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

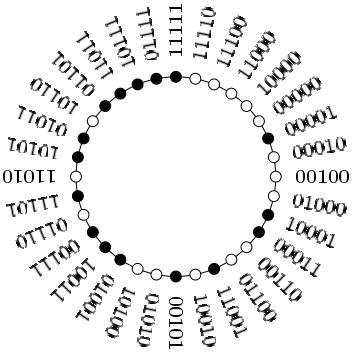


**Fig. 15.**
A de Bruijn cycle for 5-bit numbers.

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

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

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

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

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

$$00000100011001010011101011011111; \tag{53}$$

and the first $m^2$ digits of the infinite sequence

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\begin{aligned} y &= 00102122001111021220010212 2\,001\ldots \\ t &= 00100100001234001000010010 0\,001\ldots \\ x &= 00011010222012002021112212 1\,0001\ldots \end{aligned}$$

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

**Algorithm D** (*Doubly recursive de Bruijn cycle generation*). Suppose $f()$ and $g()$ 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

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

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

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

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

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

**D4.** [Invoke $g$.] 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: 12121111212121111212121111212121111212121111212121 ...**
**$x'$ in step D7: 0 11021220 11021220 11021220 11021220 11021220 1 ...**;

so the 81-cycle produced in steps D3 and **D7** is **0**0001**0**11**0**12...2222 **0**0001....

The case $m = 2$ of Algorithm R was discovered by Abraham Lempel [*IEEE Trans.* **C-19** (1970), 1204–1209]; Algorithm D was not discovered until more than 25 years later [C. J. Mitchell, T. Etzion, and K. G. Paterson, *IEEE Trans.* **IT-42** (1996), 1472–1478]. By using them together, starting with simple coroutines for $n = 2$ based on (54), we can build up an interesting family of cooperating coroutines that will generate a de Bruijn cycle of length $m^n$ for any desired $m \geq 2$ and $n \geq 2$, using only $O(\log n)$ simple computations for each digit of output. (See exercise 96.) Furthermore, in the simplest case $m = 2$, this combination "R&D method" has the property that its $k$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 $\beta$, the position of $\beta$ 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 $\alpha$ is a *prefix* of $\gamma$ and $\beta$ is a *suffix*. A prefix or suffix of $\gamma$ is called *proper* if its length is positive but less than the length of $\gamma$. Thus $\beta$ 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 $\alpha$ is less than $\beta$ when $\alpha$ is lexicographically less than $\beta$. In particular, we always have $\alpha \leq \alpha\beta$, and $\alpha < \alpha\beta$ if and only if $\beta \neq \epsilon$.)

▸ **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.

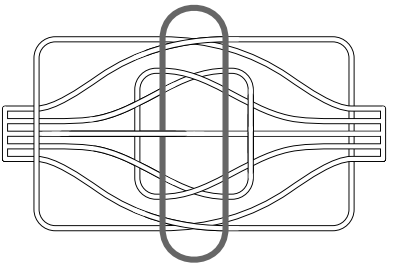

**Fig. 17.** The Loony Loop puzzle.

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

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

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

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

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

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

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

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

b) What string follows •— —•• — —• — • in your sequence for $n = 15$?

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

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

**92.** [*M30*] Is there an infinite sequence $\Phi_n$ 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 $a_0a_1a_2\ldots$ of period $p$ is interleaved with an infinite sequence $b_0b_1b_2\ldots$ of period $q$ to form the infinite cyclic sequence

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

a) Under what circumstances does $c_0c_1c_2\ldots$ have period $pq$? (The "period" of a sequence $a_0a_1a_2\ldots$, for the purposes of this exercise, is the smallest integer $p>0$ such that $a_k = a_{k+p}$ for all $k\ge 0$.)
b) Which $2n$-tuples would occur as consecutive outputs of Algorithm D if step D6 were changed to say simply "If $t'=n$ and $x'<r$, go to D4"?
c) Prove that Algorithm D outputs a de Bruijn cycle as advertised.

▸ **96.** [*M23*] Suppose a family of coroutines has been set up to generate a de Bruijn cycle of length $m^n$ using Algorithms R and D, based recursively on simple coroutines for the base case $n=2$.
a) How many coroutines of each type will there be?
b) What is the maximum number of coroutine activations needed to get one top-level digit of output?

