THE ART OF COMPUTER PROGRAMMING

PRE-FASCICLE 4A

A DRAFT OF SECTION 7.2.1.6: GENERATING ALL TREES

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.

Another page, `http://www-cs-faculty.stanford.edu/~knuth/programs.html`, contains auxiliary programs by the author.

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

Zeroth printing (revision 13), 31 March 2005

PREFACE

Explain the significance of the following sequence:
un, dos, tres, quatre, cinc, sis, set, vuit, nou, deu, . . .
— RICHARD P. STANLEY, *Enumerative Combinatorics* (1999)

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

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

To put the material in context, this is Section 7.2.1.6 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," Section 7.2.1.2, "Generating all permutations," . . . , Section 7.2.1.5, "Generating all set partitions." (Readers of the present booklet should have already looked at those sections, drafts of which are available as Pre-Fascicles 2A, 2B, 3A, and 3B.) The stage is now set for the main contents of this booklet, Section 7.2.1.6: "Generating all trees." Then will come Section 7.2.1.7, about the history of combinatorial generation. 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 tree generation turns out to be surprisingly rich, with ties to Sections 1.2.3, 1.2.6, 1.2.9, 2.2.1, 2.3, 2.3.1, 2.3.2, 2.3.3, 2.3.4.1, 2.3.4.2, 2.3.4.4, 2.3.4.5, 2.3.4.6, 2.4, 4.6.1, 5.1.1, 5.1.3, 5.1.4, 5.2.1, 5.3.4, 6.2.1, 6.2.2, 6.2.3, and 6.4 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 124 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 17, 76, 89, 101, 102, and 109; I've also implicitly posed additional unsolved questions in the answers to exercises 34, 37, 46, 59, and 103. 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 13, 15, 25, 27(e), 28(e), 29, 31, 36, 37, 42, 47, 54, 55, 60(c), 72, 74, 75, 77, 78, 80, 82, 110, 112–119, 122, 123, and the remarks about $D' \to D''$ and D^* in answer 108. Has anybody published the concept of "prepostorder" or its equivalent?

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

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

Happy reading!

Stanford, California D. E. K.
09 July 2004

A note on notation. At the beginning of Chapter 7 I'll define some operations on graphs for which many different notations are presently rampant. My current plan is to say that, if G is a graph on the vertices $U = \{u_1, \ldots, u_m\}$ and if H is a graph on the vertices $V = \{v_1, \ldots, v_n\}$, then:

- $G + H$ is the sum, aka juxtaposition, of G and H: It has the $m + n$ vertices $U \cup V$ and the edges of G and H.
- $G \mathbin{\overline{+}} H$ is the cosum, aka join, of G and H, namely the complement of the juxtaposition of their complements. (Thus its edges are those of G and H, plus all $u_j \mathrel{-\!\!-} v_k$.)
- $G \times H$ is the Cartesian product of G and H: It has the mn vertices $U \times V$; its edges are $(u, v) \mathrel{-\!\!-} (u', v)$ when $u \mathrel{-\!\!-} u'$ in G, and $(u, v) \mathrel{-\!\!-} (u, v')$ when $v \mathrel{-\!\!-} v'$ in H.
- $G \circ H$ is the direct product, aka conjunction, of G and H: Again its vertices are $U \times V$, but its edges are $(u, v) \mathrel{-\!\!-} (u', v')$ if and only if $u \mathrel{-\!\!-} u'$ in G and $v \mathrel{-\!\!-} v'$ in H.
- $G \otimes H$ is the strong product of G and H: As its symbol implies, it combines the edges of $G \times H$ and $G \circ H$.
- There also are coproducts, analogous to the cosum.

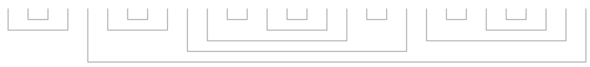
Just as in a single body there are pairs of individual members,
called by the same name but distinguished as right and left,
so when my speeches had postulated the notion of madness,
as a single generic aspect of human nature,
the speech that divided the left-hand portion
repeatedly broke it down into smaller and smaller parts.

— SOCRATES, *Phædrus* 266A (c. 370 B.C.)

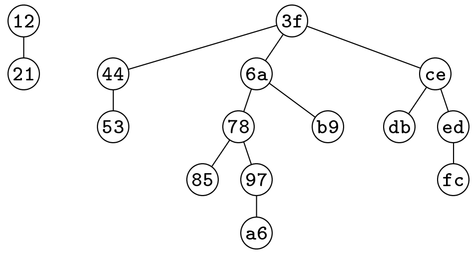
7.2.1.6. Generating all trees. We've now completed our study of the classical concepts of combinatorics: tuples, permutations, combinations, and partitions. But computer scientists have added another fundamental class of patterns to the traditional repertoire, namely the hierarchical arrangements known as trees. Trees sprout up just about everywhere in computer science, as we've seen in Section 2.3 and in nearly every subsequent section of *The Art of Computer Programming*. Therefore we turn now to the study of simple algorithms by which trees of various species can be exhaustively explored.

First let's review the basic connection between nested parentheses and forests of trees. For example,

```
1 2     3 4 5     6 7 8   9 a       b     c d   e f
( ( ) ) ( ( ( ) ) ( ( ( ) ( ( ) ) ) ( ) ) ( ( ) ( ( ) ) ) )        (1)
    1 2       3 4       5     6 7 8   9 a     b     c d e f
```

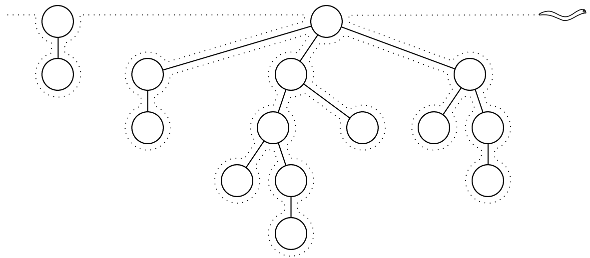


illustrates a string containing fifteen left parens '`(`' labeled `1`, `2`, ..., `f`, and fifteen right parens '`)`' also labeled `1` through `f`; gray lines beneath the string show how the parentheses match up to form fifteen pairs `12`, `21`, `3f`, `44`, `53`, `6a`, `78`, `85`, `97`, `a6`, `b9`, `ce`, `db`, `ed`, and `fc`. This string corresponds to the forest



(2)

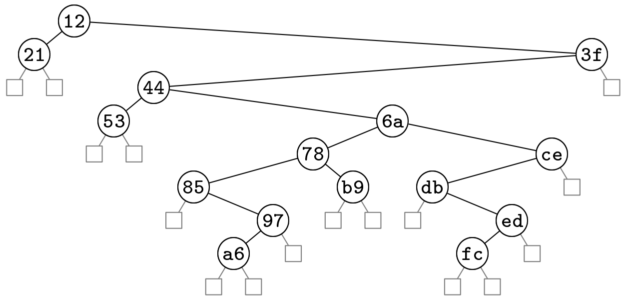
in which the nodes are 12, 21, 3f, ..., fc in preorder (sorted by first coordinates) and 21, 12, 53, ..., 3f in postorder (sorted by second coordinates). If we imagine a worm that crawls around the periphery of the forest,



(3)

seeing a '`(`' whenever it passes the left edge of a node and a '`)`' whenever it passes a node's right edge, that worm will have reconstructed the original string (1).

The forest in (2) corresponds, in turn, to the binary tree



(4)

via the "natural correspondence" discussed in Section 2.3.2; here the nodes are 21, 12, 53, ..., 3f in *symmetric* order, also known as inorder. The left subtree of node x in the binary tree is the leftmost child of x in the forest, or it is an "external node" □ if x is childless. The right subtree of x in the binary tree is its right sibling in the forest, or □ if x is the rightmost child in its family. Roots of the trees in the forest are considered to be siblings, and the leftmost root of the forest is the root of the binary tree.

Table 1

NESTED PARENTHESES AND RELATED OBJECTS WHEN $n = 4$

$a_1a_2\ldots a_8$	forest	binary tree	$d_1d_2d_3d_4$	$z_1z_2z_3z_4$	$p_1p_2p_3p_4$	$c_1c_2c_3c_4$	matching
()()()()			1111	1357	1234	0000	
()()(())			1102	1356	1243	0001	
()(())()			1021	1347	1324	0010	
()(()())			1012	1346	1342	0011	
()((()))			1003	1345	1432	0012	
(())()()			0211	1257	2134	0100	
(())(())			0202	1256	2143	0101	
(()())()			0121	1247	2314	0110	
(()()())			0112	1246	2341	0111	
(()(()))			0103	1245	2431	0112	
((()))()			0031	1237	3214	0120	
((())())			0022	1236	3241	0121	
((()()))			0013	1235	3421	0122	
(((())))			0004	1234	4321	0123	

A string $a_1a_2\ldots a_{2n}$ of parentheses is properly nested if and only if it contains n occurrences of '(' and n occurrences of ')', where the kth '(' precedes the kth ')' for $1 \le k \le n$. The easiest way to explore all strings of nested parentheses is to visit them in lexicographic order. The following algorithm, which considers ')' to be lexicographically smaller than '(', includes some refinements for efficiency suggested by I. Semba [*Inf. Processing Letters* **12** (1981), 188–192]:

Algorithm P (*Nested parentheses in lexicographic order*). Given an integer $n \geq 2$, this algorithm generates all strings $a_1 a_2 \ldots a_{2n}$ of nested parentheses.

P1. [Initialize.] Set $a_{2k-1} \leftarrow$ '(' and $a_{2k} \leftarrow$ ')' for $1 \leq k \leq n$; also set $a_0 \leftarrow$ ')' and $m \leftarrow 2n - 1$.

P2. [Visit.] Visit the nested string $a_1 a_2 \ldots a_{2n}$. (At this point $a_m =$ '(', and $a_k =$ ')' for $m < k \leq 2n$.)

P3. [Easy case?] Set $a_m \leftarrow$ ')'. Then if $a_{m-1} =$ ')', set $a_{m-1} \leftarrow$ '(', $m \leftarrow m-1$, and return to P2.

P4. [Find j.] Set $j \leftarrow m - 1$ and $k \leftarrow 2n - 1$. While $a_j =$ '(', set $a_j \leftarrow$ ')', $a_k \leftarrow$ '(', $j \leftarrow j - 1$, and $k \leftarrow k - 2$.

P5. [Increase a_j.] Terminate the algorithm if $j = 0$. Otherwise set $a_j \leftarrow$ '(', $m \leftarrow 2n - 1$, and go back to P2. ▮

We will see later that the loop in step P4 is almost always short: The operation $a_j \leftarrow$ ')' is performed only about $\frac{1}{3}$ times per nested string visited, on the average.

