbf{c} coefficients (essentially by matrix inversion).

A set of vector fields $\{\mathbf{e}_i\}$ may be linearly independent in the sense that a weighted sum of them may not be identically zero over a region, yet it may not be a basis in that region. The problem is that there may be some places in the region where the vectors are not independent. For example, two of the vectors may be parallel at a point but not parallel elsewhere in the region. At such a point \mathbf{m} the determinant of the matrix $\mathbf{c}(\mathbf{m})$ is zero. So at these points we cannot define the dual basis forms.³

The dual form fields can be used to determine the coefficients \mathbf{b} of a vector field \mathbf{v} relative to a basis \mathbf{e} , by applying the dual basis form fields $\tilde{\epsilon}$ to the vector field. Let

$$\mathbf{v}(\mathbf{f}) = \sum_i \mathbf{e}_i(\mathbf{f}) \mathbf{b}^i. \tag{4.7}$$

Then

$$\tilde{\epsilon}^j(\mathbf{v}) = \mathbf{b}^j. \tag{4.8}$$

³This is why the set of vector fields and the set of one-form fields are modules rather than vector spaces.

Define two general vector fields:

```
(define e0
  (+ (* (literal-manifold-function 'e0x R2-rect) d/dx)
       (* (literal-manifold-function 'e0y R2-rect) d/dy)))

(define e1
  (+ (* (literal-manifold-function 'e1x R2-rect) d/dx)
       (* (literal-manifold-function 'e1y R2-rect) d/dy)))
```

We use these as a vector basis and compute the dual:

```
(define e-vector-basis (down e0 e1))
(define e-dual-basis
  (vector-basis->dual e-vector-basis R2-polar))
```

The procedure `vector-basis->dual` requires an auxiliary coordinate system (here `R2-polar`) to get the c_j^k coefficient functions from which we compute the d_k^i coefficient functions. However, the final result is independent of this coordinate system. Then we can verify that the bases e and \hat{e} satisfy the dual relationship (equation 3.41) by applying the dual basis to the vector basis:

```
((e-dual-basis e-vector-basis) R2-rect-point)
(up (down 1 0) (down 0 1)))
```

Note that the dual basis was computed relative to the polar coordinate system: the resulting objects are independent of the coordinates in which they were expressed!

Or we can make a general vector field with this basis and then pick out the coefficients by applying the dual basis:

```
(define v
  (* (up (literal-manifold-function 'b^0 R2-rect)
          (literal-manifold-function 'b^1 R2-rect))
     e-vector-basis))

((e-dual-basis v) R2-rect-point)
(up (b^0 (up x0 y0)) (b^1 (up x0 y0))))
```

4.1 Change of Basis

Suppose that we have a vector field v expressed in terms of one basis e and we want to reexpress it in terms of another basis e' . We have

$$v(f) = \sum_i e_i(f)b^i = \sum_j e'_j(f)b'^j. \quad (4.9)$$

The coefficients b' can be obtained from v by applying the dual basis

$$b'^j = \tilde{e}'^j(v) = \sum_i \tilde{e}'^j(e_i)b^i. \quad (4.10)$$

Let

$$J_i^j = \tilde{e}'^j(e_i), \quad (4.11)$$

then

$$b'^j = \sum_i J_i^j b^i, \quad (4.12)$$

and

$$e_i(f) = \sum_j e'_j(f) J_i^j. \quad (4.13)$$

The Jacobian J is a structure of manifold functions. Using tuple arithmetic, we can write

$$b' = Jb \quad (4.14)$$

and

$$e(f) = e'(f)J. \quad (4.15)$$

We can write

```
(define (Jacobian to-basis from-basis)
  (s:map/r (basis->1form-basis to-basis)
            (basis->vector-basis from-basis)))
```

These are the rectangular components of a vector field:

```
(define b-rect
  ((coordinate-system->1form-basis R2-rect)
   (literal-vector-field 'b R2-rect)))
```

The polar components are:

```
(define b-polar
  (* (Jacobian (coordinate-system->basis R2-polar)
                (coordinate-system->basis R2-rect))
     b-rect))

(b-polar ((point R2-rect) (up 'x0 'y0)))
(up
 (/ (+ (* x0 (b^0 (up x0 y0))) (* y0 (b^1 (up x0 y0))))
      (sqrt (+ (expt x0 2) (expt y0 2))))
 (/ (+ (* x0 (b^1 (up x0 y0))) (* -1 y0 (b^0 (up x0 y0))))
      (+ (expt x0 2) (expt y0 2)))))
```

We can also get the polar components directly:

```
((coordinate-system->1form-basis R2-polar)
 (literal-vector-field 'b R2-rect))
 ((point R2-rect) (up 'x0 'y0)))
(up
 (/ (+ (* x0 (b^0 (up x0 y0))) (* y0 (b^1 (up x0 y0))))
      (sqrt (+ (expt x0 2) (expt y0 2))))
 (/ (+ (* x0 (b^1 (up x0 y0))) (* -1 y0 (b^0 (up x0 y0))))
      (+ (expt x0 2) (expt y0 2)))))
```

We see that they are the same.

If K is the Jacobian that relates the basis vectors in the other direction

$$e'(f) = e(f)K \quad (4.16)$$

then

$$KJ = I = JK \quad (4.17)$$

where I is a manifold function that returns the multiplicative identity.

The dual basis transforms oppositely. Let

$$\omega = \sum_i a_i \tilde{e}^i = \sum_i a'_i \tilde{e}'^i. \quad (4.18)$$

The coefficients are⁴

$$\mathbf{a}_i = \boldsymbol{\omega}(\mathbf{e}_i) = \sum_j \mathbf{a}'_j \tilde{\mathbf{e}}'^j(\mathbf{e}_i) = \sum_j \mathbf{a}'_j \mathbf{J}^j{}_i \quad (4.19)$$

or, in tuple arithmetic,

$$\mathbf{a} = \mathbf{a}' \mathbf{J}. \quad (4.20)$$

Because of equation (4.18) we can deduce

$$\tilde{\mathbf{e}} = \mathbf{K} \tilde{\mathbf{e}}'. \quad (4.21)$$

4.2 Rotation Basis

One interesting basis for rotations in 3-dimensional space is not a coordinate basis.

Rotations are the actions of the special orthogonal group $\text{SO}(3)$, which is a 3-dimensional manifold. The elements of this group may be represented by the set of 3×3 orthogonal matrices with determinant +1.

We can use a coordinate patch on this manifold with Euler angle coordinates: each element has three coordinates, θ, ϕ, ψ . A manifold point may be represented by a rotation matrix. The rotation matrix for Euler angles is a product of three simple rotations: $M(\theta, \phi, \psi) = R_z(\phi)R_x(\theta)R_z(\psi)$, where R_x and R_z are functions that take an angle and produce the matrices representing rotations about the x and z axes, respectively. We can visualize θ as the colatitude of the pole from the \hat{z} -axis, ϕ as the longitude, and ψ as the rotation around the pole.

Given a rotation specified by Euler angles, how do we change the Euler angle to correspond to an incremental rotation of size ϵ about the \hat{x} -axis? The direction (a, b, c) is constrained by the equation

$$R_x(\epsilon)M(\theta, \phi, \psi) = M(\theta + a\epsilon, \phi + b\epsilon, \psi + c\epsilon). \quad (4.22)$$

⁴We see from equations (4.15) and (4.16) that \mathbf{J} and \mathbf{K} are inverses. We can obtain their coefficients by: $\mathbf{J}_i^j = \tilde{\mathbf{e}}'^j(\mathbf{e}_i)$ and $\mathbf{K}_i^j = \tilde{\mathbf{e}}^j(\mathbf{e}'_i)$.

Linear equations for (a, b, c) can be found by taking the derivative of this equation with respect to ϵ . We find

$$0 = c \cos \theta + b, \quad (4.23)$$

$$0 = a \sin \phi - c \cos \phi \sin \theta, \quad (4.24)$$

$$1 = c \sin \phi \sin \theta + a \cos \phi, \quad (4.25)$$

with the solution

$$a = \cos \phi, \quad (4.26)$$

$$b = -\frac{\sin \phi \cos \theta}{\sin \theta}, \quad (4.27)$$

$$c = \frac{\sin \phi}{\sin \theta}. \quad (4.28)$$

Therefore, we can write the basis vector field that takes directional derivatives in the direction of incremental x rotations as

$$\begin{aligned} \mathbf{e}_x &= a \frac{\partial}{\partial \theta} + b \frac{\partial}{\partial \phi} + c \frac{\partial}{\partial \psi} \\ &= \cos \phi \frac{\partial}{\partial \theta} - \frac{\sin \phi \cos \theta}{\sin \theta} \frac{\partial}{\partial \phi} + \frac{\sin \phi}{\sin \theta} \frac{\partial}{\partial \psi}. \end{aligned} \quad (4.29)$$

Similarly, vector fields for the incremental y and z rotations are

$$\mathbf{e}_y = \frac{\cos \phi \cos \theta}{\sin \theta} \frac{\partial}{\partial \phi} + \sin \phi \frac{\partial}{\partial \theta} - \frac{\cos \phi}{\sin \theta} \frac{\partial}{\partial \psi}, \quad (4.30)$$

$$\mathbf{e}_z = \frac{\partial}{\partial \phi}. \quad (4.31)$$

4.3 Commutators

The commutator of two vector fields is defined as

$$[\mathbf{v}, \mathbf{w}](\mathbf{f}) = \mathbf{v}(\mathbf{w}(\mathbf{f})) - \mathbf{w}(\mathbf{v}(\mathbf{f})). \quad (4.32)$$

In the special case that the two vector fields are coordinate basis fields, the commutator is zero:

$$\begin{aligned}
[X_i, X_j](f) &= X_i(X_j(f)) - X_j(X_i(f)) \\
&= \partial_i \partial_j (f \circ \chi^{-1}) \circ \chi - \partial_j \partial_i (f \circ \chi^{-1}) \circ \chi \\
&= 0,
\end{aligned} \tag{4.33}$$

because the individual partial derivatives commute. The vanishing commutator is telling us that we get to the same manifold point by integrating from a point along first one basis vector field and then another as from integrating in the other order. If the commutator is zero we can use the integral curves of the basis vector fields to form a coordinate mesh.

More generally, the commutator of two vector fields is a vector field. Let v be a vector field with coefficient function $c = c \circ \chi$, and u be a vector field with coefficient function $b = b \circ \chi$, both with respect to the coordinate basis X . Then

$$\begin{aligned}
[u, v](f) &= u(v(f)) - v(u(f)) \\
&= u\left(\sum_i X_i(f)c^i\right) - v\left(\sum_j X_j(f)b^j\right) \\
&= \sum_j X_j\left(\sum_i X_i(f)c^i\right)b^j - \sum_i X_i\left(\sum_j X_j(f)b^j\right)c^i \\
&= \sum_{ij} [X_j, X_i](f)c^i b^j \\
&\quad + \sum_i X_i(f) \sum_j (X_j(c^i)b^j - X_j(b^i)c^j) \\
&= \sum_i X_i(f)a^i,
\end{aligned} \tag{4.34}$$

where the coefficient function a of the commutator vector field is

$$\begin{aligned}
a^i &= \sum_j (X_j(c^i)b^j - X_j(b^i)c^j) \\
&= u(c^i) - v(b^i).
\end{aligned} \tag{4.35}$$

We used the fact, shown above, that the commutator of two coordinate basis fields is zero.

We can check this formula for the commutator for the general vector fields e_0 and e_1 in polar coordinates:

```
(let* ((polar-basis (coordinate-system->basis R2-polar))
      (polar-vector-basis (basis->vector-basis polar-basis))
      (polar-dual-basis (basis->1form-basis polar-basis))
      (f (literal-manifold-function 'f-rect R2-rect)))
  ((- ((commutator e0 e1) f)
    (* (- (e0 (polar-dual-basis e1))
          (e1 (polar-dual-basis e0)))
       (polar-vector-basis f)))
    R2-rect-point))
  0)
```

Let e be a tuple of basis vector fields. The commutator of two basis fields can be expressed in terms of the basis vector fields:

$$[e_i, e_j](f) = \sum_k d_{ij}^k e_k(f), \quad (4.36)$$

where d_{ij}^k are functions of m , called the *structure constants* for the basis vector fields. The coefficients are

$$d_{ij}^k = \tilde{e}^k([e_i, e_j]). \quad (4.37)$$

The commutator $[u, v]$ with respect to a non-coordinate basis e_i is

$$[u, v](f) = \sum_k e_k(f) \left(u(c^k) - v(b^k) + \sum_{ij} c^i b^j d_{ji}^k \right). \quad (4.38)$$

Define the vector fields J_x , J_y , and J_z that generate rotations about the three rectangular axes in three dimensions:⁵

```
(define Jz (- (* x d/dy) (* y d/dx)))
(define Jx (- (* y d/dz) (* z d/dy)))
(define Jy (- (* z d/dx) (* x d/dz)))
```

⁵Using

```
(define R3-rect (coordinate-system-at 'rectangular 'origin R3))
(define-coordinates (up x y z) R3-rect)
(define R3-rect-point ((point R3-rect) (up 'x0 'y0 'z0)))
(define g (literal-manifold-function 'g-rect R3-rect))
```

```
((+ (commutator Jx Jy) Jz) g) R3-rect-point)
0
((+ (commutator Jy Jz) Jx) g) R3-rect-point)
0
((+ (commutator Jz Jx) Jy) g) R3-rect-point)
0
```

We see that

$$\begin{aligned} [J_x, J_y] &= -J_z \\ [J_y, J_z] &= -J_x \\ [J_z, J_x] &= -J_y. \end{aligned} \tag{4.39}$$

We can also compute the commutators for the basis vector fields e_x , e_y , and e_z in the $SO(3)$ manifold (see equations 4.29–4.31) that correspond to rotations about the x , y , and z axes, respectively:⁶

```
((+ (commutator e_x e_y) e_z) f) S03-point)
0
((+ (commutator e_y e_z) e_x) f) S03-point)
0
((+ (commutator e_z e_x) e_y) f) S03-point)
0
```

You can tell if a set of basis vector fields is a coordinate basis by calculating the commutators. If they are nonzero, then the basis is not a coordinate basis. If they are zero then the basis vector fields can be integrated to give the coordinate system.

Recall equation (3.31)

$$(e^{tv}f)(m) = (f \circ \phi_t^v)(m). \tag{4.40}$$

Iterating this equation, we find

$$(e^{sw} e^{tv} f)(m) = (f \circ \phi_t^v \circ \phi_s^w)(m). \tag{4.41}$$

⁶Using

```
(define Euler-angles (coordinate-system-at 'Euler 'Euler-patch S03))
(define Euler-angles-chi-inverse (point Euler-angles))
(define-coordinates (up theta phi psi) Euler-angles)
(define S03-point ((point Euler-angles) (up 'theta 'phi 'psi)))
(define f (literal-manifold-function 'f-Euler Euler-angles))
```

Notice that the evolution under w occurs before the evolution under v .

To illustrate the meaning of the commutator, consider the evolution around a small loop with sides made from the integral curves of two vector fields v and w . We will first follow v , then w , then $-v$, and then $-w$:

$$(e^{\epsilon v} e^{\epsilon w} e^{-\epsilon v} e^{-\epsilon w} f)(m). \quad (4.42)$$

To second order in ϵ the result is⁷

$$(e^{\epsilon^2 [v,w]} f)(m). \quad (4.43)$$

This result is illustrated in figure 4.2.

Take a point 0 in M as the origin. Then, presuming $[e_i, e_j] = 0$, the coordinates x of the point m in the coordinate system corresponding to the e basis satisfy⁸

$$m = \phi_1^{xe}(0) = \chi^{-1}(x), \quad (4.44)$$

where χ is the coordinate function being defined. Because the elements of e commute, we can translate separately along the integral curves in any order and reach the same point; the terms in the exponential can be factored into separate exponentials if needed.

Exercise 4.1: Alternate Angles

Note that the Euler angles are singular at $\theta = 0$ (where ϕ and ψ become degenerate), so the representations of e_x , e_y , and e_z (defined in equa-

⁷ For non-commuting operators A and B ,

$$\begin{aligned} e^A e^B e^{-A} e^{-B} &= \left(1 + A + \frac{A^2}{2} + \dots\right) \left(1 + B + \frac{B^2}{2} + \dots\right) \\ &\quad \times \left(1 - A + \frac{A^2}{2} + \dots\right) \left(1 - B + \frac{B^2}{2} + \dots\right) \\ &= 1 + [A, B] + \dots, \end{aligned}$$

to second order in A and B . All higher-order terms can be written in terms of higher-order commutators of A and B . An example of a higher-order commutator is $[A, [A, B]]$.

⁸ Here x is an up-tuple structure of components, and e is down-tuple structure of basis vectors. The product of the two contracts to make a scaled vector, along which we translate by one unit.

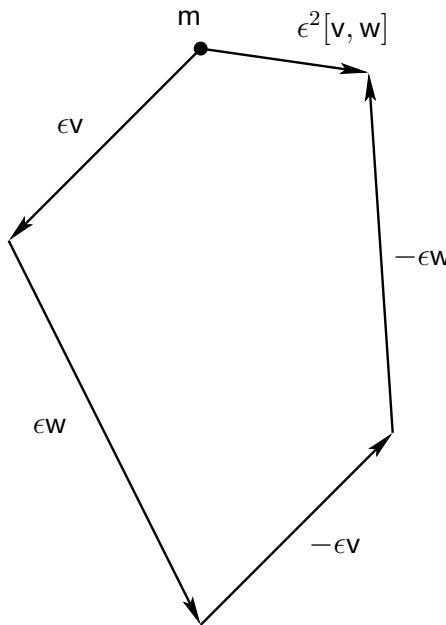


Figure 4.2 The commutator of two vector fields computes the residual of a small loop following their integral curves.

tions 4.29–4.31) have problems there. An alternate coordinate system avoids this problem, while introducing a similar problem elsewhere in the manifold.

Consider the “alternate angles” $(\theta_a, \phi_a, \psi_a)$ which define a rotation matrix via $M(\theta_a, \phi_a, \psi_a) = R_z(\phi_a) R_x(\theta_a) R_y(\psi_a)$.

- a. Where does the singularity appear in these alternate coordinates? Do you think you could define a coordinate system for rotations that has no singularities?
- b. What do the e_x , e_y , and e_z basis vector fields look like in this coordinate system?

Exercise 4.2: General Commutators

Verify equation (4.38).

Exercise 4.3: SO(3) Basis and Angular Momentum Basis

How are J_x , J_y , and J_z related to e_x , e_y , and e_z in equations (4.29–4.31)?

5

Integration

We know how to integrate real-valued functions of a real variable. We want to extend this idea to manifolds, in such a way that the integral is independent of the coordinate system used to compute it.

The integral of a real-valued function of a real variable is the limit of a sum of products of the values of the function on subintervals and the lengths of the increments of the independent variable in those subintervals:

$$\int_a^b f = \int_a^b f(x) dx = \lim_{\Delta x_i \rightarrow 0} \sum_i f(x_i) \Delta x_i. \quad (5.1)$$

If we change variables ($x = g(y)$), then the form of the integral changes:

$$\begin{aligned} \int_a^b f &= \int_a^b f(x) dx \\ &= \int_{g^{-1}(a)}^{g^{-1}(b)} f(g(y)) Dg(y) dy \\ &= \int_{g^{-1}(a)}^{g^{-1}(b)} (f \circ g) Dg. \end{aligned} \quad (5.2)$$

We can make a coordinate-independent notion of integration in the following way. An interval of the real line is a 1-dimensional manifold with boundary. We can assign a coordinate chart χ to this manifold. Let $x = \chi(m)$. The coordinate basis is associated with a coordinate-basis vector field, here $\partial/\partial x$. Let ω be a one-form on this manifold. The application of ω to $\partial/\partial x$ is a real-valued function on the manifold. If we compose this with the inverse chart, we get a real-valued function of a real variable. We can then write the usual integral of this function

$$I = \int_a^b \omega(\partial/\partial x) \circ \chi^{-1}. \quad (5.3)$$

It turns out that the value of this integral is independent of the coordinate chart used in its definition. Consider a different coordinate chart $x' = \chi'(m)$, with associated basis vector field $\partial/\partial x'$. Let $g = \chi' \circ \chi^{-1}$. We have

$$\begin{aligned}
& \int_{a'}^{b'} \omega(\partial/\partial x') \circ \chi'^{-1} \\
&= \int_{a'}^{b'} \omega(\partial/\partial x(D(\chi \circ \chi'^{-1}) \circ \chi')) \circ \chi'^{-1} \\
&= \int_{a'}^{b'} (\omega(\partial/\partial x) D(\chi \circ \chi'^{-1}) \circ \chi') \circ \chi'^{-1} \\
&= \int_{a'}^{b'} (\omega(\partial/\partial x) \circ \chi'^{-1}) D(\chi \circ \chi'^{-1}) \\
&= \int_a^b (((\omega(\partial/\partial x) \circ \chi^{-1}) D(\chi \circ \chi'^{-1})) \circ g) Dg \\
&= \int_a^b \omega(\partial/\partial x) \circ \chi^{-1},
\end{aligned} \tag{5.4}$$

where we have used the rule for coordinate transformations of basis vectors (equation 3.19), linearity of forms in the first two lines, and the rule for change-of-variables under an integral in the last line.¹

Because the integral is independent of the coordinate chart, we can write simply

$$I = \int_M \omega, \tag{5.5}$$

where M is the 1-dimensional manifold with boundary corresponding to the interval.

We are exploiting the fact that coordinate basis vectors in different coordinate systems are related by a Jacobian (see equation 3.19), which cancels the Jacobian that appears in the change-of-variables formula for integration (see equation 5.2).

¹ Note $(D(\chi \circ \chi'^{-1}) \circ (\chi' \circ \chi^{-1})) D(\chi' \circ \chi^{-1}) = 1$. With $g = \chi' \circ \chi^{-1}$ this is $(D(g^{-1}) \circ g)(Dg) = 1$.

5.1 Higher Dimensions

We have seen that we can integrate one-forms on 1-dimensional manifolds. We need higher-rank forms that we can integrate on higher-dimensional manifolds in a coordinate-independent manner.

Consider the integral of a real-valued function, $f : \mathbb{R}^n \rightarrow \mathbb{R}$, over a region U in \mathbb{R}^n . Under a coordinate transformation $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$, we have²

$$\int_U f = \int_{g^{-1}(U)} (f \circ g) \det(Dg). \quad (5.6)$$

A rank n form field takes n vector field arguments and produces a real-valued manifold function: $\omega(v, w, \dots, u)(m)$. By analogy with the 1-dimensional case, higher-rank forms are linear in each argument. Higher-rank forms must also be antisymmetric under interchange of any two arguments in order to make a coordinate-free definition of integration analogous to equation (5.3).

Consider an integral in the coordinate system χ :

$$\int_{\chi(U)} \omega(X_0, X_1, \dots) \circ \chi^{-1}. \quad (5.7)$$

Under coordinate transformations $g = \chi \circ \chi'^{-1}$, the integral becomes

$$\int_{\chi'(U)} \omega(X_0, X_1, \dots) \circ \chi'^{-1} \det(Dg). \quad (5.8)$$

Using the change-of-basis formula, equation (3.19):

$$X(f) = X'(f)(D(\chi' \circ \chi^{-1})) \circ \chi = X'(f)(D(g^{-1})) \circ \chi. \quad (5.9)$$

If we let $M = (D(g^{-1})) \circ \chi$ then

$$\begin{aligned} & (\omega(X_0, X_1, \dots) \circ \chi'^{-1}) \det(Dg) \\ &= (\omega(X'_0, X'_1, \dots) \circ \chi'^{-1}) \det(Dg) \\ &= (\omega(X'_0, X'_1, \dots) \circ \chi'^{-1}) \alpha(M_0, M_1, \dots) \det(Dg), \end{aligned} \quad (5.10)$$

²The *determinant* is the unique function of the rows of its argument that i) is linear in each row, ii) changes sign under any interchange of rows, and iii) is one when applied to the identity multiplier.

using the multilinearity of ω , where M_i is the i^{th} column of M . The function α is multilinear in the columns of M . To make a coordinate-independent integration we want the expression (5.10) to be the same as the integrand in

$$I' = \int_{X'(U)} \omega(X'_0, X'_1, \dots) \circ \chi'^{-1}. \quad (5.11)$$

For this to be the case, $\alpha(M_0, M_1, \dots)$ must be $(\det(Dg))^{-1} = \det(M)$. So α is an antisymmetric function, and thus so is ω .

Thus higher-rank form fields must be antisymmetric multilinear functions from vector fields to manifold functions. So we have a coordinate-independent definition of integration of form fields on a manifold and we can write

$$I = I' = \int_U \omega. \quad (5.12)$$

Wedge Product

There are several ways we can construct antisymmetric higher-rank forms. Given two one-form fields ω and τ we can form a two-form field $\omega \wedge \tau$ as follows:

$$(\omega \wedge \tau)(v, w) = \omega(v)\tau(w) - \omega(w)\tau(v). \quad (5.13)$$

More generally we can form the wedge of higher-rank forms. Let ω be a k -form field and τ be an l -form field. We can form a $(k+l)$ -form field $\omega \wedge \tau$ as follows:

$$\omega \wedge \tau = \frac{(k+l)!}{k! l!} \text{Alt}(\omega \otimes \tau) \quad (5.14)$$

where, if η is a function on m vectors,

$$\begin{aligned} \text{Alt}(\eta)(v_0, \dots, v_{m-1}) \\ = \frac{1}{m!} \sum_{\sigma \in \text{Perm}(m)} \text{Parity}(\sigma) \eta(v_{\sigma(0)}, \dots, v_{\sigma(m-1)}), \end{aligned} \quad (5.15)$$

and where

$$\begin{aligned} \omega \otimes \tau(v_0, \dots, v_{k-1}, v_k, \dots, v_{k+l-1}) \\ = \omega(v_0, \dots, v_{k-1}) \tau(v_k, \dots, v_{k+l-1}). \end{aligned} \quad (5.16)$$

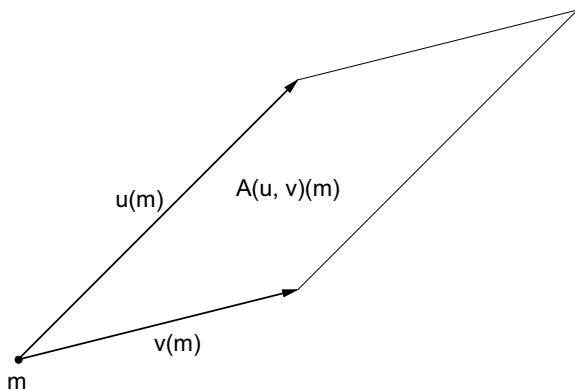


Figure 5.1 The area of the parallelogram in the (x, y) coordinate plane is given by $A(u, v)(m)$.

The wedge product is associative, and thus we need not specify the order of a multiple application. The factorial coefficients of these formulas are chosen so that

$$(dx \wedge dy \wedge \dots) (\partial/\partial x, \partial/\partial y, \dots) = 1. \quad (5.17)$$

This is true independent of the coordinate system.

Equation (5.17) gives us

$$\int_U dx \wedge dy \wedge \dots = \text{Volume}(U) \quad (5.18)$$

where $\text{Volume}(U)$ is the ordinary volume of the region corresponding to U in the Euclidean space of \mathbb{R}^n with the orthonormal coordinate system (x, y, \dots) .³

An example two-form (see figure 5.1) is the oriented area of a parallelogram in the (x, y) coordinate plane at the point m spanned by two vectors $u = u^0 \partial/\partial x + u^1 \partial/\partial y$ and $v = v^0 \partial/\partial x + v^1 \partial/\partial y$, which is given by

$$A(u, v)(m) = u^0(m)v^1(m) - v^0(m)u^1(m). \quad (5.19)$$

³By using the word “orthonormal” here we are assuming that the range of the coordinate chart is an ordinary Euclidean space with the usual Euclidean metric. The coordinate basis in that chart is orthonormal. Under these conditions we can usefully use words like “length,” “area,” and “volume” in the coordinate space.

Note that this is the area of the parallelogram in the coordinate plane, which is the range of the coordinate function. It is not the area on the manifold. To define that, we need more structure—the metric. We will put a metric on the manifold in Chapter 9.

3-Dimensional Euclidean Space

Let's specialize to 3-dimensional Euclidean space. Following equation (5.18) we can write the coordinate-area two-form in another way: $A = dx \wedge dy$. As code:

```
(define-coordinates (up x y z) R3-rect)

(define u (+ (* 'u^0 d/dx) (* 'u^1 d/dy)))
(define v (+ (* 'v^0 d/dx) (* 'v^1 d/dy)))

(((wedge dx dy) u v) R3-rect-point)
(+ (* u^0 v^1) (* -1 u^1 v^0))
```

If we use cylindrical coordinates and define cylindrical vector fields we get the analogous answer in cylindrical coordinates:

```
(define-coordinates (up r theta z) R3-cyl)

(define a (+ (* 'a^0 d/dr) (* 'a^1 d/dtheta)))
(define b (+ (* 'b^0 d/dr) (* 'b^1 d/dtheta)))

(((wedge dr dtheta) a b) ((point R3-cyl) (up 'r0 'theta0 'z0)))
(+ (* a^0 b^1) (* -1 a^1 b^0)))
```

The moral of this story is that this is the area of the parallelogram in the coordinate plane. It is not the area on the manifold!

There is a similar story with volumes. The wedge product of the elements of the coordinate basis is a three-form that measures our usual idea of coordinate volumes in R^3 with a Euclidean metric:

```
(define u (+ (* 'u^0 d/dx) (* 'u^1 d/dy) (* 'u^2 d/dz)))
(define v (+ (* 'v^0 d/dx) (* 'v^1 d/dy) (* 'v^2 d/dz)))
(define w (+ (* 'w^0 d/dx) (* 'w^1 d/dy) (* 'w^2 d/dz)))
```

```
((wedge dx dy dz) u v w) R3-rect-point)
(+ (* u^0 v^1 w^2)
  (* -1 u^0 v^2 w^1)
  (* -1 u^1 v^0 w^2)
  (* u^1 v^2 w^0)
  (* u^2 v^0 w^1)
  (* -1 u^2 v^1 w^0))
```

This last expression is the determinant of a 3×3 matrix:

```
(- (((wedge dx dy dz) u v w) R3-rect-point)
  (determinant
    (matrix-by-rows (list 'u^0 'u^1 'u^2)
                  (list 'v^0 'v^1 'v^2)
                  (list 'w^0 'w^1 'w^2))))
```

0

If we did the same operations in cylindrical coordinates we would get the analogous formula, showing that what we are computing is volume in the coordinate space, not volume on the manifold.

Because of antisymmetry, if the rank of a form is greater than the dimension of the manifold then the form is identically zero. The k -forms on an n -dimensional manifold form a module of dimension $\binom{n}{k}$. We can write a coordinate-basis expression for a k -form as

$$\omega = \sum_{i_0, \dots, i_{k-1}=0}^n \omega_{i_0, \dots, i_{k-1}} dx^{i_0} \wedge \dots \wedge dx^{i_{k-1}}. \quad (5.20)$$

The antisymmetry of the wedge product implies that

$$\omega_{i_{\sigma(0)}, \dots, i_{\sigma(k-1)}} = \text{Parity}(\sigma) \omega_{i_0, \dots, i_{k-1}}, \quad (5.21)$$

from which we see that there are only $\binom{n}{k}$ independent components of ω .

Exercise 5.1: Wedge Product

Pick a coordinate system and use the computer to verify that

- a. the wedge product is associative for forms in your coordinate system;
- b. formula (5.17) is true in your coordinate system.

5.2 Exterior Derivative

The intention of introducing the exterior derivative is to capture all of the classical theorems of “vector analysis” into one unified Stokes’s Theorem, which asserts that the integral of a form on the boundary of a manifold is the integral of the exterior derivative of the form on the interior of the manifold:⁴

$$\int_{\partial M} \omega = \int_M d\omega. \quad (5.22)$$

As we have seen in equation (3.34), the differential of a function on a manifold is a one-form field. If a function on a manifold is considered to be a form field of rank zero,⁵ then the differential operator increases the rank of the form by one. We can generalize this to k -form fields with the exterior derivative operation.

Consider a one-form ω . We define⁶

$$d\omega(v_1, v_2) = v_1(\omega(v_2)) - v_2(\omega(v_1)) - \omega([v_1, v_2]). \quad (5.23)$$

More generally, the exterior derivative of a k -form field is a $k+1$ -form field, given by:⁷

$$\begin{aligned} d\omega(v_0, \dots, v_k) &= \\ &\sum_{i=0}^k \left\{ ((-1)^i v_i(\omega(v_0, \dots, v_{i-1}, v_{i+1}, \dots, v_k)) + \right. \\ &\left. \sum_{j=i+1}^k (-1)^{i+j} \omega([v_i, v_j], v_0, \dots, v_{i-1}, v_{i+1}, \dots, v_{j-1}, v_{j+1}, \dots, v_k)) \right\}. \end{aligned} \quad (5.24)$$

This formula is coordinate-system independent. This is the way we compute the exterior derivative in our software.

⁴This is a generalization of the Fundamental Theorem of Calculus.

⁵A manifold function f induces a form field \hat{f} of rank 0 as follows:

$\hat{f}(\cdot)(m) = f(m).$

⁶The definition is chosen to make Stokes’s Theorem pretty.

⁷See Spivak, *Differential Geometry*, Volume 1, p.289.

If the form field ω is represented in a coordinate basis

$$\omega = \sum_{i_0=0, \dots, i_{k-1}=0}^{n-1} a_{i_0, \dots, i_{k-1}} dx^{i_0} \wedge \cdots \wedge dx^{i_{k-1}} \quad (5.25)$$

then the exterior derivative can be expressed as

$$d\omega = \sum_{i_0=0, \dots, i_{k-1}=0}^{n-1} da_{i_0, \dots, i_{k-1}} \wedge dx^{i_0} \wedge \cdots \wedge dx^{i_{k-1}}. \quad (5.26)$$

Though this formula is expressed in terms of a coordinate basis, the result is independent of the choice of coordinate system.

Computing Exterior Derivatives

We can test that the computation indicated by equation (5.24) is equivalent to the computation indicated by equation (5.26) in three dimensions with a general one-form field:

```
(define a (literal-manifold-function 'alpha R3-rect))
(define b (literal-manifold-function 'beta R3-rect))
(define c (literal-manifold-function 'gamma R3-rect))

(define theta (+ (* a dx) (* b dy) (* c dz)))
```

The test will require two arbitrary vector fields

```
(define X (literal-vector-field 'X-rect R3-rect))
(define Y (literal-vector-field 'Y-rect R3-rect))

(((d theta)
  (+ (wedge (d a) dx)
     (wedge (d b) dy)
     (wedge (d c) dz)))
 X Y)
 R3-rect-point)
0
```

We can also try a general two-form field in 3-dimensional space:
Let

$$\omega = ady \wedge dz + bdz \wedge dx + cdx \wedge dy, \quad (5.27)$$

where $a = \alpha \circ \chi$, $b = \beta \circ \chi$, $c = \gamma \circ \chi$, and α , β , and γ are real-valued functions of three real arguments. As a program,

```
(define omega
  (+ (* a (wedge dy dz))
    (* b (wedge dz dx))
    (* c (wedge dx dy))))
```

Here we need another vector field because our result will be a three-form field.

```
(define Z (literal-vector-field 'Z-rect R3-rect))

((( - (d omega)
  (+ (wedge (d a) dy dz)
    (wedge (d b) dz dx)
    (wedge (d c) dx dy)))
  X Y Z)
R3-rect-point)
0
```

Properties of Exterior Derivatives

The exterior derivative of the wedge of two form fields obeys the graded Leibniz rule. It can be written in terms of the exterior derivatives of the component form fields:

$$d(\omega \wedge \tau) = d\omega \wedge \tau + (-1)^k \omega \wedge d\tau, \quad (5.28)$$

where k is the rank of ω .

A form field ω that is the exterior derivative of another form field $\omega = d\theta$ is called *exact*. A form field whose exterior derivative is zero is called *closed*.

Every exact form field is a closed form field: applying the exterior derivative operator twice always yields zero:

$$d^2\omega = \mathbf{0}. \quad (5.29)$$

This is equivalent to the statement that partial derivatives with respect to different variables commute.⁸

It is easy to show equation (5.29) for manifold functions:

$$\begin{aligned} d^2f(u, v) &= d(df)(u, v) \\ &= u(df(v)) - v(df(u)) - df([u, v]) \\ &= u(v(f)) - v(u(f)) - [u, v](f) \\ &= 0 \end{aligned} \quad (5.30)$$

⁸See Spivak, *Calculus on Manifolds*, p.92

Consider the general one-form field θ defined on 3-dimensional rectangular space. Taking two exterior derivatives of θ yields a three-form field. It is zero:

```
((d (d theta)) X Y Z) R3-rect-point)
0
```

Not every closed form field is an exact form field. Whether a closed form field is exact depends on the topology of a manifold.

5.3 Stokes's Theorem

The proof of the general Stokes's Theorem for n-dimensional orientable manifolds is quite complicated, but it is easy to see how it works for a 2-dimensional region M that can be covered with a single coordinate patch.⁹

Given a coordinate chart $\chi(m) = (x(m), y(m))$ we can obtain a pair of coordinate-basis vectors $\partial/\partial x = X_0$ and $\partial/\partial y = X_1$.

The coordinate image of M can be divided into small rectangular areas in the (x, y) coordinate plane. The union of the rectangular areas gives the coordinate image of M . The clockwise integrals around the boundaries of the rectangles cancel on neighboring rectangles, because the boundary is traversed in opposite directions. But on the boundary of the coordinate image of M the boundary integrals do not cancel, yielding an integral on the boundary of M . Area integrals over the rectangular areas add to produce an integral over the entire coordinate image of M .