**97.** [*M29*] The purpose of this exercise is to analyze the de Bruijn cycles constructed by Algorithms R and D in the important special case $m=2$. Let $f_n(k)$ be the $(k+1)$st bit of the $2^n$-cycle, so that $f_n(k)=0$ for $0\le k<n$. Also let $j_n$ be the index such that $0\le j_n<2^n$ and $f_n(k)=1$ for $j_n\le k<j_n+n$.
a) Write out the cycles $(f_n(0)\ldots f_n(2^n-1))$ for $n=2$, 3, 4, and 5.
b) Prove that, for all even values of $n$, there is a number $\delta_n=\pm1$ such that we have

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

where the congruence is modulo 2. (In this formula $\Sigma f$ stands for the summation function $\Sigma f(k)=\sum_{j=0}^{k-1}f(j)$.) Hence $j_{n+1}=2^n-\delta_n$ when $n$ is even.
c) Let $(c_n(0)c_n(1)\ldots c_n(2^{2n}-5))$ be the cycle produced when the simplified version of Algorithm D in exercise 95(b) is applied to $f_n()$. Where do the $(2n-1)$-tuples $1^{2n-1}$ and $(01)^{n-1}0$ occur in this cycle?
d) Use the results of (c) to express $f_{2n}(k)$ in terms of $f_n()$.
e) Find a (somewhat) simple formula for $j_n$ as a function of $n$.

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

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

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

▸ **101.** [*M30*] (*Unique factorization of strings into nonincreasing primes.*)
a) Prove that if $\lambda$ and $\lambda'$ are prime, then $\lambda\lambda'$ is prime if $\lambda<\lambda'$.
b) Consequently every string $\alpha$ can be written in the form

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

c) In fact, only one such factorization is possible. *Hint:* Show that $\lambda_t$ must be the lexicographically smallest nonempty suffix of $\alpha$.
d) True or false: $\lambda_1$ is the longest prime prefix of $\alpha$.
e) What are the prime factors of 3141592653589793238462643383279502884197?

with $(\mathrm{—}\bullet)^{\lfloor n/3\rfloor}\bullet^{[n \bmod 3=1]}\mathrm{—}^{[n \bmod 3=2]}$. But $M_{3k+1}\bullet$, $M_{3k}^R\mathrm{—}$ is a Hamiltonian circuit in the Morse code graph when $n = 3k + 2$.

**91.** Equivalently, the $n$-tuples $a_1\bar{a}_2a_3\bar{a}_4\ldots$ have no two consecutive 1s. Such $n$-tuples correspond to Morse code sequences of length $n+1$, if we append 0 and then represent $\bullet$ and $\bullet\mathrm{—}$ 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 $1 \le j \le n$. 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 $1 \le j \le n$, except if $j+(j \bmod 3) > n$ set $r_j \leftarrow 0$ and $l_0 \leftarrow j$. (The "fringe" now contains 1, 2, 4, 5, 7, 8, ....)

**Q2.** [Visit.] Visit the $n$-tuple $(a_1, \ldots, a_n)$.

**Q3.** [Choose $p$.] Set $q \leftarrow l_0$, $p \leftarrow f_q$, $f_q \leftarrow q$.

**Q4.** [Check $a_p$.] Terminate the algorithm if $p = 0$. Otherwise set $a_p \leftarrow 1 - a_p$ and go to Q6 if $a_p + p$ is now even.

**Q5.** [Insert $p+1$.] If $p < n$, set $q \leftarrow r_p$, $l_q \leftarrow p+1$, $r_{p+1} \leftarrow q$, $r_p \leftarrow p+1$, $l_{p+1} \leftarrow p$. Go to Q7.