Why does Algorithm P work? Let A_{pq} be the sequence of all strings α that contain p left parentheses and $q \geq p$ right parentheses, where $(^{q-p}\alpha$ is properly nested, listed in lexicographic order. Then Algorithm P is supposed to generate A_{nn}, where it is easy to see that A_{pq} obeys the recursive rules

$$A_{pq} \;=\;)\,A_{p(q-1)},\ (\,A_{(p-1)q}, \quad \text{if } 0 \leq p \leq q \neq 0; \qquad A_{00} = \epsilon; \tag{5}$$

also A_{pq} is empty if $p < 0$ or $p > q$. The first element of A_{pq} is $)^{q-p}()\ldots()$, where there are p pairs '()'; the last element is $(^p)^q$. Thus the lexicographic generation process consists of scanning from the right until finding a trailing string of the form $a_j \ldots a_{2n} =)(^{p+1})^q$ and replacing it by $()^{q+1-p}()\ldots()$. Steps P4 and P5 do this efficiently, while step P3 handles the simple case $p = 0$.

Table 1 illustrates the output of Algorithm P when $n = 4$, together with the corresponding forest and binary tree as in (2) and (4). Several other equivalent combinatorial objects also appear in Table 1: For example, a string of nested parentheses can be run-length encoded as

$$()^{d_1}()^{d_2}\ldots()^{d_n}, \tag{6}$$

where the nonnegative integers $d_1 d_2 \ldots d_n$ are characterized by the constraints

$$d_1 + d_2 + \cdots + d_k \leq k \quad \text{for } 1 \leq k < n; \qquad d_1 + d_2 + \cdots + d_n = n. \tag{7}$$

We can also represent nested parentheses by the sequence $z_1 z_2 \ldots z_n$, which specifies the indices where the left parentheses appear. In essence, $z_1 z_2 \ldots z_n$ is one of the $\binom{2n}{n}$ combinations of n things from the set $\{1, 2, \ldots, 2n\}$, subject to the special constraints

$$z_{k-1} \;<\; z_k \;<\; 2k \qquad \text{for } 1 \leq k \leq n, \tag{8}$$

if we assume that $z_0 = 0$. The z's are of course related to the d's:

$$d_k \;=\; z_{k+1} - z_k - 1 \qquad \text{for } 1 \leq k < n. \tag{9}$$

Algorithm P becomes particularly simple when it is rewritten to generate the combinations $z_1 z_2 \ldots z_n$ instead of the strings $a_1 a_2 \ldots a_{2n}$. (See exercise 2.)

A parenthesis string can also be represented by the permutation $p_1p_2\ldots p_n$, where the kth right parenthesis matches the p_kth left parenthesis; in other words, the kth node of the associated forest in postorder is the p_kth node in preorder. (By exercise 2.3.2–20, node j is a descendant of node k in the forest if and only if $j < k$ and $p_j > p_k$, when we label the nodes in postorder.) The inversion table $c_1c_2\ldots c_n$ characterizes this permutation by the rule that exactly c_k elements to the right of k are less than k (see exercise 5.1.1–7); allowable inversion tables have $c_1 = 0$ and

$$0 \;\le\; c_{k+1} \;\le\; c_k + 1 \qquad \text{for } 1 \le k < n. \tag{10}$$

Moreover, exercise 3 proves that c_k is the level of the forest's kth node in preorder (the depth of the kth left parenthesis), a fact that is equivalent to the formula

$$c_k = 2k - 1 - z_k. \tag{11}$$

Table 1 and exercise 6 also illustrate a special kind of *matching*, by which $2n$ people at a circular table can simultaneously shake hands without interference.

Thus Algorithm P can be useful indeed. But if our goal is to generate all binary trees, represented by left links $l_1l_2\ldots l_n$ and right links $r_1r_2\ldots r_n$, the lexicographic sequence in Table 1 is rather awkward; the data we need to get from one tree to its successor is not readily available. Fortunately, an ingenious alternative scheme for direct generation of all linked binary trees is also available:

Algorithm B (*Binary trees*). Given $n \ge 1$, this algorithm generates all binary trees with n internal nodes, representing them via left links $l_1l_2\ldots l_n$ and right links $r_1r_2\ldots r_n$, with nodes labeled in preorder. (Thus, for example, node 1 is always the root, and l_k is either $k+1$ or 0; if $l_1 = 0$ and $n > 1$ then $r_1 = 2$.)

B1. [Initialize.] Set $l_k \leftarrow k+1$ and $r_k \leftarrow 0$ for $1 \le k < n$; also set $l_n \leftarrow r_n \leftarrow 0$, and set $l_{n+1} \leftarrow 1$ (for convenience in step B3).

B2. [Visit.] Visit the binary tree represented by $l_1l_2\ldots l_n$ and $r_1r_2\ldots r_n$.

B3. [Find j.] Set $j \leftarrow 1$. While $l_j = 0$, set $r_j \leftarrow 0$, $l_j \leftarrow j+1$, and $j \leftarrow j+1$. Then terminate the algorithm if $j > n$.

B4. [Find k and y.] Set $y \leftarrow l_j$ and $k \leftarrow 0$. While $r_y > 0$, set $k \leftarrow y$ and $y \leftarrow r_y$.

B5. [Promote y.] If $k > 0$, set $r_k \leftarrow 0$; otherwise set $l_j \leftarrow 0$. Then set $r_y \leftarrow r_j$, $r_j \leftarrow y$, and return to B2. ▌

[See W. Skarbek, *Theoretical Computer Science* **57** (1988), 153–159; step B3 uses an idea of J. Korsh.] Exercise 44 proves that the loops in steps B3 and B4 both tend to be very short. Indeed, fewer than 9 memory references are needed, on the average, to transform a linked binary tree into its successor.

Table 2 shows the fourteen binary trees that are generated when $n = 4$, together with their corresponding forests and with two related sequences: Arrays $e_1e_2\ldots e_n$ and $s_1s_2\ldots s_n$ are defined by the property that node k in preorder has e_k children and s_k descendants in the associated forest. (Thus s_k is the size of k's left subtree in the binary tree; also, $s_k + 1$ is the length of the `SCOPE` link in the sense of 2.3.3–(5).) The next column repeats the fourteen forests of Table 1 in the lexicographic ordering of Algorithm P, but mirror-reversed from left to right.

Table 2

LINKED BINARY TREES AND RELATED OBJECTS WHEN $n = 4$

$l_1l_2l_3l_4$	$r_1r_2r_3r_4$	binary tree	forest	$e_1e_2e_3e_4$	$s_1s_2s_3s_4$	colex forest	lsib/rchild
2340	0000			1110	3210		
0340	2000			0110	0210		
2040	0300			2010	3010		
2040	3000			1010	1010		
0040	2300			0010	0010		
2300	0040			1200	3200		
0300	2040			0200	0200		
2300	0400			2100	3100		
2300	4000			1100	2100		
0300	2400			0100	0100		
2000	0340			3000	3000		
2000	4300			2000	2000		
2000	3040			1000	1000		
0000	2340			0000	0000		

And the final column shows the binary tree that represents the colex forest; it also happens to represent the forest in column 4, but by links to left sibling and right child instead of to left child and right sibling. This final column provides an interesting connection between nested parentheses and binary trees, so it gives us some insight into why Algorithm B is valid (see exercise 19).

***Gray codes for trees.** Our previous experiences with other combinatorial patterns suggest that we can probably generate parentheses and trees by making only small perturbations to get from one instance to another. And indeed, there are at least three very nice ways to achieve this goal.

Consider first the case of nested parentheses, which we can represent by the sequences $z_1 z_2 \ldots z_n$ that satisfy condition (8). A "near-perfect" way to generate all such combinations, in the sense of Section 7.2.1.3, is one in which we run through all possibilities in such a way that some component z_j changes by ±1 or ±2 at each step; this means that we get from each string of parentheses to its successor by simply changing either `()` $\leftrightarrow$ `)(` or `())` $\leftrightarrow$ `))(` in the vicinity of the jth left parenthesis. Here's one way to do the job when $n = 4$:

$$1357, 1356, 1346, 1345, 1347, 1247, 1245, 1246, 1236, 1234, 1235, 1237, 1257, 1256.$$

And we can extend any solution for $n-1$ to a solution for n, by taking each pattern $z_1 z_2 \ldots z_{n-1}$ and letting z_n run through all of its legal values using *endo-order* or its reverse as in 7.2.1.3–(45), proceeding downward from $2n-2$ and then up to $2n-1$ or vice versa, and omitting all elements that are $\le z_{n-1}$.

Algorithm N (*Near-perfect nested parentheses*). This algorithm visits all n-combinations $z_1 \ldots z_n$ of $\{1, \ldots, 2n\}$ that represent the indices of left parentheses in a nested string, changing only one index at a time. The process is controlled by an auxiliary array $g_1 \ldots g_n$ that represents temporary goals.

N1. [Initialize.] Set $z_j \leftarrow 2j-1$ and $g_j \leftarrow 2j-2$ for $1 \le j \le n$.

N2. [Visit.] Visit the n-combination $z_1 \ldots z_n$. Then set $j \leftarrow n$.

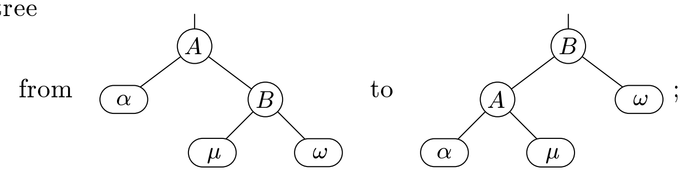
N3. [Find j.] If $z_j = g_j$, set $g_j \leftarrow g_j \oplus 1$ (thereby complementing the least significant bit), $j \leftarrow j-1$, and repeat this step.

N4. [Home stretch?] If $g_j - z_j$ is even, set $z_j \leftarrow z_j + 2$ and return to N2.