So, consider Stokes's Theorem on a small patch P of the manifold for which the coordinates form a rectangular region ($x_{min} < x < x_{max}$ and $y_{min} < y < y_{max}$). Stokes's Theorem on P states

$$\int_{\partial P} \omega = \int_P d\omega. \quad (5.31)$$

The area integral on the right can be written as an ordinary multidimensional integral using the coordinate basis vectors (recall

⁹We do not develop the machinery for integration on chains that is usually needed for a full proof of Stokes's Theorem. This is adequately done in other books. A beautiful treatment can be found in Spivak, *Calculus on Manifolds* [17].

that the integral is independent of the choice of coordinates):

$$\begin{aligned} \int_{\chi(\mathbb{P})} d\omega (\partial/\partial x, \partial/\partial y) \circ \chi^{-1} \\ = \int_{x_{min}}^{x_{max}} \int_{y_{min}}^{y_{max}} (\partial/\partial x (\omega(\partial/\partial y)) - \partial/\partial y (\omega(\partial/\partial x))) \circ \chi^{-1}. \end{aligned} \quad (5.32)$$

We have used equation (5.23) to expand the exterior derivative.

Consider just the first term of the right-hand side of equation (5.32). Then using the definition of basis vector field $\partial/\partial x$ we obtain

$$\begin{aligned} & \int_{x_{min}}^{x_{max}} \int_{y_{min}}^{y_{max}} (\partial/\partial x (\omega(\partial/\partial y)) \circ \chi^{-1}) \\ &= \int_{x_{min}}^{x_{max}} \int_{y_{min}}^{y_{max}} (X_0(\omega(\partial/\partial y)) \circ \chi^{-1}) \\ &= \int_{x_{min}}^{x_{max}} \int_{y_{min}}^{y_{max}} \partial_0 ((\omega(\partial/\partial y)) \circ \chi^{-1}). \end{aligned} \quad (5.33)$$

This integral can now be evaluated using the Fundamental Theorem of Calculus. Accumulating the results for both integrals

$$\begin{aligned} & \int_{\chi(\mathbb{P})} d\omega (\partial/\partial x, \partial/\partial y) \circ \chi^{-1} \\ &= \int_{x_{min}}^{x_{max}} ((\omega(\partial/\partial x)) \circ \chi^{-1})(x, y_{min}) dx \\ & \quad \int_{y_{min}}^{y_{max}} ((\omega(\partial/\partial y)) \circ \chi^{-1})(x_{max}, y) dy \\ & \quad - \int_{x_{min}}^{x_{max}} ((\omega(\partial/\partial x)) \circ \chi^{-1})(x, y_{max}) dx \\ & \quad - \int_{y_{min}}^{y_{max}} ((\omega(\partial/\partial y)) \circ \chi^{-1})(x_{min}, y) dy \\ &= \int_{\partial\mathbb{P}} \omega, \end{aligned} \quad (5.34)$$

as was to be shown.

5.4 Vector Integral Theorems

Green's Theorem states that for an arbitrary compact set $M \subset \mathbf{R}^2$, a 2-dimensional Euclidean space:

$$\int_{\partial M} ((\alpha \circ \chi) dx + (\beta \circ \chi) dy) = \int_M ((\partial_0 \beta - \partial_1 \alpha) \circ \chi) dx \wedge dy. \quad (5.35)$$

We can test this. By Stokes's Theorem, the integrands are related by an exterior derivative. We need some vectors to test our forms:

```
(define v (literal-vector-field 'v-rect R2-rect))
(define w (literal-vector-field 'w-rect R2-rect))
```

We can now test our integrands:¹⁰

```
(define alpha (literal-function 'alpha R2->R))
(define beta (literal-function 'beta R2->R))

(let ((dx (ref (basis->1form-basis R2-rect-basis) 0))
      (dy (ref (basis->1form-basis R2-rect-basis) 1)))
  (((- (d (+ (* (compose alpha (chart R2-rect)) dx)
              (* (compose beta (chart R2-rect)) dy)))
          (* (compose (- ((partial 0) beta)
                      ((partial 1) alpha))
                      (chart R2-rect)))
          (wedge dx dy)))
    v w)
   R2-rect-point))
  0
```

We can also compute the integrands for the Divergence Theorem: For an arbitrary compact set $M \subset \mathbf{R}^3$ and a vector field w

$$\int_M \operatorname{div}(w) dV = \int_{\partial M} w \cdot n dA \quad (5.36)$$

where n is the outward-pointing normal to the surface ∂M . Again, the integrands should be related by an exterior derivative, if this is an instance of Stokes's Theorem.

¹⁰Using `(define R2-rect-basis (coordinate-system->basis R2-rect))`.

Here we extract `dx` and `dy` from `R2-rect-basis` to avoid globally installing coordinates.

Note that even the statement of this theorem cannot be made with the machinery we have developed at this point. The concepts “outward-pointing normal,” area A , and volume V on the manifold are not definable without using a metric (see Chapter 9). However, for orthonormal rectangular coordinates in \mathbf{R}^3 we can interpret the integrands in terms of forms.

Let the vector field describing the flow of stuff be

$$w = a \frac{\partial}{\partial x} + b \frac{\partial}{\partial y} + c \frac{\partial}{\partial z}. \quad (5.37)$$

The rate of leakage of stuff through each element of the boundary is $w \cdot n dA$. We interpret this as the two-form

$$a dy \wedge dz + b dz \wedge dx + c dx \wedge dy, \quad (5.38)$$

because any part of the boundary will have $y-z$, $z-x$, and $x-y$ components, and each such component will pick up contributions from the normal component of the flux w . Formalizing this as code we have

```
(define a (literal-manifold-function 'a-rect R3-rect))
(define b (literal-manifold-function 'b-rect R3-rect))
(define c (literal-manifold-function 'c-rect R3-rect))

(define flux-through-boundary-element
  (+ (* a (wedge dy dz))
    (* b (wedge dz dx))
    (* c (wedge dx dy))))
```

The rate of production of stuff in each element of volume is $\text{div}(w) dV$. We interpret this as the three-form

$$\left(\frac{\partial}{\partial x} a + \frac{\partial}{\partial y} b + \frac{\partial}{\partial z} c \right) dx \wedge dy \wedge dz. \quad (5.39)$$

or:

```
(define production-in-volume-element
  (* (+ (d/dx a) (d/dy b) (d/dz c))
    (wedge dx dy dz)))
```

Assuming Stokes’s Theorem, the exterior derivative of the leakage of stuff per unit area through the boundary must be the rate of production of stuff per unit volume in the interior. We check this

by applying the difference to arbitrary vector fields at an arbitrary point:

```
(define X (literal-vector-field 'X-rect R3-rect))
(define Y (literal-vector-field 'Y-rect R3-rect))
(define Z (literal-vector-field 'Z-rect R3-rect))

((( - production-in-volume-element
      (d flux-through-boundary-element))
  X Y Z)
 R3-rect-point)
0
```

as expected.

Exercise 5.2: Graded Formula

Derive equation (5.28).

Exercise 5.3: Iterated Exterior Derivative

We have shown that the equation (5.29) is true for manifold functions. Show that it is true for any form field.

6

Over a Map

To deal with motion on manifolds we need to think about paths on manifolds and vectors along these paths. Tangent vectors along paths are not vector fields on the manifold because they are defined only on the path. And the path may even cross itself, which would give more than one vector at a point. Here we introduce the concept of a *vector field over a map*.¹ A vector field over a map assigns a vector to each image point of the map. In general the map may be a function from one manifold to another. If the domain of the map is the manifold of the real line, the range of the map is a 1-dimensional path on the target manifold. One possible way to define a vector field over a map is to assign a tangent vector to each image point of a path, allowing us to work with tangent vectors to paths. A *one-form field over the map* allows us to extract the components of a vector field over the map.

6.1 Vector Fields Over a Map

Let μ be a map from points n in the manifold N to points m in the manifold M . A vector over the map μ takes directional derivatives of functions on M at points $m = \mu(n)$. The vector over the map applied to the function on M is a function on N .

Restricted Vector Fields

One way to make a vector field over a map is to restrict a vector field on M to the image of N over μ , as illustrated in figure 6.1.

Let v be a vector field on M , and f a function on M . Then

$$v_\mu(f) = v(f) \circ \mu, \tag{6.1}$$

is a vector over the map μ . Note that $v_\mu(f)$ is a function on N , not M :

$$v_\mu(f)(n) = v(f)(\mu(n)). \tag{6.2}$$

¹See Bishop and Goldberg, *Tensor Analysis on Manifolds* [3].

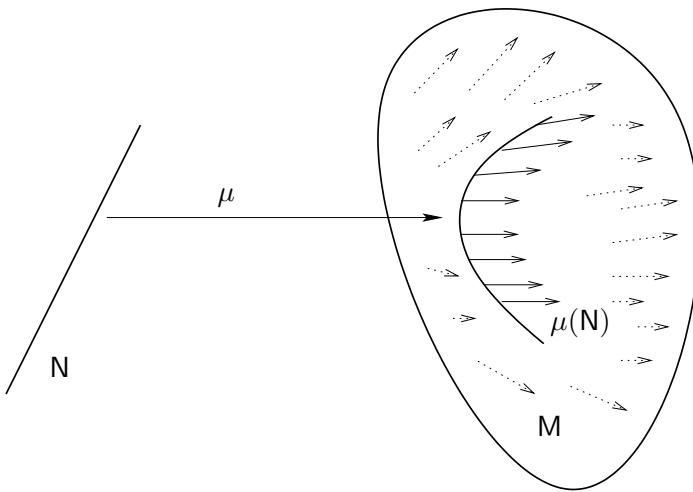


Figure 6.1 The vector field v on M is indicated by arrows. The solid arrows are v_μ , the restricted vector field over the map μ . The vector field over the map is restricted to the image of N in M .

We can implement this definition as:

```
(define ((vector-field->vector-field-over-map mu:N->M) v-on-M)
  (procedure->vector-field
    (lambda (f-on-M)
      (compose (v-on-M f-on-M) mu:N->M))))
```

Differential of a Map

Another way to construct a vector field over a map μ is to transport a vector field from the source manifold N to the target manifold M with the *differential* of the map

$$d\mu(v)(f)(n) = v(f \circ \mu)(n), \quad (6.3)$$

which takes its argument in the source manifold N . The differential of a map μ applied to a vector field v on N is a vector field over the map. A procedure to compute the differential is:

```
(define (((differential mu) v) f)
  (v (compose f mu)))
```

The nomenclature of this subject is confused. The “differential of a map between manifolds,” $d\mu$, takes one more argument than the “differential of a real-valued function on a manifold,” df , but when the target manifold of μ is the reals and I is the identity function on the reals,

$$d\mu(v)(I)(n) = (v(I \circ \mu))(n) = (v(\mu))(n) = d\mu(v)(n). \quad (6.4)$$

We avoid this problem in our notation by distinguishing d and d . In our programs we encode d as `differential` and d as `d`.

Velocity at a Time

Let μ be the map from the time line to the manifold M , and $\partial/\partial t$ be a basis vector on the time line. Then $d\mu(\partial/\partial t)$ is the vector over the map μ that computes the rate of change of functions on M along the path that is the image of μ . This is the velocity vector. We can use the differential to assign a velocity vector to each moment, solving the problem of multiple vectors at a point if the path crosses itself.

6.2 One-Form Fields Over a Map

Given a one-form ω on the manifold M , the one-form over the map $\mu : N \rightarrow M$ is constructed as follows:

$$\omega^\mu(v_\mu)(n) = \omega(u)(\mu(n)), \text{ where } u(f)(m) = v_\mu(f)(n). \quad (6.5)$$

The object u is not really a vector field on M even though we have given it that shape so that the dual vector can apply to it; $u(f)$ is evaluated only at images $m = \mu(n)$ of points n in N . If we were defining u as a vector field we would need the inverse of μ to find the point $n = \mu^{-1}(m)$, but this is not required to define the object u in a context where there is already an m associated with the n of interest. To extend this idea to k -forms, we carry each vector argument over the map.

The procedure that constructs a k -form over the map from a k -form is:

```
(define ((form-field->form-field-over-map mu:N->M) w-on-M)
  (define (make-fake-vector-field V-over-mu n)
    (define ((u f) m)
      ((V-over-mu f) n))
    (procedure->vector-field u))
  (procedure->nform-field
    (lambda vectors-over-map
      (lambda (n)
        ((apply w-on-M
          (map (lambda (V-over-mu)
            (make-fake-vector-field V-over-mu n)
            vectors-over-map))
          (mu:N->M n)))))))
  (get-rank w-on-M)))
```

The internal procedure `make-fake-vector-field` counterfeits a vector field u on M from the vector field over the map $\mu : N \rightarrow M$. This works here because the only value that is ever passed as m is $(\mu:N \rightarrow M\ n)$.

6.3 Basis Fields Over a Map

Let e be a tuple of basis vector fields, and \tilde{e} be the tuple of basis one-forms that is dual to e :

$$\tilde{e}^i(e_j)(m) = \delta_j^i. \quad (6.6)$$

The *basis vectors over the map*, e^μ , are particular cases of vectors over a map:

$$e^\mu(f) = e(f) \circ \mu. \quad (6.7)$$

And the elements of the *dual basis over the map*, \tilde{e}_μ , are particular cases of one-forms over the map. The basis and dual basis over the map satisfy

$$\tilde{e}_\mu^i(e_j^\mu)(n) = \delta_j^i. \quad (6.8)$$

Walking on a Sphere

For example, let μ map the time line to the unit sphere.² We use colatitude θ and longitude ϕ as coordinates on the sphere:

```
(define S2 (make-manifold S^2 2 3))
(define S2-spherical
  (coordinate-system-at 'spherical 'north-pole S2))
(define-coordinates (up theta phi) S2-spherical)
(define S2-basis (coordinate-system->basis S2-spherical))
```

A general path on the sphere is:³

```
(define mu
  (compose (point S2-spherical)
    (up (literal-function 'theta)
      (literal-function 'phi)))
  (chart R1-rect)))
```

The basis over the map is constructed from the basis on the sphere:

```
(define S2-basis-over-mu
  (basis->basis-over-map mu S2-basis))

(define h
  (literal-manifold-function 'h-spherical S2-spherical))

(((basis->vector-basis S2-basis-over-mu) h)
 ((point R1-rect) 't0))
(down
 (((partial 0) h-spherical) (up (theta t0) (phi t0)))
 (((partial 1) h-spherical) (up (theta t0) (phi t0))))
```

The basis vectors over the map compute derivatives of the function h evaluated on the path at the given time.

²We execute `(define-coordinates t R1-rect)` to make t the coordinate function of the real line.

³We provide a shortcut to make literal manifold maps:

```
(define mu (literal-manifold-map 'mu R1-rect S2-spherical))
```

But if we used this shortcut, the component functions would be named `mu^0` and `mu^1`. Here we wanted to use more mnemonic names for the component functions.

We can check that the dual basis over the map does the correct thing:

```
((basis->1form-basis S2-basis-over-mu)
 (basis->vector-basis S2-basis-over-mu))
 ((point R1-rect) 't0))
 (up (down 1 0) (down 0 1))
```

Components of the Velocity

Let χ be a tuple of coordinates on M , with associated basis vectors X_i , and dual basis elements dx^i . The vector basis and dual basis over the map μ are X_i^μ and dx_μ^i . The components of the velocity (rates of change of coordinates along the path μ) are obtained by applying the dual basis over the map to the velocity

$$v^i(t) = dx_\mu^i(d\mu(\partial/\partial t))(t), \quad (6.9)$$

where t is the coordinate for the point t .

For example, the coordinate velocities on a sphere are

```
((basis->1form-basis S2-basis-over-mu)
 ((differential mu) d/dt))
 ((point R1-rect) 't0))
 (up ((D theta) t0) ((D phi) t0)))
```

as expected.

6.4 Pullbacks and Pushforwards

Maps from one manifold to another can also be used to relate the vector fields and one-form fields on one manifold to those on the other. We have introduced two such relations: restricted vector fields and the differential of a function. However, there are other ways to relate the vector fields and form fields on different manifolds that are connected by a map.

Pullback and Pushforward of a Function

The *pullback* of a function f on M over the map μ is defined as

$$\mu^*f = f \circ \mu. \quad (6.10)$$

This allows us to take a function defined on M and use it to define a new function on N .

For example, the integral curve of v evolved for time t as a function of the initial manifold point m generates a map ϕ_t^v of the manifold onto itself. This is a simple currying⁴ of the integral curve of v from m as a function of time: $\phi_t^v(m) = \gamma_m^v(t)$. The evolution of the function f along an integral curve, equation (3.33), can be written in terms of the pullback over ϕ_t^v :

$$(\mathsf{E}_{t,v}f)(m) = f(\phi_t^v(m)) = ((\phi_t^v)^*f)(m). \quad (6.11)$$

This is implemented as:

```
(define ((pullback-function mu:N->M) f-on-M)
  (compose f-on-M mu:N->M))
```

A vector field over the map that was constructed by restriction (equation 6.1) can be seen as the pullback of the function constructed by application of the vector field to a function:

$$v_\mu(f) = v(f) \circ \mu = \mu^*(v(f)). \quad (6.12)$$

A vector field over the map that was constructed by a differential (equation 6.3) can be seen as the vector field applied to the pullback of the function:

$$d\mu(v)(f)(n) = v(f \circ \mu)(n) = v(\mu^*f)(n). \quad (6.13)$$

If we have an inverse for the map μ we can also define a *push-forward* of the function g , defined on the source manifold of the map:⁵

$$\mu_*g = g \circ \mu^{-1}. \quad (6.14)$$

⁴A function of two arguments may be seen as a function of one argument whose value is a function of the other argument. This can be done in two different ways, depending on which argument is supplied first. The general process of specifying a subset of the arguments to produce a new function of the others is called *currying* the function, in honor of the logician Haskell Curry (1900–1982) who, with Moses Schönfinkel (1889–1942), developed combinatory logic.

⁵Notation note: superscript asterisk indicates pullback, subscript asterisk indicates pushforward. Pullbacks and pushforwards are tightly binding operators, so, for example $\mu^*f(n) = (\mu^*f)(n)$.

Pushforward of a Vector Field

We can also define the *pushforward* of a vector field over the map μ . The pushforward takes a vector field v defined on N . The result takes directional derivatives of functions on M at a place determined by a point in M :

$$\mu_*v(f)(m) = v(\mu^*f)(\mu^{-1}(m)) = v(f \circ \mu)(\mu^{-1}(m)), \quad (6.15)$$

or

$$\mu_*v(f) = \mu_*(v(\mu^*f)). \quad (6.16)$$

Here we expressed the pushforward of the vector field in terms of pullbacks and pushforwards of functions. Note that the pushforward requires the inverse of the map.

If the map is from time to some configuration manifold and represents the time evolution of a process, we can think of the pushforward of a vector field as a velocity measured at a point on the trajectory in the configuration manifold. By contrast, the differential of the map applied to the vector field gives us the velocity vector at each moment in time. Because a trajectory may cross itself, the pushforward is not defined at any point where the crossing occurs, but the differential is always defined.

Pushforward Along Integral Curves

We can push a vector field forward over the map generated by an integral curve of a vector field w , because the inverse is always available.⁶

$$((\phi_t^w)_*v)(f)(m) = v((\phi_t^w)^*f)(\phi_{-t}^w(m)) = v(f \circ \phi_t^w)(\phi_{-t}^w(m)). \quad (6.17)$$

This is implemented as:

```
(define ((pushforward-vector mu:N->M mu^-1:M->N) v-on-N)
  (procedure->vector-field
    (lambda (f)
      (compose (v-on-N (compose f mu:N->M)) mu^-1:M->N))))
```

⁶The map ϕ_t^w is always invertible: $(\phi_t^w)^{-1} = \phi_{-t}^w$ because of the uniqueness of the solutions of the initial-value problem for ordinary differential equations.

Pullback of a Vector Field

Given a vector field v on manifold M we can pull the vector field back through the map $\mu : N \rightarrow M$ as follows:

$$\mu^*v(f)(n) = (v(f \circ \mu^{-1}))(\mu(n)) \quad (6.18)$$

or

$$\mu^*v(f) = \mu^*(v(\mu_*f)). \quad (6.19)$$

This may be useful when the map is invertible, as in the flow generated by a vector field.

This is implemented as:

```
(define (pullback-vector-field mu:N->M mu^-1:M->N)
  (pushforward-vector mu^-1:M->N mu:N->M))
```

Pullback of a Form Field

We can also pull back a one-form field ω defined on M , but an honest definition is rarely written. The pullback of a one-form field applied to a vector field is intended to be the same as the one-form field applied to the pushforward of the vector field.

The pullback of a one-form field is often described by the relation

$$\mu^*\omega(v) = \omega(\mu_*v), \quad (6.20)$$

but this is wrong, because the two sides are not functions of points in the same manifold. The one-form field ω applies to a vector field on the manifold M , which takes a directional derivative of a function defined on M and is evaluated at a point on M , but the left-hand side is evaluated at a point on the manifold N .

A more precise description would be

$$\mu^*\omega(v)(n) = \omega(\mu_*v)(\mu(n)) \quad (6.21)$$

or

$$\mu^*\omega(v) = \mu^*(\omega(\mu_*v)). \quad (6.22)$$

Although this is accurate, it may not be effective, because computing the pushforward requires the inverse of the map μ . But the inverse is available when the map is the flow generated by a vector field.

In fact it is possible to compute the pullback of a one-form field without having the inverse of the map. Instead we can use `form-field->form-field-over-map` to avoid needing the inverse:

$$\mu^* \omega(v)(n) = \omega^\mu(d\mu(v))(n). \quad (6.23)$$

The pullback of a k -form generalizes equation 6.21:

$$\mu^* \omega(u, v, \dots)(n) = \omega(\mu_* u, \mu_* v, \dots)(\mu(n)). \quad (6.24)$$

This is implemented as follows:⁷

```
(define ((pullback-form mu:N->M) omega-on-M)
  (let ((k (get-rank omega-on-M)))
    (if (= k 0)
        ((pullback-function mu:N->M) omega-on-M)
        (procedure->nform-field
         (lambda vectors-on-N
           (apply ((form-field->form-field-over-map mu:N->M)
                  omega-on-M)
                  (map (differential mu:N->M) vectors-on-N)))
         k))))
```

Properties of Pullback

The pullback through a map has many nice properties: it distributes through addition and through wedge product:

$$\mu^*(\theta + \phi) = \mu^*\theta + \mu^*\phi, \quad (6.25)$$

$$\mu^*(\theta \wedge \phi) = \mu^*\theta \wedge \mu^*\phi. \quad (6.26)$$

The pullback also commutes with the exterior derivative:

$$d(\mu^*\theta) = \mu^*(d\theta), \quad (6.27)$$

for θ a function or k -form field.

⁷There is a generic `pullback` procedure that operates on any kind of manifold object. However, to pull a vector field back requires providing the inverse map.

We can verify this by computing an example. Let μ map the rectangular plane to rectangular 3-space:

```
(define mu (literal-manifold-map 'MU R2-rect R3-rect))
```

First, let's compare the pullback of the exterior derivative of a function with the exterior derivative of the pullback of the function:

```
(define f (literal-manifold-function 'f-rect R3-rect))
(define X (literal-vector-field 'X-rect R2-rect))

((( - ((pullback mu) (d f)) (d ((pullback mu) f))) X)
 ((point R2-rect) (up 'x0 'y0)))
0
```

More generally, we can consider what happens to a form field. For a one-form field the result is as expected:

```
(define theta (literal-1form-field 'THETA R3-rect))
(define Y (literal-vector-field 'Y-rect R2-rect))

((( - ((pullback mu) (d theta)) (d ((pullback mu) theta))) X Y)
 ((point R2-rect) (up 'x0 'y0)))
0
```

Pushforward of a Form Field

By symmetry, it is possible to define the pushforward of a one-form field as

$$\mu_*(\omega(v)) = \mu_*(\omega(\mu^*v)), \quad (6.28)$$

but this is rarely useful.

Exercise 6.1: Velocities on a Globe

We can use manifold functions, vector fields, and one-forms over a map to understand how paths behave.

- a. Suppose that a vehicle is traveling east on the Earth at a given rate of change of longitude. What is the actual ground speed of the vehicle?
- b. Stereographic projection is useful for navigation because it is conformal (it preserves angles). For the situation of part a, what is the speed measured on a stereographic map? Remember that the stereographic projection is implemented with `S2-Riemann`.

7

Directional Derivatives

The vector field was a generalization of the directional derivative to functions on a manifold. When we want to generalize the directional derivative idea to operate on other manifold objects, such as directional derivatives of vector fields or of form fields, there are several useful choices. In the same way that a vector field applies to a function to produce a function, we will build directional derivatives so that when applied to any object it will produce another object of the same kind. All directional derivatives require a vector field to give the direction and scale factor.

We will have a choice of directional derivative operators that give different results for the rate of change of vector and form fields along integral curves. But all directional derivative operators must agree when computing rates of change of functions along integral curves. When applied to functions, all directional derivative operators give:

$$\mathcal{D}_v(f) = v(f). \quad (7.1)$$

Next we specify the directional derivative of a vector field u with respect to a vector field v . Let an integral curve of the vector field v be γ , parameterized by t , and let $m = \gamma(t)$. Let u' be a vector field that results from transporting the vector field u along γ for a parameter increment δ . How u is transported to make u' determines the type of derivative. We formulate the method of transport by:

$$u' = F_\delta^v u. \quad (7.2)$$

We can assume without loss of generality that $F_\delta^v u$ is a linear transformation over the reals on u , because we care about its behavior only in an incremental region around $\delta = 0$.

Let g be the comparison of the original vector field at a point with the transported vector field at that point:

$$g(\delta) = u(f)(m) - (F_\delta^v u)(f)(m). \quad (7.3)$$

So we can compute the directional derivative operator using only ordinary derivatives:

$$\mathcal{D}_v u(f)(m) = Dg(0). \quad (7.4)$$

The result $\mathcal{D}_v u$ is of type vector field.

The general pattern of constructing a directional derivative operator from a transport operator is given by the following schema:¹

```
(define (((F->directional-derivative F) v) u) f) m)
  (define (g delta)
    (- ((u f) m) (((((F v) delta) u) f) m)))
  ((D g) 0))
```

The linearity of transport implies that

$$\mathcal{D}_v(\alpha O + \beta P) = \alpha \mathcal{D}_v O + \beta \mathcal{D}_v P, \quad (7.5)$$

for any real α and β and manifold objects O and P .

The directional derivative obeys superposition in its vector-field argument:

$$\mathcal{D}_{v+w} = \mathcal{D}_v + \mathcal{D}_w. \quad (7.6)$$

The directional derivative is homogeneous over the reals in its vector-field argument:

$$\mathcal{D}_{\alpha v} = \alpha \mathcal{D}_v, \quad (7.7)$$

for any real α .² This follows from the fact that for evolution along integral curves: when α is a real number,

$$\phi_t^{\alpha v}(m) = \phi_{\alpha t}^v(m). \quad (7.8)$$

When applied to products of functions, directional derivative operators satisfy Leibniz's rule:

$$\mathcal{D}_v(fg) = f(\mathcal{D}_v g) + (\mathcal{D}_v f) g. \quad (7.9)$$

¹The directional derivative of a vector field must itself be a vector field. Thus the real program for this must make the function of f into a vector field. However, we leave out this detail here to make the structure clear.

²For some derivative operators α can be a real-valued manifold function.

The Leibniz rule is extended to applications of one-form fields to vector fields:

$$\mathcal{D}_v(\omega(y)) = \omega(\mathcal{D}_v y) + (\mathcal{D}_v \omega)(y). \quad (7.10)$$

The extension of the Leibniz rule, combined with the choice of transport of a vector field, determines the action of the directional derivative on form fields.³

7.1 Lie Derivative

The Lie derivative is one kind of directional derivative operator. We write the Lie derivative operator with respect to a vector field v as \mathcal{L}_v .

Functions

The Lie derivative of the function f with respect to the vector field v is given by

$$\mathcal{L}_v f = v(f). \quad (7.11)$$

The tangent vector v measures the rate of change of f along integral curves.

Vector Fields

For the Lie derivative of a vector field y with respect to a vector field v we choose the transport operator $F_\delta^v y$ to be the pushforward of y along the integral curves of v . Recall equation (6.15). So the Lie derivative of y with respect to v at the point m is

$$(\mathcal{L}_v y)(f)(m) = Dg(0), \quad (7.12)$$

where

$$g(\delta) = y(f)(m) - ((\phi_\delta^v)_* y)(f)(m). \quad (7.13)$$

We can construct a procedure that computes the Lie derivative of a vector field by supplying an appropriate transport operator

³The action on functions, vector fields, and one-form fields suffices to define the action on all tensor fields. See Appendix C.

(F-Lie phi) for F in our schema F->directional-derivative. In this first stab at the Lie derivative, we introduce a coordinate system and we expand the integral curve to a given order. Because in the schema we evaluate the derivative of g at 0, the dependence on the order and the coordinate system disappears. They will not be needed in the final version.

```
(define (Lie-directional coordsys order)
  (let ((Phi (phi coordsys order)))
    (F->directional-derivative (F-Lie Phi)))

(define (((F-Lie phi) v) delta)
  (pushforward-vector ((phi v) delta) ((phi v) (- delta)))))

(define ((((phi coordsys order) v) delta) m)
  ((point coordsys)
   (series:sum (((exp (* delta v)) (chart coordsys)) m)
               order)))
```

Expand the quantities in equation (7.13) to first order in δ :

$$\begin{aligned} g(\delta) &= y(f)(m) - (\phi_{\delta}^v y)(f)(m) \\ &= y(f)(m) - y(f \circ \phi_{\delta}^v)(\phi_{-\delta}^v(m)) \\ &= (y(f) - y(f + \delta v(f) + \dots) + \delta v(y(f + \delta v(f) + \dots))(m) + \dots \\ &= (-\delta y(v(f)) + \delta v(y(f)))(m) + \dots \\ &= \delta [v, y](f)(m) + \mathcal{O}(\delta^2). \end{aligned} \quad (7.14)$$

So the Lie derivative of a vector field y with respect to a vector field v is a vector field that is defined by its behavior when applied to an arbitrary manifold function f :

$$(\mathcal{L}_v y)(f) = [v, y](f) \quad (7.15)$$

Verifying this computation

```
(let ((v (literal-vector-field 'v-rect R3-rect))
      (w (literal-vector-field 'w-rect R3-rect))
      (f (literal-manifold-function 'f-rect R3-rect)))
  ((- (((Lie-directional R3-rect 2) v) w) f)
   ((commutator v w) f))
  ((point R3-rect) (up 'x0 'y0 'z0))))
```

0

Although this is tested to second order, evaluating the derivative at zero ensures that first order is enough. So we can safely define:

```
(define ((Lie-derivative-vector V) Y)
  (commutator V Y))
```

We can think of the Lie derivative as the rate of change of the manifold function $y(f)$ as we move in the v direction, adjusted to take into account that some of the variation is due to the variation of f :

$$\begin{aligned} (\mathcal{L}_v y)(f) &= [v, y](f) \\ &= v(y(f)) - y(v(f)) \\ &= v(y(f)) - y(\mathcal{L}_v(f)). \end{aligned} \tag{7.16}$$

The first term in the commutator, $v(y(f))$, measures the rate of change of the combination $y(f)$ along the integral curves of v . The change in $y(f)$ is due to both the intrinsic change in y along the curve and the change in f along the curve; the second term in the commutator subtracts this latter quantity. The result is the intrinsic change in y along the integral curves of v .

Additionally, we can extend the product rule, for any manifold function g and any vector field u :

$$\begin{aligned} \mathcal{L}_v(gu)(f) &= [v, gu](f) \\ &= v(g)u(f) + g[v, u](f) \\ &= (\mathcal{L}_v g)u(f) + g(\mathcal{L}_v u)(f). \end{aligned} \tag{7.17}$$

An Alternate View

We can write the vector field

$$y(f) = \sum_i y^i e_i(f). \tag{7.18}$$

By the extended product rule (equation 7.17) we get

$$\mathcal{L}_v y(f) = \sum_i (v(y^i)e_i(f) + y^i \mathcal{L}_v e_i(f)). \tag{7.19}$$

Because the Lie derivative of a vector field is a vector field, we can extract the components of $\mathcal{L}_v \mathbf{e}_i$ using the dual basis. We define $\Delta_j^i(v)$ to be those components:

$$\Delta_j^i(v) = \tilde{\mathbf{e}}^i(\mathcal{L}_v \mathbf{e}_j) = \tilde{\mathbf{e}}^i([v, \mathbf{e}_j]). \quad (7.20)$$

So the Lie derivative can be written

$$(\mathcal{L}_v y)(f) = \sum_i \left(v(y^i) + \sum_j \Delta_j^i(v) y^j \right) e_i(f). \quad (7.21)$$

The components of the Lie derivatives of the basis vector fields are the structure constants for the basis vector fields. (See equation 4.37.) The structure constants are antisymmetric in the lower indices:

$$\tilde{\mathbf{e}}^i(\mathcal{L}_{e_k} e_j) = \tilde{\mathbf{e}}^i([e_k, e_j]) = d_{kj}^i. \quad (7.22)$$

Resolving v into components and applying the product rule, we get

$$(\mathcal{L}_v y)(f) = \sum_k (v^k [e_k, y](f) - y(v^k) e_k(f)). \quad (7.23)$$

So $\Delta_j^i(v)$ is related to the structure constants by

$$\begin{aligned} \Delta_j^i(v) &= \tilde{\mathbf{e}}^i(\mathcal{L}_v \mathbf{e}_j) \\ &= \sum_k (v^k \tilde{\mathbf{e}}^i([e_k, e_j]) - e_j(v^k) \tilde{\mathbf{e}}^i(e_k)) \\ &= \sum_k (v^k d_{kj}^i - e_j(v^k) \delta_k^i) \\ &= \sum_k v^k d_{kj}^i - e_j(v^i). \end{aligned} \quad (7.24)$$

Note: Despite their appearance, the Δ_j^i are not form fields because $\Delta_j^i(fv) \neq f\Delta_j^i(v)$.

Form Fields

We can also define the Lie derivative of a form field ω with respect to the vector field v by its action on an arbitrary vector field y , using the extended Leibniz rule (see equation 7.10):

$$(\mathcal{L}_v(\omega))(y) \equiv v(\omega(y)) - \omega(\mathcal{L}_v y). \quad (7.25)$$

The first term computes the rate of change of the combination $\omega(y)$ along the integral curve of v , while the second subtracts ω applied to the change in y . The result is the change in ω along the curve.

The Lie derivative of a k -form field ω with respect to a vector field v is a k -form field that is defined by its behavior when applied to k arbitrary vector fields w_0, \dots, w_{k-1} . We generalize equation (7.25):

$$\begin{aligned} \mathcal{L}_v\omega(w_0, \dots, w_{k-1}) & \quad (7.26) \\ &= v(\omega(w_0, \dots, w_{k-1})) - \sum_{i=0}^{k-1} \omega(w_0, \dots, \mathcal{L}_v w_i, \dots, w_{k-1}). \end{aligned}$$

Uniform Interpretation

Consider abstracting the equations (7.16), (7.25), and (7.27). The Lie derivative of an object, a , that can apply to other objects, b , to produce manifold functions, $a(b) : M \rightarrow \mathbb{R}^n$, is

$$(\mathcal{L}_v a)(b) = v(a(b)) - a(\mathcal{L}_v b). \quad (7.27)$$

The first term in this expression computes the rate of change of the compound object $a(b)$ along integral curves of v , while the second subtracts the change in a due to the change in b along the curves. The result is a measure of the “intrinsic” change in a along integral curves of v , with b held “fixed.”

Properties of the Lie Derivative

As required by properties 7.7–7.5, the Lie derivative is linear in its arguments:

$$\mathcal{L}_{\alpha v + \beta w} = \alpha \mathcal{L}_v + \beta \mathcal{L}_w, \quad (7.28)$$

and

$$\mathcal{L}_v(\alpha \mathbf{a} + \beta \mathbf{b}) = \alpha \mathcal{L}_v \mathbf{a} + \beta \mathcal{L}_v \mathbf{b}, \quad (7.29)$$

with $\alpha, \beta \in \mathbb{R}$ and vector fields or one-form fields \mathbf{a} and \mathbf{b} .

For any k -form field ω and any vector field v the exterior derivative commutes with the Lie derivative with respect to the vector field:

$$\mathcal{L}_v(d\omega) = d(\mathcal{L}_v\omega). \quad (7.30)$$

If ω is an element of surface then $d\omega$ is an element of volume. The Lie derivative computes the rate of change of its argument under a deformation described by the vector field. The answer is the same whether we deform the surface before computing the volume or compute the volume and then deform it.

We can verify this in 3-dimensional rectangular space for a general one-form field:⁴

```
(((- ((Lie-derivative V) (d theta))
      (d ((Lie-derivative V) theta)))
  X Y)
 R3-rect-point)
0
```

and for the general two-form field:

⁴In these experiments we need some setup.

```
(define a (literal-manifold-function 'alpha R3-rect))
(define b (literal-manifold-function 'beta R3-rect))
(define c (literal-manifold-function 'gamma R3-rect))

(define-coordinates (up x y z) R3-rect)

(define theta (+ (* a dx) (* b dy) (* c dz)))

(define omega
  (+ (* a (wedge dy dz))
    (* b (wedge dz dx))
    (* c (wedge dx dy)))))

(define X (literal-vector-field 'X-rect R3-rect))
(define Y (literal-vector-field 'Y-rect R3-rect))
(define Z (literal-vector-field 'Z-rect R3-rect))
(define V (literal-vector-field 'V-rect R3-rect))
(define R3-rect-point
  ((point R3-rect) (up 'x0 'y0 'z0)))
```

```
(((- ((Lie-derivative V) (d omega))
      (d ((Lie-derivative V) omega)))
  X Y Z)
R3-rect-point)
0
```

The Lie derivative satisfies another nice elementary relationship. If v and w are two vector fields, then

$$[\mathcal{L}_v, \mathcal{L}_w] = \mathcal{L}_{[v,w]}. \quad (7.31)$$

Again, for our general one-form field θ :

```
(((( (- (commutator (Lie-derivative X) (Lie-derivative Y))
      (Lie-derivative (commutator X Y)))
    theta)
  Z)
R3-rect-point)
0
```

and for the two-form field ω :

```
((((( - (commutator (Lie-derivative X) (Lie-derivative Y))
      (Lie-derivative (commutator X Y)))
    omega)
  Z V)
R3-rect-point)
0
```

Exponentiating Lie Derivatives

The Lie derivative computes the rate of change of objects as they are advanced along integral curves. The Lie derivative of an object produces another object of the same type, so we can iterate Lie derivatives. This gives us Taylor series for objects along the curve.