**Q6.** [Delete $p+1$.] If $p < n$, set $q \leftarrow r_{p+1}$, $r_p \leftarrow q$, $l_q \leftarrow p$.

**Q7.** [Make $p$ passive.] Set $f_p \leftarrow f_{l_p}$ and $f_{l_p} \leftarrow l_p$. Return to Q2. ∎

This algorithm can also be derived as a special case of a considerably more general method due to Gang Li, Frank Ruskey, and D. E. Knuth, which extends Algorithm K by allowing the user to specify either $a_p \ge a_q$ or $a_p \le a_q$ 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 $(x_1, \ldots, x_{n-1})$ with $x_1, \ldots, x_{n-1} \le m$ and with arcs $(x_1, \ldots, x_{n-1}) \to (x_2, \ldots, x_n)$ whenever $\max(x_1, \ldots, x_n) = m$ is connected and balanced; see Theorem 2.3.4.2G. Indeed, we get such a sequence from Algorithm F if we note that the final $k^n$ elements of the prime strings of length dividing $n$, when subtracted from $m-1$, are the same for all $m \ge k$. When $n = 4$, for example, the first 81 digits of the sequence $\Phi_4$ are $2 - \alpha^R = 0\,0001\,01\,0011\ldots$, where $\alpha$ 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 $m \ge 3$. 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 $\alpha$ has length $m^n - 1$ and ends with $1^{n-1}$. The cycle generated by Algorithm R is a cyclic permutation of $\gamma = c_0 \ldots c_{m^{n+1}-1}$, where $c_k = (c_0 + b_0 + \cdots + b_{k-1}) \bmod m$ and $b_0 \ldots b_{m^{n+1}-1} = \beta = \alpha^m 1^m$.

If $x_0 \ldots x_n$ occurs in $\gamma$, say $x_j = c_{k+j}$ for $0 \le j \le n$, then $y_j = b_{k+j}$ for $0 \le j < n$, where $y_j = (x_{j+1} - x_j) \bmod m$. [This is the connection with modular $m$-ary Gray code; see exercise 78.] Now if $y_0 \ldots y_{n-1} = 1^n$ we have $m^{n+1} - m - n < k \le m^{n+1} - n$; otherwise there is an index $k'$ such that $-n < k' < m^n - n$ and $y_0 \ldots y_{n-1}$ occurs in $\beta$ at positions $k = (k' + r(m^n - 1)) \bmod m^{n+1}$ for $0 \le r < m$. In both cases the $m$ choices of $k$ have different values of $x_0$, because the sum of all elements in $\alpha$ is $m-1$

(modulo $m$) when $n \geq 2$. [Algorithm R is valid also for $n = 1$ if $m \bmod 4 \neq 2$, because $m \perp \sum \alpha$ in that case.]

**94.** 0<u>01</u>02<u>0</u>3041<u>12</u>1<u>3</u>14<u>2</u>23243344. (The underlined digits are effectively inserted into the interleaving of 00112234 with 34. Algorithm D can be used in general when $n = 1$ and $r = m - 2 \geq 0$; but it is pointless to do so, in view of (54).)

**95.** (a) Let $c_0 c_1 c_2 \ldots$ have period $r$. If $r$ is odd we have $p = q = r$, so $r = pq$ only in the trivial case when $p = q = 1$ and $a_0 = b_0$. Otherwise $r/2 = \operatorname{lcm}(p, q) = pq/\gcd(p, q)$ by 4.5.2–(10), hence $\gcd(p, q) = 2$. In the latter case the $2n$-tuples $c_l c_{l+1} \ldots c_{l+2n-1}$ that occur are $a_j b_k \ldots a_{j+n-1} b_{k+n-1}$ for $0 \leq j < p$, $0 \leq k < q$, $j \equiv k$ (modulo 2), and $b_k a_j \ldots b_{k+n-1} a_{j+n-1}$ for $0 \leq j < p$, $0 \leq k < q$, $j \not\equiv k$ (modulo 2).

(b) The output would interleave two sequences $a_0 a_1 \ldots$ and $b_0 b_1 \ldots$ whose periods are respectively $m^n + r$ and $m^n - r$; the $a$'s are the cycle of $f()$ with $x^n$ changed to $x^{n+1}$ and the $b$'s are the cycle of $g()$ 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

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

$f_n^+(k) = f_n^+(k \bmod (2^n+2))$, $f_n^-(k) = f_n^-(k \bmod (2^n-2))$. 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

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

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

(e) Consequently $j_{2n} = k_n + 1 + 2[j_n \bmod 4 < 3]$. Indeed, the elements preceding $1^{2n}$ consist of $2^{n-2}-1$ complete periods of $f_n^+()$ interleaved with $2^{n-2}$ complete periods of $f_n^-()$, with one 0 inserted and also with 10 inserted if $l_n < k_n$, followed by $f_n(1)f_n(1)f_n(2)f_n(2)\ldots f_n(j_n-1)f_n(j_n-1)$. The sum of all these elements is odd, unless $l_n < k_n$; therefore $\delta_{2n} = 1 - 2[j_n \bmod 4 = 3]$.