N5. [Decrease or turn.] Set $t \leftarrow z_j - 2$. If $t < 0$, terminate the algorithm. Otherwise, if $t \le z_{j-1}$, set $t \leftarrow t + 2[t < z_{j-1}] + 1$. Finally set $z_j \leftarrow t$ and go back to N2. ▮

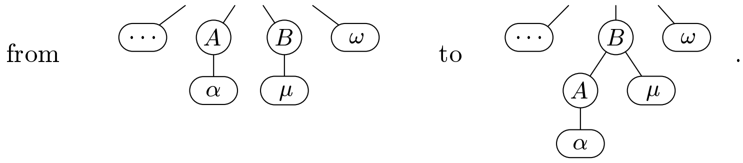
[A somewhat similar algorithm was introduced by D. Roelants van Baronaigien in *J. Algorithms* **35** (2000), 100–107; see also Xiang, Ushijima, and Tang, *Inf. Proc. Letters* **76** (2000), 169–174. F. Ruskey and A. Proskurowski, in *J. Algorithms* **11** (1990), 68–84, had previously shown how to construct *perfect* Gray codes for all tables $z_1 \ldots z_n$ when $n \ge 4$ is even, thus changing some z_j by only ±1 at every step; but their construction was quite complex, and no known perfect scheme is simple enough to be of practical use. Exercise 48 shows that perfection is impossible when $n \ge 5$ is odd.]

If our goal is to generate linked tree structures instead of strings of parentheses, perfection of the z-index changes is not good enough, because simple swaps like `()` $\leftrightarrow$ `)(` don't necessarily correspond to simple link manipulations. A far better approach can be based on the "rotation" algorithms by which we were able to keep search trees balanced in Section 6.2.3. *Rotation to the left* changes a binary tree



(12)

thus the corresponding forest is changed



(13)

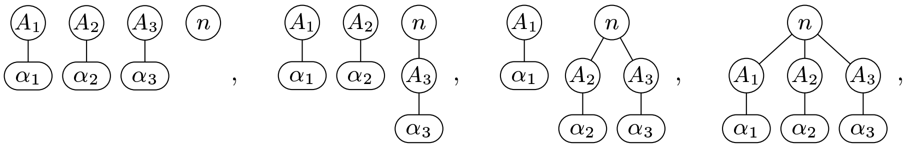
"Node (A) becomes the leftmost child of its right sibling." *Rotation to the right* is, of course, the opposite transformation: "The leftmost child of (B) becomes its left sibling." The vertical line in (12) stands for a connection to the overall context, either a left link or a right link or the pointer to the root. Any or all of the subtrees α, μ, or ω may be empty. The '$\cdots$' in (13), which represents additional siblings at the left of the family containing (B), might also be empty.

The nice thing about rotations is that only three links change: The right link from (A), the left link from (B), and the pointer from above. Rotations preserve inorder of the binary tree and postorder of the forest. (Notice also that the binary-tree form of a rotation corresponds in a natural way to an application of the *associative law*

$$(\alpha\mu)\omega \;=\; \alpha(\mu\omega) \tag{14}$$

in the midst of an algebraic formula.)

A simple scheme very much like the classical reflected Gray code for n-tuples (Algorithm 7.2.1.1H) and the method of plain changes for permutations (Algorithm 7.2.1.2P) can be used to generate all binary trees or forests via rotations. Consider any forest on $n-1$ nodes, with k roots (A_1), ..., (A_k). Then there are $k+1$ forests on n nodes that have the same postorder sequence on the first $n-1$ nodes but with node (n) last; for example, when $k=3$ they are



obtained by successively rotating (A_3), (A_2), and (A_1) to the left. Moreover, at the extremes when (n) is either at the right or at the top, we can perform any desired rotation on the other $n-1$ nodes, because node (n) isn't in the way. Therefore, as observed by J. M. Lucas, D. Roelants van Baronaigien, and F. Ruskey [*J. Algorithms* **15** (1993), 343–366], we can extend any list of the $(n-1)$-node trees to a list of all n-node trees by simply letting node (n) roam back and forth. A careful attention to low-level details makes it possible in fact to do the job with remarkable efficiency:

Algorithm L (*Linked binary trees by rotations*). This algorithm generates all pairs of arrays $l_0l_1\ldots l_n$ and $r_1\ldots r_n$ that represent left links and right links of n-node binary trees, where l_0 is the root of the tree and the links (l_k, r_k) point respectively to the left and right subtrees of the kth node in symmetric order. Equivalently, it generates all n-node forests, where l_k and r_k denote the left child and right sibling of the kth node in postorder. Each tree is obtained from its predecessor by doing a single rotation. Two auxiliary arrays $k_1\ldots k_n$ and $o_0o_1\ldots o_n$, representing backpointers and directions, are used to control the process.

L1. [Initialize.] Set $l_j \leftarrow 0$, $r_j \leftarrow j+1$, $k_j \leftarrow j-1$, and $o_j \leftarrow -1$ for $1 \le j < n$; also set $l_0 \leftarrow o_0 \leftarrow 1$, $l_n \leftarrow r_n \leftarrow 0$, $k_n \leftarrow n-1$, and $o_n \leftarrow -1$.

L2. [Visit.] Visit the binary tree or forest represented by $l_0l_1\ldots l_n$ and $r_1\ldots r_n$. Then set $j \leftarrow n$ and $p \leftarrow 0$.

L3. [Find j.] If $o_j > 0$, set $m \leftarrow l_j$ and go to L5 if $m \ne 0$. If $o_j < 0$, set $m \leftarrow k_j$; then go to L4 if $m \ne 0$, otherwise set $p \leftarrow j$. If $m = 0$ in either case, set $o_j \leftarrow -o_j$, $j \leftarrow j-1$, and repeat this step.

L4. [Rotate left.] Set $r_m \leftarrow l_j$, $l_j \leftarrow m$, $x \leftarrow k_m$, and $k_j \leftarrow x$. If $x = 0$, set $l_p \leftarrow j$, otherwise set $r_x \leftarrow j$. Return to L2.

L5. [Rotate right.] Terminate if $j = 0$. Otherwise set $l_j \leftarrow r_m$, $r_m \leftarrow j$, $k_j \leftarrow m$, $x \leftarrow k_m$. If $x = 0$, set $l_p \leftarrow m$, otherwise set $r_x \leftarrow m$. Go back to L2. ▮

Exercise 38 proves that Algorithm L needs only about 9 memory references per tree generated; thus it is almost as fast as Algorithm B. (In fact, two memory references per step could be saved by keeping the three quantities o_n, l_n, and k_n in registers. But of course Algorithm B can be speeded up too.)

Table 3 shows the sequence of binary trees and forests visited by Algorithm L when $n = 4$, with some auxiliary tables that shed further light on the process. The permutation $q_1q_2q_3q_4$ lists the nodes in preorder, when they have been numbered in postorder of the forest (symmetric order of the binary tree); it is the inverse of the permutation $p_1p_2p_3p_4$ in Table 1. The "coforest" is the conjugate (right-to-left reflection) of the forest; and the numbers $u_1u_2u_3u_4$ are its scope coordinates, analogous to $s_1s_2s_3s_4$ in Table 2. A final column shows the so-called "dual forest." The significance of these associated quantities is explored in exercises 11–13, 19, 24, 26, and 27.

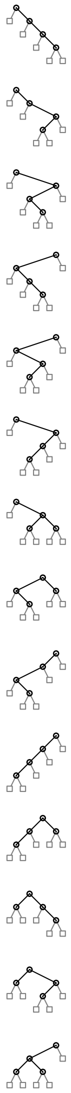
The links $l_0l_1\ldots l_n$ and $r_1\ldots r_n$ in Algorithm L and Table 3 are *not* comparable to the links $l_1\ldots l_n$ and $r_1\ldots r_n$ in Algorithm B and Table 2, because Algorithm L preserves inorder/postorder while Algorithm B preserves preorder. Node k in Algorithm L is the kth node from left to right in the binary tree, so l_0 is needed to identify the root; but node k in Algorithm B is the kth node in preorder, so the root is always node 1 in that case.

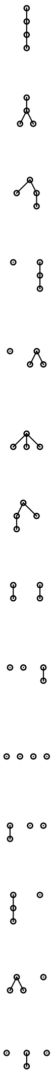
Algorithm L has the desired property that only three links change per step; but we can actually do even better in this respect if we stick to the preorder convention of Algorithm B. Exercise 25 presents an algorithm that generates

Table 3

BINARY TREES AND FORESTS GENERATED BY ROTATIONS WHEN $n = 4$

$l_0l_1l_2l_3l_4$	$r_1r_2r_3r_4$	$k_1k_2k_3k_4$	binary tree	forest	$q_1q_2q_3q_4$	coforest	$u_1u_2u_3u_4$	dual
10000	2340	0123			1234		0000	
10003	2400	0122			1243		1000	
10002	4300	0121			1423		2000	
40001	2300	0120			4123		3000	
40021	3000	0110			4132		3100	
10023	4000	0111			1432		2100	
10020	3040	0113			1324		0100	
30010	2040	0103			3124		0200	
40013	2000	0100			4312		3200	
40123	0000	0000			4321		3210	
30120	0040	0003			3214		0210	
20100	0340	0023			2134		0010	
20103	0400	0022			2143		1010	
40102	0300	0020			4213		3010	





all linked binary trees or forests by changing just two links per step, preserving preorder. One link becomes zero while another becomes nonzero. This prune-and-graft algorithm, which is the third of the three "very nice Gray codes for trees" promised above, has only one downside: Its controlling mechanism is a bit trickier than that of Algorithm L, so it needs about 40% more time to do the calculations when we include the cost of deciding what links to change at each step.

The number of trees. There's a simple formula for the total number of outputs that are generated by Algorithms P, B, N, and L, namely

$$C_n \;=\; \frac{1}{n+1}\binom{2n}{n} \;=\; \binom{2n}{n} - \binom{2n}{n-1}; \qquad (15)$$

we proved this fact in Eq. 2.3.4.4–(14). The first few values are

$n =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$C_n =$	1	1	2	5	14	42	132	429	1430	4862	16796	58786	208012	742900

and they are called *Catalan numbers* because of some influential papers written by Eugène Catalan [*Journal de math.* **3** (1838), 508–515; **4** (1839), 95–99]. Stirling's approximation tells us the asymptotic value,

$$C_n \;=\; \frac{4^n}{\sqrt{\pi}\,n^{3/2}}\Bigl(1-\frac{9}{8n}+\frac{145}{128n^2}-\frac{1155}{1024n^3}+\frac{36939}{32768n^4}+O(n^{-5})\Bigr); \qquad (16)$$

in particular we can conclude that

$$\frac{C_{n-k}}{C_n} \;=\; \frac{1}{4^k}\Bigl(1+\frac{3k}{2n}+O\Bigl(\frac{k^2}{n^2}\Bigr)\Bigr) \qquad \text{when } |k|\le\frac{n}{2}. \qquad (17)$$

(And of course C_{n-1}/C_n is equal to $(n+1)/(4n-2)$, exactly, by (15).) In Section 2.3.4.4 we also derived the generating function

$$C(z) \;=\; C_0+C_1z+C_2z^2+C_3z^3+\cdots \;=\; \frac{1-\sqrt{1-4z}}{2z} \qquad (18)$$

and proved the important formula

$$[z^n]\,C(z)^r \;=\; \frac{r}{n+r}\binom{2n+r-1}{n} \;=\; \binom{2n+r-1}{n}-\binom{2n+r-1}{n-1}; \qquad (19)$$

see the answer to exercise 2.3.4.4–33, and *CMath* equation (5.70).