The operator $e^{t\mathcal{L}_v} = 1 + t\mathcal{L}_v + \frac{t^2}{2!}\mathcal{L}_v^2 + \dots$ evolves objects along the curve by parameter t . For example, the exponential of a Lie derivative applied to a vector field is

$$\begin{aligned} e^{t\mathcal{L}_v} y &= y + t\mathcal{L}_v y + \frac{t^2}{2}\mathcal{L}_v^2 y + \dots \\ &= y + t[v, y] + \frac{t^2}{2}[v, [v, y]] + \dots \end{aligned} \quad (7.32)$$

Consider a simple case. We advance the coordinate-basis vector field $\partial/\partial y$ by an angle a around the circle. Let $J_z = x\partial/\partial y - y\partial/\partial x$, the circular vector field. We recall

```
(define Jz (- (* x d/dy) (* y d/dx)))
```

We can apply the exponential of the Lie derivative with respect to J_z to $\partial/\partial y$. We examine how the result affects a general function on the manifold:

```
(series:for-each print-expression
 (((exp (* 'a (Lie-derivative Jz))) d/dy)
  (literal-manifold-function 'f-rect R3-rect))
  ((point R3-rect) (up 1 0 0)))
 5)
(((partial 0) f-rect) (up 1 0))
(* -1 a (((partial 1) f-rect) (up 1 0)))
(* -1/2 (expt a 2) (((partial 0) f-rect) (up 1 0)))
(* 1/6 (expt a 3) (((partial 1) f-rect) (up 1 0)))
(* 1/24 (expt a 4) (((partial 0) f-rect) (up 1 0)))
;Value: ...
```

Apparently the result is

$$\exp(a \mathcal{L}_{(x\partial/\partial y - y\partial/\partial x)}) \frac{\partial}{\partial y} = -\sin(a) \frac{\partial}{\partial x} + \cos(a) \frac{\partial}{\partial y}. \quad (7.33)$$

Interior Product

There is a simple but useful operation available between vector fields and form fields called *interior product*. This is the substitution of a vector field v into the first argument of a p -form field ω to produce a $p-1$ -form field:

$$(i_v \omega)(v_1, \dots, v_{p-1}) = \omega(v, v_1, \dots, v_{p-1}). \quad (7.34)$$

There is a mundane identity corresponding to the product rule for the Lie derivative of an interior product:

$$\mathcal{L}_v(i_y \omega) = i_{\mathcal{L}_v y} \omega + i_y(\mathcal{L}_v \omega). \quad (7.35)$$

And there is a rather nice identity for the Lie derivative in terms of the interior product and the exterior derivative, called *Cartan's formula*:

$$\mathcal{L}_v \omega = i_v(d\omega) + d(i_v \omega). \quad (7.36)$$

We can verify Cartan's formula in a simple case with a program:

```
(define X (literal-vector-field 'X-rect R3-rect))
(define Y (literal-vector-field 'Y-rect R3-rect))
(define Z (literal-vector-field 'Z-rect R3-rect))

(define a (literal-manifold-function 'alpha R3-rect))
(define b (literal-manifold-function 'beta R3-rect))
(define c (literal-manifold-function 'gamma R3-rect))

(define omega
  (+ (* a (wedge dx dy))
    (* b (wedge dy dz))
    (* c (wedge dz dx)))))

(define ((L1 X) omega)
  (+ ((interior-product X) (d omega))
    (d ((interior-product X) omega)))))

((- (((Lie-derivative X) omega) Y Z)
  (((L1 X) omega) Y Z))
  ((point R3-rect) (up 'x0 'y0 'z0)))
  0)
```

Note that $i_v \circ i_u + i_u \circ i_v = 0$. One consequence of this is that $i_v \circ i_v = 0$.

7.2 Covariant Derivative

The covariant derivative is another kind of directional derivative operator. We write the covariant derivative operator with respect to a vector field v as ∇_v . This is pronounced “covariant derivative with respect to v ” or “nabla v .”

Covariant Derivative of Vector Fields

We may also choose our $F_\delta^v u$ to define what we mean by “parallel” transport of the vector field u along an integral curve of the vector field v . This may correspond to our usual understanding of parallel in situations where we have intuitive insight.

The notion of parallel transport is path dependent. Remember our example from the Introduction, page 1: Start at the North Pole carrying a stick along a line of longitude to the Equator, always pointing it south, parallel to the surface of the Earth. Then

proceed eastward for some distance, still pointing the stick south. Finally, return to the North Pole along this new line of longitude, keeping the stick pointing south all the time. At the pole the stick will not point in the same direction as it did at the beginning of the trip, and the discrepancy will depend on the amount of eastward motion.⁵

So if we try to carry a stick parallel to itself and tangent to the sphere, around a closed path, the stick generally does not end up pointing in the same direction as it started. The result of carrying the stick from one point on the sphere to another depends on the path taken. However, the direction of the stick at the endpoint of a path does not depend on the rate of transport, just on the particular path on which it is carried. Parallel transport over a zero-length path is the identity.

A vector may be resolved as a linear combination of other vectors. If we parallel-transport each component, and form the same linear combination, we get the transported original vector. Thus parallel transport on a particular path for a particular distance is a linear operation.

So the transport function F_δ^v is a linear operator on the components of its argument, and thus

$$F_\delta^v u(f)(m) = \sum_{i,j} (A_j^i(\delta)(u^j \circ \phi_{-\delta}^v) e_i(f))(m) \quad (7.37)$$

for some functions A_j^i that depend on the particular path (hence its tangent vector v) and the initial point. We reach back along the integral curve to pick up the components of u and then parallel-transport them forward by the matrix $A_j^i(\delta)$ to form the components of the parallel-transported vector at the advanced point.

As before, we compute

$$\nabla_v u(f)(m) = Dg(0), \quad (7.38)$$

where

$$g(\delta) = u(f)(m) - (F_\delta^v u)(f)(m). \quad (7.39)$$

⁵In the introduction the stick was always kept east-west rather than pointing south, but the phenomenon is the same!

Expanding with respect to a basis $\{\mathbf{e}_i\}$ we get

$$g(\delta) = \sum_i \left(\mathbf{u}^i \mathbf{e}_i(\mathbf{f}) - \sum_j A_j^i(\delta)(\mathbf{u}^j \circ \phi_{-\delta}^\mathbf{v}) \mathbf{e}_i(\mathbf{f}) \right) (\mathbf{m}). \quad (7.40)$$

By the product rule for derivatives,

$$\begin{aligned} Dg(\delta) &= \\ &\sum_{ij} (A_j^i(\delta)((\mathbf{v}(\mathbf{u}^j)) \circ \phi_{-\delta}^\mathbf{v}) \mathbf{e}_i(\mathbf{f}) - DA_j^i(\delta)(\mathbf{u}^j \circ \phi_{-\delta}^\mathbf{v}) \mathbf{e}_i(\mathbf{f})) (\mathbf{m}). \end{aligned} \quad (7.41)$$

So, since $A_j^i(0)(\mathbf{m})$ is the identity multiplier, and $\phi_0^\mathbf{v}$ is the identity function,

$$Dg(0) = \sum_i \left(\mathbf{v}(\mathbf{u}^i)(\mathbf{m}) \mathbf{e}_i(\mathbf{f}) - \sum_j DA_j^i(0) \mathbf{u}^j(\mathbf{m}) \mathbf{e}_i(\mathbf{f}) \right) (\mathbf{m}). \quad (7.42)$$

We need $DA_j^i(0)$. Parallel transport depends on the path, but not on the parameterization of the path. From this we can deduce that $DA_j^i(0)$ can be written as one-form fields applied to the vector field \mathbf{v} , as follows.

Introduce B to make the dependence of A s on \mathbf{v} explicit:

$$A_j^i(\delta) = B_j^i(\mathbf{v})(\delta). \quad (7.43)$$

Parallel transport depends on the path but not on the rate along the path. Incrementally, if we scale the vector field \mathbf{v} by ξ ,

$$\frac{d}{d\xi} (B(\mathbf{v})(\delta)) = \frac{d}{d\xi} (B(\xi\mathbf{v})(\delta/\xi)). \quad (7.44)$$

Using the chain rule

$$D(B(\mathbf{v}))(\delta) = \frac{1}{\xi} D(B(\xi\mathbf{v}))\left(\frac{\delta}{\xi}\right), \quad (7.45)$$

so, for $\delta = 0$,

$$\xi D(B(\mathbf{v}))(0) = D(B(\xi\mathbf{v}))(0). \quad (7.46)$$

The scale factor ξ can vary from place to place. So $DA_j^i(0)$ is homogeneous in \mathbf{v} over manifold functions. This is stronger than the homogeneity required by equation (7.7).

The superposition property (equation (7.6)) is true of the ordinary directional derivative of manifold functions. By analogy we require it to be true of directional derivatives of vector fields.

These two properties imply that $DA_j^i(0)$ is a one-form field:

$$DA_j^i(0) = -\varpi_j^i(v), \quad (7.47)$$

where the minus sign is a matter of convention.

As before, we can take a stab at computing the covariant derivative of a vector field by supplying an appropriate transport operator for F in $F \rightarrow \text{directional-derivative}$. Again, this is expanded to a given order with a given coordinate system. These will be unnecessary in the final version.

```
(define (covariant-derivative-vector omega coordsys order)
  (let ((Phi (phi coordsys order)))
    (F->directional-derivative
      (F-parallel omega Phi coordsys)))))

(define (((((F-parallel omega phi coordsys) v) delta) u) f) m)
  (let ((basis (coordinate-system->basis coordsys)))
    (let ((etilde (basis->1form-basis basis))
          (e (basis->vector-basis basis)))
      (let ((m0 (((phi v) (- delta)) m)))
        (let ((Aij (+ (identity-like ((omega v) m0))
                      (* delta (- ((omega v) m0)))))
              (ui ((etilde u) m0)))
          (* ((e f) m) (* Aij ui)))))))
```

So

$$Dg(0) = \sum_i \left(v(u^i)(m) + \sum_j \varpi_j^i(v)(m)u^j(m) \right) e_i(f)(m). \quad (7.48)$$

Thus the covariant derivative is

$$\nabla_v u(f) = \sum_i \left(v(u^i) + \sum_j \varpi_j^i(v)u^j \right) e_i(f). \quad (7.49)$$

The one-form fields ϖ_j^i are called the *Cartan one-forms*, or the *connection one-forms*. They are defined with respect to the basis e .

As a program, the covariant derivative is:⁶

```
(define (((covariant-derivative-vector Cartan) V) U) f)
  (let ((basis (Cartan->basis Cartan))
        (Cartan-forms (Cartan->forms Cartan)))
    (let ((vector-basis (basis->vector-basis basis))
          (1form-basis (basis->1form-basis basis)))
      (let ((u-components (1form-basis U)))
        (* (vector-basis f)
           (+ (V u-components)
              (* (Cartan-forms V) u-components)))))))
```

An important property of $\nabla_v u$ is that it is linear over manifold functions g in the first argument

$$\nabla_{gv} u(f) = g \nabla_v u(f), \quad (7.50)$$

consistent with the fact that the Cartan forms ϖ_j^i share the same property.

Additionally, we can extend the product rule, for any manifold function g and any vector field u :

$$\begin{aligned} \nabla_v(gu)(f) &= \sum_i \left(v(gu^i) + \sum_j \varpi_j^i(v) gu^j \right) e_i(f) \\ &= \sum_i v(g) u^i e_i(f) + g \nabla_v(u)(f) \\ &= (\nabla_v g) u(f) + g \nabla_v(u)(f). \end{aligned} \quad (7.51)$$

An Alternate View

As we did with the Lie derivative (equations 7.18–7.21), we can write the vector field

$$u(f)(m) = \sum_i u^i(m) e_i(f)(m). \quad (7.52)$$

By the extended product rule, equation (7.51), we get:

$$\nabla_v u(f) = \sum_i (v(u^i) e_i(f) + u^i \nabla_v e_i(f)). \quad (7.53)$$

⁶This program is incomplete. It must construct a vector field; it must make a differential operator; and it does not apply to functions or forms.

Because the covariant derivative of a vector field is a vector field we can extract the components of $\nabla_v e_i$ using the dual basis:

$$\varpi_j^i(v) = \tilde{e}^i(\nabla_v e_j). \quad (7.54)$$

This gives an alternate expression for the Cartan one forms. So

$$\nabla_v u(f) = \sum_i \left(v(u^i) + \sum_j \varpi_j^i(v) u^j \right) e_i(f). \quad (7.55)$$

This analysis is parallel to the analysis of the Lie derivative, except that here we have the Cartan form fields ϖ_j^i and there we had Δ_j^i , which are not form fields.

Notice that the Cartan forms appear here (equation 7.53) in terms of the covariant derivatives of the basis vectors. By contrast, in the first derivation (see equation 7.42) the Cartan forms appear as the derivatives of the linear forms that accomplish the parallel transport of the coefficients.

The Cartan forms can be constructed from the dual basis one-forms:

$$\varpi_j^i(v)(m) = \sum_k \Gamma_{jk}^i(m) \tilde{e}^k(v)(m). \quad (7.56)$$

The connection coefficient functions Γ_{jk}^i are called the *Christoffel coefficients* (traditionally called *Christoffel symbols*).⁷ Making use of the structures,⁸ the Cartan forms are

$$\varpi(v) = \Gamma \tilde{e}(v). \quad (7.57)$$

Conversely, the Christoffel coefficients may be obtained from the Cartan forms

$$\Gamma_{jk}^i = \varpi_j^i(e_k). \quad (7.58)$$

⁷This terminology may be restricted to the case in which the basis is a coordinate basis.

⁸The structure of the Cartan forms ϖ together with this equation forces the shape of the Christoffel coefficient structure.

Covariant Derivative of One-Form Fields

The covariant derivative of a vector field induces a compatible covariant derivative for a one-form field. Because the application of a one-form field to a vector field yields a manifold function, we can evaluate the covariant derivative of such an application. Let τ be a one-form field and w be a vector field. Then

$$\begin{aligned}\nabla_v(\tau(w)) &= v \left(\sum_j \tau_j w^j \right) \\ &= \sum_j (v(\tau_j) w^j + \tau_j v(w^j)) \\ &= \sum_j \left(v(\tau_j) w^j + \tau_j \left(\tilde{e}^j (\nabla_v w) - \sum_k \varpi_k^j(v) w^k \right) \right) \\ &= \sum_j \left(v(\tau_j) w^j - \tau_j \sum_k \varpi_k^j(v) w^k \right) + \tau(\nabla_v w) \\ &= \sum_j \left(v(\tau_j) \tilde{e}^j - \tau_j \sum_k \varpi_k^j(v) \tilde{e}^k \right) (w) + \tau(\nabla_v w).\end{aligned}$$

So if we define the covariant derivative of a one-form field to be

$$\nabla_v(\tau) = \sum_k \left(v(\tau_k) - \sum_j \tau_j \varpi_k^j(v) \right) \tilde{e}^k, \quad (7.59)$$

then the generalized product rule holds:

$$\nabla_v(\tau(u)) = (\nabla_v \tau)(u) + \tau(\nabla_v u). \quad (7.60)$$

Alternatively, assuming the generalized product rule forces the definition of covariant derivative of a one-form field.

As a program this is

```
(define (((covariant-derivative-1form Cartan) V) tau) U)
  (let ((nabla_V ((covariant-derivative-vector Cartan) V)))
    (- (V (tau U)) (tau (nabla_V U)))))
```

This program extends naturally to higher-rank form fields:

```
(define (((covariant-derivative-form Cartan) V) tau) vs)
  (let ((k (get-rank tau))
        (nabla_V ((covariant-derivative-vector Cartan) V)))
    (- (V (apply tau vs))
        (sigma (lambda (i)
                  (apply tau
                         (list-with-substituted-coord vs i
                           (nabla_V (list-ref vs i))))))
        0 (- k 1)))))
```

Change of Basis

The basis-independence of the covariant derivative implies a relationship between the Cartan forms in one basis and the equivalent Cartan forms in another basis. Recall (equation 4.13) that the basis vector fields of two bases are always related by a linear transformation. Let J be the matrix of coefficient functions and let e and e' be down tuples of basis vector fields. Then

$$e(f) = e'(f)J. \quad (7.61)$$

We want the covariant derivative to be independent of basis. This will determine how the connection transforms with a change of basis:

$$\begin{aligned} \nabla_v u(f) &= \sum_i e_i(f) \left(v(u^i) + \sum_j \varpi_j^i(v) u^j \right) \\ &= \sum_{ijk} e'_i(f) J^i_j \left(v((J^{-1})_k^j (u')^k) + \sum_l \varpi_k^j(v) (J^{-1})_l^k (u')^l \right) \\ &= \sum_i e'_i(f) \left(v((u')^i) + \sum_{jk} J^i_j v((J^{-1})_k^j) (u')^k \right. \\ &\quad \left. + \sum_{jkl} J^i_j \varpi_k^j(v) (J^{-1})_l^k (u')^l \right) \\ &= \sum_i e'_i(f) \left(v((u')^i) + \sum_j (\varpi')_j^i(v) (u')^j \right). \end{aligned} \quad (7.62)$$

The last line of equation (7.62) gives the formula for the covariant derivative we would have written down naturally in the primed coordinates; comparing with the next-to-last line, we see that

$$\varpi'(v) = Jv(J^{-1}) + J\varpi(v)J^{-1}. \quad (7.63)$$

This transformation rule is weird. It is not a linear transformation of ϖ because the first term is an offset that depends on v . So it is not required that $\varpi' = 0$ when $\varpi = 0$. Thus ϖ is not a tensor field. See Appendix C.

We can write equation (7.61) in terms of components

$$e_i(f) = \sum_j e'_j(f) J^j{}_i. \quad (7.64)$$

Let $K = J^{-1}$, so $\sum_j K^i{}_j(m) J^j{}_k(m) = \delta^i{}_k$. Then

$$\varpi'^i_l(v) = \sum_j J^i{}_j v(K^j{}_l) + \sum_{jk} J^i{}_j \varpi^j_k(v) K^k{}_l. \quad (7.65)$$

The transformation rule for ϖ is implemented in the following program:

```
(define (Cartan-transform Cartan basis-prime)
  (let ((basis (Cartan->basis Cartan))
        (forms (Cartan->forms Cartan))
        (prime-dual-basis (basis->1form-basis basis-prime))
        (prime-vector-basis (basis->vector-basis basis-prime)))
    (let ((vector-basis (basis->vector-basis basis))
          (1form-basis (basis->1form-basis basis)))
      (let ((J-inv (s:map/r 1form-basis prime-vector-basis))
            (J (s:map/r prime-dual-basis vector-basis)))
        (let ((omega-prime-forms
              (procedure->1form-field
               (lambda (v)
                 (+ (* J (v J-inv))
                    (* J (* (forms v) J-inv)))))))
          (make-Cartan omega-prime-forms basis-prime))))))
```

The `s:map/r` procedure constructs a tuple of the same shape as its second argument whose elements are the result of applying the first argument to the corresponding elements of the second argument.

We can illustrate that the covariant derivative is independent of the coordinate system in a simple case, using rectangular and polar coordinates in the plane.⁹ We can choose Christoffel coefficients for rectangular coordinates that are all zero:¹⁰

```
(define R2-rect-Christoffel
  (make-Christoffel
    (let ((zero (lambda (m) 0)))
      (down (down (up zero zero)
                  (up zero zero))
            (down (up zero zero)
                  (up zero zero))))))
  R2-rect-basis))
```

With these Christoffel coefficients, parallel transport preserves the components relative to the rectangular basis. This corresponds to our usual notion of parallel in the plane. We will see later in Chapter 9 that these Christoffel coefficients are a natural choice for the plane. From these we obtain the Cartan form:¹¹

```
(define R2-rect-Cartan
  (Christoffel->Cartan R2-rect-Christoffel))
```

And from equation (7.63) we can get the corresponding Cartan form for polar coordinates:

```
(define R2-polar-Cartan
  (Cartan-transform R2-rect-Cartan R2-polar-basis))
```

⁹We will need a few definitions:

```
(define R2-rect-basis (coordinate-system->basis R2-rect))
(define R2-polar-basis (coordinate-system->basis R2-polar))
(define-coordinates (up x y) R2-rect)
(define-coordinates (up r theta) R2-polar)
```

¹⁰Since the Christoffel coefficients are basis-dependent they are packaged with the basis.

¹¹The code for making the Cartan forms is as follows:

```
(define (Christoffel->Cartan Christoffel)
  (let ((basis (Christoffel->basis Christoffel))
        (Christoffel-symbols (Christoffel->symbols Christoffel)))
    (make-Cartan
      (* Christoffel-symbols (basis->1form-basis basis))
      basis)))
```

The vector field $\partial/\partial\theta$ generates a rotation in the plane (the same as `circular`). The covariant derivative with respect to $\partial/\partial x$ of $\partial/\partial\theta$ applied to an arbitrary manifold function is:

```
(define circular (- (* x d/dy) (* y d/dx)))

(define f (literal-manifold-function 'f-rect R2-rect))
(define R2-rect-point ((point R2-rect) (up 'x0 'y0)))

(((covariant-derivative R2-rect-Cartan) d/dx)
  circular)
  f)
R2-rect-point)
(((partial 1) f-rect) (up x0 y0))
```

Note that this is the same thing as $\partial/\partial y$ applied to the function:

```
((d/dy f) R2-rect-point)
(((partial 1) f-rect) (up x0 y0)))
```

In rectangular coordinates, where the Christoffel coefficients are zero, the covariant derivative $\nabla_u v$ is the vector whose coefficients are obtained by applying u to the coefficients of v . Here, only one coefficient of $\partial/\partial\theta$ depends on x , the coefficient of $\partial/\partial y$, and it depends linearly on x . So $\nabla_{\partial/\partial x} \partial/\partial\theta = \partial/\partial y$. (See figure 7.1.)

Note that we get the same answer if we use polar coordinates to compute the covariant derivative:

```
((((covariant-derivative R2-polar-Cartan) d/dx) J) f)
R2-rect-point)
(((partial 1) f-rect) (up x0 y0)))
```

In rectangular coordinates the Christoffel coefficients are all zero; in polar coordinates there are nonzero coefficients, but the value of the covariant derivative is the same. In polar coordinates the basis elements vary with position, and the Christoffel coefficients compensate for this.

Of course, this is a pretty special situation. Let's try something more general:

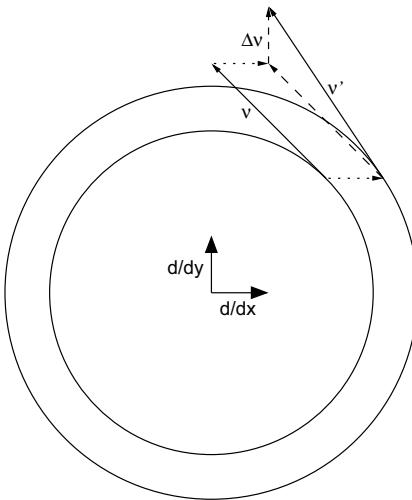


Figure 7.1 If v and v' are “arrow” representations of vectors in the circular field and we parallel-transport v in the $\partial/\partial x$ direction, then the difference between v' and the parallel transport of v is in the $\partial/\partial y$ direction.

```
(define V (literal-vector-field 'V-rect R2-rect))
(define W (literal-vector-field 'W-rect R2-rect))

(((((- (covariant-derivative R2-rect-Cartan)
       (covariant-derivative R2-polar-Cartan))
      V)
     W)
    f)
   R2-rect-point)
  0
```

7.3 Parallel Transport

We have defined parallel transport of a vector field along integral curves of another vector field. But not all paths are integral curves of a vector field. For example, paths that cross themselves are not integral curves of any vector field.

Here we extend the idea of parallel transport of a stick to make sense for arbitrary paths on the manifold. Any path can be written as a map γ from the real-line manifold to the manifold M . We

construct a vector field over the map u_γ by parallel-transporting the stick to all points on the path γ .

For any path γ there are locally directional derivatives of functions on M defined by tangent vectors to the curve. The vector over the map $w_\gamma = d\gamma(\partial/\partial t)$ is a directional derivative of functions on the manifold M along the path γ .

Our goal is to determine the equations satisfied by the vector field over the map u_γ . Consider the parallel-transport $F_\delta^{w_\gamma} u_\gamma$.¹² So a vector field u_γ is parallel-transported to itself if and only if $u_\gamma = F_\delta^{w_\gamma} u_\gamma$. Restricted to a path, the equation analogous to equation (7.40) is

$$g(\delta) = \sum_i \left(u^i(t) - \sum_j A_j^i(\delta) u^j(t - \delta) \right) e_i^\gamma(f)(t), \quad (7.66)$$

where the coefficient function u^i is now a function on the real-line parameter manifold and where we have rewritten the basis as a basis over the map γ .¹³ Here $g(\delta) = 0$ if u_γ is parallel-transported into itself.

Taking the derivative and setting $\delta = 0$ we find

$$0 = \sum_i \left(Du^i(t) + \sum_j {}^\gamma \varpi_j^i(w_\gamma)(t) u^j(t) \right) e_i^\gamma(f)(t). \quad (7.67)$$

But this implies that

$$0 = Du^i(t) + \sum_j {}^\gamma \varpi_j^i(w_\gamma)(t) u^j(t), \quad (7.68)$$

an ordinary differential equation in the coefficients of u_γ .

¹²The argument w_γ makes sense because our parallel-transport operator never depended on the vector field tangent to the integral curve existing off of the curve. Because the connection is a form field (see equation 7.47), it does not depend on the value of its vector argument anywhere except at the point where it is being evaluated.

The argument u_γ is more difficult. We must modify equation (7.37):

$$F_\delta^{w_\gamma} u_\gamma(f)(t) = \sum_{i,j} A_j^i(\delta) u^j(t - \delta) e_i^\gamma(f)(t).$$

¹³You may have noticed that t and t appear here. The real-line manifold point t has coordinate t .

We can abstract these equations of parallel transport by inventing a covariant derivative over a map. We also generalize the time line to a source manifold N .

$$\nabla_v^\gamma u_\gamma(f)(n) = \sum_i \left(v(u^i)(n) + \sum_j {}^\gamma \varpi_j^i(d\gamma(v))(n) u^j(n) \right) e_i^\gamma(f)(n), \quad (7.69)$$

where the map $\gamma : N \rightarrow M$, v is a vector on N , u_γ is a vector over the map γ , f is a function on M , and n is a point in N . Indeed, if w is a vector field on M , f is a manifold function on M , and if $d\gamma(v) = w_\gamma$ then

$$\nabla_v^\gamma u_\gamma(f)(n) = \nabla_w u(f)(\gamma(n)). \quad (7.70)$$

This is why we are justified in calling ∇_v^γ a covariant derivative.

Respecializing the source manifold to the real line, we can write the equations governing the parallel transport of u_γ as

$$\nabla_{\partial/\partial t}^\gamma u_\gamma = 0. \quad (7.71)$$

We obtain the set of differential equations (7.68) for the coordinates of u_γ , the vector over the map γ , that is parallel-transported along the curve γ :

$$Du^i(t) + \sum_j {}^\gamma \varpi_j^i(d\gamma(\partial/\partial t))(t) u^j(t) = 0. \quad (7.72)$$

Expressing the Cartan forms in terms of the Christoffel coefficients we obtain

$$Du^i(t) + \sum_{j,k} \Gamma_{jk}^i(\gamma(t)) D\sigma^k(t) u^j(t) = 0 \quad (7.73)$$

where $\sigma = \chi_M \circ \gamma \circ \chi_R^{-1}$ are the coordinates of the path (χ_M and χ_R are the coordinate functions for M and the real line).

On a Sphere

Let's figure out what the equations of parallel transport of u_γ , an arbitrary vector over the map γ , along an arbitrary path γ on a sphere are. We start by constructing the necessary manifold.

```
(define sphere (make-manifold S^2 2 3))
(define S2-spherical
  (coordinate-system-at 'spherical 'north-pole sphere))
(define S2-basis
  (coordinate-system->basis S2-spherical))
```

We need the path γ , which we represent as a map from the real line to M , and w , the parallel-transported vector over the map:

```
(define gamma
  (compose (point S2-spherical)
    (up (literal-function 'alpha)
      (literal-function 'beta)))
  (chart R1-rect)))
```

where `alpha` is the colatitude and `beta` is the longitude.

We also need an arbitrary vector field `u_gamma` over the map `gamma`. To make this we multiply the structure of literal component functions by the vector basis structure.

```
(define basis-over-gamma
  (basis->basis-over-map gamma S2-basis))

(define u_gamma
  (* (up (compose (literal-function 'u^0)
    (chart R1-rect))
  (compose (literal-function 'u^1)
    (chart R1-rect)))
  (basis->vector-basis basis-over-gamma)))
```

We specify a connection by giving the Christoffel coefficients.¹⁴

```
(define S2-Christoffel
  (make-Christoffel
    (let ((zero (lambda (point) 0)))
      (down (down (up zero zero)
        (up zero (/ 1 (tan theta))))
      (down (up zero (/ 1 (tan theta)))
        (up (- (* (sin theta) (cos theta))) zero))))
    S2-basis))

(define sphere-Cartan (Christoffel->Cartan S2-Christoffel))
```

¹⁴We will show later that these Christoffel coefficients are a natural choice for the sphere.

Finally, we compute the residual of the equation (7.71) that governs parallel transport for this situation:¹⁵

```
(define-coordinates t R1-rect)

(s:map/r
(lambda (omega)
((omega
  (((covariant-derivative sphere-Cartan gamma)
    d/dt)
   u_gamma))
 ((point R1-rect) 'tau)))
(basis->1form-basis basis-over-gamma))
(up (+ (* -1
            (sin (alpha tau))
            (cos (alpha tau))
            ((D beta) tau)
            (u^1 tau))
            ((D u^0) tau))
      (/ (+ (* (u^0 tau) (cos (alpha tau)) ((D beta) tau))
            (* ((D alpha) tau) (cos (alpha tau)) (u^1 tau))
            (* ((D u^1) tau) (sin (alpha tau))))
         (sin (alpha tau))))
```

Thus the equations governing the evolution of the components of the transported vector are:

$$\begin{aligned} Du^0(\tau) &= \sin(\alpha(\tau)) \cos(\alpha(\tau)) D\beta(\tau) u^1(\tau), \\ Du^1(\tau) &= -\frac{\cos(\alpha(\tau))}{\sin(\alpha(\tau))} (D\beta(\tau) u^0(\tau) + D\alpha(\tau) u^1(\tau)). \end{aligned} \quad (7.74)$$

These equations describe the transport on a sphere, but more generally they look like

$$Du(\tau) = f(\sigma(\tau), D\sigma(\tau)) u(\tau), \quad (7.75)$$

where σ is the tuple of the coordinates of the path on the manifold and u is the tuple of the components of the vector. The equation is linear in u and is driven by the path σ , as in a variational equation.

¹⁵If we give `covariant-derivative` an extra argument, in addition to the Cartan form, the covariant derivative treats the extra argument as a map and transforms the Cartan form to work over the map.

We now set this up for numerical integration. Let $s(t) = (t, u(t))$ be a state tuple, combining the time and the coordinates of u_γ at that time. Then we define g :

$$g(s(t)) = Ds(t) = (1, Du(t)), \quad (7.76)$$

where $Du(t)$ is the tuple of right-hand sides of equation (7.72).

On a Great Circle

We illustrate parallel transport in a case where we should know the answer: we carry a vector along a great circle of a sphere. Given a path and Cartan forms for the manifold we can produce a state derivative suitable for numerical integration. Such a state derivative takes a state and produces the derivative of the state.

```
(define (g gamma Cartan)
  (let ((omega
         ((Cartan->forms
           (Cartan->Cartan-over-map Cartan gamma))
          ((differential gamma) d/dt))))
    (define ((the-state-derivative) state)
      (let ((t ((point R1-rect) (ref state 0)))
            (u (ref state 1)))
        (up 1 (* -1 (omega t) u)))
      the-state-derivative)))
```

The path on the sphere will be the target of a map from the real line. We choose one that starts at the origin of longitudes on the equator and follows the great circle that makes a given tilt angle with the equator.

```
(define ((transform tilt) coords)
  (let ((colat (ref coords 0))
        (long (ref coords 1)))
    (let ((x (* (sin colat) (cos long)))
          (y (* (sin colat) (sin long)))
          (z (cos colat)))
      (let ((vp ((rotate-x tilt) (up x y z))))
        (let ((colatp (acos (ref vp 2)))
              (longp (atan (ref vp 1) (ref vp 0))))
          (up colatp longp))))))
```

```
(define (tilted-path tilt)
  (define (coords t)
    ((transform tilt) (up :pi/2 t)))
  (compose (point S2-spherical)
            coords
            (chart R1-rect)))
```

A southward pointing vector, with components (up 1 0), is transformed to an initial vector for the tilted path by multiplying by the derivative of the tilt transform at the initial point. We then parallel transport this vector by numerically integrating the differential equations. In this example we tilt by 1 radian, and we advance for $\pi/2$ radians. In this case we know the answer: by advancing by $\pi/2$ we walk around the circle a quarter of the way and at that point the transported vector points south:

```
((state-advancer (g (tilted-path 1) sphere-Cartan))
 (up 0 (* ((D (transform 1)) (up :pi/2 0)) (up 1 0)))
 pi/2)
(up 1.5707963267948957
 (up .9999999999997626 7.376378522558262e-13))
```

However, if we transport by 1 radian rather than $\pi/2$, the numbers are not so pleasant, and the transported vector no longer points south:

```
((state-advancer (g (tilted-path 1) sphere-Cartan))
 (up 0 (* ((D (transform 1)) (up :pi/2 0)) (up 1 0)))
 1)
(up 1. (up .7651502649360408 .9117920272006472))
```

But the transported vector can be obtained by tilting the original southward-pointing vector after parallel-transporting along the equator.¹⁶

```
(* ((D (transform 1)) (up :pi/2 1)) (up 1 0))
(up .7651502649370375 .9117920272004736)
```

¹⁶A southward-pointing vector remains southward-pointing when it is parallel-transported along the equator. To do this we do not have to integrate the differential equations, because we know the answer.

7.4 Geodesic Motion

In geodesic motion the velocity vector is parallel-transported by itself. Recall (equation 6.9) that the velocity is the differential of the vector $\partial/\partial t$ over the map γ . The equation of geodesic motion is¹⁷

$$\nabla_{\partial/\partial t}^\gamma d\gamma(\partial/\partial t) = 0. \quad (7.78)$$

In coordinates, this is

$$D^2\sigma^i(t) + \sum_{jk} \Gamma_{jk}^i(\gamma(t)) D\sigma^j(t) D\sigma^k(t) = 0, \quad (7.79)$$

where $\sigma(t)$ is the coordinate path corresponding to the manifold path γ .

For example, let's consider geodesic motion on the surface of a unit sphere. We let `gamma` be a map from the real line to the sphere, with colatitude `alpha` and longitude `beta`, as before. The geodesic equation is:

```
(show-expression
 (((((covariant-derivative sphere-Cartan gamma)
      d/dt)
     ((differential gamma) d/dt))
   (chart S2-spherical))
  ((point R1-rect) 't0)))
```

$$\begin{pmatrix} -\cos(\alpha(t_0)) \sin(\alpha(t_0)) (D\beta(t_0))^2 + D^2\alpha(t_0) \\ \frac{2D\beta(t_0) \cos(\alpha(t_0)) D\alpha(t_0)}{\sin(\alpha(t_0))} + D^2\beta(t_0) \end{pmatrix}$$

¹⁷The equation of a geodesic path is often said to be

$$\nabla_v v = 0, \quad (7.77)$$

but this is nonsense. The geodesic equation is a constraint on the path, but the path does not appear in this equation. Further, the velocity along a path is not a vector field, so it cannot appear in either argument to the covariant derivative.

What is true is that a vector field v all of whose integral curves are geodesics satisfies equation (7.77).

The geodesic equation is the same as the Lagrange equation for free motion constrained to the surface of the unit sphere. The Lagrangian for motion on the sphere is the composition of the free-particle Lagrangian and the state transformation induced by the coordinate constraint:¹⁸

```
(define (Lfree s)
  (* 1/2 (square (velocity s)))))

(define (sphere->R3 s)
  (let ((q (coordinate s)))
    (let ((theta (ref q 0)) (phi (ref q 1)))
      (up (* (sin theta) (cos phi))
           (* (sin theta) (sin phi))
           (cos theta)))))

(define Lsphere
  (compose Lfree (F->C sphere->R3)))
```

Then the Lagrange equations are:

```
(show-expression
 (((Lagrange-equations Lsphere)
   (up (literal-function 'alpha)
        (literal-function 'beta)))
  't))
```

$$\begin{bmatrix} - (D\beta(t))^2 \sin(\alpha(t)) \cos(\alpha(t)) + D^2\alpha(t) \\ 2D\alpha(t) D\beta(t) \sin(\alpha(t)) \cos(\alpha(t)) + D^2\beta(t) (\sin(\alpha(t)))^2 \end{bmatrix}$$

The Lagrange equations are true of the same paths as the geodesic equations. The second Lagrange equation is the second geodesic equation multiplied by $(\sin(\alpha(t)))^2$, and the Lagrange equations are arranged in a down tuple, whereas the geodesic equations are arranged in an up tuple.¹⁹ The two systems are equivalent unless $\alpha(t) = 0$, where the coordinate system is singular.