Let $n = 2^t q$, where $q$ is odd and $n > 2$. The recurrences imply that, if $q = 1$, we have $j_n = 2^{n-1} + b_t$ where $b_t = 2^t/3 - (-1)^t/3$. And if $q > 1$ we have $j_n = 2^{n-1} \pm b_{t+2}$, where the $+$ sign is chosen if and only if $\lfloor \lg q \rfloor + [\lfloor 4q/2^{\lfloor \lg q \rfloor} \rfloor = 5]$ is even.

**98.** If $f(k) = g(k)$ when $k$ lies in a certain range, there's a constant $C$ such that $\Sigma f(k) = C + \Sigma g(k)$ for $k$ in that range. We can therefore continue almost mindlessly to derive additional recurrences: If $n > 1$ we have

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

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

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

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

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

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

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

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

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

And then, aha, there is closure:

$$\Sigma\Sigma c_n(2k) = \Sigma f_n^+(k), \qquad \Sigma\Sigma c_n(2k+1) = \Sigma f_n^-(k).$$

If $n = 2^t q$ where $q$ is odd, the running time to evaluate $f_n(k)$ by this system of recursive formulas is $O(t + S(q))$, where $S(1) = 1$, $S(2k) = 1 + 2S(k)$, and $S(2k+1) = 1 + S(k)$. Clearly $S(k) < 2k$, so the evaluations involve at most $O(n)$ simple operations on $n$-bit numbers. In fact, the method is often significantly faster: If we average $S(k)$ over all $k$ with $\lfloor \lg k \rfloor = s$ we get $(3^{s+1} - 2^{s+1})/2^s$, which is less than $3k^{\lg(3/2)} < 3k^{0.59}$. (Incidentally, if $k = 2^{s+1} - 1 - (2^{s-e_1} + 2^{s-e_2} + \cdots + 2^{s-e_t})$ we have $S(k) = s + 1 + e_t + 2e_{t-1} + 4e_{t-2} + \cdots + 2^t e_1$.)

**99.** A string that starts at position $k$ in $f_n()$ starts at position $k^+ = k + 1 + [k > j_n]$ in $f_n^+()$ and at position $k^- = k - 1 - [k > j_n]$ in $f_n^-()$, except that $0^n$ and $1^n$ occur twice in $f_n^+()$ but not at all in $f_n^-()$.

To find $\gamma = a_0 b_0 \ldots a_{n-1} b_{n-1}$ in the cycle $f_{2n}()$, let $\alpha = a_0 \ldots a_{n-1}$ and $\beta = b_0 \ldots b_{n-1}$. Suppose $\alpha$ starts at position $j$ and $\beta$ at position $k$ in $f_n()$, and assume that neither $\alpha$ nor $\beta$ 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 $\gamma$ 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 $\beta = b_0 b_1 \ldots b_n$ in $f_{n+1}()$ when $n$ is even, suppose that the $n$-bit string $(b_0 \oplus b_1) \ldots (b_{n-1} \oplus b_n)$ starts at position $j$ in $f_n()$. Then $\beta$ 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 $\beta$ be a proper suffix of $\lambda\lambda'$ with $\beta \leq \lambda\lambda'$. Either $\beta$ is a suffix of $\lambda'$, 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 $\gamma$ is a suffix of $\beta$ with $1 \leq |\gamma| = |\beta| - |\lambda| < |\lambda'|$; hence $\gamma$ is a proper suffix of $\lambda'$, 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 $\lambda$ be the smallest suffix of $\lambda_1 \ldots \lambda_t$. Then $\lambda$ is prime by definition, and it has the form $\beta\gamma$ where $\beta$ is a nonempty suffix of some $\lambda_j$. Therefore $\lambda_t \leq \lambda_j \leq \beta \leq \beta\gamma = \lambda \leq \lambda_t$, so we must have $\lambda = \lambda_t$. Remove $\lambda_t$ and repeat until $t = 0$.

(d) True. For if we had $\alpha = \lambda\beta$ for some prime $\lambda$ with $|\lambda| > |\lambda_1|$, we could append the factors of $\beta$ to obtain another factorization of $\alpha$.

(e) $3 \cdot 14159265358979323846264338327 95 \cdot 02884197$. (Knowing more digits of $\pi$ 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.)