These facts give us more than enough information to analyze Algorithm P, our algorithm for lexicographic generation of nested parentheses. Step P2 is obviously performed C_n times; then P3 usually makes a simple change and goes back to P2. How often do we need to go on to step P4? Easy: It's the number of times that step P2 finds $m = 2n-1$. And m is the location of the rightmost '(', so we have $m = 2n-1$ in exactly C_{n-1} cases. Thus the probability that P3 sets $m \leftarrow m-1$ and returns immediately to P2 is $(C_n - C_{n-1})/C_n \approx 3/4$, by (17). On the other hand when we do get to step P4, suppose we need to set $a_j \leftarrow$ ')' and $a_k \leftarrow$ '(' exactly $h-1$ times in that step. The number of cases with $h > x$ is the number of nested strings of length $2n$ that end with x trivial pairs `()` … `()`, namely C_{n-x}. Therefore the total number of times the algorithm changes a_j and a_k in step P4 is

$$\begin{aligned} C_{n-1}+C_{n-2}+\cdots+C_1 \;&=\; C_n\Bigl(\frac{C_{n-1}}{C_n}+\frac{C_{n-2}}{C_n}+\cdots+\frac{C_1}{C_n}\Bigr)\\ &=\; \frac{1}{3}C_n\Bigl(1+\frac{2}{n}+O\Bigl(\frac{1}{n^2}\Bigr)\Bigr), \end{aligned} \qquad (20)$$

by (17); we have proved the claim for efficiency made earlier.

For a deeper understanding it is helpful to study the recursive structure underlying Algorithm P, as expressed in (5). The sequences A_{pq} in that formula have C_{pq} elements, where

$$C_{pq} = C_{p(q-1)} + C_{(p-1)q}, \quad \text{if } 0 \le p \le q \ne 0; \qquad C_{00} = 1; \tag{21}$$

and $C_{pq} = 0$ if $p < 0$ or $p > q$. Thus we can form the triangular array

$$\begin{array}{lllllll} C_{00} \\ C_{01} & C_{11} \\ C_{02} & C_{12} & C_{22} \\ C_{03} & C_{13} & C_{23} & C_{33} \\ C_{04} & C_{14} & C_{24} & C_{34} & C_{44} \\ C_{05} & C_{15} & C_{25} & C_{35} & C_{45} & C_{55} \\ C_{06} & C_{16} & C_{26} & C_{36} & C_{46} & C_{56} & C_{66} \end{array} \;=\; \begin{array}{ccccccc} 1 \\ 1 & 1 \\ 1 & 2 & 2 \\ 1 & 3 & 5 & 5 \\ 1 & 4 & 9 & 14 & 14 \\ 1 & 5 & 14 & 28 & 42 & 42 \\ 1 & 6 & 20 & 48 & 90 & 132 & 132 \end{array} \tag{22}$$

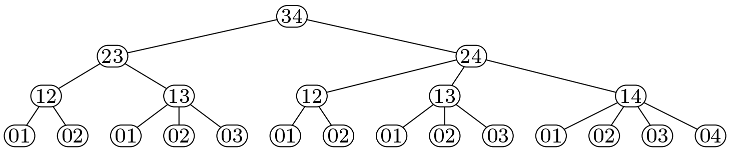
in which every entry is the sum of its nearest neighbors above and to the left; the Catalan numbers $C_n = C_{nn}$ appear on the diagonal. The elements of this triangle, which themselves have a venerable pedigree going back to de Moivre in 1711, are called "ballot numbers," because they represent sequences of $p+q$ ballots for which a running tabulation never favors a candidate with p votes over an opponent who receives q votes. The general formula

$$C_{pq} \;=\; \frac{q-p+1}{q+1}\binom{p+q}{p} \;=\; \binom{p+q}{p} - \binom{p+q}{p-1} \tag{23}$$

can be proved by induction or in a variety of more interesting ways; see exercise 39 and the answer to exercise 2.2.1–4. Notice that, because of (19), we have

$$C_{pq} \;=\; [z^p]\, C(z)^{q-p+1}. \tag{24}$$

When $n = 4$, Algorithm P essentially describes the recursion tree



(25)

because the specification (5) implies that $A_{nn} = (\,A_{(n-1)n}$ and that

$$A_{pq} \;=\;)^{q-p}(A_{(p-1)p},\;)^{q-p-1}(\,A_{(p-1)(p+1)},\;)^{q-p-2}(\,A_{(p-1)(p+2)},\; \ldots,\; (\,A_{(p-1)q} \qquad \text{when } 0 \le p < q. \tag{26}$$

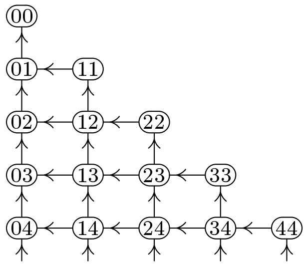
The number of leaves below node (pq) in this recursion tree is C_{pq}, and node (pq) appears exactly $C_{(n-q)(n-1-p)}$ times on level $n-1-p$; therefore we must have

$$\sum_q C_{(n-q)(n-1-p)} C_{pq} \;=\; C_n, \qquad \text{for } 0 \le p < n. \tag{27}$$

The fourteen leaves of (25), from left to right, correspond to the fourteen rows of Table 1, from top to bottom. Notice that the entries in column $c_1c_2c_3c_4$ of that table assign the respective numbers 0000, 0001, 0010, ..., 0123 to the leaves of (25), in accord with "Dewey decimal notation" for tree nodes (but with indices starting at 0 instead of 1, and with an extra 0 tacked on at the beginning).

A worm that crawls from one leaf to the next, around the bottom of the recursion tree, will ascend and descend h levels when h of the coordinates $c_1 \ldots c_n$ are changed, namely when Algorithm P resets the values of h '`(`'s and h '`)`'s. This observation makes it easy to understand our previous conclusion that the condition $h > x$ occurs exactly C_{n-x} times during a complete crawl.

Yet another way to understand Algorithm P arises when we contemplate an infinite directed graph that is suggested by the recursion (21):



(28)

Clearly C_{pq} is the number of paths from pq to 00 in this digraph, because of (21). And indeed, every string of parentheses in A_{pq} corresponds directly to such a path, with '`(`' signifying a step to the left and '`)`' signifying a step upward. Algorithm P explores all such paths systematically by trying first to go upward when extending a partial path.

Therefore it is easy to determine the Nth string of nested parentheses that is visited by Algorithm P, by starting at node nn and doing the following calculation when at node pq: If $p = q = 0$, stop; otherwise, if $N \le C_{p(q-1)}$, emit '`)`', set $q \leftarrow q - 1$, and continue; otherwise set $N \leftarrow N - C_{p(q-1)}$, emit '`(`', set $p \leftarrow p - 1$, and continue. The following algorithm [Frank Ruskey, Ph.D. thesis (University of California at San Diego, 1978), 16–24] avoids the need to precompute the Catalan triangle by evaluating C_{pq} on the fly as it goes:

Algorithm U (*Unrank a string of nested parentheses*). Given n and N, where $1 \le N \le C_n$, this algorithm computes the Nth output $a_1 \ldots a_{2n}$ of Algorithm P.

U1. [Initialize.] Set $q \leftarrow n$ and $m \leftarrow p \leftarrow c \leftarrow 1$. While $p < n$, set $p \leftarrow p + 1$ and $c \leftarrow ((4p - 2)c)/(p + 1)$.

U2. [Done?] Terminate the algorithm if $q = 0$.

U3. [Go up?] Set $c' \leftarrow ((q + 1)(q - p)c)/((q + p)(q - p + 1))$. (At this point we have $1 \le N \le c = C_{pq}$ and $c' = C_{p(q-1)}$.) If $N \le c'$, set $q \leftarrow q - 1$, $c \leftarrow c'$, $a_m \leftarrow$ '`)`', $m \leftarrow m + 1$, and return to U2.

U4. [Go left.] Set $p \leftarrow p - 1$, $c \leftarrow c - c'$, $N \leftarrow N - c'$, $a_m \leftarrow$ '`(`', $m \leftarrow m + 1$, and return to U3. ▮

Random trees. We could choose a string $a_1 a_2 \ldots a_{2n}$ of nested parentheses at random by simply applying Algorithm U to a random integer N between 1 and C_n. But that idea isn't really very good, when n is bigger than 32 or so, because C_n can be quite large. A simpler and better way, proposed by D. B. Arnold and M. R. Sleep [*ACM Trans. Prog. Languages and Systems* **2** (1980), 122–128], is to generate a random "worm walk" by starting at nn in (28) and repeatedly taking leftward or upward branches with the appropriate probabilities. The resulting algorithm is almost the same as Algorithm U, but it deals only with nonnegative integers less than n^2+n+1:

Algorithm W (*Uniformly random strings of nested parentheses*). This algorithm generates a random string $a_1a_2\ldots a_{2n}$ of properly nested `(`s and `)`s.

W1. [Initialize.] Set $p \leftarrow q \leftarrow n$ and $m \leftarrow 1$.

W2. [Done?] Terminate the algorithm if $q=0$.

W3. [Go up?] Let X be a random integer in the range $0 \le X < (q+p)(q-p+1)$. If $X < (q+1)(q-p)$, set $q \leftarrow q-1$, $a_m \leftarrow$ '`)`', $m \leftarrow m+1$, and return to W2.

W4. [Go left.] Set $p \leftarrow p-1$, $a_m \leftarrow$ '`(`', $m \leftarrow m+1$, and return to W3. ▮

A worm's walk can be regarded as a sequence $w_0w_1\ldots w_{2n}$, where w_m is the worm's current depth after m steps. Thus, $w_0=0$; $w_m=w_{m-1}+1$ when $a_m =$ '`(`'; $w_m = w_{m-1}-1$ when $a_m =$ '`)`'; and we have $w_m \ge 0$, $w_{2n}=0$. The sequence $w_0w_1\ldots w_{30}$ corresponding to (1) and (2) is 0121012321234345432321232343210. At step W3 of Algorithm W we have $q+p=2n+1-m$ and $q-p=w_{m-1}$.