¹⁸The method of formulating a system with constraints by composing a free system with the state-space coordinate transformation that represents the constraints can be found in [19], Section 1.6.3. The procedure **F->C** takes a coordinate transformation and produces a corresponding transformation of Lagrangian state.

¹⁹The geodesic equations and the Lagrange equations are related by a contraction with the metric.

Exercise 7.1: Hamiltonian Evolution

We have just seen that the Lagrange equations for the motion of a free particle constrained to the surface of a sphere determine the geodesics on the sphere. We can investigate this phenomenon in the Hamiltonian formulation. The Hamiltonian is obtained from the Lagrangian by a Legendre transformation:

```
(define Hsphere
  (Lagrangian->Hamiltonian Lsphere))
```

We can get the coordinate representation of the Hamiltonian vector field as follows:

```
((phase-space-derivative Hsphere)
  (up 't (up 'theta 'phi) (down 'p_theta 'p_phi)))
(up 1
  (up p_theta
    (/ p_phi (expt (sin theta) 2)))
  (down (/ (* (expt p_phi 2) (cos theta))
            (expt (sin theta) 3))
         0)))
```

The state space for Hamiltonian evolution has five dimensions: time, two dimensions of position on the sphere, and two dimensions of momentum:

```
(define state-space
  (make-manifold R^n 5))
(define states
  (coordinate-system-at 'rectangular 'origin state-space))
(define-coordinates
  (up t (up theta phi) (down p_theta p_phi))
  states)
```

So now we have coordinate functions and the coordinate-basis vector fields and coordinate-basis one-form fields.

- Define the Hamiltonian vector field as a linear combination of these fields.
- Obtain the first few terms of the Taylor series for the evolution of the coordinates (θ, ϕ) by exponentiating the Lie derivative of the Hamiltonian vector field.

Exercise 7.2: Lie Derivative and Covariant Derivative

How are the Lie derivative and the covariant derivative related?

- Prove that for every vector field there exists a connection such that the covariant derivative for that connection and the given vector field is equivalent to the Lie derivative with respect to that vector field.
- Show that there is no connection that for every vector field makes the Lie derivative the same as the covariant derivative with the chosen connection.

8

Curvature

If the intrinsic curvature of a manifold is not zero, a vector parallel-transported around a small loop will end up different from the vector that started. We saw the consequence of this before, on page 1 and on page 93. The Riemann tensor encapsulates this idea.

The Riemann curvature operator is

$$\mathcal{R}(w, v) = [\nabla_w, \nabla_v] - \nabla_{[w, v]}. \quad (8.1)$$

The traditional Riemann tensor is¹

$$R(\omega, u, w, v) = \omega((\mathcal{R}(w, v))(u)), \quad (8.2)$$

where ω is a one-form field that measures the incremental change in the vector field u caused by parallel-transporting it around the loop defined by the vector fields w and v . R allows us to compute the *intrinsic curvature* of a manifold at a point.

The Riemann curvature is computed by

```
(define ((Riemann-curvature nabla) w v)
  (- (commutator (nabla w) (nabla v))
      (nabla (commutator w v))))
```

The `Riemann-curvature` procedure is parameterized by the relevant covariant-derivative operator `nabla`, which implements ∇ . The `nabla` is itself dependent on the connection, which provides the details of the local geometry. The same `Riemann-curvature` procedure works for ordinary covariant derivatives and for covariant derivatives over a map. Given two vector fields, the result of `((Riemann-curvature nabla) w v)` is a procedure that takes a vector field and produces a vector field so we can implement the Riemann tensor as

¹ [11], [4], and [14] use our definition. [20] uses a different convention for the order of arguments and a different sign. See Appendix C for a definition of tensors.

```
(define ((Riemann nabla) omega u w v)
  (omega (((Riemann-curvature nabla) w v) u)))
```

So, for example,²

```
((((Riemann (covariant-derivative sphere-Cartan))
  dphi d/dtheta d/dphi d/dtheta)
  ((point S2-spherical) (up 'theta0 'phi0)))
  1
```

Here we have computed the ϕ component of the result of carrying a $\partial/\partial\theta$ basis vector around the parallelogram defined by $\partial/\partial\phi$ and $\partial/\partial\theta$. The result shows a net rotation in the ϕ direction.

Most of the sixteen coefficients of the Riemann tensor for the sphere are zero. The following are the nonzero coefficients:

$$\begin{aligned} R \left(d\theta, \frac{\partial}{\partial\phi}, \frac{\partial}{\partial\theta}, \frac{\partial}{\partial\phi} \right) (\chi^{-1}(q^\theta, q^\phi)) &= (\sin(q^\theta))^2, \\ R \left(d\theta, \frac{\partial}{\partial\phi}, \frac{\partial}{\partial\phi}, \frac{\partial}{\partial\theta} \right) (\chi^{-1}(q^\theta, q^\phi)) &= -(\sin(q^\theta))^2, \\ R \left(d\phi, \frac{\partial}{\partial\theta}, \frac{\partial}{\partial\theta}, \frac{\partial}{\partial\phi} \right) (\chi^{-1}(q^\theta, q^\phi)) &= -1, \\ R \left(d\phi, \frac{\partial}{\partial\theta}, \frac{\partial}{\partial\phi}, \frac{\partial}{\partial\theta} \right) (\chi^{-1}(q^\theta, q^\phi)) &= 1. \end{aligned} \quad (8.3)$$

8.1 Explicit Transport

We will show that the result of the Riemann calculation of the change in a vector, as we traverse a loop, is what we get by explicitly calculating the transport. The coordinates of the vector to be transported are governed by the differential equations (see equation 7.72)

$$Du^i(t) = - \sum_j \varpi_j^i(v)(\chi^{-1}(\sigma(t))) u^j(t) \quad (8.4)$$

²The connection specified by `sphere-Cartan` is defined on page 107.

and the coordinates as a function of time, $\sigma = \chi \circ \gamma \circ \chi_R^{-1}$, of the path γ , are governed by the differential equations³

$$D\sigma(t) = v(\chi)(\chi^{-1}(\sigma(t))). \quad (8.5)$$

We have to integrate these equations (8.4, 8.5) together to transport the vector over the map u_γ a finite distance along the vector field v .

Let $s(t) = (\sigma(t), u(t))$ be a state tuple, combining σ the coordinates of γ , and u the coordinates of u_γ . Then

$$Ds(t) = (D\sigma(t), Du(t)) = g(s(t)), \quad (8.6)$$

where g is the tuple of right-hand sides of equations (8.4, 8.5).

The differential equations describing the evolution of a function h of state s along the state path are

$$D(h \circ s) = (Dh \circ s)(g \circ s) = L_g h \circ s, \quad (8.7)$$

defining the operator L_g .

Exponentiation gives a finite evolution:⁴

$$h(s(t + \epsilon)) = (e^{\epsilon L_g} h)(s(t)). \quad (8.8)$$

The finite parallel transport of the vector with components u is

$$u(t + \epsilon) = (e^{\epsilon L_g} U)(s(t)), \quad (8.9)$$

where the selector $U(\sigma, u) = u$, and the initial state is $s(t) = (\sigma(t), u(t))$.

Consider parallel-transporting a vector u around a parallelogram defined by two coordinate-basis vector fields w and v . The vector u is really a vector over a map, where the map is the parametric curve describing our parallelogram. This map is implicitly defined in terms of the vector fields w and v . Let g_w and g_v be the right-hand sides of the differential equations for parallel transport

³The map γ takes points on the real line to points on the target manifold. The chart χ gives coordinates of points on the target manifold while χ_R gives a time coordinate on the real line.

⁴The series may not converge for large increments in the independent variable. In this case it is appropriate to numerically integrate the differential equations directly.

along w and v respectively. Then evolution along w for interval ϵ , then along v for interval ϵ , then reversing w , and reversing v , brings σ back to where it started to second order in ϵ .

The state $s = (\sigma, u)$ after transporting s_0 around the loop is⁵

$$\begin{aligned} & (e^{-\epsilon L_{gv}} I) \circ (e^{-\epsilon L_{gw}} I) \circ (e^{\epsilon L_{gv}} I) \circ (e^{\epsilon L_{gw}} I)(s_0) \\ &= (e^{\epsilon L_{gw}} e^{\epsilon L_{gv}} e^{-\epsilon L_{gw}} e^{-\epsilon L_{gv}} I)(s_0) \\ &= (e^{\epsilon^2 [L_{gw}, L_{gv}] + \dots} I)(s_0). \end{aligned} \quad (8.10)$$

So the lowest-order change in the transported vector is

$$\epsilon^2 U(([L_{gw}, L_{gv}] I)(s_0)), \quad (8.11)$$

where $U(\sigma, u) = u$.

However, if w and v do not commute, the indicated loop does not bring σ back to the starting point, to second order in ϵ . We must account for the commutator. (See figure 4.2.) In the general case the lowest order change in the transported vector is

$$\epsilon^2 U((([L_{gw}, L_{gv}] - L_{g_{[w,v]}}) I)(s_0)). \quad (8.12)$$

This is what the Riemann tensor computation gives, scaled by ϵ^2 .

Verification in Two Dimensions

We can verify this in two dimensions. We need to make the structure representing a state:

```
(define (make-state sigma u) (vector sigma u))

(define (Sigma state) (ref state 0))

(define (U-select state) (ref state 1))
```

⁵ The parallel-transport operators are evolution operators, and therefore descend into composition:

$$e^A(F \circ G) = F \circ (e^A G),$$

for any state function G and any compatible F . As a consequence, we have the following identity:

$$e^A e^B I = e^A ((e^B I) \circ I) = (e^B I) \circ (e^A I),$$

where I is the identity function on states.

And now we get to the meat of the matter: First we find the rate of change of the components of the vector u as we carry it along the vector field v .⁶

```
(define ((Du v) state)
  (let ((CF (Cartan->forms general-Cartan-2)))
    (* -1
       ((CF v) (Chi-inverse (Sigma state)))
       (U-select state))))
```

We also need to determine the rate of change of the coordinates of the integral curve of v .

```
(define ((Dsigma v) state)
  ((v Chi) (Chi-inverse (Sigma state))))
```

Putting these together to make the derivative of the state vector

```
(define ((g v) state)
  (make-state ((Dsigma v) state) ((Du v) state)))
```

gives us just what we need to construct the differential operator for evolution of the combined state:

```
(define (L v)
  (define ((l h) state)
    (* ((D h) state) ((g v) state)))
  (make-operator l))
```

So now we can demonstrate that the lowest-order change resulting from explicit parallel transport of a vector around an infinitesimal loop is what is computed by the Riemann curvature.

⁶ The setup for this experiment is a bit complicated. We need to make a manifold with a general connection.

```
(define Chi-inverse (point R2-rect))
(define Chi (chart R2-rect))
```

We now make the Cartan forms from the most general 2-dimensional Christoffel coefficient structure:

```
(define general-Cartan-2
  (Christoffel->Cartan
    (literal-Christoffel-2 'Gamma R2-rect)))
```

```
(let ((U (literal-vector-field 'U-rect R2-rect))
      (W (literal-vector-field 'W-rect R2-rect))
      (V (literal-vector-field 'V-rect R2-rect))
      (sigma (up 'sigma0 'sigma1)))
  (let ((nabla (covariant-derivative general-Cartan-2))
        (m (Chi-inverse sigma)))
    (let ((s (make-state sigma ((U Chi) m))))
      (- (((- (commutator (L V) (L W))
                (L (commutator V W)))
            U-select)
           s)
          (((((Riemann-curvature nabla) W V) U) Chi) m)))))
  (up 0 0))
```

Geometrically

The explicit transport above was done with differential equations operating on a state consisting of coordinates and components of the vector being transported. We can simplify this so that it is entirely built on manifold objects, eliminating the state. After a long algebraic story we find that

$$\begin{aligned} & ((\mathcal{R}(w, v))(u))(f) \\ &= e(f) \{ (w(\varpi(v)) - v(\varpi(w)) - \varpi([w, v])) \tilde{e}(u) \\ & \quad + \varpi(w)\varpi(v)\tilde{e}(u) - \varpi(v)\varpi(w)\tilde{e}(u) \} \end{aligned} \quad (8.13)$$

or as a program:

```
(define ((((curvature-from-transport Cartan) w v) u) f)
  (let* ((CF (Cartan->forms Cartan))
         (basis (Cartan->basis Cartan))
         (fi (basis->1form-basis basis))
         (ei (basis->vector-basis basis)))
    (* (ei f)
       (+ (* (- (- (w (CF v)) (v (CF w)))
                  (CF (commutator w v)))
              (fi u))
            (- (* (CF w) (* (CF v) (fi u)))
               (* (CF v) (* (CF w) (fi u))))))))
```

This computes the same operator as the traditional Riemann curvature operator:

```
(define (test coordsys Cartan)
  (let ((m (typical-point coordsys))
        (u (literal-vector-field 'u-coord coordsys))
        (w (literal-vector-field 'w-coord coordsys))
        (v (literal-vector-field 'v-coord coordsys))
        (f (literal-manifold-function 'f-coord coordsys)))
    (let ((nabla (covariant-derivative Cartan)))
      (- (((curvature-from-transport Cartan) w v) u) f) m)
      (((Riemann-curvature nabla) w v) u) f) m)))))

(test R2-rect general-Cartan-2)
0

(test R2-polar general-Cartan-2)
0
```

Terms of the Riemann Curvature

Since the Riemann curvature is defined as in equation (8.1),

$$\mathcal{R}(w, v) = [\nabla_w, \nabla_v] - \nabla_{[w, v]}, \quad (8.14)$$

it is natural⁷ to identify these terms with the corresponding terms in

$$(([L_{g_w}, L_{g_v}] - L_{g_{[w, v]}})U)(s_0). \quad (8.15)$$

Unfortunately, this does not work, as demonstrated below:

```
(let ((U (literal-vector-field 'U-rect R2-rect))
      (V (literal-vector-field 'V-rect R2-rect))
      (W (literal-vector-field 'W-rect R2-rect))
      (nabla (covariant-derivative general-Cartan-2))
      (sigma (up 'sigma0 'sigma1)))
  (let ((m (Chi-inverse sigma)))
    (let ((s (make-state sigma ((U Chi) m))))
      (- (((commutator (L W) (L V)) U-select) s)
          (((commutator (nabla W) (nabla V)) U) Chi)
          m)))))

a nonzero mess
```

⁷People often say “Geodesic evolution is exponentiation of the covariant derivative.” But this is wrong. The evolution is by exponentiation of L_g .

The obvious identification does not work, but neither does the other one!

```
(let ((U (literal-vector-field 'U-rect R2-rect))
      (V (literal-vector-field 'V-rect R2-rect))
      (W (literal-vector-field 'W-rect R2-rect))
      (nabla (covariant-derivative general-Cartan-2))
      (sigma (up 'sigma0 'sigma1)))
  (let ((m (Chi-inverse sigma)))
    (let ((s (make-state sigma ((U Chi) m))))
      (- (((commutator (L W) (L V)) U-select) s)
          (((nabla (commutator W V)) U) Chi)
          m)))))
a nonzero mess
```

Let's compute the two parts of the Riemann curvature operator and see how this works out. First, recall

$$\nabla_v u(f) = \sum_i e_i(f) \left(v(\tilde{e}^i(u)) + \sum_j \varpi_j^i(v) \tilde{e}^j(u) \right) \quad (8.16)$$

$$= e(f) (v(\tilde{e}(u)) + \varpi(v)\tilde{e}(u)), \quad (8.17)$$

where the second form uses tuple arithmetic. Now let's consider the first part of the Riemann curvature operator:

$$\begin{aligned} [\nabla_w, \nabla_v] u &= \nabla_w \nabla_v u - \nabla_v \nabla_w u \\ &= e \{ w(v(\tilde{e}(u)) + \varpi(v)\tilde{e}(u)) + \varpi(w)(v(\tilde{e}(u)) + \varpi(v)\tilde{e}(u)) \} \\ &\quad - e \{ v(w(\tilde{e}(u)) + \varpi(w)\tilde{e}(u)) + \varpi(v)(w(\tilde{e}(u)) + \varpi(w)\tilde{e}(u)) \} \\ &= e \{ [w, v]\tilde{e}(u) \\ &\quad + w(\varpi(v))\tilde{e}(u) - v(\varpi(w))\tilde{e}(u) \\ &\quad + \varpi(w)\varpi(v)\tilde{e}(u) - \varpi(v)\varpi(w)\tilde{e}(u) \}. \end{aligned} \quad (8.18)$$

The second term of the Riemann curvature operator is

$$\nabla_{[w,v]} u = e \{ [w, v]\tilde{e}(u) + \varpi([w, v])\tilde{e}(u) \}. \quad (8.19)$$

The difference of these is the Riemann curvature operator. Notice that the first term in each cancels, and the rest gives equation (8.13).

Ricci Curvature

One measure of the curvature is the Ricci tensor, which is computed from the Riemann tensor by

$$R(u, v) = \sum_i R(\tilde{e}^i, u, e_i, v). \quad (8.20)$$

Expressed as a program:

```
(define ((Ricci nabla basis) u v)
  (contract (lambda (ei wi) ((Riemann nabla) wi u ei v))
            basis))
```

Einstein's field equation (9.27) for gravity, which we will encounter later, is expressed in terms of the Ricci tensor.

Exercise 8.1: Ricci of a Sphere

Compute the components of the Ricci tensor of the surface of a sphere.

Exercise 8.2: Pseudosphere

A pseudosphere is a surface in 3-dimensional space. It is a surface of revolution of a tractrix about its asymptote (along the \hat{z} -axis). We can make coordinates for the surface (t, θ) where t is the coordinate along the asymptote and θ is the angle of revolution. We embed the pseudosphere in rectangular 3-dimensional space with

```
(define (pseudosphere q)
  (let ((t (ref q 0)) (theta (ref q 1)))
    (up (* (sech t) (cos theta))
         (* (sech t) (sin theta))
         (- t (tanh t)))))
```

The structure of Christoffel coefficients for the pseudosphere is

```
(down
  (down (up (/ (+ (* 2 (expt (cosh t) 2) (expt (sinh t) 2))
                  (* -2 (expt (sinh t) 4)) (expt (cosh t) 2))
                  (* -2 (expt (sinh t) 2)))
                  (+ (* (cosh t) (expt (sinh t) 3))
                     (* (cosh t) (sinh t))))
                  0)
        (up 0
            (/ (* -1 (sinh t)) (cosh t))))
  (down (up 0
            (/ (* -1 (sinh t)) (cosh t)))
        (up (/ (cosh t) (+ (expt (sinh t) 3) (sinh t)))
            0)))
```

Note that this is independent of θ .

Compute the components of the Ricci tensor.

8.2 Torsion

There are many connections that describe the local properties of any particular manifold. A connection has a property called *torsion*, which is computed as follows:

$$\mathcal{T}(u, v) = \nabla_u v - \nabla_v u - [u, v]. \quad (8.21)$$

The torsion takes two vector fields and produces a vector field. The torsion depends on the covariant derivative, which is constructed from the connection.

We account for this dependency by parameterizing the program by `nabla`.

```
(define ((torsion-vector nabla) u v)
  (- (- ((nabla u) v) ((nabla v) u))
      (commutator u v)))

(define ((torsion nabla) omega u v)
  (omega ((torsion-vector nabla) u v)))
```

The torsion for the connection for the 2-sphere specified by the Christoffel coefficients `S2-Christoffel` above is zero. We demonstrate this by applying the torsion to the basis vector fields:

```
(for-each
  (lambda (x)
    (for-each
      (lambda (y)
        (print-expression
          (((torsion-vector (covariant-derivative sphere-Cartan))
            x y)
            (literal-manifold-function 'f S2-spherical))
          ((point S2-spherical) (up 'theta0 'phi0))))
        (list d/dtheta d/dphi)))
      (list d/dtheta d/dphi)))
  0
  0
  0
  0)
```

Torsion Doesn't Affect Geodesics

There are multiple connections that give the same geodesic curves. Among these connections there is always one with zero torsion. Thus, if you care about only geodesics, it is appropriate to use a torsion-free connection.

Consider a basis \mathbf{e} and its dual $\tilde{\mathbf{e}}$. The components of the torsion are

$$\tilde{\mathbf{e}}^k(\mathbf{T}(\mathbf{e}_i, \mathbf{e}_j)) = \Gamma_{ij}^k - \Gamma_{ji}^k + \mathbf{d}_{ij}^k, \quad (8.22)$$

where \mathbf{d}_{ij}^k are the structure constants of the basis. See equations (4.37, 4.38). For a commuting basis the structure constants are zero, and the components of the torsion are the antisymmetric part of Γ with respect to the lower indices.

Recall the geodesic equation (7.79):

$$D^2\sigma^i(t) + \sum_{jk} \Gamma_{jk}^i(\gamma(t)) D\sigma^j(t) D\sigma^k(t) = 0. \quad (8.23)$$

Observe that the lower indices of Γ are contracted with two copies of the velocity. Because the use of Γ is symmetrical here, any asymmetry of Γ in its lower indices is irrelevant to the geodesics. Thus one can study the geodesics of any connection by first symmetrizing the connection, eliminating torsion. The resulting equations will be simpler.

8.3 Geodesic Deviation

Geodesics may converge and intersect (as in the lines of longitude on a sphere) or they may diverge (for example, on a saddle). To capture this notion requires some measure of the convergence or divergence, but this requires metrics (see Chapter 9). But even in the absence of a metric we can define a quantity, the *geodesic deviation*, that can be interpreted in terms of relative acceleration of neighboring geodesics from a reference geodesic.

Let there be a one-parameter family of geodesics, with parameter s , and let \mathbf{T} be the vector field of tangent vectors to those geodesics:

$$\nabla_{\mathbf{T}} \mathbf{T} = 0. \quad (8.24)$$

We can parameterize travel along the geodesics with parameter t : a geodesic curve $\gamma_s(t) = \phi_t^{\mathbf{T}}(\mathbf{m}_s)$ where

$$\mathbf{f} \circ \phi_t^{\mathbf{T}}(\mathbf{m}_s) = (e^{tT} \mathbf{f})(\mathbf{m}_s). \quad (8.25)$$

Let $U = \partial/\partial s$ be the vector field corresponding to the displacement of neighboring geodesics. Locally, (t, s) is a coordinate system on the 2-dimensional submanifold formed by the family of geodesics. The vector fields T and U are a coordinate basis for this coordinate system, so $[T, U] = 0$.

The geodesic deviation vector field is defined as:

$$\nabla_T(\nabla_T U). \quad (8.26)$$

If the connection has zero torsion, the geodesic deviation can be related to the Riemann curvature:

$$\nabla_T(\nabla_T U) = -\mathcal{R}(U, T)(T), \quad (8.27)$$

as follows, using equation (8.21),

$$\nabla_T(\nabla_T U) = \nabla_T(\nabla_U T), \quad (8.28)$$

because both the torsion is zero and $[T, U] = 0$. Continuing

$$\begin{aligned} \nabla_T(\nabla_T U) &= \nabla_T(\nabla_U T) \\ &= \nabla_T(\nabla_U T) + \nabla_U(\nabla_T T) - \nabla_U(\nabla_T T) \\ &= \nabla_U(\nabla_T T) - \mathcal{R}(U, T)(T) \\ &= -\mathcal{R}(U, T)(T). \end{aligned} \quad (8.29)$$

In the last line the first term was dropped because T satisfies the geodesic equation (8.24).

The geodesic deviation is defined without using a metric, but it helps to have a metric (see Chapter 9) to interpret the geodesic deviation. Consider two neighboring geodesics, with parameters s and $s + \Delta s$. Given a metric we can assume that t is proportional to path length along each geodesic, and we can define a distance $\delta(s, t, \Delta s)$ between the geodesics at the same value of the parameter t . So the velocity of separation of the two geodesics is

$$(\nabla_T U)\Delta s = \partial_1 \delta(s, t, \Delta s) \hat{s} \quad (8.30)$$

where \hat{s} is a unit vector in the direction of increasing s . So $\nabla_T U$ is the factor of increase of velocity with increase of separation. Similarly, the geodesic deviation can be interpreted as the factor of increase of acceleration with increase of separation:

$$\nabla_T(\nabla_T U)\Delta s = \partial_1 \partial_1 \delta(s, t, \Delta s) \hat{s}. \quad (8.31)$$

Longitude Lines on a Sphere

Consider longitude lines on the unit sphere.⁸ Let theta be colatitude and phi be longitude. These are the parameters s and t , respectively. Then let T be the vector field $d/d\theta$ that is tangent to the longitude lines.

We can verify that every longitude line is a geodesic:

```
((omega (((covariant-derivative Cartan) T) T)) m)
0
```

where ω is an arbitrary one-form field.

Now let U be $d/d\phi$, then U commutes with T :

```
((((commutator U T) f) m)
0
```

The torsion for the usual connection for the sphere is zero:

```
(let ((X (literal-vector-field 'X-sphere S2-spherical))
      (Y (literal-vector-field 'Y-sphere S2-spherical)))
  (((torsion-vector nabla) X Y) f) m))
0
```

So we can compute the geodesic deviation using **Riemann**

```
((+ (omega ((nabla T) ((nabla T) U)))
     ((Riemann nabla) omega T U T))
 m)
0
```

confirming equation (8.29).

Lines of longitude are geodesics. How do the lines of longitude behave? As we proceed from the North Pole, the lines of constant longitude diverge. At the Equator they are parallel and they converge towards the South Pole.

Let's compute $\nabla_T U$ and $\nabla_T(\nabla_T U)$. We know that the distance is purely in the ϕ direction, so

⁸The setup for this example is:

```
(define-coordinates (up theta phi) S2-spherical)
(define T d/dtheta)
(define U d/dphi)
(define m ((point S2-spherical) (up 'theta0 'phi0)))
(define Cartan (Christoffel->Cartan S2-Christoffel))
(define nabla (covariant-derivative Cartan))
```

```
((dphi ((nabla T) U)) m)
(/ (cos theta0) (sin theta0))

((dphi ((nabla T) ((nabla T) U))) m)
-1
```

Let's interpret these results. On a sphere of radius R the distance at colatitude θ between two geodesics separated by $\Delta\phi$ is $d(\phi, \theta, \Delta\phi) = R \sin(\theta) \Delta\phi$. Assuming that θ is uniformly increasing with time, the magnitude of the velocity is just the θ -derivative of this distance:

```
(define ((delta R) phi theta Delta-phi)
  (* R (sin theta) Delta-phi))

(((partial 1) (delta 'R)) 'phi0 'theta0 'Delta-phi)
(* Delta-phi R (cos theta0))
```

The direction of the velocity is the unit vector in the ϕ direction:

```
(define phi-hat
  (* (/ 1 (sin theta)) d/dphi))
```

This comes from the fact that the separation of lines of longitude is proportional to the sine of the colatitude. So the velocity vector field is the product.

We can measure the ϕ component with $d\phi$:

```
((dphi (* (((partial 1) (delta 'R))
            'phi0 'theta0 'Delta-phi)
            phi-hat))
 m)
(/ (* Delta-phi R (cos theta0)) (sin theta0))
```

This agrees with $\nabla_T U \Delta\phi$ for the unit sphere. Indeed, the lines of longitude diverge until they reach the Equator and then they converge.

Similarly, the magnitude of the acceleration is

```
((((partial 1) ((partial 1) (delta 'R)))
  'phi0 'theta0 'Delta-phi)
  (* -1 Delta-phi R (sin theta0)))
```

and the acceleration vector is the product of this result with $\hat{\phi}$. Measuring this with $d\phi$ we get:

```
((dphi (* (((partial 1) ((partial 1) (delta 'R)))
           'phi0 'theta0 'Delta-phi)
           phi-hat))
 m)
(* -1 Delta-phi R)
```

And this agrees with the calculation of $\nabla_T \nabla_T U \Delta\phi$ for the unit sphere. We see that the separation of the lines of longitude are uniformly decelerated as they progress from pole to pole.

8.4 Bianchi Identities

There are some important mathematical properties of the Riemann curvature. These identities will be used to constrain the possible geometries that can occur.

A system with a symmetric connection, $\Gamma_{jk}^i = \Gamma_{kj}^i$, is torsion free.⁹

```
(define nabla
  (covariant-derivative
   (Christoffel->Cartan
    (symmetrize-Christoffel
     (literal-Christoffel-2 'C R4-rect)))))

(((torsion nabla) omega X Y)
 (typical-point R4-rect))
0
```

The Bianchi identities are defined in terms of a cyclic-summation operator, which is most easily described as a Scheme procedure:

```
(define ((cyclic-sum f) x y z)
  (+ (f x y z)
      (f y z x)
      (f z x y)))
```

⁹Setup for this section:

```
(define omega (literal-1form-field 'omega-rect R4-rect))
(define X (literal-vector-field 'X-rect R4-rect))
(define Y (literal-vector-field 'Y-rect R4-rect))
(define Z (literal-vector-field 'Z-rect R4-rect))
(define V (literal-vector-field 'V-rect R4-rect))
```

The first Bianchi identity is

$$R(\omega, x, y, z) + R(\omega, y, z, x) + R(\omega, z, x, y) = 0, \quad (8.32)$$

or, as a program:

```
((cyclic-sum
  (lambda (x y z)
    ((Riemann nabla) omega x y z)))
 X Y Z)
(typical-point R4-rect))
0
```

The second Bianchi identity is

$$\nabla_x R(\omega, v, y, z) + \nabla_y R(\omega, v, z, x) + \nabla_z R(\omega, v, x, y) = 0 \quad (8.33)$$

or, as a program:

```
((cyclic-sum
  (lambda (x y z)
    (((nabla x) (Riemann nabla))
     omega V y z)))
 X Y Z)
(typical-point R4-rect))
0
```

Things get more complicated when there is torsion. We can make a general connection, which has torsion:

```
(define nabla
  (covariant-derivative
   (Christoffel->Cartan
    (literal-Christoffel-2 'C R4-rect)))))

(define R (Riemann nabla))
(define T (torsion-vector nabla))

(define (TT omega x y)
  (omega (T x y)))
```

The first Bianchi identity is now:¹⁰

```
((cyclic-sum
  (lambda (x y z)
    (- (R omega x y z)
        (+ (omega (T (T x y) z))
           (((nabla x) TT) omega y z))))))
  X Y Z)
(typical-point R4-rect))
0
```

and the second Bianchi identity for a general connection is

```
((cyclic-sum
  (lambda (x y z)
    (+ (((nabla x) R) omega V y z)
        (R omega V (T x y) z))))
  X Y Z)
(typical-point R4-rect))
0
```

¹⁰The Bianchi identities are much nastier to write in traditional mathematical notation than as Scheme programs.

9

Metrics

We often want to impose further structure on a manifold to allow us to define lengths and angles. This is done by generalizing the idea of the Euclidean dot product, which allows us to compute lengths of vectors and angles between vectors in traditional vector algebra.

For vectors $\vec{u} = u^x \hat{x} + u^y \hat{y} + u^z \hat{z}$ and $\vec{v} = v^x \hat{x} + v^y \hat{y} + v^z \hat{z}$ the dot product is $\vec{u} \cdot \vec{v} = u^x v^x + u^y v^y + u^z v^z$. The generalization is to provide coefficients for these terms and to include cross terms, consistent with the requirement that the function of two vectors is symmetric. This symmetric, bilinear, real-valued function of two vector fields is called a *metric field*.

For example, the natural metric on a sphere of radius R is

$$g(u, v) = R^2(d\theta(u)d\theta(v) + (\sin \theta)^2 d\phi(u)d\phi(v)), \quad (9.1)$$

and the Minkowski metric on the 4-dimensional space of special relativity is

$$g(u, v) = dx(u)dx(v) + dy(u)dy(v) + dz(u)dz(v) - c^2 dt(u)dt(v). \quad (9.2)$$

Although these examples are expressed in terms of a coordinate basis, the value of the metric on vector fields does not depend on the coordinate system that is used to specify the metric.

Given a metric field g and a vector field v the scalar field $g(v, v)$ is the squared length of the vector at each point of the manifold.

Metric Music

The metric can be used to construct a one-form field ω_u from a vector field u , such that for any vector field v we have

$$\omega_u(v) = g(v, u). \quad (9.3)$$

The operation of constructing a one-form field from a vector field using a metric is called “lowering” the vector field. It is sometimes notated as

$$\omega_u = g^\flat(u). \quad (9.4)$$

There is also an inverse metric that takes two one-form fields. It is defined by the relation

$$\delta_k^i = \sum_j g^{-1}(\tilde{e}^i, \tilde{e}^j)g(e_j, e_k), \quad (9.5)$$

where e and \tilde{e} are any basis and its dual basis.

The inverse metric can be used to construct a vector field v_ω from a one-form field ω , such that for any one-form field τ we have

$$\tau(v_\omega) = g^{-1}(\omega, \tau). \quad (9.6)$$

This definition is implicit, but the vector field can be explicitly computed from the one-form field with respect to a basis as follows:

$$v_\omega = \sum_i g^{-1}(\omega, \tilde{e}^i)e_i. \quad (9.7)$$

The operation of constructing a vector field from a one-form field using a metric is called “raising” the one-form field. It is sometimes notated

$$v_\omega = g^\sharp(\omega). \quad (9.8)$$

The raising and lowering operations allow one to interchange the vector fields and the one-form fields. However they should not be confused with the dual operation that allows one to construct a dual one-form basis from a vector basis or construct a vector basis from a one-form basis. The dual operation that interchanges bases is defined without assigning a metric structure on the space.

Lowering a vector field with respect to a metric is a simple program:

```
(define ((lower metric) u)
  (define (omega v) (metric v u))
  (procedure->1form-field omega))
```

But raising a one-form field to make a vector field is a bit more complicated:

```
(define (raise metric basis)
  (let ((gi (metric:invert metric basis)))
    (lambda (omega)
      (contract (lambda (e_i w^i)
                  (* (gi omega w^i) e_i))
                basis))))
```

where `contract` is the trace over a basis of a two-argument function that takes a vector field and a one-form field as its arguments.¹

```
(define (contract proc basis)
  (let ((vector-basis (basis->vector-basis basis))
        (1form-basis (basis->1form-basis basis)))
    (s:sigma/r proc
                vector-basis
                1form-basis)))
```

9.1 Metric Compatibility

A connection is said to be compatible with a metric g if the covariant derivative for that connection obeys the “product rule”:

$$\nabla_X(g(Y, Z)) = g(\nabla_X(Y), Z) + g(Y, \nabla_X(Z)). \quad (9.9)$$

For a metric there is a unique torsion-free connection that is compatible with it. The Christoffel coefficients of the first kind are computed from the metric by the following:

$$\bar{\Gamma}_{ijk} = \frac{1}{2}(e_k(g(e_i, e_j)) + e_j(g(e_i, e_k)) - e_i(g(e_j, e_k))), \quad (9.10)$$

for the coordinate basis e . We can then construct the Christoffel coefficients of the second kind (the ones used previously to define a connection) by “raising the first index.” To do this we define a function of three vectors, with a weird currying:

$$\tilde{\Gamma}(v, w)(u) = \sum_{ijk} \bar{\Gamma}_{ijk} \tilde{e}^i(u) \tilde{e}^j(v) \tilde{e}^k(w). \quad (9.11)$$

¹Notice that `raise` and `lower` are not symmetrical. This is because vector fields and form fields are not symmetrical: a vector field takes a manifold function as its argument, whereas a form field takes a vector field as its argument. This asymmetry is not apparent in traditional treatments based on index notation.

This function takes two vector fields and produces a one-form field. We can use it with equation (9.7) to construct a new function that takes two vector fields and produces a vector field:

$$\hat{\Gamma}(v, w) = \sum_i g^{-1}(\tilde{\Gamma}(v, w), \tilde{e}^i) e_i. \quad (9.12)$$

We can now construct the Christoffel coefficients of the second kind:

$$\Gamma_{jk}^i = \tilde{e}^i(\hat{\Gamma}(e_j, e_k)) = \sum_m \bar{\Gamma}_{mjk} g^{-1}(\tilde{e}^m, \tilde{e}^i). \quad (9.13)$$

The Cartan forms are then just

$$\varpi_j^i = \sum_k \Gamma_{jk}^i \tilde{e}^k = \sum_k \tilde{e}^i(\hat{\Gamma}(e_j, e_k)) \tilde{e}^k. \quad (9.14)$$

So, for example, we can compute the Christoffel coefficients for the sphere from the metric for the sphere. First, we need the metric:

```
(define ((g-sphere R) u v)
  (* (square R)
    (+ (* (dtheta u) (dtheta v))
      (* (compose (square sin) theta)
        (dphi u)
        (dphi v))))))
```

The Christoffel coefficients of the first kind are a complex structure with all three indices down:

```
((Christoffel->symbols
  (metric->Christoffel-1 (g-sphere 'R) S2-basis))
 ((point S2-spherical) (up 'theta0 'phi0)))
 (down
  (down (down 0 0)
    (down 0 (* (* (cos theta0) (sin theta0)) (expt R 2))))
  (down (down 0 (* (* (cos theta0) (sin theta0)) (expt R 2)))
    (down (* (-1 (* (cos theta0) (sin theta0)) (expt R 2))
      0))))
```

And the Christoffel coefficients of the second kind have the innermost index up:

```
((Christoffel->symbols
  (metric->Christoffel-2 (g-sphere 'R) S2-basis))
  ((point S2-spherical) (up 'theta0 'phi0)))
  (down (down (up 0 0)
    (up 0 (/ (cos theta0) (sin theta0))))
    (down (up 0 (/ (cos theta0) (sin theta0)))
      (up (* -1 (cos theta0) (sin theta0)) 0))))
```

Exercise 9.1: Metric Compatibility

The connections constructed from a metric by equation (9.13) are “metric compatible,” as described in equation (9.9). Demonstrate that this is true for a literal metric, as described on page 6, in \mathbf{R}^4 . Your program should produce a zero.

9.2 Metrics and Lagrange Equations

In the Introduction (Chapter 1) we showed that the Lagrange equations for a free particle constrained to a 2-dimensional surface are equivalent to the geodesic equations for motion on that surface. We illustrated that in detail in Section 7.4 for motion on a sphere.

Here we expand this understanding to show that the Christoffel symbols can be derived from the Lagrange equations. Specifically, if we solve the Lagrange equations for the acceleration (the highest-order derivatives) we find that the Christoffel symbols are the symmetrized coefficients of the quadratic velocity terms.

Consider the Lagrange equations for a free particle, with Lagrangian

$$L_2(t, x, v) = \frac{1}{2}g(x)(v, v). \quad (9.15)$$

If we solve the Lagrange equations for the accelerations, the accelerations can be expressed with the geodesic equations (7.79):

$$D^2q^i + \sum_{jk}(\Gamma_{jk}^i \circ \chi^{-1} \circ q)Dq^j Dq^k = 0. \quad (9.16)$$

We can verify this computationally. Given a metric, we can construct a Lagrangian where the kinetic energy is the metric applied to the velocity twice: The kinetic energy is proportional to the squared length of the velocity vector.

```
(define (metric->Lagrangian metric coordsys)
  (define (L state)
    (let ((q (ref state 1)) (qd (ref state 2)))
      (define v
        (components->vector-field (lambda (m) qd) coordsys))
      ((* 1/2 (metric v v)) ((point coordsys) q))))
  L)
```

The following code compares the Christoffel symbols with the coefficients of the terms of second order in velocity appearing in the accelerations, determined by solving the Lagrange equations for the highest-order derivative.² We extract these terms by taking two partials with respect to the structure of velocities. Because the elementary partials commute we get two copies of each coefficient, requiring a factor of 1/2.

```
(let* ((metric (literal-metric 'g R3-rect))
      (q (typical-coords R3-rect))
      (L2 (metric->Lagrangian metric R3-rect)))
  (+ (* 1/2
         (((expt (partial 2) 2) (Lagrange-explicit L2))
          (up 't q (corresponding-velocities q))))
     ((Christoffel->symbols
       (metric->Christoffel-2 metric
         (coordinate-system->basis R3-rect)))
      (point R3-rect) q)))
  (down (down (up 0 0 0) (up 0 0 0) (up 0 0 0))
        (down (up 0 0 0) (up 0 0 0) (up 0 0 0))
        (down (up 0 0 0) (up 0 0 0) (up 0 0 0))))
```

We get a structure of zeros, demonstrating the correspondence between Christoffel symbols and coefficients of the Lagrange equations.

Thus, if we have a metric specifying an inner product, the geodesic equations are equivalent to the Lagrange equations for

²The procedure `Lagrange-explicit` produces the accelerations of the coordinates. In this code the division operator (/) multiplies its first argument on the left by the inverse of its second argument.

```
(define (Lagrange-explicit L)
  (let ((P ((partial 2) L)))
    (F ((partial 1) L)))
  (/ (- F (+ ((partial 0) P) (* ((partial 1) P) velocity)))
      ((partial 2) P))))
```

the Lagrangian that is equal to the inner product of the generalized velocities with themselves.

Kinetic Energy or Arc Length

A geodesic is a path of stationary length with respect to variations in the path that keep the endpoints fixed. On the other hand, the solutions of the Lagrange equations are paths of stationary action that keep the endpoints fixed. How are these solutions related?

The integrand of the traditional action is the Lagrangian, which is in this case the Lagrangian L_2 , the kinetic energy. The integrand of the arc length is

$$L_1(t, x, v) = \sqrt{g(x)(v, v)} = \sqrt{2L_2(t, x, v)} \quad (9.17)$$

and the path length is

$$\tau = \int_{t_1}^{t_2} L_1(t, q(t), Dq(t)) dt. \quad (9.18)$$

If we compute the Lagrange equations for L_2 we get the Lagrange equations for L_1 with a correction term. Since

$$L_2(t, x, v) = \frac{1}{2}(L_1(t, x, v))^2, \quad (9.19)$$

and the Lagrange operator for L_2 is³

$$\mathbf{E}[L_2] = D_t \partial_2 L_2 - \partial_1 L_2,$$

we find

$$\mathbf{E}[L_2] = L_1 \mathbf{E}[L_1] + \partial_2 L_1 D_t L_1. \quad (9.20)$$

L_2 is the kinetic energy. It is conserved along solution paths, since there is no explicit time dependence. Because of the relation between L_1 and L_2 , L_1 is also a conserved quantity. Let L_1 take the constant value a on the geodesic coordinate path q we are

³ \mathbf{E} is the Euler-Lagrange operator, which gives the residuals of the Lagrange equations for a Lagrangian. Γ extends a configuration-space path q to make a state-space path, with as many terms as needed: $\Gamma[q](t) = (t, q(t), Dq(t), \dots)$. The total time derivative D_t is defined by $D_t F \circ \Gamma[q] = D(F \circ \Gamma[q])$ for any state function F and path q . The Lagrange equations are $\mathbf{E}[L] \circ \Gamma[q] = 0$. See [19] for more details.

considering. Then $\tau = a(t_2 - t_1)$. Since L_1 is conserved, $(D_t L_1) \circ \Gamma[q] = 0$ on the geodesic path q , and both $\mathbf{E}[L_1] \circ \Gamma[q] = 0$ and $\mathbf{E}[L_2] \circ \Gamma[q] = 0$, as required by equation (9.20).

Since L_2 is homogeneous of degree 2 in the velocities, L_1 is homogeneous of degree 1. So we cannot solve for the highest-order derivative in the Lagrange-Euler equations derived from L_1 : The Lagrange equations of the Lagrangian L_1 are dependent. But although they do not uniquely specify the evolution, they do specify the geodesic path.

On the other hand, we can solve for the highest-order derivative in $\mathbf{E}[L_2]$. This is because $L_1 \mathbf{E}[L_1]$ is homogeneous of degree 2. So the equations derived from L_2 uniquely determine the time evolution along the geodesic path.

For Two Dimensions

We can show this is true for a 2-dimensional system with a general metric. We define the Lagrangians in terms of this metric:

```
(define L2
  (metric->Lagrangian (literal-metric 'm R2-rect)
                        R2-rect))

(define (L1 state)
  (sqrt (* 2 (L2 state))))
```

Although the mass matrix of L_2 is nonsingular

```
(determinant
 (((partial 2) ((partial 2) L2))
  (up 't (up 'x 'y) (up 'vx 'vy)))
 (+ (* (m_00 (up x y)) (m_11 (up x y)))
    (* -1 (expt (m_01 (up x y)) 2))))
```

the mass matrix of L_1 has determinant zero

```
(determinant
 (((partial 2) ((partial 2) L1))
  (up 't (up 'x 'y) (up 'vx 'vy)))
 0)
```

showing that these Lagrange equations are dependent.

We can show this dependence explicitly, for a simple system. Consider the simplest possible system, a geodesic (straight line) in a plane:

```
(define (L1 state)
  (sqrt (square (velocity state)))))

(((Lagrange-equations L1)
  (up (literal-function 'x) (literal-function 'y)))
 't)
(down
 (/ (+ (* (((expt D 2) x) t) (expt ((D y) t) 2))
        (* -1 ((D x) t) ((D y) t) (((expt D 2) y) t)))
    (expt (+ (expt ((D x) t) 2) (expt ((D y) t) 2)) 3/2))
 (/ (+ (* -1 (((expt D 2) x) t) ((D x) t) ((D y) t))
        (* (expt ((D x) t) 2) (((expt D 2) y) t)))
    (expt (+ (expt ((D x) t) 2) (expt ((D y) t) 2)) 3/2)))
```

These residuals must be zero; so the numerators must be zero.⁴
They are:

$$\begin{aligned} D^2x(Dy)^2 &= Dx Dy D^2y \\ D^2x DxDy &= (Dx)^2 D^2y \end{aligned}$$

Note that the only constraint is $D^2x Dy = Dx D^2y$, so the resulting Lagrange equations are dependent.

This is enough to determine that the result is a straight line, without specifying the rate along the line. Suppose $y = f(x)$, for path $(x(t), y(t))$. Then

$$Dy = Df(x) Dx \text{ and } D^2y = D^2f(x) Dx + Df(x) D^2x.$$

Substituting, we get

$$Df(x) Dx D^2x = Dx(D^2f(x) Dx + Df(x) D^2x)$$

or

$$Df(x) D^2x = D^2f(x) Dx + Df(x) D^2x,$$

so $D^2f(x) = 0$. Thus f is a straight line, as required.

Reparameterization

More generally, a differential equation system $F[q](t) = 0$ is said to be *reparameterized* if the coordinate path q is replaced with a

⁴We cheated: We hand-simplified the denominator to make the result more obvious.

new coordinate path $q \circ f$. For example, we may change the scale of the independent variable. The system $F[q \circ f] = 0$ is said to be independent of the parameterization if and only if $F[q] \circ f = 0$. So the differential equation system is satisfied by $q \circ f$ if and only if it is satisfied by q .

The Lagrangian L_1 is homogeneous of degree 1 in the velocities; so

$$\mathbf{E}[L_1] \circ \Gamma[q \circ f] - (\mathbf{E}[L_1] \circ \Gamma[q] \circ f) Df = 0. \quad (9.21)$$

We can check this in a simple case. For two dimensions $q = (x, y)$, the condition under which a reparameterization f of the geodesic paths with coordinates q satisfies the Lagrange equations for L_1 is:

```
(let ((x (literal-function 'x))
      (y (literal-function 'y))
      (f (literal-function 'f)))
  (E1 (Euler-Lagrange-operator L1)))
  ((- (compose E1
                (Gamma (up (compose x f)
                           (compose y f)))
                4))
     (* (compose E1
                  (Gamma (up x y) 4)
                  f)
        (D f)))
   't))
  (down 0 0))
```

This residual is identically satisfied, showing that the Lagrange equations for L_1 are independent of the parameterization of the independent variable.

The Lagrangian L_2 is homogeneous of degree 2 in the velocities; so

$$\mathbf{E}[L_2][q \circ f] - (\mathbf{E}[L_2][q] \circ f)(Df)^2 = (\partial_2 L_2 \circ \Gamma[q] \circ f)(D^2 f). \quad (9.22)$$

Although the Euler-Lagrange equations for L_1 are invariant under an arbitrary reparameterization ($Df \neq 0$), the Euler-Lagrange equations for L_2 are invariant only for a restricted set of f . The conditions under which a reparameterization f of geodesic paths with coordinates q satisfies the Lagrange equations for L_2 are:

```
(let ((q (up (literal-function 'x) (literal-function 'y)))
      (f (literal-function 'f)))
    ((- (compose (Euler-Lagrange-operator L2)
                 (Gamma (compose q f) 4))
        (* (compose (Euler-Lagrange-operator L2)
                    (Gamma q 4)
                    f)
           (expt (D f) 2)))
     't))
  (down
   (* (+ (* ((D x) (f t)) (m_00 (up (x (f t)) (y (f t)))))))
      (* ((D y) (f t)) (m_01 (up (x (f t)) (y (f t)))))))
   (((expt D 2) f) t)))
  (* (+ (* ((D x) (f t)) (m_01 (up (x (f t)) (y (f t)))))))
     (* ((D y) (f t)) (m_11 (up (x (f t)) (y (f t)))))))
   (((expt D 2) f) t))))
```

We see that if these expressions must be zero, then $D^2 f = 0$. This tells us that f is at most affine in t : $f(t) = at + b$.

Exercise 9.2: SO(3) Geodesics

We have derived a basis for $\text{SO}(3)$ in terms of incremental rotations around the rectangular axes. See equations (4.29, 4.30, 4.31). We can use the dual basis to define a metric on $\text{SO}(3)$.

```
(define (SO3-metric v1 v2)
  (+ (* (e^x v1) (e^x v2))
      (* (e^y v1) (e^y v2))
      (* (e^z v1) (e^z v2))))
```

This metric determines a connection. Show that uniform rotation about an arbitrary axis traces a geodesic on $\text{SO}(3)$.

Exercise 9.3: Curvature of a Spherical Surface

The 2-dimensional surface of a 3-dimensional sphere can be embedded in three dimensions with a metric that depends on the radius:

```
(define M (make-manifold S^2-type 2 3))
(define spherical
  (coordinate-system-at 'spherical 'north-pole M))
(define-coordinates (up theta phi) spherical)
(define spherical-basis (coordinate-system->basis spherical))

(define ((spherical-metric r) v1 v2)
  (* (square r)
     (+ (* (dtheta v1) (dtheta v2))
        (* (square (sin theta))
           (dphi v1) (dphi v2))))))
```

If we raise one index of the Ricci tensor (see equation 8.20) by contracting it with the inverse of the metric tensor we can further contract it to obtain a scalar manifold function:

$$R = \sum_{ij} g(\tilde{e}^i, \tilde{e}^j) R(e^i, e^j). \quad (9.23)$$

The `trace2down` procedure converts a tensor that takes two vector fields into a tensor that takes a vector field and a one-form field, and then it contracts the result over a basis to make a trace. It is useful for getting the Ricci scalar from the Ricci tensor, given a metric and a basis.

```
(define ((trace2down metric basis) tensor)
  (let ((inverse-metric-tensor
        (metric:invert metric-tensor basis)))
    (contract
      (lambda (v1 w1)
        (contract
          (lambda (v w)
            (* (inverse-metric-tensor w1 w)
               (tensor v v1)))
          basis))
      basis)))
```

Evaluate the Ricci scalar for a sphere of radius r to obtain a measure of its intrinsic curvature. You should obtain the answer $2/r^2$.

Exercise 9.4: Curvature of a Pseudosphere

Compute the scalar curvature of the pseudosphere (see exercise 8.2). You should obtain the value -2 .

9.3 General Relativity

By analogy to Newtonian mechanics, relativistic mechanics has two parts. There are equations of motion that describe how particles move under the influence of “forces” and there are field equations that describe how the forces arise. In general relativity the only force considered is gravity. However, gravity is not treated as a force. Instead, gravity arises from curvature in the spacetime, and the equations of motion are motion along geodesics of that space.

The geodesic equations for a spacetime with the metric

$$\begin{aligned} g(v_1, v_2) = & -c^2 \left(1 + \frac{2V}{c^2} \right) dt(v_1)dt(v_2) \\ & + dx(v_1)dx(v_2) \\ & + dy(v_1)dy(v_2) \\ & + dz(v_1)dz(v_2) \end{aligned} \quad (9.24)$$

are Newton's equations to lowest order in V/c^2 :

$$D^2\vec{x}(t) = -\text{grad}V(\vec{x}(t)). \quad (9.25)$$

Exercise 9.5: Newton's Equations

Verify that Newton's equations (9.25) are indeed the lowest-order terms of the geodesic equations for the metric (9.24).

Einstein's field equations tell how the local energy-momentum distribution determines the local shape of the spacetime, as described by the metric tensor g . The equations are traditionally written

$$R_{\mu\nu} - \frac{1}{2}Rg_{\mu\nu} + \Lambda g_{\mu\nu} = \frac{8\pi G}{c^4}T_{\mu\nu} \quad (9.26)$$

where $R_{\mu\nu}$ are the components of the Ricci tensor (equation 8.20), R is the Ricci scalar (equation 9.23),⁵ and Λ is the cosmological constant.

$T_{\mu\nu}$ are the components of the stress-energy tensor describing the energy-momentum distribution. Equivalently, one can write

$$R_{\mu\nu} = \frac{8\pi G}{c^4} \left(T_{\mu\nu} - \frac{1}{2}Tg_{\mu\nu} \right) - \Lambda g_{\mu\nu} \quad (9.27)$$

where $T = T_{\mu\nu}g^{\mu\nu}$.⁶

⁵The tensor with components $G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2}Rg_{\mu\nu}$ is called the *Einstein tensor*. In his search for an appropriate field equation for gravity, Einstein demanded *general covariance* (independence of coordinate system) and local Lorentz invariance (at each point transformations must preserve the line element). These considerations led Einstein to look for a tensor equation (see Appendix C).

⁶Start with equation (9.26). Raise one index of both sides, and then contract. Notice that the trace $g^\mu_\mu = 4$, the dimension of spacetime. This gets $R = -(8\pi G/c^4)T$, from which we can deduce equation (9.27).

Einstein's field equations arise from a heuristic derivation by analogy to the Poisson equation for a Newtonian gravitational field:

$$\text{Lap}(V) = 4\pi G\rho \quad (9.28)$$

where V is the gravitational potential field at a point, ρ is the mass density at that point, and Lap is the Laplacian operator.

The time-time component of the Ricci tensor derived from the metric (9.24) is the Laplacian of the potential, to lowest order.

```
(define (Newton-metric M G c V)
  (let ((a
         (+ 1 (* (/ 2 (square c))
                  (compose V (up x y z))))))
    (define (g v1 v2)
      (+ (* -1 (square c) a (dt v1) (dt v2))
          (* (dx v1) (dx v2))
          (* (dy v1) (dy v2))
          (* (dz v1) (dz v2))))
    g))

(define (Newton-connection M G c V)
  (Christoffel->Cartan
   (metric->Christoffel-2 (Newton-metric M G c V)
                           spacetime-rect-basis)))

(define nabla
  (covariant-derivative
   (Newton-connection 'M 'G ':c
                      (literal-function 'V (-> (UP Real Real Real) Real)))))

(((Ricci nabla (coordinate-system->basis spacetime-rect))
  d/dt d/dt)
 ((point spacetime-rect) (up 't 'x 'y 'z)))
mess
```

The leading terms of the mess are

```
(+ (((partial 0) ((partial 0) V)) (up x y z))
    (((partial 1) ((partial 1) V)) (up x y z))
    (((partial 2) ((partial 2) V)) (up x y z)))
```

which is the Laplacian of V . The other terms are smaller by V/c^2 .

Now consider the right-hand side of equation (9.27). In the Poisson equation the source of the gravitational potential is the density of matter. Let the time-time component of the stress-

energy tensor T^{00} be the matter density ρ . Here is a program for the stress-energy tensor:

```
(define (Tdust rho)
  (define (T w1 w2)
    (* rho (w1 d/dt) (w2 d/dt)))
  T)
```

If we evaluate the right-hand side expression we obtain⁷

```
(let ((g (Newton-metric 'M 'G ':c V)))
  (let ((T_ij ((drop2 g spacetime-rect-basis) (Tdust 'rho))))
    (let ((T ((trace2down g spacetime-rect-basis) T_ij)))
      ((- (T_ij d/dt d/dt) (* 1/2 T (g d/dt d/dt)))
       ((point spacetime-rect) (up 't 'x 'y 'z))))))
  (* 1/2 (expt :c 4) rho))
```

So, to make the Poisson analogy we get

$$R_{\mu\nu} = \frac{8\pi G}{c^4} \left(T_{\mu\nu} - \frac{1}{2} T g_{\mu\nu} \right) - \Lambda g_{\mu\nu} \quad (9.29)$$

as required.

Exercise 9.6: Curvature of Schwarzschild Spacetime

In spherical coordinates around a nonrotating gravitating body the metric of Schwarzschild spacetime is given as:⁸

⁷The procedure `trace2down` is defined on page 144. This expression also uses `drop2`, which converts a tensor field that takes two one-form fields into a tensor field that takes two vector fields. Its definition is

```
(define ((drop2 metric-tensor basis) tensor)
  (lambda (v1 v2)
    (contract
      (lambda (e1 w1)
        (contract
          (lambda (e2 w2)
            (* (metric-tensor v1 e1) (tensor w1 w2) (metric-tensor e2 v2)))
            basis)))
      basis)))
```

⁸The spacetime manifold is built from \mathbf{R}^4 with the addition of appropriate coordinate systems:

```
(define spacetime (make-manifold R^n 4))
(define spacetime-rect
  (coordinate-system-at 'rectangular 'origin spacetime))
(define spacetime-sphere
  (coordinate-system-at 'spacetime-spherical 'origin spacetime))
```

```
(define-coordinates (up t r theta phi) spacetime-sphere)

(define (Schwarzschild-metric M G c)
  (let ((a (- 1 (/ (* 2 G M) (* (square c) r))))))
    (lambda (v1 v2)
      (+ (* -1 (square c) a (dt v1) (dt v2))
          (* (/ 1 a) (dr v1) (dr v2))
          (* (square r)
              (+ (* (dtheta v1) (dtheta v2))
                  (* (square (sin theta))
                      (dphi v1) (dphi v2)))))))
```

Show that the Ricci curvature of the Schwarzschild spacetime is zero. Use the definition of the Ricci tensor in equation (8.20).

Exercise 9.7: Circular Orbits in Schwarzschild Spacetime

Test particles move along geodesics in spacetime. Now that we have a metric for Schwarzschild spacetime (page 147) we can use it to construct the geodesic equations and determine how test particles move. Consider circular orbits. For example, the circular orbit along a line of constant longitude is a geodesic, so it should satisfy the geodesic equations. Here is the equation of a circular path along the zero longitude line.

```
(define (prime-meridian r omega)
  (compose (point spacetime-sphere)
            (lambda (t) (up t r (* omega t) 0))
            (chart R1-rect)))
```

This equation will satisfy the geodesic equations for compatible values of the radius r and the angular velocity ω . If you substitute this into the geodesic equation and set the residual to zero you will obtain a constraint relating r and ω . Do it.

Surprise: You should find out that $\omega^2 r^3 = GM$ —Kepler's law!

Exercise 9.8: Stability of Circular Orbits

In Schwarzschild spacetime there are stable circular orbits if the coordinate r is large enough, but below that value all orbits are unstable. The critical value of r is larger than the Schwarzschild horizon radius. Let's find that value.

For example, we can consider a perturbation of the orbit of constant longitude. Here is the result of adding an exponential variation of size ϵ :

```
(define (prime-meridian+X r epsilon X)
  (compose
    (point spacetime-sphere)
    (lambda (t)
      (up (+ t (* epsilon (* (ref X 0) (exp (* 'lambda t))))))
        (+ r (* epsilon (* (ref X 1) (exp (* 'lambda t))))))
        (+ (* (sqrt (/ (* 'G 'M) (expt r 3))) t)
            (* epsilon (* (ref X 2) (exp (* 'lambda t))))))
        0)))
  (chart R1-rect)))
```

Plugging this into the geodesic equation yields a structure of residuals:

```
(define (geodesic-equation+X-residuals eps X)
  (let ((gamma (prime-meridian+X 'r eps X)))
    (((((covariant-derivative Cartan gamma) d/dtau)
        ((differential gamma) d/dtau))
       (chart spacetime-sphere))
      ((point R1-rect) 't))))
```

The characteristic equation in the eigenvalue `lambda` can be obtained as the numerator of the expression:

```
(determinant
  (submatrix (((* (partial 1) (partial 0))
               geodesic-equation+X-residuals)
              0
              (up 0 0 0))
             0 3 0 3))
```

Show that the orbits are unstable if $r < 6GM/c^2$.

Exercise 9.9: Friedmann-Lemaître-Robertson-Walker

The Einstein tensor $G_{\mu\nu}$ (see footnote 5) can be expressed as a program:

```
(define (Einstein coordinate-system metric-tensor)
  (let* ((basis (coordinate-system->basis coordinate-system))
         (connection
          (Christoffel->Cartan
           (metric->Christoffel-2 metric-tensor basis)))
         (nabla (covariant-derivative connection))
         (Ricci-tensor (Ricci nabla basis))
         (Ricci-scalar
          ((trace2down metric-tensor basis) Ricci-tensor)))
    (define (Einstein-tensor v1 v2)
      (- (Ricci-tensor v1 v2)
          (* 1/2 Ricci-scalar (metric-tensor v1 v2))))
    Einstein-tensor))

(define (Einstein-field-equation
         coordinate-system metric-tensor Lambda stress-energy-tensor)
  (let ((Einstein-tensor
         (Einstein coordinate-system metric-tensor)))
    (define EFE-residuals
      (- (+ Einstein-tensor (* Lambda metric-tensor))
          (* (/ (* 8 :pi :G) (expt :c 4))
              stress-energy-tensor)))
    EFE-residuals))
```

One exact solution to the Einstein equations was found by Alexander Friedmann in 1922. He showed that a metric for an isotropic and homogeneous spacetime was consistent with a similarly isotropic and homogeneous stress-energy tensor in Einstein's equations. In this case

the residuals of the Einstein equations gave ordinary differential equations for the time-dependent scale of the universe. These are called the Robertson-Walker equations. Friedmann's metric is:

```
(define (FLRW-metric c k R)
  (define-coordinates (up t r theta phi) spacetime-sphere)
  (let ((a (/ (square (compose R t)) (- 1 (* k (square r)))))))
    (b (square (* (compose R t) r))))
  (define (g v1 v2)
    (+ (* -1 (square c) (dt v1) (dt v2))
       (* a (dr v1) (dr v2))
       (* b (+ (* (dtheta v1) (dtheta v2))
                 (* (square (sin theta))
                    (dphi v1) (dphi v2)))))))
  g))
```

Here c is the speed of light, k is the intrinsic curvature, and R is a length scale that is a function of time.

The associated stress-energy tensor is

```
(define (Tperfect-fluid rho p c metric)
  (define-coordinates (up t r theta phi) spacetime-sphere)
  (let* ((basis (coordinate-system->basis spacetime-sphere))
         (inverse-metric (metric:invert metric basis)))
    (define (T w1 w2)
      (+ (* (+ (compose rho t)
                (/ (compose p t) (square c)))
            (w1 d/dt) (w2 d/dt))
          (* (compose p t) (inverse-metric w1 w2))))
    T))
```

where ρ is the energy density, and p is the pressure in an ideal fluid model.

The Robertson-Walker equations are:

$$\begin{aligned} \left(\frac{DR(t)}{R(t)} \right)^2 + \frac{kc^2}{(R(t))^2} - \frac{\Lambda c^2}{3} &= \frac{8\pi G}{3} \rho(t), \\ 2 \frac{D^2 R(t)}{R(t)} - \frac{2}{3} \Lambda c^2 &= -8\pi G \left(\frac{\rho(t)}{3} + \frac{p(t)}{c^2} \right). \end{aligned} \quad (9.30)$$

Use the programs supplied to derive the Robertson-Walker equations.

Exercise 9.10: Cosmology

For energy to be conserved, the stress-energy tensor must be constrained so that its covariant divergence is zero

$$\sum_{\mu} \nabla_{e_{\mu}} T(\tilde{e}^{\mu}, \omega) = 0 \quad (9.31)$$

for every one-form ω .

- a. Show that for the perfect fluid stress-energy tensor and the FLRW metric this constraint is equivalent to the differential equation

$$D(c^2 \rho R^3) + p D(R^3) = 0. \quad (9.32)$$

- b. Assume that in a “matter-dominated universe” radiation pressure is negligible, so $p = 0$. Using the Robertson-Walker equations (9.30) and the energy conservation equation (9.32) show that the observation of an expanding universe is compatible with a negative curvature universe, a flat universe, or a positive curvature universe: $k \in \{-1, 0, +1\}$.

10

Hodge Star and Electrodynamics

The vector space of p -form fields on an n -dimensional manifold has dimension $n!/((n-p)!p!)$. This is the same dimension as the space of $(n-p)$ -form fields. So these vector spaces are isomorphic. If we have a metric there is a natural isomorphism: for each p -form field ω on an n -dimensional manifold there is an $(n-p)$ -form field $g^*\omega$, called its *Hodge dual*.¹ The Hodge dual should not be confused with the duality of vector bases and one-form bases, which is defined without reference to a metric. The Hodge dual is useful for the elegant formalization of electrodynamics.

In Euclidean 3-space, if we think of a one-form as a foliation of the space, then the dual is a two-form, which can be thought of as a pack of square tubes, whose axes are perpendicular to the leaves of the foliation. The original one-form divides these tubes up into volume elements. For example, the dual of the basis one-form dx is the two-form $g^*dx = dy \wedge dz$. We may think of dx as a set of planes perpendicular to the \hat{x} -axis. Then g^*dx is a set of tubes parallel to the \hat{x} -axis. In higher-dimensional spaces the visualization is more complicated, but the basic idea is the same. The Hodge dual of a two-form in four dimensions is a two-form that is perpendicular to the given two-form. However, if the metric is indefinite (e.g., the Lorentz metric) there is an added complication with the signs.

The Hodge dual is a linear operator, so it can be defined by its action on the basis elements. Let $\{\partial/\partial x^0, \dots, \partial/\partial x^{n-1}\}$ be an orthonormal basis of vector fields² and let $\{dx^0, \dots, dx^{n-1}\}$ be the ordinary dual basis for the one-forms. Then the $(n-p)$ -form $g^*\omega$ that is the Hodge dual of the p -form ω can be defined by its coefficients with respect to the basis, using indices, as

$$(g^*\omega)_{j_p \dots j_{n-1}} = \sum_{i_0 \dots i_{p-1} j_0 \dots j_{p-1}} \frac{1}{p!} \omega_{i_0 \dots i_{p-1}} g^{i_0 j_0} \dots g^{i_{p-1} j_{p-1}} \epsilon_{j_0 \dots j_{n-1}}, \quad (10.1)$$

¹The traditional notion is to just use an asterisk; we use g^* to emphasize that this duality depends on the choice of metric g .

²We have a metric, so we can define “orthonormal” and use it to construct an orthonormal basis given any basis. The Gram-Schmidt procedure does the job.

where g^{ij} are the coefficients of the inverse metric and $\epsilon_{j_0 \dots j_{n-1}}$ is either -1 or $+1$ if the permutation $\{0 \dots n-1\} \rightarrow \{j_0 \dots j_{n-1}\}$ is odd or even, respectively.

Relationship to Vector Calculus

In 3-dimensional Euclidean space the traditional vector derivative operations are gradient, curl, and divergence. If $\hat{x}, \hat{y}, \hat{z}$ are the usual orthonormal rectangular vector basis, f a function on the space, and \vec{v} a vector field on the space, then

$$\begin{aligned}\text{grad}(f) &= \frac{\partial f}{\partial x} \hat{x} + \frac{\partial f}{\partial y} \hat{y} + \frac{\partial f}{\partial z} \hat{z}, \\ \text{curl}(\vec{v}) &= \left(\frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z} \right) \hat{x} + \left(\frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x} \right) \hat{y} + \left(\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right) \hat{z}, \\ \text{div}(\vec{v}) &= \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}.\end{aligned}$$

Recall the meaning of the traditional vector operations. Traditionally we assume that there is a metric that allows us to determine distances between locations and angles between vectors. Such a metric establishes local scale factors relating coordinate increments to actual distances. The vector gradient, $\text{grad}(f)$, points in the direction of steepest increase in the function with respect to actual distances. By contrast, the gradient one-form, df , does not depend on a metric, so there is no concept of distance built in to it. Nevertheless, the concepts are related. The gradient one-form is given by

$$df = \left(\frac{\partial}{\partial x} f \right) dx + \left(\frac{\partial}{\partial y} f \right) dy + \left(\frac{\partial}{\partial z} f \right) dz. \quad (10.2)$$

The traditional gradient vector field is then just the raised gradient one-form (see equation 9.8). So

$$\text{grad}(f) = g^\sharp(df) \quad (10.3)$$

is computed by

```
(define (gradient metric basis)
  (compose (raise metric basis) d))
```

Let θ be a one-form field:

$$\theta = \theta_x dx + \theta_y dy + \theta_z dz. \quad (10.4)$$

We compute

$$\begin{aligned} d\theta &= \left(\frac{\partial \theta_z}{\partial y} - \frac{\partial \theta_y}{\partial z} \right) dy \wedge dz + \left(\frac{\partial \theta_x}{\partial z} - \frac{\partial \theta_z}{\partial x} \right) dz \wedge dx \\ &\quad + \left(\frac{\partial \theta_y}{\partial x} - \frac{\partial \theta_x}{\partial y} \right) dx \wedge dy. \end{aligned} \quad (10.5)$$

So the exterior-derivative expression corresponding to the vector-calculus curl is:

$$\begin{aligned} g^*(d\theta) &= \left(\frac{\partial \theta_z}{\partial y} - \frac{\partial \theta_y}{\partial z} \right) dx + \left(\frac{\partial \theta_x}{\partial z} - \frac{\partial \theta_z}{\partial x} \right) dy \\ &\quad + \left(\frac{\partial \theta_y}{\partial x} - \frac{\partial \theta_x}{\partial y} \right) dz. \end{aligned} \quad (10.6)$$

Thus, the curl of a vector field v is

$$\text{curl}(v) = g^\sharp(g^*(d(g^\flat(v)))), \quad (10.7)$$

which can be computed with

```
(define (curl metric orthonormal-basis)
  (let ((star (Hodge-star metric orthonormal-basis))
        (sharp (raise metric orthonormal-basis))
        (flat (lower metric)))
    (compose sharp star d flat)))
```

Also, we compute

$$d(g^*\theta) = \left(\frac{\partial \theta_x}{\partial x} + \frac{\partial \theta_y}{\partial y} + \frac{\partial \theta_z}{\partial z} \right) dx \wedge dy \wedge dz. \quad (10.8)$$

So the exterior-derivative expression corresponding to the vector-calculus div is

$$g^*d(g^*\theta) = \frac{\partial \theta_x}{\partial x} + \frac{\partial \theta_y}{\partial y} + \frac{\partial \theta_z}{\partial z}. \quad (10.9)$$

Thus, the divergence of a vector field v is

$$\text{div}(v) = g^*(d(g^*(g^\flat(v)))). \quad (10.10)$$

It is easily computed:

```
(define (divergence metric orthonormal-basis)
  (let ((star (Hodge-star metric orthonormal-basis))
        (flat (lower metric)))
    (compose star d star flat)))
```

The divergence is defined even if we don't have a metric, but have only a connection. In that case the divergence can be computed with

```
(define (((divergence Cartan) v) point)
  (let ((basis (Cartan->basis Cartan))
        (nabla (covariant-derivative Cartan)))
    (contract
      (lambda (ei wi)
        ((wi ((nabla ei) v)) point))
      basis)))
```

If the Cartan form is derived from a metric these programs yield the same answer.

The Laplacian is, as expected, the composition of the divergence and the gradient:

```
(define (Laplacian metric orthonormal-basis)
  (compose (divergence metric orthonormal-basis)
           (gradient metric orthonormal-basis)))
```

Spherical Coordinates

We can illustrate these by computing the formulas for the vector-calculus operators in spherical coordinates. We start with a 3-dimensional manifold, and we set up the conditions for spherical coordinates.

```
(define spherical R3-rect)

(define-coordinates (up r theta phi) spherical)

(define R3-spherical-point
  ((point spherical) (up 'r0 'theta0 'phi0)))
```

The geometry is specified by the metric:

```
(define (spherical-metric v1 v2)
  (+ (* (dr v1) (dr v2))
      (* (square r)
          (+ (* (dtheta v1) (dtheta v2))
              (* (expt (sin theta) 2)
                  (dphi v1) (dphi v2))))))
```

We also need an orthonormal basis for the spherical coordinates. The coordinate basis is orthogonal but not normalized.

```
(define e_0 d/dr)
(define e_1 (* (/ 1 r) d/dtheta))
(define e_2 (* (/ 1 (* r (sin theta))) d/dphi))

(define orthonormal-spherical-vector-basis
  (down e_0 e_1 e_2))

(define orthonormal-spherical-1form-basis
  (vector-basis-> dual orthonormal-spherical-vector-basis
    spherical))

(define orthonormal-spherical-basis
  (make-basis orthonormal-spherical-vector-basis
    orthonormal-spherical-1form-basis))
```

The components of the gradient of a scalar field are obtained using the dual basis:

```
((orthonormal-spherical-1form-basis
  ((gradient spherical-metric orthonormal-spherical-basis)
   (literal-manifold-function 'f spherical)))
 R3-spherical-point)
(up (((partial 0) f) (up r0 theta0 phi0))
  (/ (((partial 1) f) (up r0 theta0 phi0))
    r0)
  (/ (((partial 2) f) (up r0 theta0 phi0))
    (* r0 (sin theta0))))
```

To get the formulas for curl and divergence we need a vector field with components with respect to the normalized basis.

```
(define v
  (+ (* (literal-manifold-function 'v^0 spherical) e_0)
      (* (literal-manifold-function 'v^1 spherical) e_1)
      (* (literal-manifold-function 'v^2 spherical) e_2)))
```

The curl is a bit complicated:

```
((orthonormal-spherical-1form-basis
  ((curl spherical-metric orthonormal-spherical-basis) v))
R3-spherical-point)
(up
(/ (+ (* (sin theta0)
      (((partial 1) v^2) (up r0 theta0 phi0)))
  (* (cos theta0) (v^2 (up r0 theta0 phi0)))
  (* -1 (((partial 2) v^1) (up r0 theta0 phi0))))
  (* r0 (sin theta0)))
(/ (+ (* -1 r0 (sin theta0)
      (((partial 0) v^2) (up r0 theta0 phi0)))
  (* -1 (sin theta0) (v^2 (up r0 theta0 phi0)))
  (((partial 2) v^0) (up r0 theta0 phi0)))
  (* r0 (sin theta0)))
(/ (+ (* r0 (((partial 0) v^1) (up r0 theta0 phi0)))
  (v^1 (up r0 theta0 phi0))
  (* -1 (((partial 1) v^0) (up r0 theta0 phi0))))
  r0))
```

But the divergence and Laplacian are simpler

```
((divergence spherical-metric orthonormal-spherical-basis) v)
R3-spherical-point)
(+ (((partial 0) v^0) (up r0 theta0 phi0))
  (/ (* 2 (v^0 (up r0 theta0 phi0))) r0)
  (/ (((partial 1) v^1) (up r0 theta0 phi0)) r0)
  (/ (* (v^1 (up r0 theta0 phi0)) (cos theta0))
    (* r0 (sin theta0)))
  (/ (((partial 2) v^2) (up r0 theta0 phi0))
    (* r0 (sin theta0)))))

((Laplacian spherical-metric orthonormal-spherical-basis)
 (literal-manifold-function 'f spherical))
R3-spherical-point)
(+ (((partial 0) ((partial 0) f)) (up r0 theta0 phi0))
  (/ (* 2 (((partial 0) f) (up r0 theta0 phi0)))
    r0)
  (/ (((partial 1) ((partial 1) f)) (up r0 theta0 phi0))
    (expt r0 2))
  (/ (* (cos theta0) (((partial 1) f) (up r0 theta0 phi0)))
    (* (expt r0 2) (sin theta0)))
  (/ (((partial 2) ((partial 2) f)) (up r0 theta0 phi0))
    (* (expt r0 2) (expt (sin theta0) 2))))
```

10.1 The Wave Equation

The kinematics of special relativity can be formulated on a flat 4-dimensional spacetime manifold.

```
(define SR R4-rect)
(define-coordinates (up ct x y z) SR)
(define an-event ((point SR) (up 'ct0 'x0 'y0 'z0)))

(define a-vector
  (+ (* (literal-manifold-function 'v^t SR) d/dct)
     (* (literal-manifold-function 'v^x SR) d/dx)
     (* (literal-manifold-function 'v^y SR) d/dy)
     (* (literal-manifold-function 'v^z SR) d/dz)))
```

The Minkowski metric is³

$$g(u, v) = -c^2 dt(u) dt(v) + dx(u) dx(v) + dy(u) dy(v) + dz(u) dz(v). \quad (10.11)$$

As a program:

```
(define (g-Minkowski u v)
  (+ (* -1 (dct u) (dct v))
     (* (dx u) (dx v))
     (* (dy u) (dy v))
     (* (dz u) (dz v))))
```

The length of a vector is described in terms of the metric:

$$\sigma = g(v, v). \quad (10.12)$$

If σ is positive the vector is *spacelike* and its square root is the *proper length* of the vector. If σ is negative the vector is *timelike* and the square root of its negation is the *proper time* of the vector. If σ is zero the vector is *lightlike* or *null*.

```
((g-Minkowski a-vector a-vector) an-event)
(+ (* -1 (expt (v^t (up ct0 x0 y0 z0)) 2))
  (expt (v^x (up ct0 x0 y0 z0)) 2)
  (expt (v^y (up ct0 x0 y0 z0)) 2)
  (expt (v^z (up ct0 x0 y0 z0)) 2))
```

³The metric in relativity is not positive definite, so nonzero vectors can have zero length.