Let's say that the *outline* of a forest is the path that runs through the points $(m,-w_m)$ in the plane, for $0 \le m \le 2n$, where $w_0w_1\ldots w_{2n}$ is the worm walk corresponding to the associated string $a_1\ldots a_{2n}$ of nested parentheses. Figure 36 shows what happens if we plot the outlines of all 50-node forests and darken each point according to the number of forests that lie above it. For example, w_1 is always 1, so the triangular region at the upper left of Fig. 36 is solid black. But w_2 is either 0 or 2, and 0 occurs in $C_{49} \approx C_{50}/4$ cases; so the adjacent diamond-shaped area is a 75% shade of gray. Thus Fig. 36 illustrates the shape of a random forest, analogous to the shapes of random partitions that we've seen in Figs. 30, 31, and 35 of Sections 7.2.1.4 and 7.2.1.5.

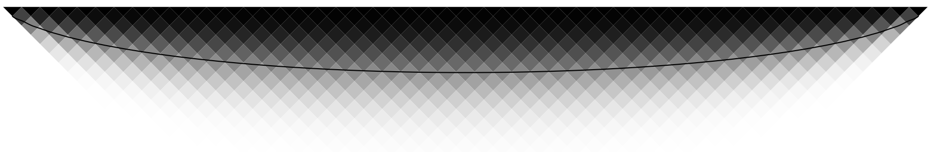


Fig. 36. The shape of a random 50-node forest.

Of course we can't really draw the outlines of all those forests, since there are $C_{50} = 1{,}978{,}261{,}657{,}756{,}160{,}653{,}623{,}774{,}456$ of them. But with the help of mathematics we can pretend that we've done so. The probability that $w_{2m}=2k$ is $C_{(m-k)(m+k)}C_{(n-m-k)(n-m+k)}/C_n$, because there are $C_{(m-k)(m+k)}$ ways to start with $m+k$ `(`s and $m-k$ `)`s, and $C_{(n-m-k)(n-m+k)}$ ways to finish with

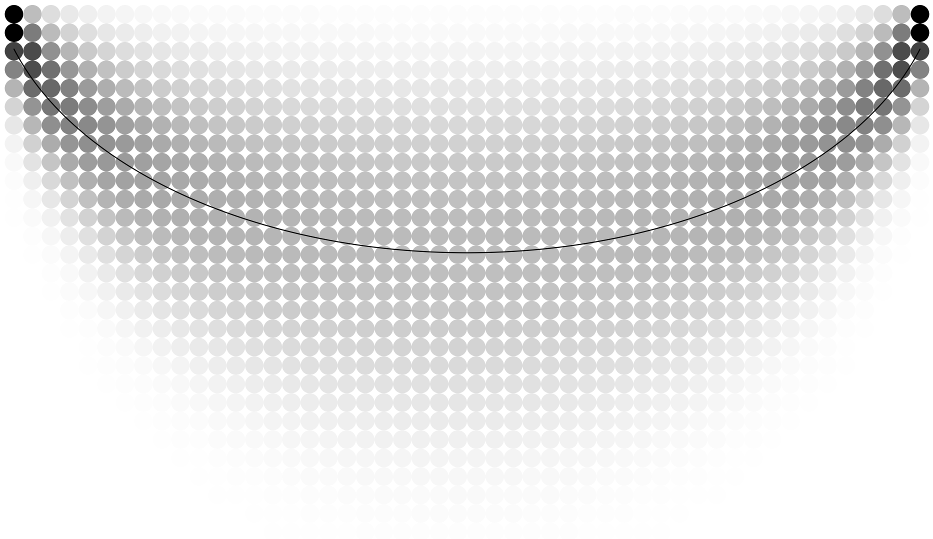


Fig. 37. Locations of the internal nodes in a random 50-node binary tree.

$n-(m+k)$ (s and $n-(m-k)$)s. By (23) and Stirling's approximation, this probability is

$$\begin{aligned}&\frac{(2k+1)^2(n+1)}{(m+k+1)(n-m+k+1)}\binom{2m}{m-k}\binom{2n-2m}{n-m+k}\Big/\binom{2n}{n}\\&\quad=\frac{(2k+1)^2}{\sqrt{\pi}\,\big(\theta(1-\theta)n\big)^{3/2}}\,e^{-k^2/(\theta(1-\theta)n)}\Big(1+O\Big(\frac{k+1}{n}\Big)+O\Big(\frac{k^3}{n^2}\Big)\Big)\end{aligned}\tag{29}$$

when $m=\theta n$ and $n\to\infty$, for $0<\theta<1$. The average value of w_{2m} is worked out in exercise 57; it comes to

$$\frac{(4m(n-m)+n)\binom{2m}{m}\binom{2n-2m}{n-m}}{n\binom{2n}{n}}-1\;=\;4\sqrt{\frac{\theta(1-\theta)n}{\pi}}-1+O(n^{-1/2}),\tag{30}$$

and it is illustrated for $n=50$ as a curved line in Fig. 36.

When n is large, worm walks approach the so-called "Brownian excursion," which is an important concept in probability theory. See, for example, Paul Lévy, *Processus Stochastiques et Mouvement Brownien* (1948), 225–237; Guy Louchard, *J. Applied Prob.* **21** (1984), 479–499, and *BIT* **26** (1986), 17–34; David Aldous, *Electronic Communications in Probability* **3** (1998), 79–90; Jon Warren, *Electronic Communications in Probability* **4** (1999), 25–29; J.-F. Marckert, *Random Structures and Algorithms* **24** (2004), 118–132.

What is the shape of a random *binary* tree? This question was investigated by Frank Ruskey in *SIAM J. Algebraic and Discrete Methods* **1** (1980), 43–50, and the answer turns out to be quite interesting. Suppose we draw a binary tree

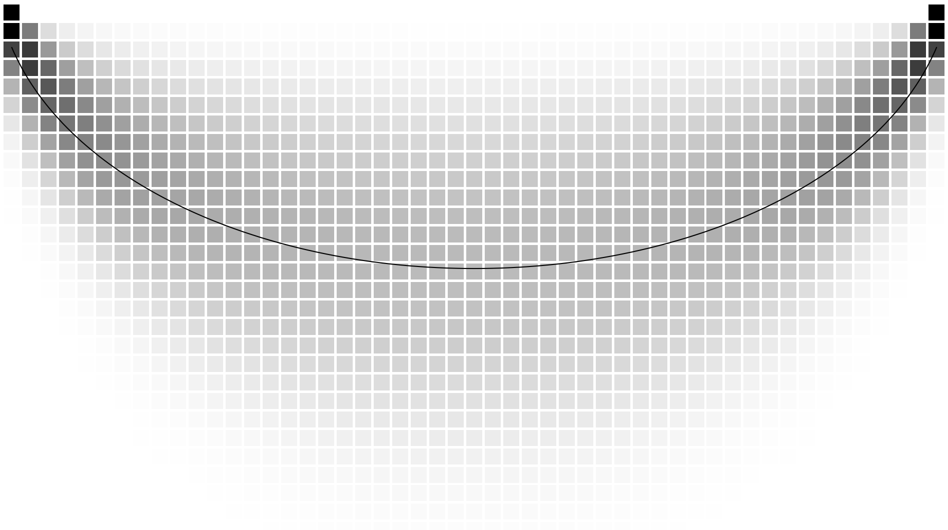


Fig. 38. Locations of the external nodes in a random 50-node binary tree.

as in (4), with the mth internal node at horizontal position m when the nodes are numbered in symmetric order. If all of the 50-node binary trees are drawn in this way and superimposed on each other, we get the distribution of node positions shown in Fig. 37. Similarly, if we number the *external* nodes from 0 to n in symmetric order and place them at horizontal positions .5, 1.5, ..., $n+.5$, the "fringes" of all 50-node binary trees form the distribution shown in Fig. 38. Notice that the root node is most likely to be either number 1 or number n, at the extreme left or right; it is least likely to be either $\lfloor (n+1)/2 \rfloor$ or $\lceil (n+1)/2 \rceil$, in the middle.

As in Fig. 36, the smooth curves in Figs. 37 and 38 show the average node depths; exact formulas are derived in exercises 58 and 59. Asymptotically, the average depth of external node m is

$$8\sqrt{\frac{\theta(1-\theta)n}{\pi}} - 1 + O\Bigl(\frac{1}{\sqrt{n}}\Bigr), \qquad \text{when } m = \theta n \text{ and } n \to \infty, \tag{31}$$

for all fixed ratios θ with $0 < \theta < 1$, curiously like (30); and the average depth of *internal* node m is asymptotically the same, but with '-1' replaced by '-3'. Thus we can say that *the average shape of a random binary tree is approximately the lower half of an ellipse, n units wide and $4\sqrt{n/\pi}$ levels deep.*

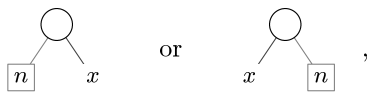
Three other noteworthy ways to generate random encodings of forests are discussed in exercises 60, 61, and 62. They are less direct than Algorithm W, yet they have substantial combinatorial interest. The first one begins with an arbitrary random string containing n `(`s and n `)`s, not necessarily nested; each of the $\binom{2n}{n}$ possibilities is equally likely. It then proceeds to convert every such string into a sequence that is properly nested, in such a way that exactly $n+1$

strings map into each final outcome. The second method is similar, but it starts with a sequence of $n+1$ 0s and n 2s, mapping them in such a way that exactly $2n+1$ original strings produce each possible result. And the third method produces each output from exactly n of the bit strings that contain exactly $n-1$ 1s and $n+1$ 0s. In other words, the three methods provide combinatorial proofs of the fact that C_n is simultaneously equal to $\binom{2n}{n}/(n+1)$, $\binom{2n+1}{n}/(2n+1)$, and $\binom{2n}{n-1}/n$. For example, when $n=4$ we have $14 = 70/5 = 126/9 = 56/4$.

If we want to generate random binary trees directly in linked form, we can use a beautiful method suggested by J. L. Rémy [*RAIRO Informatique Théorique* **19** (1985), 179–195]. His approach is particularly instructive because it shows how random Catalan trees might actually occur "in nature," using a deliciously simple mechanism based on a classical idea of Olinde Rodrigues [*J. de Math.* **3** (1838), 549]. Let us suppose that our goal is to obtain not only an ordinary n-node binary tree, but a *decorated* binary tree, namely an extended binary tree in which the external nodes have been labeled with the numbers 0 to n in some order. There are $(n+1)!$ ways to decorate any given binary tree; so the total number of decorated binary trees with n internal nodes is

$$D_n \;=\; (n+1)!\,C_n \;=\; \frac{(2n)!}{n!} \;=\; (4n-2)\,D_{n-1}. \tag{32}$$

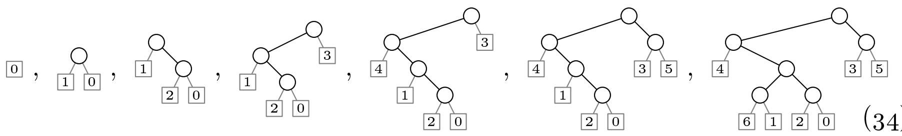
Rémy observed that there are $4n-2$ easy ways to build a decorated tree of order n from a given decorated tree of order $n-1$: We simply choose any one of the $2n-1$ nodes (internal or external) in the given tree, say x, and replace it by either



(33)

thus inserting a new internal node and a new leaf while moving x and its descendants (if any) down one level.