As an example of vector calculus in four dimensions, we can compute the wave equation for a scalar field in 4-dimensional spacetime.

We need an orthonormal basis for the spacetime:

```
(define SR-vector-basis (coordinate-system->vector-basis SR))
```

We check that it is orthonormal with respect to the metric:

```
((g-Minkowski SR-vector-basis SR-vector-basis) an-event)
(down (down -1 0 0 0)
      (down 0 1 0 0)
      (down 0 0 1 0)
      (down 0 0 0 1))
```

So, the Laplacian of a scalar field is the wave equation!

```
(define p (literal-manifold-function 'phi SR))

(((Laplacian g-Minkowski SR-basis) p) an-event)
(+ (((partial 0) ((partial 0) phi)) (up ct0 x0 y0 z0))
   (* -1 (((partial 1) ((partial 1) phi)) (up ct0 x0 y0 z0)))
   (* -1 (((partial 2) ((partial 2) phi)) (up ct0 x0 y0 z0)))
   (* -1 (((partial 3) ((partial 3) phi)) (up ct0 x0 y0 z0))))
```

10.2 Electrodynamics

Using Hodge duals we can represent electrodynamics in an elegant way. Maxwell's electrodynamics is invariant under Lorentz transformations. We use 4-dimensional rectangular coordinates for the flat spacetime of special relativity.

In this formulation of electrodynamics the electric and magnetic fields are represented together as a two-form field, the *Faraday tensor*. Under Lorentz transformations the individual components are mixed. The Faraday tensor is.⁴

```
(define (Faraday Ex Ey Ez Bx By Bz)
  (+ (* Ex (wedge dx dct))
     (* Ey (wedge dy dct))
     (* Ez (wedge dz dct))
     (* Bx (wedge dy dz))
     (* By (wedge dz dx))
     (* Bz (wedge dx dy))))
```

⁴This representation is from Misner, Thorne, and Wheeler, *Gravitation*, p.108.

The Hodge dual of the Faraday tensor exchanges the electric and magnetic fields, negating the components that will involve time. The result is called the *Maxwell tensor*:

```
(define (Maxwell Ex Ey Ez Bx By Bz)
  (+ (* -1 Bx (wedge dx dct))
      (* -1 By (wedge dy dct))
      (* -1 Bz (wedge dz dct))
      (* Ex (wedge dy dz))
      (* Ey (wedge dz dx)))
  (* Ez (wedge dx dy))))
```

We make a Hodge dual operator for this situation:

```
(define SR-star (Hodge-star g-Minkowski SR-basis))
```

And indeed, it transforms the Faraday tensor into the Maxwell tensor:

```
(((- (SR-star (Faraday 'Ex 'Ey 'Ez 'Bx 'By 'Bz))
      (Maxwell 'Ex 'Ey 'Ez 'Bx 'By 'Bz))
      (literal-vector-field 'u SR)
      (literal-vector-field 'v SR))
  an-event)
  0
```

One way to get electric fields is to have charges; magnetic fields can arise from motion of charges. In this formulation we combine the charge density and the current to make a one-form field:

```
(define (J charge-density Ix Iy Iz)
  (- (* (/ 1 :c) (+ (* Ix dx) (* Iy dy) (* Iz dz)))
      (* charge-density dct)))
```

The coefficient ($/ 1 :c$) makes the components of the one-form uniform with respect to units.

To develop Maxwell's equations we need a general Faraday field and a general current-density field:

```
(define F
  (Faraday (literal-manifold-function 'Ex SR)
            (literal-manifold-function 'Ey SR)
            (literal-manifold-function 'Ez SR)
            (literal-manifold-function 'Bx SR)
            (literal-manifold-function 'By SR)
            (literal-manifold-function 'Bz SR)))
```

```
(define 4-current
  (J (literal-manifold-function 'rho SR)
    (literal-manifold-function 'Ix SR)
    (literal-manifold-function 'Iy SR)
    (literal-manifold-function 'Iz SR)))
```

Maxwell's Equations

Maxwell's equations in the language of differential forms are

$$dF = 0, \quad (10.13)$$

$$d(g^* F) = 4\pi g^* J. \quad (10.14)$$

The first equation gives us what would be written in vector notation as

$$\operatorname{div} \vec{B} = 0, \quad (10.15)$$

$$\operatorname{curl} \vec{E} = -\frac{1}{c} \frac{d\vec{B}}{dt}. \quad (10.16)$$

The second equation gives us what would be written in vector notation as

$$\operatorname{div} \vec{E} = 4\pi\rho, \quad (10.17)$$

$$\operatorname{curl} \vec{B} = \frac{1}{c} \frac{d\vec{E}}{dt} + \frac{4\pi}{c} \vec{J}. \quad (10.18)$$

To see how these work out, we evaluate each component of dF and $d(g^* F) - 4\pi g^* J$. Since these are both two-form fields, their exterior derivatives are three-form fields, so we have to provide three basis vectors to get each component. Each component equation will yield one of Maxwell's equations, written in coordinates, without vector notation. So, the purely spatial component $dF(\partial/\partial x, \partial/\partial y, \partial/\partial z)$ of equation 10.13 is equation 10.15:

```
((d F) d/dx d/dy d/dz) an-event
(+ (((partial 1) Bx) (up ct0 x0 y0 z0))
  (((partial 2) By) (up ct0 x0 y0 z0))
  (((partial 3) Bz) (up ct0 x0 y0 z0)))
```

$$\frac{\partial B_x}{\partial x} + \frac{\partial B_y}{\partial y} + \frac{\partial B_z}{\partial z} = 0 \quad (10.19)$$

The three mixed space and time components of equation 10.13 are equation 10.16:

$$\begin{aligned} & (((d F) d/dct d/dy d/dz) \text{ an-event}) \\ & (+ (((partial 0) Bx) (up ct0 x0 y0 z0)) \\ & \quad (((partial 2) Ez) (up ct0 x0 y0 z0))) \\ & \quad (* -1 (((partial 3) Ey) (up ct0 x0 y0 z0)))) \end{aligned}$$

$$\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} = \frac{1}{c} \frac{\partial B_x}{\partial t}, \quad (10.20)$$

$$\begin{aligned} & (((d F) d/dct d/dz d/dx) \text{ an-event}) \\ & (+ (((partial 0) By) (up ct0 x0 y0 z0)) \\ & \quad (((partial 3) Ex) (up ct0 x0 y0 z0))) \\ & \quad (* -1 (((partial 1) Ez) (up ct0 x0 y0 z0)))) \end{aligned}$$

$$\frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x} = \frac{1}{c} \frac{\partial B_y}{\partial t}, \quad (10.21)$$

$$\begin{aligned} & (((d F) d/dct d/dx d/dy) \text{ an-event}) \\ & (+ (((partial 0) Bz) (up ct0 x0 y0 z0)) \\ & \quad (((partial 1) Ey) (up ct0 x0 y0 z0))) \\ & \quad (* -1 (((partial 2) Ex) (up ct0 x0 y0 z0)))) \end{aligned}$$

$$\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} = \frac{1}{c} \frac{\partial B_z}{\partial t}. \quad (10.22)$$

The purely spatial component of equation 10.14 is equation 10.17:

$$\begin{aligned} & (((- (d (SR-star F)) (* 4 :pi (SR-star 4-current)))) \\ & \quad d/dx d/dy d/dz) \\ & \quad \text{an-event}) \\ & (+ (* -4 :pi (rho (up ct0 x0 y0 z0)))) \\ & \quad (((partial 1) Ex) (up ct0 x0 y0 z0)) \\ & \quad (((partial 2) Ey) (up ct0 x0 y0 z0))) \\ & \quad (((partial 3) Ez) (up ct0 x0 y0 z0))) \end{aligned}$$

$$\frac{\partial E_x}{\partial x} + \frac{\partial E_y}{\partial y} + \frac{\partial E_z}{\partial z} = 4\pi\rho. \quad (10.23)$$

And finally, the three mixed time and space components of equation 10.14 are equation 10.18:

```
(((- (d (SR-star F)) (* 4 :pi (SR-star 4-current)))
  d/dct d/dy d/dz)
 an-event)
(+ (((partial 0) Ex) (up ct0 x0 y0 z0))
  (* -1 (((partial 2) Bz) (up ct0 x0 y0 z0)))
  (((partial 3) By) (up ct0 x0 y0 z0))
  (/ (* 4 :pi (Ix (up ct0 x0 y0 z0))) :c))
```

$$\frac{\partial B_y}{\partial z} - \frac{\partial B_z}{\partial y} = -\frac{1}{c} \frac{\partial E_x}{\partial t} - \frac{4\pi}{c} I_x, \quad (10.24)$$

```
(((- (d (SR-star F)) (* 4 :pi (SR-star 4-current)))
  d/dct d/dz d/dx)
 an-event)
(+ (((partial 0) Ey) (up ct0 x0 y0 z0))
  (* -1 (((partial 3) Bx) (up ct0 x0 y0 z0)))
  (((partial 1) Bz) (up ct0 x0 y0 z0))
  (/ (* 4 :pi (Iy (up ct0 x0 y0 z0))) :c))
```

$$\frac{\partial B_z}{\partial x} - \frac{\partial B_x}{\partial z} = -\frac{1}{c} \frac{\partial E_y}{\partial t} - \frac{4\pi}{c} I_y, \quad (10.25)$$

```
(((- (d (SR-star F)) (* 4 :pi (SR-star 4-current)))
  d/dct d/dx d/dy)
 an-event)
(+ (((partial 0) Ez) (up ct0 x0 y0 z0))
  (* -1 (((partial 1) By) (up ct0 x0 y0 z0)))
  (((partial 2) Bx) (up ct0 x0 y0 z0))
  (/ (* 4 :pi (Iz (up ct0 x0 y0 z0))) :c))
```

$$\frac{\partial B_x}{\partial y} - \frac{\partial B_y}{\partial x} = -\frac{1}{c} \frac{\partial E_z}{\partial t} - \frac{4\pi}{c} I_z. \quad (10.26)$$

Lorentz Force

The classical force on a charged particle moving in a electromagnetic field is

$$\vec{f} = q \left(\vec{E} + \frac{1}{c} \vec{v} \times \vec{B} \right). \quad (10.27)$$

We can compute this in coordinates. We construct arbitrary \vec{E} and \vec{B} vector fields and an arbitrary velocity:

```
(define E
  (up (literal-manifold-function 'Ex SR)
       (literal-manifold-function 'Ey SR)
       (literal-manifold-function 'Ez SR)))

(define B
  (up (literal-manifold-function 'Bx SR)
       (literal-manifold-function 'By SR)
       (literal-manifold-function 'Bz SR)))

(define V (up 'V_x 'V_y 'V_z))
```

The 3-space force that results is a mess:

```
(* 'q (+ (E an-event) (cross-product V (B an-event))))
(up (+ (* q (Ex (up ct0 x0 y0 z0)))
        (* q V_y (Bz (up ct0 x0 y0 z0)))
        (* -1 q V_z (By (up ct0 x0 y0 z0))))
    (+ (* q (Ey (up ct0 x0 y0 z0)))
        (* -1 q V_x (Bz (up ct0 x0 y0 z0)))
        (* q V_z (Bx (up ct0 x0 y0 z0))))
    (+ (* q (Ez (up ct0 x0 y0 z0)))
        (* q V_x (By (up ct0 x0 y0 z0)))
        (* -1 q V_y (Bx (up ct0 x0 y0 z0))))))
```

The relativistic Lorentz 4-force is usually written in coordinates as

$$f^\nu = - \sum_{\alpha, \mu} q U^\mu F_{\mu\alpha} \eta^{\alpha\nu}, \quad (10.28)$$

where U is the 4-velocity of the charged particle, F is the Faraday tensor, and $\eta^{\alpha\nu}$ are the components of the inverse of the Minkowski metric. Here is a program that computes a component of the force in terms of the Faraday tensor. The desired component is specified by a one-form.

```
(define (Force charge F 4velocity component)
  (* -1 charge
     (contract (lambda (a b)
                  (contract (lambda (e w)
                               (* (w 4velocity)
                                  (F e a)
                                  (eta-inverse b component)))
                  SR-basis))
                 SR-basis)))
```

So, for example, the force in the \hat{x} direction for a stationary particle is

```
((Force 'q F d/dct dx) an-event)
(* q (Ex (up ct0 x0 y0 z0)))
```

Notice that the 4-velocity $\partial/\partial ct$ is the 4-velocity of a stationary particle!

If we give a particle a more general timelike 4-velocity in the \hat{x} direction we can see how the \hat{y} component of the force involves both the electric and magnetic field:

```
(define (Ux beta)
  (+ (* (/ 1 (sqrt (- 1 (square beta)))) d/dct)
      (* (/ beta (sqrt (- 1 (square beta)))) d/dx)))

((Force 'q F (Ux 'v/c) dy) an-event)
(/ (+ (* -1 q v/c (Bz (up ct0 x0 y0 z0)))
      (* q (Ey (up ct0 x0 y0 z0))))
   (sqrt (+ 1 (* -1 (expt v/c 2))))))
```

Exercise 10.1: Relativistic Lorentz Force

Compute all components of the 4-force for a general timelike 4-velocity.

- a. Compare these components to the components of the nonrelativistic force given above. Interpret the differences.
- b. What is the meaning of the time component? For example, consider:

```
((Force 'q F (Ux 'v/c) dct) an-event)
(/ (* q v/c (Ex (up ct0 x0 y0 z0)))
  (sqrt (+ 1 (* -1 (expt v/c 2))))))
```

- c. Subtract the structure of components of the relativistic 3-space force from the structure of the spatial components of the 4-space force to show that they are equal.

11

Special Relativity

Although the usual treatments of special relativity begin with the Michelson-Morley experiment, this is not how Einstein began. In fact, Einstein was impressed with Maxwell's work and he was emulating Maxwell's breakthrough.

Maxwell was preceded by Faraday, Ampere, Oersted, Coulomb, Gauss, and Franklin. These giants discovered electromagnetism and worked out empirical equations that described the phenomena. They understood the existence of conserved charges and fields. Faraday invented the idea of lines of force by which fields can be visualized.

Maxwell's great insight was noticing and resolving the contradiction between the empirically-derived laws of electromagnetism and conservation of charge. He did this by introducing the then experimentally undetectable displacement-current term into one of the empirical equations. The modified equations implied a wave equation and the propagation speed of the wave predicted by the new equation turned out to be the speed of light, as measured by the eclipses of the Galilean satellites of Jupiter. The experimental confirmation by Hertz of the existence of electromagnetic radiation that obeyed Maxwell's equations capped the discovery.

By analogy, Einstein noticed that Maxwell's equations were inconsistent with Galilean relativity. In free space, where electromagnetic waves propagate, Maxwell's equations say that the vector source of electric fields is the time rate of change of the magnetic field and the vector source of magnetic field is the time rate of change of the electric field. The combination of these ideas yields the wave equation. The wave equation itself is not invariant under the Galilean transformation: As Einstein noted, if you run with the propagation speed of the wave there is no time variation in the field you observe, so there is no space variation either, contradicting the wave equation. But the Maxwell theory is beautiful, and it can be verified to a high degree of accuracy, so there must be something wrong with Galilean relativity. Einstein resolved the contradiction by generalizing the meaning of the Lorentz transformation, which was invented to explain the failure of the Michelson-Morley experiment. Lorentz and his colleagues

decided that the problem with the Michelson-Morley experiment was that matter interacting with the luminiferous ether contracts in the direction of motion. To make this consistent he had to invent a “local time” which had no clear interpretation. Einstein took the Lorentz transformation to be a fundamental replacement for the Galilean transformation in all of mechanics.

Now to the details. Before Maxwell the empirical laws of electromagnetism were as follows. Electric fields arise from charges, with the inverse square law of Coulomb. This is Carl Friedrich Gauss’s law for electrostatics:

$$\operatorname{div} \vec{E} = 4\pi\rho. \quad (11.1)$$

Magnetic fields do not have a scalar source. This is Gauss’s law for magnetostatics:

$$\operatorname{div} \vec{B} = 0. \quad (11.2)$$

Magnetic fields are produced by electric currents, as discovered by Hans Christian Oersted and quantified by André-Marie Ampère:

$$\operatorname{curl} \vec{B} = \frac{4\pi}{c} \vec{I}. \quad (11.3)$$

Michael Faraday (and Joseph Henry) discovered that electric fields are produced by moving magnetic fields:

$$\operatorname{curl} \vec{E} = \frac{-1}{c} \frac{\partial \vec{B}}{\partial t}. \quad (11.4)$$

Benjamin Franklin was the first to understand that electrical charges are conserved:

$$\operatorname{div} \vec{I} + \frac{\partial \rho}{\partial t} = 0. \quad (11.5)$$

Although these equations are written in terms of the speed of light c , these laws were originally written in terms of electrical permittivity and magnetic permeability of free space, which could be determined by measurement of the forces for given currents and charges.

It is easy to see that these equations are mutually contradictory. Indeed, if we take the divergence of equation (11.3) we get

$$\operatorname{div} \operatorname{curl} \vec{B} = 0 = \frac{4\pi}{c} \operatorname{div} \vec{I}, \quad (11.6)$$

which directly contradicts conservation of charge (11.5).

Maxwell patched this bug by adding in the displacement current, changing equation (11.3) to read

$$\operatorname{curl} \vec{B} = \frac{1}{c} \frac{\partial \vec{E}}{\partial t} + \frac{4\pi}{c} \vec{I}. \quad (11.7)$$

Maxwell proceeded by taking the curl of equation (11.4) to get

$$\operatorname{curl} \operatorname{curl} \vec{E} = \frac{-1}{c} \frac{\partial}{\partial t} \operatorname{curl} \vec{B}. \quad (11.8)$$

Expanding the left-hand side

$$\operatorname{grad} \operatorname{div} \vec{E} - \operatorname{Lap} \vec{E} = \frac{-1}{c} \frac{\partial}{\partial t} \operatorname{curl} \vec{B}, \quad (11.9)$$

substituting from equations (11.7) and (11.1), and rearranging the terms we get the inhomogeneous wave equation:

$$\operatorname{Lap} \vec{E} - \frac{1}{c^2} \frac{\partial^2 \vec{E}}{\partial t^2} = 4\pi \left(\operatorname{grad} \rho + \frac{1}{c^2} \vec{I} \right). \quad (11.10)$$

We see that in free space (in the absence of any charges or currents) we have the familiar homogeneous linear wave equation. A similar equation can be derived for the magnetic field.

Lorentz, whom Einstein also greatly respected, developed a general formula to describe the force on a particle with charge q moving with velocity \vec{v} in an electromagnetic field:

$$\vec{F} = q\vec{E} + \frac{q}{c}\vec{v} \times \vec{B}. \quad (11.11)$$

A crucial point in Einstein's inspiration for relativity is, quoting Einstein (in English translation), "During that year [1895–1896] in Aarau the question came to me: If one runs after a light wave with light velocity, then one would encounter a time-independent

wavefield. However, something like that does not seem to exist!”¹ This was the observation of the inconsistency.

Let’s be more precise about this. Consider a plane sinusoidal wave moving in the \hat{x} direction with velocity c in free space ($\rho = 0$ and $\vec{I} = 0$). This is a perfectly good solution of the wave equation. Now suppose that an observer is moving with the wave in the \hat{x} direction with velocity c . Such an observer will see no time variation of the field. So the wave equation reduces to Laplace’s equation. But a sinusoidal variation in space is not a solution of Laplace’s equation.

Einstein believed that the Maxwell-Lorentz electromagnetic theory was fundamentally correct, though he was unhappy with an apparent asymmetry in the formulation. Consider a system consisting of a conductor and a magnet. If the conductor is moved and the magnet is held stationary (a stationary magnetic field) then the charge carriers in the conductor are subject to the Lorentz force (11.11), causing them to move. However, if the magnet is moved past a stationary conductor then the changing magnetic field induces an electric field in the conductor by equation (11.4), which causes the charge carriers in the conductor to move. The actual current which results is identical for both explanations if the relative velocity of the magnet and the conductor are the same. To Einstein, there should not have been two explanations for the same phenomenon.

Invariance of the Wave Equation

Let $u = (t, x, y, z)$ be a tuple of time and space coordinates that specify a point in spacetime.² If $\phi(t, x, y, z)$ is a scalar field over time and space, the homogeneous linear wave equation is

$$\frac{\partial^2 \phi(u)}{\partial x^2} + \frac{\partial^2 \phi(u)}{\partial y^2} + \frac{\partial^2 \phi(u)}{\partial z^2} - \frac{1}{c^2} \frac{\partial^2 \phi(u)}{\partial t^2} = 0. \quad (11.12)$$

The characteristics for this equation are the “light cones.” If we define a function of spacetime points and increments, length,

¹The quote is from Pais [12], p. 131.

²Points in spacetime are often called *events*.

such that for an incremental tuple in position and time $\xi = (\Delta t, \Delta x, \Delta y, \Delta z)$ we have³

$$\text{length}_u(\xi) = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2 - (c \Delta t)^2}, \quad (11.13)$$

then the light cones are the hypersurfaces, for which

$$\text{length}_u(\Delta t, \Delta x, \Delta y, \Delta z) = 0. \quad (11.14)$$

This “length” is called the *interval*.

What is the class of transformations of time and space coordinates that leave the Maxwell-Lorentz theory invariant? The transformations that preserve the wave equation are exactly those that leave its characteristics invariant. We consider a transformation $u = A(u')$ of time and space coordinates:

$$t = A^0(t', x', y', z'), \quad (11.15)$$

$$x = A^1(t', x', y', z'), \quad (11.16)$$

$$y = A^2(t', x', y', z'), \quad (11.17)$$

$$z = A^3(t', x', y', z'). \quad (11.18)$$

If we define a new field $\psi(t', x', y', z')$ such that $\psi = \phi \circ A$, or

$$\psi(t', x', y', z') = \phi(A(t', x', y', z')), \quad (11.19)$$

then ψ will satisfy the wave equation

$$\frac{\partial^2 \psi(u')}{\partial x'^2} + \frac{\partial^2 \psi(u')}{\partial y'^2} + \frac{\partial^2 \psi(u')}{\partial z'^2} - \frac{1}{c^2} \frac{\partial^2 \psi(u')}{\partial t'^2} = 0, \quad (11.20)$$

if and only if

$$\text{length}_{u'}(\xi') = \text{length}_{A(u')}(D A \xi') = \text{length}_u(\xi). \quad (11.21)$$

But this is just a statement that the velocity of light is invariant under change of the coordinate system. The class of transformations that satisfy equation (11.21) are the Poincaré transformations.

³Here the length is independent of the spacetime point specified by u . In General Relativity we find that the metric, and thus the length function needs to vary with the point in spacetime.

11.1 Lorentz Transformations

Special relativity is usually presented in terms of global Lorentz frames, with rectangular spatial coordinates. In this context the Lorentz transformations (and, more generally, the Poincaré transformations) can be characterized as the set of affine transformations (linear transformations plus shift) of the coordinate tuple (time and spatial rectangular coordinates) that preserve the length of incremental spacetime intervals as measured by

$$f(\xi) = -(\xi^0)^2 + (\xi^1)^2 + (\xi^2)^2 + (\xi^3)^2, \quad (11.22)$$

where ξ is an incremental 4-tuple that could be added to the coordinate 4-tuple (ct, x, y, z) .⁴ The Poincaré-Lorentz transformations are of the form

$$x = \Lambda x' + a, \quad (11.23)$$

where Λ is the tuple representation of a linear transformation and a is a 4-tuple shift. Because the 4-tuple includes the time, these transformations include transformations to a uniformly moving frame. A transformation that does not rotate or shift, but just introduces relative velocity, is sometimes called a *boost*.

In general relativity, global Lorentz frames do not exist, and so global affine transformations are irrelevant. In general relativity Lorentz invariance is a local property of incremental 4-tuples at a point.

Incremental 4-tuples transform as

$$\xi = \Lambda \xi'. \quad (11.24)$$

This places a constraint on the allowed Λ

$$f(\xi') = f(\Lambda \xi'), \quad (11.25)$$

for arbitrary ξ' .

The possible Λ that are consistent with the preservation of the interval can be completely specified and conveniently parameterized.

⁴Incrementally, $\xi = \xi^0 \partial/\partial ct + \xi^1 \partial/\partial x + \xi^2 \partial/\partial y + \xi^3 \partial/\partial z$. The length of this vector, using the Minkowski metric (see equation 10.11), is the Lorentz interval, the right-hand side of equation (11.22).

Simple Lorentz Transformations

Consider the linear transformation, in the first two coordinates,

$$\begin{aligned}\xi^0 &= p(\xi')^0 + q(\xi')^1 \\ \xi^1 &= r(\xi')^0 + s(\xi')^1.\end{aligned}\tag{11.26}$$

The requirement to preserve the interval gives the constraints

$$\begin{aligned}p^2 - r^2 &= 1, \\ pq - rs &= 0, \\ q^2 - s^2 &= -1.\end{aligned}\tag{11.27}$$

There are four parameters to determine, and only three equations, so the solutions have a free parameter. It turns out that a good choice is $\beta = q/p$. Solve to find

$$p = \frac{1}{\sqrt{1 - \beta^2}} = \gamma(\beta),\tag{11.28}$$

and also $p = s$ and $q = r = \beta p$. This defines γ . Written out, the transformation is

$$\begin{aligned}\xi^0 &= \gamma(\beta)((\xi')^0 + \beta(\xi')^1) \\ \xi^1 &= \gamma(\beta)(\beta(\xi')^0 + (\xi')^1).\end{aligned}\tag{11.29}$$

Simple physical arguments⁵ show that this mathematical result relates the time and space coordinates for two systems in uniform relative motion. The parameter β is related to the relative velocity.

Consider incremental vectors as spacetime vectors relative to an origin in a global inertial frame. So, for example, $\xi = (ct, x)$, ignoring y and z for a moment. The unprimed coordinate origin $x = 0$ corresponds, in primed coordinates, to (using equations 11.29)

$$x = 0 = \gamma(\beta)(x' + \beta ct'),\tag{11.30}$$

so

$$\beta = -\frac{x'}{ct'} = -\frac{v'}{c},\tag{11.31}$$

⁵See, for instance, Mermin, “Space and Time in Special Relativity.”

with the definition $v' = x'/t'$. We see that β is minus $1/c$ times the velocity (v') of the unprimed system (which moves with its origin) as “seen” in the primed coordinates.

To check the consistency of our interpretation, we can find the velocity of the origin of the primed system ($x' = 0$) as seen by the unprimed system. Using both of equations (11.29), we find

$$\beta = \frac{x}{ct} = \frac{v}{c}. \quad (11.32)$$

So $v' = -v$.

A consistent interpretation is that the origin of the primed system moves with velocity $v = \beta c$ along the \hat{x} -axis of the unprimed system. And the unprimed system moves with the same velocity in the other direction, when viewed in terms of the primed system.

What happened to the other coordinates: y and z ? We did not need them to find this one-parameter family of Lorentz transformations. They are left alone. This mathematical result has a physical interpretation: Lengths are not affected by perpendicular boosts. Think about two observers on a collision course, each carrying a meter stick perpendicular to their relative velocity. At the moment of impact, the meter sticks must coincide. The symmetry of the situation does not permit one observer to conclude that one meter stick is shorter than the other, because the other observer must come to the same conclusion. Both observers can put their conclusions to the test upon impact.

We can fill in the components of this simple boost:

$$\begin{aligned}\xi^0 &= \gamma(\beta)((\xi')^0 + \beta(\xi')^1) \\ \xi^1 &= \gamma(\beta)(\beta(\xi')^0 + (\xi')^1) \\ \xi^2 &= (\xi')^2 \\ \xi^3 &= (\xi')^3.\end{aligned} \quad (11.33)$$

More General Lorentz Transformations

One direction was special in our consideration of simple boosts. We can make use of this fact to find boosts in any direction.

Let $c\beta = (v^0, v^1, v^2)$ be the tuple of components of the relative velocity of the origin of the primed system in the unprimed system. The components are with respect to the same rectangular basis used to define the spatial components of any incremental vector.

An incremental vector can be decomposed into vectors parallel and perpendicular to the velocity. Let ξ be the tuple of spatial components of ξ , and ξ^0 be the time component. Then,

$$\xi = \xi^\perp + \xi^\parallel, \quad (11.34)$$

where $\beta \cdot \xi^\perp = 0$. (This is the ordinary dot product in three dimensions.) Explicitly,

$$\xi^\parallel = \frac{\beta}{\beta} (\frac{\beta}{\beta} \cdot \xi), \quad (11.35)$$

where $\beta = \|\beta\|$, the magnitude of β , and

$$\xi^\perp = \xi - \xi^\parallel. \quad (11.36)$$

In the simple boost of equation (11.33) we can identify ξ^1 with the magnitude $|\xi^\parallel|$ of the parallel component. The perpendicular component is unchanged:

$$\begin{aligned} \xi^0 &= \gamma(\beta)((\xi')^0 + \beta|(\xi')^\parallel|), \\ |\xi^\parallel| &= \gamma(\beta)(\beta(\xi')^0 + |(\xi')^\parallel|), \\ \xi^\perp &= (\xi')^\perp. \end{aligned} \quad (11.37)$$

Putting the components back together, this leads to

$$\begin{aligned} \xi^0 &= \gamma(\beta)((\xi')^0 + \beta \cdot \xi') \\ \xi &= \gamma(\beta)\beta(\xi')^0 + \xi' + \frac{\gamma(\beta) - 1}{\beta^2}\beta(\beta \cdot \xi'), \end{aligned} \quad (11.38)$$

which gives the components of the general boost B along velocity $c\beta$:

$$\xi = B(\beta)(\xi'). \quad (11.39)$$

Implementation

We represent a 4-tuple as a flat up-tuple of components.

```
(define (make-4tuple ct space)
  (up ct (ref space 0) (ref space 1) (ref space 2)))
```

```
(define (4tuple->ct v) (ref v 0))
(define (4tuple->space v)
  (up (ref v 1) (ref v 2) (ref v 3)))
```

The invariant interval is then

```
(define (proper-space-interval 4tuple)
  (sqrt (- (square (4tuple->space 4tuple))
            (square (4tuple->ct 4tuple)))))
```

This is a real number for space-like intervals. A space-like interval is one where spatial distance is larger than can be traversed by light in the time interval.

It is often convenient for the interval to be real for time-like intervals, where light can traverse the spatial distance in less than the time interval.

```
(define (proper-time-interval 4tuple)
  (sqrt (- (square (4tuple->ct 4tuple))
            (square (4tuple->space 4tuple)))))
```

The general boost B is

```
(define ((general-boost beta) xi-p)
  (let ((gamma (expt (- 1 (square beta)) -1/2)))
    (let ((factor (/ (- gamma 1) (square beta))))
      (let ((xi-p-time (4tuple->ct xi-p))
            (xi-p-space (4tuple->space xi-p)))
        (let ((beta-dot-xi-p (dot-product beta xi-p-space)))
          (make-4-tuple
            (* gamma (+ xi-p-time beta-dot-xi-p))
            (+ (* gamma beta xi-p-time)
                xi-p-space
                (* factor beta beta-dot-xi-p))))))))
```

We can check that the interval is invariant:

```
(- (proper-space-interval
      ((general-boost (up 'vx 'vy 'vz))
       (make-4tuple 'ct (up 'x 'y 'z))))
      (proper-space-interval
       (make-4tuple 'ct (up 'x 'y 'z)))))

0
```

It is inconvenient that the general boost as just defined does not work if β is zero. An alternate way to specify a boost is through the magnitude of v/c and a direction:

```
(define ((general-boost2 direction v/c) 4tuple-prime)
  (let ((delta-ct-prime (4tuple->ct 4tuple-prime))
        (delta-x-prime (4tuple->space 4tuple-prime)))
    (let ((betasq (square v/c)))
      (let ((bx (dot-product direction delta-x-prime))
            (gamma (/ 1 (sqrt (- 1 betasq)))))
        (let ((alpha (- gamma 1)))
          (let ((delta-ct
                  (* gamma (+ delta-ct-prime (* bx v/c))))
                (delta-x
                  (+ (* gamma v/c direction delta-ct-prime)
                     delta-x-prime
                     (* alpha direction bx))))
                (make-4tuple delta-ct delta-x))))))
    ))))
```

This is well behaved as v/c goes to zero.

Rotations

A linear transformation that does not change the magnitude of the spatial and time components, individually, leaves the interval invariant. So a transformation that rotates the spatial coordinates and leaves the time component unchanged is also a Lorentz transformation. Let R be a 3-dimensional rotation. Then the extension to a Lorentz transformation \mathcal{R} is defined by

$$(\xi^0, \boldsymbol{\xi}) = \mathcal{R}(R)((\xi')^0, \boldsymbol{\xi}') = ((\xi')^0, R(\boldsymbol{\xi}')). \quad (11.40)$$

Examining the expression for the general boost, equation (11.38), we see that the boost transforms simply as the arguments are rotated. Indeed,

$$B(\boldsymbol{\beta}) = (\mathcal{R}(R))^{-1} \circ B(R(\boldsymbol{\beta})) \circ \mathcal{R}(R). \quad (11.41)$$

Note that $(\mathcal{R}(R))^{-1} = \mathcal{R}(R^{-1})$. The functional inverse of the extended rotation is the extension of the inverse rotation. We could use this property of boosts to think of the general boost as a combination of a rotation and a simple boost along some special direction.

The extended rotation can be implemented:

```
(define ((extended-rotation R) xi)
  (make-4tuple
    (4tuple->ct xi)
    (R (4tuple->space xi))))
```

In terms of this we can check the relation between boosts and rotations:

```
(let ((beta (up 'bx 'by 'bz))
      (xi (make-4tuple 'ct (up 'x 'y 'z))))
      (R (compose
            (rotate-x 'theta)
            (rotate-y 'phi)
            (rotate-z 'psi)))
      (R-inverse (compose
                  (rotate-z (- 'psi))
                  (rotate-y (- 'phi))
                  (rotate-x (- 'theta))))))
  (- ((general-boost beta) xi)
      ((compose (extended-rotation R-inverse)
                (general-boost (R beta))
                (extended-rotation R))
       xi)))
  (up 0 0 0 0))
```

General Lorentz Transformations

A Lorentz transformation carries an incremental 4-tuple to another 4-tuple. A general linear transformation on 4-tuples has sixteen free parameters. The interval is a symmetric quadratic form, so the requirement that the interval be preserved places only ten constraints on these parameters. Evidently there are six free parameters to the general Lorentz transformation. We already have three parameters that specify boosts (the three components of the boost velocity). And we have three more parameters in the extended rotations. The general Lorentz transformation can be constructed by combining generalized rotations and boosts.