For example, here's one way to construct a decorated tree of order 6:



(34)

Notice that every decorated tree is obtained by this process in exactly one way, because the predecessor of each tree must be the tree we get by striking out the highest-numbered leaf. Therefore Rémy's construction produces decorated trees that are uniformly random; and if we ignore the external nodes, we get random binary trees of the ordinary, undecorated variety.

One appealing way to implement Rémy's procedure is to maintain a table of links $L_0L_1\ldots L_{2n}$ where external (leaf) nodes have even numbers and internal (branch) nodes have odd numbers. The root is node L_0; the left and right children of branch node $2k-1$ are respectively L_{2k-1} and L_{2k}, for $1 \le k \le n$. Then the program is short and sweet:

Algorithm R (*Growing a random binary tree*). This algorithm constructs the linked representation $L_0L_1\dots L_{2N}$ of a uniformly random binary tree with N internal nodes, using the conventions explained above.

R1. [Initialize.] Set $n \leftarrow 0$ and $L_0 \leftarrow 0$.

R2. [Done?] (At this point the links $L_0L_1\dots L_{2n}$ represent a random n-node binary tree.) Terminate the algorithm if $n = N$.

R3. [Advance n.] Let X be a random integer between 0 and $4n+1$, inclusive. Set $n \leftarrow n+1$, $b \leftarrow X \bmod 2$, $k \leftarrow \lfloor X/2 \rfloor$, $L_{2n-b} \leftarrow 2n$, $L_{2n-1+b} \leftarrow L_k$, $L_k \leftarrow 2n-1$, and return to R2. ▌

***Chains of subsets.** Now that we've got trees and parentheses firmly in mind, it's a good time to discuss the *Christmas tree pattern*,* which is a remarkable way to arrange the set of all 2^n bit strings of length n into $\binom{n}{\lfloor n/2 \rfloor}$ rows and $n+1$ columns, discovered by de Bruijn, van Ebbenhorst Tengbergen, and Kruyswijk [*Nieuw Archief voor Wiskunde* (2) **23** (1951), 191–193].

The Christmas tree pattern of order 1 is the single row '0 1'; and the pattern of order 2 is

$$\begin{array}{ccc} & 10 & \\ 00 & 01 & 11 \end{array}\,. \tag{35}$$

In general we get the Christmas tree pattern of order $n+1$ by taking every row '$\sigma_1\ \sigma_2\ \dots\ \sigma_s$' of the order-$n$ pattern and replacing it by the two rows

$$\begin{array}{cccccc} & \sigma_2 0 & \dots & \sigma_s 0 & & \\ \sigma_1 0 & \sigma_1 1 & \dots & \sigma_{s-1} 1 & \sigma_s 1 & \end{array}\,. \tag{36}$$

(The first of these rows is omitted when $s = 1$.)

Proceeding in this way, we obtain for example the pattern of order 8 that appears in Table 4 on the next page. It is easy to verify by induction that

i) Each of the 2^n bit strings appears exactly once in the pattern.
ii) The bit strings with k 1s all appear in the same column.
iii) Within each row, consecutive bit strings differ by changing a 0 to a 1.

If we think of the bit strings as representing subsets of $\{1,\dots,n\}$, with 1-bits to indicate the members of a set, property (iii) says that each row represents a *chain* in which each subset is covered by its successor. In symbols, using the notation of Section 7.1, every row $\sigma_1\ \sigma_2\ \dots\ \sigma_s$ has the property that $\sigma_j \subseteq \sigma_{j+1}$ and $\nu(\sigma_{j+1}) = \nu(\sigma_j) + 1$ for $1 \le j < s$.

Properties (i) and (ii) tell us that there are exactly $\binom{n}{k}$ elements in column k, if we number the columns from 0 to n. This observation, together with the fact that each row is centered among the columns, proves that the total number of rows is $\max_{0\le k\le n}\binom{n}{k} = \binom{n}{\lfloor n/2 \rfloor}$, as claimed. Let us call this number M_n.

* This name was chosen for sentimental reasons, because the pattern has a general shape not unlike that of a festive tree, and because it was the subject of the author's ninth annual "Christmas Tree Lecture" at Stanford University in December 2002.

Table 4

THE CHRISTMAS TREE PATTERN OF ORDER 8

				10101010				
			10101000	10101001	10101011			
				10101100				
			10100100	10100101	10101101			
			10100010	10100110	10101110			
		10100000	10100001	10100011	10100111	10101111		
				10110010				
			10110000	10110001	10110011			
				10110100				
			10010100	10010101	10110101			
			10010010	10010110	10110110			
		10010000	10010001	10010011	10010111	10110111		
				10111000				
			10011000	10011001	10111001			
			10001010	10011010	10111010			
		10001000	10001001	10001011	10011011	10111011		
			10001100	10011100	10111100			
		10000100	10000101	10001101	10011101	10111101		
		10000010	10000110	10001110	10011110	10111110		
	10000000	10000001	10000011	10000111	10001111	10011111	10111111	
				11001010				
			11001000	11001001	11001011			
				11001100				
			11000100	11000101	11001101			
			11000010	11000110	11001110			
		11000000	11000001	11000011	11000111	11001111		
				11010010				
			11010000	11010001	11010011			
				11010100				
			01010100	01010101	11010101			
			01010010	01010110	11010110			
		01010000	01010001	01010011	01010111	11010111		
				11011000				
			01011000	01011001	11011001			
			01001010	01011010	11011010			
		01001000	01001001	01001011	01011011	11011011		
			01001100	01011100	11011100			
		01000100	01000101	01001101	01011101	11011101		
		01000010	01000110	01001110	01011110	11011110		
	01000000	01000001	01000011	01000111	01001111	01011111	11011111	
				11100010				
			11100000	11100001	11100011			
				11100100				
			01100100	01100101	11100101			
			01100010	01100110	11100110			
		01100000	01100001	01100011	01100111	11100111		
				11101000				
			01101000	01101001	11101001			
			00101010	01101010	11101010			
		00101000	00101001	00101011	01101011	11101011		
			00101100	01101100	11101100			
		00100100	00100101	00101101	01101101	11101101		
		00100010	00100110	00101110	01101110	11101110		
	00100000	00100001	00100011	00100111	00101111	01101111	11101111	
				11110000				
			01110000	01110001	11110001			
			00110010	01110010	11110010			
		00110000	00110001	00110011	01110011	11110011		
			00110100	01110100	11110100			
		00010100	00010101	00110101	01110101	11110101		
		00010010	00010110	00110110	01110110	11110110		
	00010000	00010001	00010011	00010111	00110111	01110111	11110111	
			00111000	01111000	11111000			
		00011000	00011001	00111001	01111001	11111001		
		00001010	00011010	00111010	01111010	11111010		
	00001000	00001001	00001011	00011011	00111011	01111011	11111011	
		00001100	00011100	00111100	01111100	11111100		
	00000100	00000101	00001101	00011101	00111101	01111101	11111101	
	00000010	00000110	00001110	00011110	00111110	01111110	11111110	
00000000	00000001	00000011	00000111	00001111	00011111	00111111	01111111	11111111

A set C of bit strings is called a *clutter*, or an "antichain of subsets," if its bit strings are incomparable in the sense that $\sigma \not\subseteq \tau$ whenever σ and τ are distinct elements of C. A famous theorem of Emanuel Sperner [*Math. Zeitschrift* **27** (1928), 544–548] asserts that no clutter on $\{1,\ldots,n\}$ can have more than M_n elements; and the Christmas tree pattern provides a simple proof, because no clutter can contain more than one element of each row.

Indeed, the Christmas tree pattern can be used to show that much more is true. Let's note first that exactly $\binom{n}{k}-\binom{n}{k-1}$ rows of length $n+1-2k$ are present, for $0 \le k \le n/2$, because there are exactly $\binom{n}{k}$ elements in column k. For example, Table 4 has one row of length 9, namely the bottom row; it also has $\binom{8}{1}-\binom{8}{0}=7$ rows of length 7, $\binom{8}{2}-\binom{8}{1}=20$ rows of length 5, $\binom{8}{3}-\binom{8}{2}=28$ of length 3, and $\binom{8}{4}-\binom{8}{3}=14$ of length 1. Moreover, these numbers $\binom{n}{k}-\binom{n}{k-1}$ appear in the Catalan triangle (22), because they're equal to $C_{k(n-k)}$ according to Eq. (23).

Further study reveals that this Catalan connection is not simply a coincidence; nested parentheses are, in fact, the key to a deeper understanding of the Christmas tree pattern, because the theory of parentheses tells us where an arbitrary bit string fits into the array. Suppose we use the symbols (and) instead of 1 and 0, respectively. Any string of parentheses, nested or not, can be written uniquely in the form

$$\alpha_0\,)\,\ldots\,\alpha_{p-1}\,)\,\alpha_p\,(\,\alpha_{p+1}\,\ldots\,(\,\alpha_q \tag{37}$$

for some p and q with $0 \le p \le q$, where the substrings $\alpha_0, \ldots, \alpha_q$ are properly nested and possibly empty; exactly p of the right parentheses and $q-p$ of the left parentheses are "free" in the sense that they have no mate. For example, the string

$$\mathtt{)\,(\,(\,)\,)\,(\,)\,)\,(\,)\,)\,)\,)\,(\,(\,(\,(\,(\,(\,)\,(\,(\,)\,(\,)\,(\,(\,(\,)\,)} \tag{38}$$

has $p=5$, $q=12$, $\alpha_0=\epsilon$, $\alpha_1=\mathtt{(())()}$, $\alpha_2=\mathtt{()}$, $\alpha_3=\epsilon$, $\ldots$, $\alpha_{12}=\mathtt{(())}$. In general, the string (37) is part of a chain of length $q+1$,

$$\alpha_0\,)\ldots\alpha_{q-1}\,)\,\alpha_q,\quad \alpha_0\,)\ldots\alpha_{q-2}\,)\,\alpha_{q-1}\,(\,\alpha_q,\quad \ldots,\quad \alpha_0\,(\,\alpha_1\ldots(\,\alpha_q, \tag{39}$$

in which we start with q free)s and change them one by one into free (s. Every row of the Christmas tree pattern is obtained in exactly this manner, but using 1 and 0 instead of (and); for if the chain $\sigma_1 \ldots \sigma_s$ corresponds to the nested strings $\alpha_0, \ldots, \alpha_{s-1}$, its successor chains in (36) correspond respectively to $\alpha_0, \ldots, \alpha_{s-3}, \alpha_{s-2}\,(\,\alpha_{s-1}\,)$ and to $\alpha_0, \ldots, \alpha_{s-3}, \alpha_{s-2}, \alpha_{s-1}, \epsilon$. [See Curtis Greene and Daniel J. Kleitman, *J. Combinatorial Theory* **A20** (1976), 80–88.]

Notice furthermore that the rightmost elements in each row of the pattern— such as 10101010, 10101011, 10101100, 10101101, ..., 11111110, 11111111 in the case $n=8$—are in lexicographic order. Thus, for example, the fourteen rows of length 1 in Table 4 correspond precisely to the fourteen strings of nested parentheses in Table 1. This observation makes it easy to generate the rows of Table 8 sequentially from bottom to top, with a method analogous to Algorithm P; see exercise 77.

Let $f(x_1,\ldots,x_n)$ be any monotone Boolean function of n variables. If $\sigma = a_1\ldots a_n$ is any bit string of length n, we can write $f(\sigma) = f(a_1,\ldots,a_n)$ for convenience. Any row $\sigma_1 \;\ldots\; \sigma_s$ of the Christmas tree pattern forms a chain, so we have

$$0 \;\le\; f(\sigma_1) \;\le\; \cdots \;\le\; f(\sigma_s) \;\le\; 1. \tag{40}$$

In other words, there is an index t such that $f(\sigma_j) = 0$ for $j < t$ and $f(\sigma_j) = 1$ for $j \ge t$; we will know the value of $f(\sigma)$ for all 2^n bit strings σ if we know the indices t for each row of the pattern.

Georges Hansel [*Comptes Rendus Acad. Sci.* (A) **262** (Paris, 1966), 1088–1090] noticed that the Christmas tree pattern has another important property: If σ_{j-1}, σ_j, and σ_{j+1} are three consecutive entries of any row, the bit string

$$\sigma'_j \;=\; \sigma_{j-1} \oplus \sigma_j \oplus \sigma_{j+1} \tag{41}$$

lies in a *previous* row. In fact, σ'_j lies in the same column as σ_j, and it satisfies

$$\sigma_{j-1} \;\subseteq\; \sigma'_j \;\subseteq\; \sigma_{j+1}; \tag{42}$$

it is called the relative complement of σ_j in the interval $(\sigma_{j-1}\,..\,\sigma_{j+1})$. Hansel's observation is easy to prove by induction, because of the recursive rule (36) that defines the Christmas tree pattern. He used it to show that we can deduce the values of $f(\sigma)$ for all σ by actually evaluating the function at relatively few well-chosen places; for if we know the value of $f(\sigma'_j)$, we will know either $f(\sigma_{j-1})$ or $f(\sigma_{j+1})$ because of relation (42).

Algorithm H (*Learning a monotone Boolean function*). Let $f(x_1,\ldots,x_n)$ be a Boolean function that is nondecreasing in each Boolean variable, but otherwise unknown. Given a bit string σ of length n, let $r(\sigma)$ be the number of the row in which σ appears in the Christmas tree pattern, where $1 \le r(\sigma) \le M_n$. If $1 \le m \le M_n$, let $s(m)$ be the number of bit strings in row m; also let $\chi(m,k)$ be the bit string in column k of that row, for $(n+1-s(m))/2 \le k \le (n-1+s(m))/2$. This algorithm determines the sequence of threshold values $t(1)$, $t(2)$, …, $t(M_n)$ such that

$$f(\sigma) = 1 \quad\iff\quad \nu(\sigma) \ge t\bigl(r(\sigma)\bigr), \tag{43}$$

by evaluating f at no more than two points per row.

H1. [Loop on m.] Perform steps H2 through H4 for $m = 1$, …, M_n; then stop.

H2. [Begin row m.] Set $a \leftarrow (n+1-s(m))/2$ and $z \leftarrow (n-1+s(m))/2$.

H3. [Do a binary search.] If $z \le a+1$, go to H4. Otherwise set $k \leftarrow \lfloor (a+z)/2 \rfloor$, and

$$\sigma \;\leftarrow\; \chi(m,k-1) \oplus \chi(m,k) \oplus \chi(m,k+1). \tag{44}$$

If $k \ge t\bigl(r(\sigma)\bigr)$, set $z \leftarrow k$; otherwise set $a \leftarrow k$. Repeat step H3.

H4. [Evaluate.] If $f(\chi(m,a)) = 1$, set $t(m) \leftarrow a$; otherwise, if $a = z$, set $t(m) \leftarrow a+1$; otherwise set $t(m) \leftarrow z+1-f(\chi(m,z))$. ▮

Hansel's algorithm is *optimum*, in the sense that it evaluates f at the fewest possible points in the worst case. For if f happens to be the threshold function

$$f(\sigma) \;=\; \bigl[\nu(\sigma) > n/2\bigr], \tag{45}$$

any valid algorithm that learns f on the first m rows of the Christmas tree pattern must evaluate $f(\sigma)$ in column $\lfloor n/2\rfloor$ of each row, and in column $\lfloor n/2\rfloor+1$ of each row that has size greater than 1. Otherwise we could not distinguish f from a function that differs from it only at an unexamined point. [See V. K. Korobkov, *Problemy Kibernetiki* **13** (1965), 5–28, Theorem 5.]

Oriented trees and forests. Let's turn now to another kind of tree, in which the parent-child relationship is important but the order of children in each family is not. An *oriented forest* of n nodes can be defined by a sequence of pointers $p_1 \ldots p_n$, where p_j is the parent of node j (or $p_j = 0$ if j is a root); the directed graph on vertices $\{0, 1, \ldots, n\}$ with arcs $\{j \to p_j \mid 1 \le j \le n\}$ will have no oriented cycles. An *oriented tree* is an oriented forest with exactly one root. (See Section 2.3.4.2.) Every n-node oriented forest is equivalent to an $(n+1)$-node oriented tree, because the root of that tree can be regarded as the parent of all the roots of the forest. We saw in Section 2.3.4.4 that there are A_n oriented trees with n nodes, where the first few values are

$$\begin{array}{rcccccccccccccc} n = & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ A_n = & 1 & 1 & 2 & 4 & 9 & 20 & 48 & 115 & 286 & 719 & 1842 & 4766 & 12486 & 32973 \end{array}\;; \tag{46}$$

asymptotically, $A_n = c\alpha^n n^{-3/2} + O(\alpha^n n^{-5/2})$ where $\alpha \approx 2.9558$ and $c \approx 0.4399$. Thus, for example, only 9 of the 14 forests in Table 1 are distinct when we ignore the horizontal left-to-right ordering and consider only the vertical orientation.

Every oriented forest corresponds to a unique ordered forest if we sort the members of each family appropriately, using an ordering on trees introduced by H. I. Scoins [*Machine Intelligence* **3** (1968), 43–60]: Recall from (11) that ordered forests can be characterized by their level codes $c_1 \ldots c_n$, where node j in preorder appears on level c_j. An ordered forest is called *canonical* if the level code sequences for the subtrees in each family are in nonincreasing lexicographic order. For example, the canonical forests in Table 1 are those whose level codes $c_1c_2c_3c_4$ are 0000, 0100, 0101, 0110, 0111, 0120, 0121, 0122, and 0123. The level sequence 0112 is not canonical, because the subtrees of the root have respective level codes 1 and 12; the string 1 is lexicographically less than 12. We can readily verify by induction that *the canonical level codes are lexicographically largest, among all ways of reordering the subtrees of a given oriented forest.*

T. Beyer and S. M. Hedetniemi [*SICOMP* **9** (1980), 706–712] noticed that there is a remarkably simple way to generate oriented forests if we visit them in *decreasing* lexicographic order of the canonical level codes. Suppose $c_1 \ldots c_n$ is canonical, where $c_k > 0$ and $c_{k+1} = \cdots = c_n = 0$. The next smallest sequence is obtained by decreasing c_k, then increasing $c_{k+1} \ldots c_n$ to the largest levels consistent with canonicity; and those levels are easy to compute. For if $j = p_k$ is the parent of node k, we have $c_j = c_k - 1 < c_l$ for $j < l \le k$, hence the levels $c_j \ldots c_k$

represent the subtree currently rooted at node j. To get the largest sequence of levels less than $c_1 \ldots c_n$ we therefore replace $c_k \ldots c_n$ by the first $n+1-k$ elements of the infinite sequence $(c_j \ldots c_{k-1})^\infty = c_j \ldots c_{k-1} c_j \ldots c_{k-1} c_j \ldots$. (The effect is to remove k from its current position as the rightmost child of j, then to append new subtrees that are siblings of j, by cloning j and its descendants as often as possible. This cloning process may terminate in the midst of the sequence $c_j \ldots c_{k-1}$, but that causes no difficulty because every prefix of a canonical level sequence is canonical.) For example, to obtain the successor of any sequence of canonical codes that ends with 23443433000000000, we replace the final 3000000000 by 2344343234.

Algorithm O (*Oriented forests*). This algorithm generates all oriented forests on n nodes, by visiting all canonical n-node forests in decreasing lexicographic order of their level codes $c_1 \ldots c_n$. The level codes are not computed explicitly, however; each canonical forest is represented directly by its sequence of parent pointers $p_1 \ldots p_n$, in preorder of the nodes. To generate all oriented trees on $n+1$ nodes, we can imagine that node 0 is the root.

O1. [Initialize.] Set $p_k \leftarrow k-1$ for $0 \le k \le n$. (In particular, this step makes p_0 nonzero, for use in termination testing; see step O4.)

O2. [Visit.] Visit the forest represented by parent pointers $p_1 \ldots p_n$.

O3. [Easy case?] If $p_n > 0$, set $p_n \leftarrow p_{p_n}$ and return to step O2.

O4. [Find j and k.] Find the largest $k < n$ such that $p_k \ne 0$. Terminate the algorithm if $k = 0$; otherwise set $j \leftarrow p_k$ and $d \leftarrow k - j$.