Any Lorentz transformation has a unique decomposition as a generalized rotation followed by a general boost. Any Λ that preserves the interval can be written uniquely:

$$\Lambda = B(\boldsymbol{\beta})\mathcal{R}. \quad (11.42)$$

We can use property (11.41) to see this. Suppose we follow a general boost by a rotation. A new boost can be defined to absorb this rotation, but only if the boost is preceded by a suitable rotation:

$$\mathcal{R}(R) \circ B(\boldsymbol{\beta}) = B(R(\boldsymbol{\beta})) \circ \mathcal{R}(R). \quad (11.43)$$

Exercise 11.1: Lorentz Decomposition

The counting of free parameters supports the conclusion that the general Lorentz transformation can be constructed by combining generalized rotations and boosts. Then the decomposition (11.42) follows from property (11.41). Find a more convincing proof.

11.2 Special Relativity Frames

A new frame is defined by a Poincaré transformation from a given frame (see equation 11.23). The transformation is specified by a boost magnitude and a unit-vector boost direction, relative to the given frame, and the position of the origin of the frame being defined in the given frame.

Points in spacetime are called events. It must be possible to compare two events to determine if they are the same. This is accomplished in any particular experiment by building all frames involved in that experiment from a base frame, and representing the events as coordinates in that base frame.

When one frame is built upon another, to determine the event from frame-specific coordinates or to determine the frame-specific coordinates for an event requires composition of the boosts that relate the frames to each other. The two procedures that are required to implement this strategy are⁶

```
(define ((coordinates->event ancestor-frame this-frame
                           boost-direction v/c origin)
        coords)
  ((point ancestor-frame)
   (make-SR-coordinates ancestor-frame
     (+ ((general-boost2 boost-direction v/c) coords)
        origin)))))

(define ((event->coordinates ancestor-frame this-frame
                           boost-direction v/c origin)
        event)
  (make-SR-coordinates this-frame
    ((general-boost2 (- boost-direction) v/c)
     (- ((chart ancestor-frame) event) origin))))
```

⁶The procedure `make-SR-coordinates` labels the given coordinates with the given frame. The procedures that manipulate coordinates, such as `(point ancestor-frame)`, check that the coordinates they are given are in the appropriate frame. This error checking makes it easier to debug relativity procedures.

With these two procedures, the procedure `make-SR-frame` constructs a new relativistic frame by a Poincaré transformation from a given frame.

```
(define make-SR-frame
  (frame-maker coordinates->event event->coordinates))
```

Velocity Addition Formula

For example, we can derive the traditional velocity addition formula. Assume that we have a base frame called `home`. We can make a frame `A` by a boost from `home` in the \hat{x} direction, with components $(1, 0, 0)$, and with a dimensionless measure of the speed v_a/c . We also specify that the 4-tuple origin of this new frame coincides with the origin of `home`.

```
(define A
  (make-SR-frame 'A home
    (up 1 0 0)
    'va/c
    (make-SR-coordinates home (up 0 0 0))))
```

Frame `B` is built on frame `A` similarly, boosted by v_b/c .

```
(define B
  (make-SR-frame 'B A
    (up 1 0 0)
    'vb/c
    (make-SR-coordinates A (up 0 0 0))))
```

So any point at rest in frame `B` will have a speed relative to `home`. For the spatial origin of frame `B`, with `B` coordinates $(\text{up } 'ct 0 0 0)$, we have

```
(let ((B-origin-home-coords
      ((chart home)
       ((point B)
        (make-SR-coordinates B (up 'ct 0 0 0))))))
  (/ (ref B-origin-home-coords 1)
      (ref B-origin-home-coords 0)))
  (/ (+ va/c vb/c) (+ 1 (* va/c vb/c))))
```

obtaining the traditional velocity-addition formula. (Note that the resulting velocity is represented as a fraction of the speed of light.) This is a useful result, so:

```
(define (add-v/cs va/c vb/c)
  (/ (+ va/c vb/c)
      (+ 1 (* va/c vb/c)))))
```

11.3 Twin Paradox

Special relativity engenders a traditional conundrum: consider two twins, one of whom travels and the other stays at home. When the traveller returns it is discovered that the traveller has aged less than the twin who stayed at home. How is this possible?

The experiment begins at the start event, which we arbitrarily place at the origin of the home frame.

```
(define start-event
  ((point home)
   (make-SR-coordinates home (up 0 0 0))))
```

There is a homebody and a traveller. The traveller leaves home at the start event and proceeds at $24/25$ of the speed of light in the \hat{x} direction. We define a frame for the traveller, by boosting from the home frame.

```
(define outgoing
  (make-SR-frame 'outgoing           ; for debugging
                 home              ; base frame
                 (up 1 0 0)          ; x direction
                 24/25              ; velocity as fraction of c
                 ((chart home)
                  start-event)))
```

After 25 years of home time the traveller is 24 light-years out. We define that event using the coordinates in the home frame. Here we scale the time coordinate by the speed of light so that the units of ct slot in the 4-vector are the same as the units in the spatial slots. Since $v/c = 24/25$ we must multiply that by the speed of light to get the velocity. This is multiplied by 25 years to get the \hat{x} coordinate of the traveller in the home frame at the turning point.

```
(define traveller-at-turning-point-event
  ((point home)
   (make-SR-coordinates home
     (up (* :c 25) (* 25 24/25 :c) 0 0))))
```

Note that the first component of the coordinates of an event is the speed of light multiplied by time. The other components are distances. For example, the second component (the \hat{x} component) is the distance travelled in 25 years at $24/25$ the speed of light. This is 24 light-years.

If we examine the displacement of the traveller in his own frame we see that the traveller has aged 7 years and he has not moved from his spatial origin.

```
(- ((chart outgoing) traveller-at-turning-point-event)
    ((chart outgoing) start-event))
(up (* 7 :c) 0 0 0)
```

But in the frame of the homebody we see that the time has advanced by 25 years.

```
(- ((chart home) traveller-at-turning-point-event)
    ((chart home) start-event))
(up (* 25 :c) (* 24 :c) 0 0)
```

The proper time interval is 7 years, as seen in any frame, because it measures the aging of the traveller:

```
(proper-time-interval
 (- ((chart outgoing) traveller-at-turning-point-event)
    ((chart outgoing) start-event)))
(* 7 :c)

(proper-time-interval
 (- ((chart home) traveller-at-turning-point-event)
    ((chart home) start-event)))
(* 7 :c)
```

When the traveller is at the turning point, the event of the homebody is:

```
(define halfway-at-home-event
  ((point home)
   (make-SR-coordinates home (up (* :c 25) 0 0 0))))
```

and the homebody has aged

```
(proper-time-interval
 (- ((chart home) halfway-at-home-event)
    ((chart home) start-event)))
(* 25 :c)
```

```
(proper-time-interval
  (- ((chart outgoing) halfway-at-home-event)
      ((chart outgoing) start-event)))
(* 25 :c)
```

as seen from either frame.

As seen by the traveller, home is moving in the $-\hat{x}$ direction at $24/25$ of the velocity of light. At the turning point (7 years by his time) home is at:

```
(define home-at-outgoing-turning-point-event
  ((point outgoing)
   (make-SR-coordinates outgoing
     (up (* 7 :c) (* 7 -24/25 :c) 0 0))))
```

Since home is speeding away from the traveller, the twin at home has aged less than the traveller. This may seem weird, but it is OK because this event is different from the halfway event in the home frame.

```
(proper-time-interval
  (- ((chart home) home-at-outgoing-turning-point-event)
      ((chart home) start-event)))
(* 49/25 :c)
```

The traveller turns around abruptly at this point (painful!) and begins the return trip. The incoming trip is the reverse of the outgoing trip, with origin at the turning-point event:

```
(define incoming
  (make-SR-frame 'incoming home
    (up -1 0 0) 24/25
    ((chart home)
     traveller-at-turning-point-event)))
```

After 50 years of home time the traveller reunites with the homebody:

```
(define end-event
  ((point home)
   (make-SR-coordinates home (up (* :c 50) 0 0 0))))
```

Indeed, the traveller comes home after 7 more years in the incoming frame:

```
(- ((chart incoming) end-event)
  (make-SR-coordinates incoming
    (up (* :c 7) 0 0 0)))
(up 0 0 0 0)

(- ((chart home) end-event)
  ((chart home)
   ((point incoming)
    (make-SR-coordinates incoming
      (up (* :c 7) 0 0 0))))
  (up 0 0 0 0))
```

The traveller ages only 7 years on the return segment, so his total aging is 14 years:

```
(+ (proper-time-interval
  (- ((chart outgoing) traveller-at-turning-point-event)
    ((chart outgoing) start-event)))
(proper-time-interval
  (- ((chart incoming) end-event)
    ((chart incoming) traveller-at-turning-point-event))))
(* 14 :c))
```

But the homebody ages 50 years:

```
(proper-time-interval
  (- ((chart home) end-event)
    ((chart home) start-event)))
(* 50 :c))
```

At the turning point of the traveller the homebody is at

```
(define home-at-incoming-turning-point-event
  ((point incoming)
   (make-SR-coordinates incoming
     (up 0 (* 7 -24/25 :c) 0 0))))
```

The time elapsed for the homebody between the reunion and the turning point of the homebody, as viewed by the incoming traveller, is about 2 years.

```
(proper-time-interval
  (- ((chart home) end-event)
    ((chart home) home-at-incoming-turning-point-event)))
(* 49/25 :c))
```

Thus the aging of the homebody occurs at the turnaround, from the point of view of the traveller.

A Scheme

Programming languages should be designed not by piling feature on top of feature, but by removing the weaknesses and restrictions that make additional features appear necessary. Scheme demonstrates that a very small number of rules for forming expressions, with no restrictions on how they are composed, suffice to form a practical and efficient programming language that is flexible enough to support most of the major programming paradigms in use today.

IEEE Standard for the Scheme Programming Language [10], p. 3

Here we give an elementary introduction to Scheme.¹ For a more precise explanation of the language see the IEEE standard [10]; for a longer introduction see the textbook [1].

Scheme is a simple programming language based on expressions. An expression names a value. For example, the numeral 3.14 names an approximation to a familiar number. There are primitive expressions, such as a numeral, that we directly recognize, and there are compound expressions of several kinds.

Procedure Calls

A *procedure call* is a kind of compound expression. A procedure call is a sequence of expressions delimited by parentheses. The first subexpression in a procedure call is taken to name a procedure, and the rest of the subexpressions are taken to name the arguments to that procedure. The value produced by the procedure when applied to the given arguments is the value named by the procedure call. For example,

¹Many of the statements here are valid only assuming that no assignments are used.

```
(+ 1 2.14)
```

```
3.14
```

```
(+ 1 (* 2 1.07))
```

```
3.14
```

are both compound expressions that name the same number as the numeral 3.14.² In these cases the symbols `+` and `*` name procedures that add and multiply, respectively. If we replace any subexpression of any expression with an expression that names the same thing as the original subexpression, the thing named by the overall expression remains unchanged. In general, a procedure call is written

```
( operator operand-1 ... operand-n )
```

where *operator* names a procedure and *operand-i* names the *i*th argument.³

Lambda Expressions

Just as we use numerals to name numbers, we use λ -expressions to name procedures.⁴ For example, the procedure that squares its input can be written:

```
(lambda (x) (* x x))
```

This expression can be read: “The procedure of one argument, *x*, that multiplies *x* by *x*.” Of course, we can use this expression in any context where a procedure is needed. For example,

```
((lambda (x) (* x x)) 4)
```

```
16
```

The general form of a λ -expression is

²In examples we show the value that would be printed by the Scheme system using slanted characters following the input expression.

³In Scheme every parenthesis is essential: you cannot add extra parentheses or remove any.

⁴The logician Alonzo Church [5] invented λ -notation to allow the specification of an anonymous function of a named parameter: $\lambda x[\text{expression in } x]$. This is read, “That function of one argument that is obtained by substituting the argument for *x* in the indicated expression.”

```
(lambda formal-parameters body)
```

where *formal-parameters* is a list of symbols that will be the names of the arguments to the procedure and *body* is an expression that may refer to the formal parameters. The value of a procedure call is the value of the body of the procedure with the arguments substituted for the formal parameters.

Definitions

We can use the `define` construct to give a name to any object. For example, if we make the definitions⁵

```
(define pi 3.141592653589793)  
(define square (lambda (x) (* x x)))
```

we can then use the symbols `pi` and `square` wherever the numeral or the λ -expression could appear. For example, the area of the surface of a sphere of radius 5 meters is

```
(* 4 pi (square 5))  
314.1592653589793
```

Procedure definitions may be expressed more conveniently using “syntactic sugar.” The squaring procedure may be defined

```
(define (square x) (* x x))
```

which we may read: “To square *x* multiply *x* by *x*.”

In Scheme, procedures may be passed as arguments and returned as values. For example, it is possible to make a procedure that implements the mathematical notion of the composition of two functions:⁶

⁵The definition of `square` given here is not the definition of `square` in the Scmutils system. In Scmutils, `square` is extended for tuples to mean the sum of the squares of the components of the tuple. However, for arguments that are not tuples the Scmutils `square` does multiply the argument by itself.

⁶The examples are indented to help with readability. Scheme does not care about extra white space, so we may add as much as we please to make things easier to read.

```
(define compose
  (lambda (f g)
    (lambda (x)
      (f (g x)))))

((compose square sin) 2)
.826821810431806

(square (sin 2))
.826821810431806
```

Using the syntactic sugar shown above, we can write the definition more conveniently. The following are both equivalent to the definition above:

```
(define (compose f g)
  (lambda (x)
    (f (g x)))

(define ((compose f g) x)
  (f (g x)))
```

Conditionals

Conditional expressions may be used to choose among several expressions to produce a value. For example, a procedure that implements the absolute value function may be written:

```
(define (abs x)
  (cond ((< x 0) (- x))
        ((= x 0) x)
        ((> x 0) x)))
```

The conditional `cond` takes a number of clauses. Each clause has a predicate expression, which may be either true or false, and a consequent expression. The value of the `cond` expression is the value of the consequent expression of the first clause for which the corresponding predicate expression is true. The general form of a conditional expression is

```
(cond ( predicate-1 consequent-1)
      ...
      ( predicate-n consequent-n))
```

For convenience there is a special predicate expression `else` that can be used as the predicate in the last clause of a `cond`. The `if`

construct provides another way to make a conditional when there is only a binary choice to be made. For example, because we have to do something special only when the argument is negative, we could have defined `abs` as:

```
(define (abs x)
  (if (< x 0)
      (- x)
      x))
```

The general form of an `if` expression is

```
(if predicate consequent alternative)
```

If the *predicate* is true the value of the `if` expression is the value of the *consequent*, otherwise it is the value of the *alternative*.

Recursive Procedures

Given conditionals and definitions, we can write recursive procedures. For example, to compute the *n*th factorial number we may write:

```
(define (factorial n)
  (if (= n 0)
      1
      (* n (factorial (- n 1)))))

(factorial 6)
720

(factorial 40)
815915283247897734345611269596115894272000000000
```

Local Names

The `let` expression is used to give names to objects in a local context. For example,

```
(define (f radius)
  (let ((area (* 4 pi (square radius)))
        (volume (* 4/3 pi (cube radius))))
    (/ volume area)))

(f 3)
1
```

The general form of a `let` expression is

```
(let ((variable-1 expression-1)
     ...
     (variable-n expression-n))
  body)
```

The value of the `let` expression is the value of the `body` expression in the context where the variables *variable-i* have the values of the expressions *expression-i*. The expressions *expression-i* may not refer to any of the variables.

A slight variant of the `let` expression provides a convenient way to express looping constructs. We can write a procedure that implements an alternative algorithm for computing factorials as follows:

```
(define (factorial n)
  (let factlp ((count 1) (answer 1))
    (if (> count n)
        answer
        (factlp (+ count 1) (* count answer)))))

(factorial 6)
```

720

Here, the symbol `factlp` following the `let` is locally defined to be a procedure that has the variables `count` and `answer` as its formal parameters. It is called the first time with the expressions 1 and 1, initializing the loop. Whenever the procedure named `factlp` is called later, these variables get new values that are the values of the operand expressions `(+ count 1)` and `(* count answer)`.

Compound Data—Lists and Vectors

Data can be glued together to form compound data structures. A list is a data structure in which the elements are linked sequentially. A Scheme vector is a data structure in which the elements are packed in a linear array. New elements can be added to lists, but to access the *n*th element of a list takes computing time proportional to *n*. By contrast a Scheme vector is of fixed length, and its elements can be accessed in constant time. All data structures in this book are implemented as combinations of lists and Scheme vectors. Compound data objects are constructed from components by procedures called constructors and the components are accessed by selectors.

The procedure `list` is the constructor for lists. The selector `list-ref` gets an element of the list. All selectors in Scheme are zero-based. For example,

```
(define a-list (list 6 946 8 356 12 620))

a-list
(6 946 8 356 12 620)

(list-ref a-list 3)
356

(list-ref a-list 0)
6
```

Lists are built from pairs. A pair is made using the constructor `cons`. The selectors for the two components of the pair are `car` and `cdr` (pronounced “could-er”).⁷ A list is a chain of pairs, such that the `car` of each pair is the list element and the `cdr` of each pair is the next pair, except for the last `cdr`, which is a distinguishable value called the empty list and is written `()`. Thus,

```
(car a-list)
6

(cdr a-list)
(946 8 356 12 620)

(car (cdr a-list))
946

(define another-list
  (cons 32 (cdr a-list)))

another-list
(32 946 8 356 12 620)

(car (cdr another-list))
946
```

Both `a-list` and `another-list` share the same tail (their `cdr`).

⁷These names are accidents of history. They stand for “Contents of the Address part of Register” and “Contents of the Decrement part of Register” of the IBM 704 computer, which was used for the first implementation of Lisp in the late 1950s. Scheme is a dialect of Lisp.

There is a predicate `pair?` that is true of pairs and false on all other types of data.

Vectors are simpler than lists. There is a constructor `vector` that can be used to make vectors and a selector `vector-ref` for accessing the elements of a vector:

```
(define a-vector
  (vector 37 63 49 21 88 56))
```

```
a-vector
#(37 63 49 21 88 56)
```

```
(vector-ref a-vector 3)
21
```

```
(vector-ref a-vector 0)
37
```

Notice that a vector is distinguished from a list on printout by the character `#` appearing before the initial parenthesis.

There is a predicate `vector?` that is true of vectors and false for all other types of data.

The elements of lists and vectors may be any kind of data, including numbers, procedures, lists, and vectors. Numerous other procedures for manipulating list-structured data and vector-structured data can be found in the Scheme online documentation.

Symbols

Symbols are a very important kind of primitive data type that we use to make programs and algebraic expressions. You probably have noticed that Scheme programs look just like lists. In fact, they are lists. Some of the elements of the lists that make up programs are symbols, such as `+` and `vector`.⁸ If we are to make programs that can manipulate programs, we need to be able to write an expression that names such a symbol. This is accomplished by the mechanism of *quotation*. The name of the symbol `+` is the expression `'+`, and in general the name of an expression is the expression preceded by a single quote character. Thus the name of the expression `(+ 3 a)` is `'(+ 3 a)`.

⁸Symbols may have any number of characters. A symbol may not contain whitespace or a delimiter character, such as parentheses, brackets, quotation marks, comma, or `#`.

We can test if two symbols are identical by using the predicate `eq?`. For example, we can write a program to determine if an expression is a sum:

```
(define (sum? expression)
  (and (pair? expression)
    (eq? (car expression) '+)))

(sum? '(+ 3 a))
#t

(sum? '(* 3 a))
#f
```

Here `#t` and `#f` are the printed representations of the boolean values true and false.

Consider what would happen if we were to leave out the quote in the expression `(sum? '(+ 3 a))`. If the variable `a` had the value 4 we would be asking if 7 is a sum. But what we wanted to know was whether the expression `(+ 3 a)` is a sum. That is why we need the quote.

B

Our Notation

An adequate notation should be understood by at least two people, one of whom may be the author.
Abdus Salam (1950).

We adopt a *functional mathematical notation* that is close to that used by Spivak in his *Calculus on Manifolds* [17]. The use of functional notation avoids many of the ambiguities of traditional mathematical notation that can impede clear reasoning. Functional notation carefully distinguishes the function from the value of the function when applied to particular arguments. In functional notation mathematical expressions are unambiguous and self-contained.

We adopt a *generic arithmetic* in which the basic arithmetic operations, such as addition and multiplication, are extended to a wide variety of mathematical types. Thus, for example, the addition operator $+$ can be applied to numbers, tuples of numbers, matrices, functions, etc. Generic arithmetic formalizes the common informal practice used to manipulate mathematical objects.

We often want to manipulate aggregate quantities, such as the collection of all of the rectangular coordinates of a collection of particles, without explicitly manipulating the component parts. Tensor arithmetic provides a traditional way of manipulating aggregate objects: Indices label the parts; conventions, such as the summation convention, are introduced to manipulate the indices. We introduce a *tuple arithmetic* as an alternative way of manipulating aggregate quantities that usually lets us avoid labeling the parts with indices. Tuple arithmetic is inspired by tensor arithmetic but it is more general: not all of the components of a tuple need to be of the same size or type.

The mathematical notation is in one-to-one correspondence with expressions of the computer language *Scheme* [10]. Scheme is based on the λ -calculus [5] and directly supports the manipulation of functions. We augment Scheme with symbolic, numerical, and generic features to support our applications. For a simple

introduction to Scheme, see Appendix A. The correspondence between the mathematical notation and Scheme requires that mathematical expressions be unambiguous and self-contained. Scheme provides immediate feedback in verification of mathematical deductions and facilitates the exploration of the behavior of systems.

Functions

The expression $f(x)$ denotes the value of the function f at the given argument x ; when we wish to denote the function we write just f . Functions may take several arguments. For example, we may have the function that gives the Euclidean distance between two points in the plane given by their rectangular coordinates:

$$d(x_1, y_1, x_2, y_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}. \quad (\text{B.1})$$

In Scheme we can write this as:

```
(define (d x1 y1 x2 y2)
  (sqrt (+ (square (- x2 x1)) (square (- y2 y1)))))
```

Functions may be composed if the range of one overlaps the domain of the other. The composition of functions is constructed by passing the output of one to the input of the other. We write the composition of two functions using the \circ operator:

$$(f \circ g) : x \mapsto (f \circ g)(x) = f(g(x)). \quad (\text{B.2})$$

A procedure `h` that computes the cube of the sine of its argument may be defined by composing the procedures `cube` and `sin`:

```
(define h (compose cube sin))

(h 2)
.7518269446689928
```

which is the same as

```
(cube (sin 2))
.7518269446689928
```

Arithmetic is extended to the manipulation of functions: the usual mathematical operations may be applied to functions. Examples are addition and multiplication; we may add or multiply two functions if they take the same kinds of arguments and if their

values can be added or multiplied:

$$(f + g)(x) = f(x) + g(x), \\ (fg)(x) = f(x)g(x). \quad (\text{B.3})$$

A procedure `g` that multiplies the cube of its argument by the sine of its argument is

```
(define g (* cube sin))

(g 2)
7.274379414605454

(* (cube 2) (sin 2))
7.274379414605454
```

Symbolic Values

As in usual mathematical notation, arithmetic is extended to allow the use of symbols that represent unknown or incompletely specified mathematical objects. These symbols are manipulated as if they had values of a known type. By default, a Scheme symbol is assumed to represent a real number. So the expression '`a`' is a literal Scheme symbol that represents an unspecified real number:

```
((compose cube sin) 'a)
(expt (sin a) 3)
```

The default printer simplifies the expression,¹ and displays it in a readable form. We can use the simplifier to verify a trigonometric identity:

```
((- (+ (square sin) (square cos)) 1) 'a)
0
```

Just as it is useful to be able to manipulate symbolic numbers, it is useful to be able to manipulate symbolic functions. The procedure `literal-function` makes a procedure that acts as a function having no properties other than its name. By default, a literal function is defined to take one real argument and produce

¹The procedure `print-expression` can be used in a program to print a simplified version of an expression. The default printer in the user interface incorporates the simplifier.

one real value. For example, we may want to work with a function $f : \mathbf{R} \rightarrow \mathbf{R}$:

```
((literal-function 'f) 'x)
(f x)

((compose (literal-function 'f) (literal-function 'g)) 'x)
(f (g x))
```

We can also make literal functions of multiple, possibly structured arguments that return structured values. For example, to denote a literal function named g that takes two real arguments and returns a real value ($g : \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}$) we may write:

```
(define g (literal-function 'g (-> (X Real Real) Real)))

(g 'x 'y)
(g x y)
```

We may use such a literal function anywhere that an explicit function of the same type may be used.

There is a whole language for describing the type of a literal function in terms of the number of arguments, the types of the arguments, and the types of the values. Here we describe a function that maps pairs of real numbers to real numbers with the expression $(\rightarrow (X \text{ Real} \text{ Real}) \text{ Real})$. Later we will introduce structured arguments and values and show extensions of literal functions to handle these.

Tuples

There are two kinds of tuples: *up* tuples and *down* tuples. We write tuples as ordered lists of their components; a tuple is delimited by parentheses if it is an up tuple and by square brackets if it is a down tuple. For example, the up tuple v of velocity components v^0 , v^1 , and v^2 is

$$v = (v^0, v^1, v^2). \quad (\text{B.4})$$

The down tuple p of momentum components p_0 , p_1 , and p_2 is

$$p = [p_0, p_1, p_2]. \quad (\text{B.5})$$

A component of an up tuple is usually identified by a superscript. A component of a down tuple is usually identified by a subscript. We use zero-based indexing when referring to tuple elements. This notation follows the usual convention in tensor arithmetic.

We make tuples with the constructors up and down:

```
(define v (up 'v^0 'v^1 'v^2))

v
(up v^0 v^1 v^2)

(define p (down 'p_0 'p_1 'p_2))

p
(down p_0 p_1 p_2)
```

Note that v^0 and p_2 are just symbols. The caret and underline characters are symbol constituents, so there is no meaning other than mnemonic to the structure of these symbols. However, our software can also display expressions using TeX, and then these decorations turn into superscripts and subscripts.

Tuple arithmetic is different from the usual tensor arithmetic in that the components of a tuple may also be tuples and different components need not have the same structure. For example, a tuple structure s of phase-space states is

$$s = (t, (x, y), [p_x, p_y]). \quad (\text{B.6})$$

It is an up tuple of the time, the coordinates, and the momenta. The time t has no substructure. The coordinates are an up tuple of the coordinate components x and y . The momentum is a down tuple of the momentum components p_x and p_y . In Scheme this is written:

```
(define s (up 't (up 'x 'y) (down 'p_x 'p_y)))
```

In order to reference components of tuple structures there are selector functions, for example:

$$\begin{aligned} I(s) &= s \\ I_0(s) &= t \\ I_1(s) &= (x, y) \\ I_2(s) &= [p_x, p_y] \\ I_{1,0}(s) &= x \\ &\dots \\ I_{2,1}(s) &= p_y. \end{aligned} \quad (\text{B.7})$$

The sequence of integer subscripts on the selector describes the access chain to the desired component.

The procedure `component` is the general selector procedure that implements the selector function I_z :

```
((component 0 1) (up (up 'a 'b) (up 'c 'd)))
b
```

To access a component of a tuple we may also use the selector procedure `ref`, which takes a tuple and an index and returns the indicated element of the tuple:

```
(ref (up 'a 'b 'c) 1)
b
```

We use zero-based indexing everywhere. The procedure `ref` can be used to access any substructure of a tree of tuples:

```
(ref (up (up 'a 'b) (up 'c 'd)) 0 1)
b
```

Two up tuples of the same length may be added or subtracted, elementwise, to produce an up tuple, if the components are compatible for addition. Similarly, two down tuples of the same length may be added or subtracted, elementwise, to produce a down tuple, if the components are compatible for addition.

Any tuple may be multiplied by a number by multiplying each component by the number. Numbers may, of course, be multiplied. Tuples that are compatible for addition form a vector space.

For convenience we define the square of a tuple to be the sum of the squares of the components of the tuple. Tuples can be multiplied, as described below, but the square of a tuple is not the product of the tuple with itself.

The meaning of multiplication of tuples depends on the structure of the tuples. Two tuples are compatible for contraction if they are of opposite types, they are of the same length, and corresponding elements have the following property: either they are both tuples and are compatible for contraction, or at least one is not a tuple. If two tuples are compatible for contraction then generic multiplication is interpreted as contraction: the result is the sum of the products of corresponding components of the tuples. For example, p and v introduced in equations (B.4) and (B.5) above are compatible for contraction; the product is

$$pv = p_0v^0 + p_1v^1 + p_2v^2. \quad (\text{B.8})$$

So the product of tuples that are compatible for contraction is an inner product. Using the tuples p and v defined above gives us

$$(* \ p \ v) \\ (+ (* p_{-0} \ v^0) \ (* p_{-1} \ v^1) \ (* p_{-2} \ v^2))$$

Contraction of tuples is commutative: $pv = vp$. Caution: Multiplication of tuples that are compatible for contraction is, in general, not associative. For example, let $u = (5, 2)$, $v = (11, 13)$, and $g = [[3, 5], [7, 9]]$. Then $u(gv) = 964$, but $(ug)v = 878$. The expression ugv is ambiguous. An expression that has this ambiguity does not occur in this book.

The rule for multiplying two structures that are not compatible for contraction is simple. If A and B are not compatible for contraction, the product AB is a tuple of type B whose components are the products of A and the components of B . The same rule is applied recursively in multiplying the components. So if $B = (B^0, B^1, B^2)$, the product of A and B is

$$AB = (AB^0, AB^1, AB^2). \quad (\text{B.9})$$

If A and C are not compatible for contraction and $C = [C_0, C_1, C_2]$, the product is

$$AC = [AC_0, AC_1, AC_2]. \quad (\text{B.10})$$

Tuple structures can be made to represent linear transformations. For example, the rotation commonly represented by the matrix

$$\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (\text{B.11})$$

can be represented as a tuple structure:²

$$\left[\begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix} \begin{pmatrix} -\sin \theta \\ \cos \theta \end{pmatrix} \right]. \quad (\text{B.12})$$

²To emphasize the relationship of simple tuple structures to matrix notation we often format **up** tuples as vertical arrangements of components and **down** tuples as horizontal arrangements of components. However, we could just as well have written this tuple as $[(\cos \theta, \sin \theta), (-\sin \theta, \cos \theta)]$.

Such a tuple is compatible for contraction with an up tuple that represents a vector. So, for example:

$$\left[\begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix} \begin{pmatrix} -\sin \theta \\ \cos \theta \end{pmatrix} \right] \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{pmatrix}. \quad (\text{B.13})$$

The product of two tuples that represent linear transformations—which are not compatible for contraction—represents the composition of the linear transformations. For example, the product of the tuples representing two rotations is

$$\begin{aligned} & \left[\begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix} \begin{pmatrix} -\sin \theta \\ \cos \theta \end{pmatrix} \right] \left[\begin{pmatrix} \cos \varphi \\ \sin \varphi \end{pmatrix} \begin{pmatrix} -\sin \varphi \\ \cos \varphi \end{pmatrix} \right] \\ &= \left[\begin{pmatrix} \cos(\theta + \varphi) \\ \sin(\theta + \varphi) \end{pmatrix} \begin{pmatrix} -\sin(\theta + \varphi) \\ \cos(\theta + \varphi) \end{pmatrix} \right]. \end{aligned} \quad (\text{B.14})$$

Multiplication of tuples that represent linear transformations is associative but generally not commutative, just as the composition of the transformations is associative but not generally commutative.

Derivatives

The derivative of a function f is a function, denoted by Df . Our notational convention is that D is a high-precedence operator. Thus D operates on the adjacent function before any other application occurs: $Df(x)$ is the same as $(Df)(x)$. Higher-order derivatives are described by exponentiating the derivative operator. Thus the n th derivative of a function f is notated as $D^n f$.

The Scheme procedure for producing the derivative of a function is named `D`. The derivative of the `sin` procedure is a procedure that computes `cos`:

```
(define derivative-of-sine (D sin))

(derivative-of-sine 'x)
(cos x)
```

The derivative of a function f is the function Df whose value for a particular argument is something that can be multiplied by an increment Δx in the argument to get a linear approximation to the increment in the value of f :

$$f(x + \Delta x) \approx f(x) + Df(x)\Delta x. \quad (\text{B.15})$$

For example, let f be the function that cubes its argument ($f(x) = x^3$); then Df is the function that yields three times the square of its argument ($Df(y) = 3y^2$). So $f(5) = 125$ and $Df(5) = 75$. The value of f with argument $x + \Delta x$ is

$$f(x + \Delta x) = (x + \Delta x)^3 = x^3 + 3x^2\Delta x + 3x\Delta x^2 + \Delta x^3 \quad (\text{B.16})$$

and

$$Df(x)\Delta x = 3x^2\Delta x. \quad (\text{B.17})$$

So $Df(x)$ multiplied by Δx gives us the term in $f(x + \Delta x)$ that is linear in Δx , providing a good approximation to $f(x + \Delta x) - f(x)$ when Δx is small.

Derivatives of compositions obey the chain rule:

$$D(f \circ g) = ((Df) \circ g) \cdot Dg. \quad (\text{B.18})$$

So at x ,

$$(D(f \circ g))(x) = Df(g(x)) \cdot Dg(x). \quad (\text{B.19})$$

D is an example of an *operator*. An operator is like a function except that multiplication of operators is interpreted as composition, whereas multiplication of functions is multiplication of the values (see equation B.3). If D were an ordinary function, then the rule for multiplication would imply that D^2f would just be the product of Df with itself, which is not what is intended. A product of a number and an operator scales the operator. So, for example

```
(((* 5 D) cos) 'x)
(* -5 (sin x)))
```

Arithmetic is extended to allow manipulation of operators. A typical operator is $(D+I)(D-I) = D^2 - I$, where I is the identity operator, subtracts a function from its second derivative. Such an operator can be constructed and used in Scheme as follows:

```
(((* (+ D I) (- D I)) (literal-function 'f)) 'x)
(+ (((expt D 2) f) x) (* -1 (f x))))
```

Derivatives of Functions of Multiple Arguments

The derivative generalizes to functions that take multiple arguments. The derivative of a real-valued function of multiple arguments is an object whose contraction with the tuple of increments in the arguments gives a linear approximation to the increment in the function's value.

A function of multiple arguments can be thought of as a function of an up tuple of those arguments. Thus an incremental argument tuple is an up tuple of components, one for each argument position. The derivative of such a function is a down tuple of the partial derivatives of the function with respect to each argument position.

Suppose we have a real-valued function g of two real-valued arguments, and we want to approximate the increment in the value of g from its value at x, y . If the arguments are incremented by the tuple $(\Delta x, \Delta y)$ we compute:

$$\begin{aligned} Dg(x, y) \cdot (\Delta x, \Delta y) &= [\partial_0 g(x, y), \partial_1 g(x, y)] \cdot (\Delta x, \Delta y) \\ &= \partial_0 g(x, y) \Delta x + \partial_1 g(x, y) \Delta y. \end{aligned} \quad (\text{B.20})$$

Using the two-argument literal function `g` defined on page 198, we have:

```
((D g) 'x 'y)
(down (((partial 0) g) x y) (((partial 1) g) x y))
```

In general, partial derivatives are just the components of the derivative of a function that takes multiple arguments (or structured arguments or both; see below). So a partial derivative of a function is a composition of a component selector and the derivative of that function.³ Indeed:

$$\partial_0 g = I_0 \circ Dg, \quad (\text{B.21})$$

$$\partial_1 g = I_1 \circ Dg. \quad (\text{B.22})$$

Concretely, if

$$g(x, y) = x^3 y^5 \quad (\text{B.23})$$

³Partial derivative operators such as `(partial 2)` are operators, so `(expt (partial 1) 2)` is a second partial derivative.

then

$$Dg(x, y) = [3x^2y^5, 5x^3y^4] \quad (\text{B.24})$$

and the first-order approximation of the increment for changing the arguments by Δx and Δy is

$$\begin{aligned} g(x + \Delta x, y + \Delta y) - g(x, y) &\approx [3x^2y^5, 5x^3y^4] \cdot (\Delta x, \Delta y) \\ &= 3x^2y^5\Delta x + 5x^3y^4\Delta y. \end{aligned} \quad (\text{B.25})$$

Partial derivatives of compositions also obey a chain rule:

$$\partial_i(f \circ g) = ((Df) \circ g) \cdot \partial_i g. \quad (\text{B.26})$$

So if x is a tuple of arguments, then

$$(\partial_i(f \circ g))(x) = Df(g(x)) \cdot \partial_i g(x). \quad (\text{B.27})$$

Mathematical notation usually does not distinguish functions of multiple arguments and functions of the tuple of arguments. Let $h((x, y)) = g(x, y)$. The function h , which takes a tuple of arguments x and y , is not distinguished from the function g that takes arguments x and y . We use both ways of defining functions of multiple arguments. The derivatives of both kinds of functions are compatible for contraction with a tuple of increments to the arguments. Scheme comes in handy here:

```
(define (h s)
  (g (ref s 0) (ref s 1)))

(h (up 'x 'y))
(g x y)

((D g) 'x 'y)
(down (((partial 0) g) x y) (((partial 1) g) x y))

((D h) (up 'x 'y))
(down (((partial 0) g) x y) (((partial 1) g) x y))
```

A phase-space state function is a function of time, coordinates, and momenta. Let H be such a function. The value of H is $H(t, (x, y), [p_x, p_y])$ for time t , coordinates (x, y) , and momenta $[p_x, p_y]$. Let s be the phase-space state tuple as in (B.6):

$$s = (t, (x, y), [p_x, p_y]). \quad (\text{B.28})$$

The value of H for argument tuple s is $H(s)$. We use both ways of writing the value of H .

We often show a function of multiple arguments that include tuples by indicating the boundaries of the argument tuples with semicolons and separating their components with commas. If H is a function of phase-space states with arguments t , (x, y) , and $[p_x, p_y]$, we may write $H(t; x, y; p_x, p_y)$. This notation loses the up/down distinction, but our semicolon-and-comma notation is convenient and reasonably unambiguous.

The derivative of H is a function that produces an object that can be contracted with an increment in the argument structure to produce an increment in the function's value. The derivative is a down tuple of three partial derivatives. The first partial derivative is the partial derivative with respect to the numerical argument. The second partial derivative is a down tuple of partial derivatives with respect to each component of the up-tuple argument. The third partial derivative is an up tuple of partial derivatives with respect to each component of the down-tuple argument:

$$\begin{aligned} DH(s) &= [\partial_0 H(s), \partial_1 H(s), \partial_2 H(s)] \\ &= [\partial_0 H(s), [\partial_{1,0} H(s), \partial_{1,1} H(s)], (\partial_{2,0} H(s), \partial_{2,1} H(s))], \end{aligned} \quad (\text{B.29})$$

where $\partial_{1,0}$ indicates the partial derivative with respect to the first component (index 0) of the second argument (index 1) of the function, and so on. Indeed, $\partial_z F = I_z \circ DF$ for any function F and access chain z . So, if we let Δs be an incremental phase-space state tuple,

$$\Delta s = (\Delta t, (\Delta x, \Delta y), [\Delta p_x, \Delta p_y]), \quad (\text{B.30})$$

then

$$\begin{aligned} DH(s)\Delta s &= \partial_0 H(s)\Delta t \\ &\quad + \partial_{1,0} H(s)\Delta x + \partial_{1,1} H(s)\Delta y \\ &\quad + \partial_{2,0} H(s)\Delta p_x + \partial_{2,1} H(s)\Delta p_y. \end{aligned} \quad (\text{B.31})$$

Caution: Partial derivative operators with respect to different structured arguments generally do not commute.

In Scheme we must make explicit choices. We usually assume that phase-space state functions are functions of the tuple. For example,

```
(define H
  (literal-function 'H
    (-> (UP Real (UP Real Real) (DOWN Real Real)) Real)))

(H s)
(H (up t (up x y) (down p-x p-y)))

((D H) s)
(down
 (((partial 0) H) (up t (up x y) (down p-x p-y)))
 (down (((partial 1 0) H) (up t (up x y) (down p-x p-y)))
        (((partial 1 1) H) (up t (up x y) (down p-x p-y))))
 (up (((partial 2 0) H) (up t (up x y) (down p-x p-y)))
      (((partial 2 1) H) (up t (up x y) (down p-x p-y))))))
```

Structured Results

Some functions produce structured outputs. A function whose output is a tuple is equivalent to a tuple of component functions each of which produces one component of the output tuple.

For example, a function that takes one numerical argument and produces a structure of outputs may be used to describe a curve through space. The following function describes a helical path around the \hat{z} -axis in 3-dimensional space:

$$h(t) = (\cos t, \sin t, t) = (\cos, \sin, I)(t). \quad (\text{B.32})$$

The derivative is just the up tuple of the derivatives of each component of the function:

$$Dh(t) = (-\sin t, \cos t, 1). \quad (\text{B.33})$$

In Scheme we can write

```
(define (helix t)
  (up (cos t) (sin t) t))
```

or just

```
(define helix (up cos sin identity))
```

Its derivative is just the up tuple of the derivatives of each component of the function:

```
((D helix) 't)
(up (* -1 (sin t)) (cos t) 1)
```

In general, a function that produces structured outputs is just treated as a structure of functions, one for each of the components. The derivative of a function of structured inputs that produces structured outputs is an object that when contracted with an incremental input structure produces a linear approximation to the incremental output. Thus, if we define function g by

$$g(x, y) = ((x + y)^2, (y - x)^3, e^{x+y}), \quad (\text{B.34})$$

then the derivative of g is

$$Dg(x, y) = \left[\begin{pmatrix} 2(x + y) \\ -3(y - x)^2 \\ e^{x+y} \end{pmatrix}, \begin{pmatrix} 2(x + y) \\ 3(y - x)^2 \\ e^{x+y} \end{pmatrix} \right]. \quad (\text{B.35})$$

In Scheme:

```
(define (g x y)
  (up (square (+ x y)) (cube (- y x)) (exp (+ x y)))))

((D g) 'x 'y)
(down (up (+ (* 2 x) (* 2 y))
           (+ (* -3 (expt x 2)) (* 6 x y) (* -3 (expt y 2)))
           (* (exp y) (exp x)))
      (up (+ (* 2 x) (* 2 y))
           (+ (* 3 (expt x 2)) (* -6 x y) (* 3 (expt y 2)))
           (* (exp y) (exp x))))
```

Exercise B.1: Chain Rule

Let $F(x, y) = x^2y^3$, $G(x, y) = (F(x, y), y)$, and $H(x, y) = F(F(x, y), y)$, so that $H = F \circ G$.

- a. Compute $\partial_0 F(x, y)$ and $\partial_1 F(x, y)$.
- b. Compute $\partial_0 F(F(x, y), y)$ and $\partial_1 F(F(x, y), y)$.
- c. Compute $\partial_0 G(x, y)$ and $\partial_1 G(x, y)$.
- d. Compute $DF(a, b)$, $DG(3, 5)$ and $DH(3a^2, 5b^3)$.

Exercise B.2: Computing Derivatives

We can represent functions of multiple arguments as procedures in several ways, depending upon how we wish to use them. The simplest idea is to identify the procedure arguments with the function's arguments.

For example, we could write implementations of the functions that occur in exercise B.1 as follows:

```
(define (f x y)
  (* (square x) (cube y)))
```

```
(define (g x y)
  (up (f x y) y))

(define (h x y)
  (f (f x y) y))
```

With this choice it is awkward to compose a function that takes multiple arguments, such as *f*, with a function that produces a tuple of those arguments, such as *g*. Alternatively, we can represent the function arguments as slots of a tuple data structure, and then composition with a function that produces such a data structure is easy. However, this choice requires the procedures to build and take apart structures.

For example, we may define procedures that implement the functions above as follows:

```
(define (f v)
  (let ((x (ref v 0))
        (y (ref v 1)))
    (* (square x) (cube y)))

(define (g v)
  (let ((x (ref v 0))
        (y (ref v 1)))
    (up (f v) y)))

(define h (compose f g))
```

Repeat exercise B.1 using the computer. Explore both implementations of multiple-argument functions.

C

Tensors

There are a variety of objects that have meaning independent of any particular basis. Examples are form fields, vector fields, covariant derivative, and so on. We call objects that are independent of basis *geometric objects*. Some of these are functions that take other geometric objects, such as vector fields and form fields, as arguments and produce further geometric objects. We refer to such functions as *geometric functions*. We want the laws of physics to be independent of the coordinate systems. How we describe an experiment should not affect the result. If we use only geometric objects in our descriptions then this is automatic.

A geometric function of vector fields and form fields that is linear in each argument with functions as multipliers is called a *tensor*. For example, let T be a geometric function of a vector field and form field that gives a real-number result at the manifold point m . Then

$$T(fu + gv, \omega) = f T(u, \omega) + g T(v, \omega) \quad (C.1)$$

$$T(u, f\omega + g\theta) = f T(u, \omega) + g T(u, \theta), \quad (C.2)$$

where u and v are vector fields, ω and θ are form fields, and f and g are manifold functions. That a tensor is linear over functions and not just constants is important.

The multilinearity over functions implies that the components of the tensor transform in a particularly simple way as the basis is changed. The components of a real-valued geometric function of vector fields and form fields are obtained by evaluating the function on a set of basis vectors and their dual form basis. In our example,

$$T_j^i = T(e_j, \tilde{e}^i), \quad (C.3)$$

for basis vector fields e_j and dual form fields \tilde{e}^i . On the left, T_j^i is a function of place (manifold point); on the right, T is a function of a vector field and a form field that returns a function of place.

Now we consider a change of basis, $e(f) = e'(f) J$ or

$$e_i(f) = \sum_j e'_j(f) J^j_i, \quad (\text{C.4})$$

where J typically depends on place. The corresponding dual basis transforms as

$$\tilde{e}^i(v) = \sum_j K^i_j \tilde{e}'^j(v), \quad (\text{C.5})$$

where $K = J^{-1}$ or $\sum_j K^i_j(m) J^j_k(m) = \delta^i_k$.

Because the tensor is multilinear over functions, we can deduce that the tensor components in the two bases are related by, in our example,

$$T^i_j = \sum_{kl} K^i_k T^k_l J^l_j, \quad (\text{C.6})$$

or

$$T^i_j = \sum_{kl} J^i_k T^k_l K^l_j. \quad (\text{C.7})$$

Tensors are a restricted set of mathematical objects that are geometric, so if we restrict our descriptions to tensor expressions they are *prima facie* independent of the coordinates used to represent them. So if we can represent the physical laws in terms of tensors we have built in the coordinate-system independence.

Let's test whether the geometric function R , which we have called the Riemann tensor (see equation 8.2), is indeed a tensor field. A real-valued geometric function is a tensor if it is linear (over the functions) in each of its arguments. We can try it for 3-dimensional rectangular coordinates:

```
(let ((cs R3-rect))
  (let ((u (literal-vector-field 'u-coord cs))
        (v (literal-vector-field 'v-coord cs))
        (w (literal-vector-field 'w-coord cs))
        (x (literal-vector-field 'x-coord cs))
        (omega (literal-1form-field 'omega-coord cs))
        (nu (literal-1form-field 'nu-coord cs))
        (f (literal-manifold-function 'f-coord cs))
        (g (literal-manifold-function 'g-coord cs))
        (nabla (covariant-derivative (literal-Cartan 'G cs)))
        (m (typical-point cs)))
    (let ((F (Riemann nabla)))
      ((up (- (F (+ (* f omega) (* g nu)) u v w)
                (+ (* f (F omega u v w)) (* g (F nu u v w))))
            (- (F omega (+ (* f u) (* g x))) v w)
            (+ (* f (F omega u v w)) (* g (F omega x v w))))
            (- (F omega v (+ (* f u) (* g x))) w)
            (+ (* f (F omega v u w)) (* g (F omega v x w))))
            (- (F omega v w (+ (* f u) (* g x))) )
            (+ (* f (F omega v w u)) (* g (F omega v w x))))))
        m))))
  (up 0 0 0 0))
```

Now that we are convinced that the Riemann tensor is indeed a tensor, we know how its components change under a change of basis. Let

$$R^i_{jkl} = R(\tilde{e}^i, e_j, e_k, e_l), \quad (C.8)$$

then

$$R^i_{jkl} = \sum_{mnpq} K_m^i R'_{npq} J_j^n J_k^p J_l^q, \quad (C.9)$$

or

$$R^i_{jkl} = \sum_{mnpq} J_m^i R_{npq} K_j^n K_k^p K_l^q. \quad (C.10)$$

Whew!

It is easy to generalize these formulas to tensors with general arguments. We have formulated the general tensor test as a program **tensor-test** that takes the procedure **T** to be tested, a list of argument types, and a coordinate system to be used. It tests each argument for linearity (over functions). If the function passed as **T** is a tensor, the result will be a list of zeros.

So, for example, Riemann proves to be a tensor

```
(tensor-test
  (Riemann (covariant-derivative (literal-Cartan 'G R3-rect)))
  '(1form vector vector)
  R3-rect)
(0 0 0 0)
```

and so does the torsion (see equation 8.21):

```
(tensor-test
  (torsion (covariant-derivative (literal-Cartan 'G R3-rect)))
  '(1form vector vector)
  R3-rect)
(up 0 0 0)
```

But not all geometric functions are tensors. The covariant derivative is an interesting and important case. The function F , defined by

$$F(\omega, u, v) = \omega(\nabla_u v), \quad (\text{C.11})$$

is a geometric object, since the result is independent of the coordinate system used to represent the ∇ . For example:

```
(define ((F nabla) omega u v)
  (omega ((nabla u) v)))

((( - (F (covariant-derivative
            (Christoffel->Cartan
              (metric->Christoffel-2
                (coordinate-system->metric S2-spherical)
                (coordinate-system->basis S2-spherical))))))
  (F (covariant-derivative
        (Christoffel->Cartan
          (metric->Christoffel-2
            (coordinate-system->metric S2-stereographic)
            (coordinate-system->basis S2-stereographic))))))
  (literal-1form-field 'omega S2-spherical)
  (literal-vector-field 'u S2-spherical)
  (literal-vector-field 'v S2-spherical))
  ((point S2-spherical) (up 'theta 'phi)))
  0
```

But it is not a tensor field:

```
(tensor-test
  (F (covariant-derivative (literal-Cartan 'G R3-rect)))
  '(1form vector vector)
  R3-rect)
(0 0 MESS)
```

This result tells us that the function F is linear in its first two arguments but not in its third argument.

That the covariant derivative is not linear over functions in the second vector argument is easy to understand. The first vector argument takes derivatives of the coefficients of the second vector argument, so multiplying these coefficients by a manifold function changes the derivative.

References

- [1] Harold Abelson and Gerald Jay Sussman with Julie Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, MA, 1996.
- [2] Harold Abelson and Andrea deSessa, *Turtle Geometry*, MIT Press, Cambridge, MA, 1980.
- [3] R. L. Bishop and S. I. Goldberg, *Tensor Analysis on Manifolds*, MacMillan, New York, 1968.
- [4] S. Carroll, *Spacetime and Geometry: An Introduction to General Relativity*, Benjamin Cummings, 2003.
- [5] Alonzo Church, *The Calculi of Lambda-Conversion*, Princeton University Press, 1941.
- [6] Harley Flanders, *Differential Forms with Applications to the Physical Sciences*, Academic Press, New York, 1963, Dover, New York, 1989.
- [7] Theodore Frankel, *The Geometry of Physics*, Cambridge University Press, 1997.
- [8] Galileo Galilei, *Il Saggiatore (The Assayer)*, 1623.
- [9] S. W. Hawking and G. F. R. Ellis, *The Large Scale Structure of Space-Time*, Cambridge University Press, 1973.
- [10] IEEE Std 1178-1990, *IEEE Standard for the Scheme Programming Language*, Institute of Electrical and Electronic Engineers, Inc., 1991.
- [11] Charles W. Misner, Kip S. Thorne, and John Archibald Wheeler, *Gravitation*, W. H. Freeman and Company, San Francisco, 1973.
- [12] Abraham Pais, *Subtle is the Lord: The Science and the Life of Albert Einstein*, Oxford University Press, Oxford, UK, 1982.
- [13] Seymour A. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, 1980.

- [14] B. Schutz *A First Course in General Relativity*, Cambridge University Press, 1985.
- [15] I. M. Singer and John A. Thorpe, *Lecture Notes on Elementary Topology and Geometry*, Scott, Foresman and Company, Glenview, Illinois, 1967.
- [16] Michael Spivak, *A Comprehensive Introduction to Differential Geometry*, Publish or Perish, Houston, Texas, 1970.
- [17] Michael Spivak, *Calculus on Manifolds*, W. A. Benjamin, New York, NY, 1965.
- [18] Gerald Jay Sussman and Jack Wisdom, *The Role of Programming in the Formulation of Ideas*, Artificial Intelligence Laboratory memo AIM-2002-018, November 2002.
- [19] Gerald Jay Sussman and Jack Wisdom with Meinhard E. Mayer, *Structure and Interpretation of Classical Mechanics*, MIT Press, Cambridge, MA, 2001.
- [20] Robert M. Wald, *General Relativity*, University of Chicago Press, 1984.
- [21] Free software is available at:
groups.csail.mit.edu/mac/users/gjs/6946/linux-install.htm.

Index

Any inaccuracies in this index may be explained by the fact that it has been prepared with the help of a computer.

Donald E. Knuth, *Fundamental Algorithms*
(Volume 1 of *The Art of Computer Programming*)

Page numbers for Scheme procedure definitions are in italics.

Page numbers followed by n indicate footnotes.

- (composition), 196
- ▷, 133
- ¶, 134
- D (derivative $\mathbf{R}^n \rightarrow \mathbf{R}^m$), 202
- ∂ (partial derivative), 206
- df (differential of manifold function), 33, 35
- $d\omega$ (exterior derivative of form field), 63
- δ_j^i , 34 n
- ∇ (nabla), 8
- $\gamma_m^v(t)$, 29
- I_j (selector), 199
- \mathcal{L}_v (Lie derivative), 85
- λ -calculus, 195
- λ -expression, 186–187
- λ -notation, 186 n
- $\phi_t^v(m)$, 29
- ϖ_j^i , 96
- ' (quote in Scheme), 192
- , in tuple, 206
- ; in tuple, 206
- # in Scheme, 192
- [] for down tuples, 198
- () for up tuples, 198
- () in Scheme, 185, 186 n , 191
- `4tuple->ct`, 176
- `4tuple->space`, 176
- Abuse of notation, 17 n
- Access chain, 199
- Alternative in conditional, 189
- Ampere, 167
- Angular momentum, $SO(3)$ and, 53 (ex. 4.3)
- Arguments, in Scheme, 185
- Arithmetic
 - generic, 195
 - on functions, 196
 - on operators, 203
 - on symbolic values, 197
 - on tuples, 195, 199–202
- Associativity and
 - non-associativity of tuple multiplication, 201, 202
- Atlas, on manifold, 11
- Basis fields, 41–53
 - change of basis, 44–47
 - dual forms, 41
 - general vector as linear combination, 41
- Jacobian, 45
- linear independence, 43
- over a map, 74
- rotation basis, 47–48
- `basis->1form-basis`, 45
- `basis->vector-basis`, 45
- Bianchi identity, 129–131
 - first, 130
 - second, 130
- Boost, 172
 - general, 174
 - perpendicular, 174

- Boost (*continued*)
 transformation under rotation,
 177
 Brackets for down tuples, 198
- car**, 191
 Cardioid, 18 (ex. 2.1)
 Cartan one-forms, 96
 Christoffel coefficients and, 98
 expression in terms of covariant derivative, 98
 linearity over functions, 97
 parallel transport and, 96
 transformation rule, 101
 Cartan's formula for Lie derivative, 92
Cartan-transform, 101
cdr, 191
 Chain rule
 for derivatives, 203, 208 (ex. B.1)
 for partial derivatives, 205, 208 (ex. B.1)
 for vector fields, 26
chart, 7, 14
 Chart, on manifold, 11
 Christoffel coefficients, 3, 8, 98
 first kind, 135
 from metric, 135
 Lagrange equations and, 3, 138
 metric and, 9
 second kind, 136
Christoffel->Cartan, 9, 102 *n*
 Church, Alonzo, 186 *n*
circular, 31
 Circular orbit in Schwarzschild spacetime, 148 (ex. 9.7)
 stability of, 148 (ex. 9.8)
 Closed form field, 64
 Coefficient functions
 one-form field, 34
 vector field, 23
 Combinatory logic, 77 *n*
 Comma in tuple, 206
 Commutativity of some tuple multiplication, 201
 Commutator, 48
 for rotation basis, 50–51
- meaning, 52
 non-coordinate basis, 53 (ex. 4.2)
 zero for coordinate basis fields, 48
- component**, 200
 Components of the velocity, 76
components->1form-field, 35
components->vector-field, 24
compose, 188
 Composition
 of functions, 196, 209 (ex. B.2)
 of linear transformations, 202
 of operators, 203
 Compound data in Scheme, 190–192
cond, 188
 Conditionals in Scheme, 188–189
 Confusion: differential of map and of manifold function, 73
 Connection
 Cartan one-forms, 96
 Christoffel coefficients and, 98
 from Lagrange equations, 137
 from metric, 135
cons, 191
 Consequent in conditional, 188
 Constraint, 3, 112
 Constructors in Scheme, 190
contract, 135
 Contraction of tuples, 200
 Contravariant, 29
 Coordinate basis
 one-form field, 34
 one-form field traditional notation, 36
 vector field, 26–29
 vector field traditional notation, 27
 Coordinate component functions
 one-form field, 34
 vector field, 23
 Coordinate function, 11–14
 Coordinate independence, 38
 integration, 55
 manifold functions, 15
 one-form field, 38
 vector field, 22, 28

- Coordinate patch, on manifold, 11
Coordinate representation
manifold functions, 14
one-form fields, 35
vector fields, 25
Coordinate transformations
one-form field, 38
vector field, 28
coordinate-system, 13
coordinate-system-at, 13 *n*
coordinate-system->basis, 9, 46
coordinatize, 25
Cosmology, 149 (ex. 9.9), 150 (ex. 9.10)
Coulomb, 167
Covariant, 28
Covariant derivative, 8, 93–104
change of basis, 100
directional derivative and, 93
generalized product rule, 99
geodesic, 8
Lie derivative and, 113 (ex. 7.2)
nabla (∇) notation, 8
not a tensor, 214
of one-form field, 99
of vector field, 93
over a map, 106
parallel transport and, 93
product rule, 97
covariant-derivative, 9
covariant-derivative-form, 100
covariant-derivative-vector, 97
curl, 154
curl, 155
Curry, Haskell, 77 *n*
Currying, 77
Curvature, 115–131
by explicit transport, 116
intrinsic, 115
pseudosphere, 123 (ex. 8.2), 144 (ex. 9.4)
Riemann curvature operator and, 115
Schwarzschild spacetime, 147 (ex. 9.6)
spherical surface, 123 (ex. 8.1), 143 (ex. 9.3)
universe, 150 (ex. 9.10)
d ω (exterior derivative of form field), 63
D, D (derivative $\mathbf{R}^n \rightarrow \mathbf{R}^m$), 202
define, 187
define-coordinates, 16, 27, 36
Definitions in Scheme, 187–188
Derivative, 202–207, *See also*
Covariant derivative;
Directional derivative;
Exterior derivative; Lie derivative;
Partial derivative;
Vector field
as best linear approximation, 21
as operator, 203
chain rule, 203, 208 (ex. B.1)
in Scheme programs: **D**, 202
notation: D , 202
of function of multiple arguments, 204–207
of function with structured inputs and outputs, 208
precedence of, 202
Determinant, 57 *n*, 61
df (differential of manifold function), 33, 35
Difference between points on manifold, 21
Differential
in a coordinate basis, 35
of manifold function (**df**), 33
of map, 72
pullback and, 77
differential, 10, 72
Differential equation, integral curve of, 29
Dimension of manifold, 12
Directional derivative, 83–114
all agree on functions, 83
covariant derivative, 93
extended Leibniz rule, 85

- Directional derivative (*continued*)
 formulation of method of transport, 83
 general properties, 84
 Leibniz rule, 84
 Lie derivative, 85
 of form field, 83
 of manifold function, 22
 of vector field, 22, 83
 using ordinary derivative, 84
div, 154
divergence, 156
 Divergence Theorem, 67
down, 199
 Down tuple, 198
drop2, 147 *n*
 Dual basis, 42
 over a map, 74
 Dual forms used to determine vector field coefficients, 43
 Duality, 41
 Hodge, 153
 illustrated, 36
 one-form field, 34
 Dust stress-energy tensor, 147
- Einstein**
 field equations, 145
 special relativity, 167
 tensor, 145 *n*
Einstein, 149 (ex. 9.9)
Einstein-field-equation, 149 (ex. 9.9)
 Electrodynamics, 160–170
else, 188
 Empty list, 191
eq?, 193
 Euler angles, 47
 alternate angles, 52 (ex. 4.1)
 Euler-Lagrange equations, 2
 Event, 179
 Evolution
 Hamiltonian, 113 (ex. 7.1)
 integral curve, 30
 Evolution operator, 31
 Exact form field, 64
 Exact forms are closed, 64
- Exponentiating Lie derivatives, 91
 Expressions in Scheme, 185
 Extended rotation, 177
 Exterior derivative, 33 *n*, 62–65
 Cartan's formula and, 92
 commutes with Lie derivative, 90
 commutes with pullback, 80
 coordinate-system independent, 62
 general definition, 62
 graded formula, 69 (ex. 5.2)
 iterated, 69 (ex. 5.3)
 obeys graded Leibniz rule, 64
 of one-form, 62
 Stokes's Theorem and, 62
- F->C**, 4
 Faraday, 167
Faraday, 160
 Faraday tensor, 160
 Field equations
 Einstein, 145
 Maxwell, 162
FLRW-metric, 150 (ex. 9.9)
 Force, Lorentz, 164
 relativistic, 166 (ex. 10.1)
 Form field
 closed, 64
 exact, 64
 pushforward, 81
form-field->form-field-over-map, 74
 Formal parameters of a procedure, 187
 Franklin, 167
 Friedmann metric, 150 (ex. 9.9)
 Friedmann-Lemaître-Robertson-Walker, 149 (ex. 9.9)
 Function(s), 196–197
 arithmetic operations on, 196
 composition of, 196, 209 (ex. B.2)
 operators vs., 203
 selector, 199
 tuple of, 207

- vs. value when applied, 195, 196
with multiple arguments, 204,
205, 208 (ex. B.2)
with structured arguments, 205,
208 (ex. B.2)
with structured output, 207,
208 (ex. B.2)
- Functional mathematical
notation, 195
- Fundamental Theorem of
Calculus, 66
- g-Minkowski**, 159
- Galileo Galilei, 1
- Gauss, 167
- General relativity, 144–151, 172
- general-boost**, 176
- general-boost2**, 177
- Generic arithmetic, 195
- Geodesic deviation, 125–129
relative acceleration of
neighboring geodesics and, 125
- Riemann curvature and, 126
vector field, 126
- Geodesic motion, 2, 111
governing equation, 8, 111
governing equation, in
coordinates, 9, 111
- Lagrange equations and, 2, 112
 $SO(3)$ on, 143 (ex. 9.2)
- Geometric function, 211
- Geometric object, 211
- Global Lorentz frame, 172
- grad, 154
- gradient**, 154
- Green's Theorem, 67
- Hamiltonian Evolution, 113 (ex.
7.1)
- Higher-rank forms, 57
linear and antisymmetric, 57
- Hill climbing, power expended,
39 (ex. 3.3)
- Hodge dual, 153
- Hodge star, 153–158
- Honest definition rare, 79
- I_j (selector), 199
- if**, 188
- Inner product of tuples, 201
- Integral
coordinate-independent
definition, 58
higher dimensions, 57–58
higher-rank forms, 57
one-form, 56
- Integral curve, 29–32
differential equation, 29
evolution, 30
Taylor series, 30
- Integration, 55–69
coordinate-independent notion
of, 55
- Interior product, 92
- Iteration in Scheme, 190
- Jacobian, 28
basis fields, 45
- Knuth, Donald E., 219
- Kronecker delta, 34 *n*
- \mathcal{L}_v (Lie derivative), 85
- Lagrange equations, 2 *n*
Christoffel coefficients and, 138
geodesic equations and, 137
- Lagrange-explicit**, 138 *n*
- Lagrangian, 2
constraint, 3
free particle, 3, 137
metric and, 137
- lambda**, 186
- Lambda calculus, 195
- Lambda expression, 186–187
- Laplacian, 156
wave equation and, 160
- Laplacian**, 156
- Leibniz rule (product rule) for
vector field, 26
- Lemniscate of Bernoulli, 18 (ex.
2.1)
- let**, 189
- Lie derivative, 85–93
alternate formulation, 88
as commutator, 86
commutes with exterior
derivative, 90

- Lie derivative (*continued*)
 covariant derivative and, 113
 (ex. 7.2)
 directional derivative and, 85
 exponentiation, 91
 interpretation, 87
 of form field, 89
 of function, 85
 of vector field, 85
 properties, 89
 transport operator is
 pushforward, 85
 uniform interpretation, 89
- Lie-derivative-vector**, 87
- Linear independence and basis fields, 43
- Linear transformations as tuples, 201
- Linearity
 one-form field, 33
 vector field, 26
- Lisp, 191 *n*
- list**, 191
- list-ref**, 191
- Lists in Scheme, 190–192
- Literal symbol in Scheme, 192–193
- literal-1form-field**, 35
- literal-Christoffel-2**, 119 *n*
- literal-function**, 16, 197, 206
- literal-manifold-function**, 16 *n*
- literal-manifold-map**, 7 *n*
- literal-metric**, 6 *n*
- literal-vector-field**, 24
- Local names in Scheme, 189–190
- Loops in Scheme, 190
- Lorentz, 169
- Lorentz decomposition, 179 (ex. 11.1)
- Lorentz force, 164
 relativistic, 166 (ex. 10.1)
- Lorentz frame, global, 172
- Lorentz interval, 172
- Lorentz transformation, 172
 general, 178
 simple, 173
 unique decomposition, 178
- lower**, 134
- Lowering a vector field, 133
- make-4tuple**, 175
- make-fake-vector-field**, 74
- make-manifold**, 12
- make-SR-coordinates**, 179 *n*
- make-SR-frame**, 180
- Manifold, 11–19
 atlas, 11
 chart, 11
 coordinate function, 11–14
 coordinate patch, 11
 coordinate-independence of manifold functions, 15
 difference between points, 21
 dimension of, 12
 motion and paths, 71
 naming coordinate functions, 16
- Manifold function, 14–17
 coordinate representation, 14
 directional derivative, 22
- Matrix as tuple, 201
- Maxwell, 167
- Maxwell**, 161
- Maxwell tensor, 161
- Maxwell's equations, 162
- Metric, 5, 133–151
 Christoffel coefficients and, 9, 135
 connection compatible with, 135
 Friedmann, 150 (ex. 9.9)
 Lagrangian and, 137
 Minkowski, 133, 159
- metric->Christoffel-2**, 9
- metric->Lagrangian**, 138
- Minkowski metric, 133, 159
- Motion
 geodesic in General Relativity, 144
 on a sphere, 10 (ex. 1.1)
 on manifolds, 71
 Schwarzschild spacetime, 148
 (ex. 9.7), 148 (ex. 9.8)
- Multiplication of operators as composition, 203
- Multiplication of tuples, 200–202
 as composition, 202
 as contraction, 200

- Nabla (∇), 8
Newton-connection, 146
Newton-metric, 146
Newton's equations, 32 (ex. 3.1)
Non-associativity and
 associativity of tuple multiplication, 201, 202
Non-commutativity
 of some partial derivatives, 206
 of some tuple multiplication, 202
Non-coordinate basis, 41, 47
 commutator, 53 (ex. 4.2)
Notation, 195–209
 ∇ , 8
 () for up tuples, 198
 [] for down tuples, 198
 abuse of, 17 *n*
 ambiguities of traditional, 195
 derivative, partial: ∂ , 206
 derivative: D , 202
 functional, 195
 selector function: I_j , 199
Notation for coordinate basis
 one-form field, 36
 vector field, 27
- Oersted, 167
- One-form field, 32–39
 coefficient functions, 34
 coordinate basis, 34
 coordinate independence, 38
 coordinate representation, 35
 coordinate transformations, 38
 differential, 33
 duality, 34
 general, 34
 linearity, 33
 not all are differentials, 37
 over a map, 71, 73
 raising, 134
- Operator, 203
 arithmetic operations on, 203
 function vs., 203
- Oriented area of a parallelogram, 59
- Over a map, 71–81
 basis fields, 74
- covariant derivative, 106
dual basis, 74
one-form field, 73
vector field, 71
- pair?**, 192
- Pairs in Scheme, 191
- Parallel transport, 3, 93, 104–112
 Cartan one-forms and, 96
 equations similar to variational equations, 108
 for arbitrary paths, 104
 geodesic motion and, 111
 governing equations, 106
 independent of rate of transport, 94
 numerical integration, 109
 on a sphere, 106
 path-dependent, 93
- Parentheses
 in Scheme, 185, 186 *n*
 for up tuples, 198
- Partial derivative, 204–206
 chain rule, 205, 208 (ex. B.1)
 commutativity of, 37
 notation: ∂ , 206
- patch**, 12
- Patch, on manifold, 11
- Paths and manifolds, 71
- Perfect-fluid stress-energy tensor, 150 (ex. 9.9)
- Phase-space state function, 205
 in Scheme, 206
- Poincaré transformation, 172
- point**, 7, 14
- Power expended in hill climbing, 39 (ex. 3.3)
- Predicate in conditional, 188
- print-expression**, 31, 197 *n*
- Procedure calls, 185–186
- procedure->vector-field**, 24
- Product rule (Leibniz rule) for vector field, 26
- Projection, stereographic, 19 (ex. 2.2)
- Proper length, 159
- proper-space-interval**, 176
- Proper time, 159

- proper-time-interval**, 176
- Pullback**
- commutes with exterior derivative, 80
 - differential and, 77
 - of form field, 79
 - of function, 76
 - of vector field, 79
 - properties, 80
 - vector field over a map as, 77
- pullback-form**, 80
- pullback-function**, 77
- pullback-vector-field**, 79
- Pushforward**
- along integral curves, 78
 - of form field, 81
 - of function, 76
 - of vector field, 78
- pushforward-vector**, 78
- Quotation in Scheme, 192–193
- R_x, R_z (rotations), 47
- R2-polar**, 7 n
- R2-polar-Cartan**, 102
- R2->R**, 16
- R2-rect**, 7 n, 13 n
- R2-rect-Cartan**, 102
- R2-rect-Christoffel**, 102
- R2-rect-point**, 16
- R3-cyl**, 18 (ex. 2.1)
- raise**, 135
- Raising a one-form field, 134
- Recursive procedures, 189
- ref**, 200
- Reparameterization, 141
- Residual, xvi
- Restricted vector field, 71
- Ricci**, 123
- Ricci scalar, 144 (ex. 9.3)
- Ricci tensor, 123
- Riemann**, 116
- Riemann curvature
- in terms of Cartan one-forms, 120
 - way to compute, 120
- Riemann-curvature**, 115
- Riemann curvature operator, 115
- Riemann tensor, 115
- by explicit transport, 116
 - for sphere, 116
 - is a tensor, 212
- Robertson-Walker equations, 150 (ex. 9.9)
- Rotation**
- extended to Lorentz transformation, 177
 - as tuples, 201
- S2-basis**, 75
- S2-Christoffel**, 107
- S2-spherical**, 75
- s:map/r**, 45
- Salam, Abdus, 195
- Scheme, 185–193, 195
- lists, 191
 - quotation, 192–193
 - vectors, 192
- Schönfinkel, Moses, 77 n
- Schwarzschild-metric**, 148 (ex. 9.6)
- Schwarzschild spacetime
- circular orbit, 148 (ex. 9.7)
 - circular orbit stability, 148 (ex. 9.8)
 - curvature, 147 (ex. 9.6)
- Scmutils, 195–209
- generic arithmetic, 195
 - simplification of expressions, 197
- Selector functions, 190, 199
- Semicolon in tuple, 206
- series:for-each**, 31
- Simplification of expressions, 197
- $SO(3)$, 47
- angular momentum and, 53 (ex. 4.3)
 - geodesics, 143 (ex. 9.2)
- SO3-metric**, 143 (ex. 9.2)
- Spacelike, 159
- Spacetime, 144
- Special orthogonal group— $SO(3)$, 47
- Special relativity, 167–184
- electrodynamics, 160–170
 - frame, 179

- Minkowski metric, 159
twin paradox, 181
velocity addition, 180
sphere-Cartan, 107
sphere->R3, 4
spherical-metric, 143 (ex. 9.3)
Spherical surface, curvature, 143
(ex. 9.3)
Spivak, Michael, 195
on notation, 28 *n*
square, 187
for tuples, 187 *n*
Stability of circular orbits in
Schwarzschild spacetime, 148
(ex. 9.8)
State derivative, 32 (ex. 3.1)
Stereographic projection, 18 (ex.
2.2)
Stokes's Theorem, 65–66
proof, 65
states, 65
Stress-energy tensor, 145
dust, 147
perfect fluid, 150 (ex. 9.9)
Structure constants, 50, 125
Subscripts
for down-tuple components, 198
for selectors, 199
Superscripts
for up-tuple components, 12,
198
Symbolic values, 197–198
Symbols in Scheme, 192–193
Syntactic sugar, 187

Taylor series of integral curve, 30
Tdust, 147
Tensor, 211–215
components, 211
linear over functions, 211
stress-energy, 145
transformation with change of
basis, 211
Tensor arithmetic vs. tuple
arithmetic, 195, 199
tensor-test, 213
Timelike, 159

Torsion, 124–125
components in a non-coordinate
basis, 125
is a tensor, 214
torsion, 124
Tperfect-fluid, 150 (ex. 9.9)
trace2down, 144 (ex. 9.3)
Traditional notation
coordinate-basis one-form field,
36
coordinate-basis vector field, 27
Tuples, 198–202
arithmetic on, 195, 199–202
commas and semicolons in, 206
component selector: I_j , 199
composition and, 202
contraction, 200
down and up, 198
of functions, 207
inner product, 201
linear transformations as, 201
matrices as, 201
multiplication of, 200–202
rotations as, 201
squaring, 187 *n*, 200
up and down, 198
Twin paradox, 181–184

up, 199
Up tuple, 12, 198

Vector, in Scheme, 190–192
vector, 192
vector?, 192
vector-ref, 192
vector-basis->dual, 44
Vector calculus, 154–158
Vector field, 21–32
as linear combination of partial
derivatives, 23
chain rule, 26
coefficient functions, 23
commutator of, 48
coordinate basis, 26–29
coordinate independence, 22, 28
coordinate representation, 25
coordinate transformations, 28

Vector field (*continued*)
 directional derivative, 22
 Leibniz rule, 26
 linearity, 26
 lowering, 133
 module, 25
 operator, 24
 over a map, 71
 product rule, 26
 properties, 25–26
 restricted, 71
 traditional notation for
 coordinate-basis vector fields,
 27
Vector field over a map, 71–73
 as pullback, 77
vector-field->vector-field-
 over-map, 72
Vector integral theorems, 67–69
 Divergence Theorem, 67
 Green’s Theorem, 67
Vector space of tuples, 200
Velocity
 addition of in special relativity,
 180
 at a time, 73
 components, 76
 differential and, 73
 on a globe, 81 (ex. 6.1)
Volume, 59

Walking on a sphere, 75
Wave equation, 159
 Laplacian and, 160
Wedge product, 58
 antisymmetry, 61
 associativity of, 59
 construction of antisymmetric
 higher-rank forms, 58
 determinant and, 61
 oriented area of a
 parallelogram, 59

Zero-based indexing, 191, 198,
 200