O5. [Clone.] If $p_{k-d} = p_j$, set $p_k \leftarrow p_j$; otherwise set $p_k \leftarrow p_{k-d} + d$. Return to step O2 if $k = n$; otherwise set $k \leftarrow k+1$ and repeat this step. ▮

As in other algorithms we've been seeing, the loops in steps O4 and O5 tend to be quite short; see exercise 88. Exercise 90 proves that slight changes to this algorithm suffice to generate all arrangements of edges that form *free* trees.

Spanning trees. Now let's consider the minimal subgraphs that "span" a given graph. If G is a connected graph on n vertices, the *spanning trees* of G are the subsets of $n-1$ edges that contain no cycles; equivalently, they are the subsets of edges that form a free tree connecting all the vertices. Spanning trees are important in many applications, especially in the study of networks, so the problem of generating all spanning trees has been treated by many authors. In fact, systematic ways to list them all were developed early in the 20th century by Wilhelm Feussner [*Annalen der Physik* (4) **9** (1902), 1304–1329], long before anybody thought about generating other kinds of trees.

In the following discussion we will allow graphs to have any number of edges between two vertices; but we disallow loops from a vertex to itself, because self-loops cannot be part of a tree. Feussner's basic idea was very simple, yet eminently suited for calculation: If e is any edge of G, a spanning tree either contains e or it doesn't. Suppose e joins vertex u to vertex v, and suppose it is part of a spanning tree; then the other $n-2$ edges of that tree span the graph

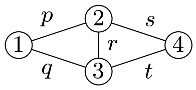
$G\,/\,e$ that we obtain by regarding u and v as identical. In other words, the spanning trees that contain e are essentially the same as the spanning trees of the contracted graph $G\,/\,e$ that results when we shrink e down to a single point. On the other hand the spanning trees that do *not* contain e are spanning trees of the reduced graph $G \setminus e$ that results when we eliminate edge e. Symbolically, therefore, the set $S(G)$ of all spanning trees of G satisfies

$$S(G) \;=\; e\,S(G\,/\,e) \;\cup\; S(G \setminus e). \tag{47}$$

Malcolm J. Smith, in his Master's thesis at the University of Victoria (1999), introduced a nice way to carry out the recursion (47) by finding all spanning trees in a "revolving-door Gray code" order: Each tree in his scheme is obtained from its predecessor by simply removing one edge and substituting another. Such orderings are not difficult to find, but the trick is to do the job efficiently.

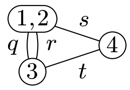
The basic idea of Smith's algorithm is to generate $S(G)$ in such a way that the first spanning tree includes a given *near tree*, namely a set of $n-2$ edges containing no cycle. This task is trivial if $n=2$; we simply list all the edges. If $n>2$ and if the given near tree is $\{e_1,\ldots,e_{n-2}\}$, we proceed as follows: Assume that G is connected; otherwise there are no spanning trees. Form $G\,/\,e_1$ and append e_1 to each of its spanning trees, beginning with one that contains $\{e_2,\ldots,e_{n-2}\}$; notice that $\{e_2,\ldots,e_{n-2}\}$ is a near tree of G/e_1, so this recursion makes sense. If the last spanning tree found in this way for $G\,/\,e_1$ is $f_1\ldots f_{n-2}$, complete the task by listing all spanning trees for $G \setminus e_1$, beginning with one that contains the near tree $\{f_1,\ldots,f_{n-2}\}$.

For example, suppose G is the graph

$G \;=$ 

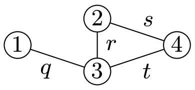
(48)

with four vertices and five edges $\{p,q,r,s,t\}$. Starting with the near tree $\{p,q\}$, Smith's procedure first forms the contracted graph

$G\,/\,p \;=$ 

(49)

and lists its spanning trees, beginning with one that contains q. This list might be qs, qt, ts, tr, rs; thus the trees pqs, pqt, pts, ptr, and prs span G. The remaining task is to list the spanning trees of

$G \setminus p \;=$  ,

(50)

starting with one that contains $\{r,s\}$; they are rsq, rqt, qts.

A detailed implementation of Smith's algorithm turns out to be quite instructive. As usual we represent the graph by letting two arcs $u \to v$ and $v \to u$ correspond to each edge $u \mathbin{—} v$, and we maintain lists of "arc nodes" to represent the arcs that leave each vertex. We'll need to shrink and unshrink the graph's edges, so we will make these lists doubly linked. If a points to an arc node that represents $u \to v$, then

$a \oplus 1$ points to the "mate" of a, which represents $v \to u$;
t_a is the "tip" of a, namely v (hence $t_{a\oplus 1} = u$);
i_a is an optional name that identifies this edge (and equals $i_{a\oplus 1}$);
n_a points to the next element of u's arc list;
p_a points to the previous element of u's arc list;
and l_a is a link used for undeleting arcs as explained below.

The vertices are represented by integers $\{1,\ldots,n\}$; and arc number $v-1$ is a header node for vertex v's doubly linked arc list. A header node a is recognizable by the fact that its tip, t_a, is 0. We let d_v be the degree of vertex v. Thus, for example, the graph (48) might be represented by $(d_1, d_2, d_3, d_4) = (2, 3, 3, 2)$ and by the following fourteen nodes of arc data:

$a =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$t_a =$	0	0	0	0	1	2	1	3	2	3	2	4	3	4
$i_a =$					p	p	q	q	r	r	s	s	t	t
$n_a =$	5	4	6	10	9	7	8	0	13	11	12	1	3	2
$p_a =$	7	11	13	12	1	0	2	5	6	4	3	9	10	8

The implicit recursion of Smith's algorithm can be controlled conveniently by using an array of arc pointers $a_1 \ldots a_{n-1}$. At level l of the process, arcs $a_1 \ldots a_{l-1}$ denote edges that have been included in the current spanning tree; a_l is ignored; and arcs $a_{l+1} \ldots a_{n-1}$ denote edges of a near tree on the contracted graph $(\ldots(G/a_1)\ldots)/a_{l-1}$ that should be part of the next spanning tree visited.

There's also another array of arc pointers $s_1 \ldots s_{n-2}$, representing stacks of arcs that have been temporarily removed from the current graph. The top element of the stack for level l is s_l, and each arc a links to its successor, l_a (which is 0 at the bottom of the stack).

An edge whose removal would disconnect a connected graph is called a *bridge*. One of the key points in the algorithm that follows is the fact that we want to keep the current graph connected; therefore we don't set $G \leftarrow G \setminus e$ when e is a bridge.

Algorithm S (*All spanning trees*). Given a connected graph represented with the data structures explained above, this algorithm visits all of its spanning trees.

A technique called "dancing links," which we will discuss extensively in Section 7.2.2.1, is used here to remove and restore items from and to doubly linked lists. The abbreviation "delete(a)" in the steps below is shorthand for the pair of operations

$$n_{p_a} \leftarrow n_a, \quad p_{n_a} \leftarrow p_a; \tag{51}$$

similarly, "undelete(a)" stands for

$$p_{n_a} \leftarrow a, \quad n_{p_a} \leftarrow a. \tag{52}$$

S1. [Initialize.] Set $a_1 \ldots a_{n-1}$ to a spanning tree of the graph. (See exercise 94.) Also set $x \leftarrow 0$, $l \leftarrow 1$, and $s_1 \leftarrow 0$. If $n = 2$, set $v \leftarrow 1$, $e \leftarrow n_0$, and go to S5.

S2. [Enter level l.] Set $e \leftarrow a_{l+1}$, $u \leftarrow t_e$, and $v \leftarrow t_{e\oplus 1}$. If $d_u > d_v$, interchange $v \leftrightarrow u$ and set $e \leftarrow e \oplus 1$.

S3. [Shrink e.] (Now we will make u identical to v by inserting u's adjacency list into v's. We also must delete all former edges between u and v, including e itself, because such edges would otherwise become loops. Deleted edges are linked together so that we can restore them later in step S7.) Set $k \leftarrow d_u + d_v$, $f \leftarrow n_{u-1}$, and $g \leftarrow 0$. While $t_f \neq 0$, do the following: If $t_f = v$, delete(f), delete($f \oplus 1$), and set $k \leftarrow k - 2$, $l_f \leftarrow g$, $g \leftarrow f$; otherwise set $t_{f\oplus 1} \leftarrow v$. Then set $f \leftarrow n_f$ and repeat these operations until $t_f = 0$. Finally set $l_e \leftarrow g$, $d_v \leftarrow k$, $g \leftarrow v - 1$, $n_{p_f} \leftarrow n_g$, $p_{n_g} \leftarrow p_f$, $p_{n_f} \leftarrow g$, $n_g \leftarrow n_f$, and $a_l \leftarrow e$.

S4. [Advance l.] Set $l \leftarrow l + 1$. If $l < n - 1$, set $s_l \leftarrow 0$ and return to S2. Otherwise set $e \leftarrow n_{v-1}$.

S5. [Visit.] (The current graph now has only two vertices, one of which is v.) Set $a_{n-1} \leftarrow e$ and visit the spanning tree $a_1 \ldots a_{n-1}$. (If $x = 0$, this is the first spanning tree to be visited; otherwise it differs from its predecessor by deleting x and inserting e.) Set $x \leftarrow e$ and $e \leftarrow n_e$. Repeat step S5 if $t_e \neq 0$.

S6. [Decrease l.] Set $l \leftarrow l - 1$. Terminate the algorithm if $l = 0$; otherwise set $e \leftarrow a_l$, $u \leftarrow t_e$, and $v \leftarrow t_{e\oplus 1}$.

S7. [Unshrink e.] Set $f \leftarrow u - 1$, $g \leftarrow v - 1$, $n_g \leftarrow n_{p_f}$, $p_{n_g} \leftarrow g$, $n_{p_f} \leftarrow f$, $p_{n_f} \leftarrow f$, and $f \leftarrow p_f$. While $t_f \neq 0$, set $t_{f\oplus 1} \leftarrow u$ and $f \leftarrow p_f$. Then set $f \leftarrow l_e$, $k \leftarrow d_v$; while $f \neq 0$ set $k \leftarrow k + 2$, undelete($f \oplus 1$), undelete(f), and set $f \leftarrow l_f$. Finally set $d_v \leftarrow k - d_u$.

S8. [Test for bridge.] If e is a bridge, go to S9. (See exercise 95 for one way to perform this test.) Otherwise set $x \leftarrow e$, $l_e \leftarrow s_l$, $s_l \leftarrow e$; delete(e) and delete($e \oplus 1$). Set $d_u \leftarrow d_u - 1$, $d_v \leftarrow d_v - 1$, and go to S2.

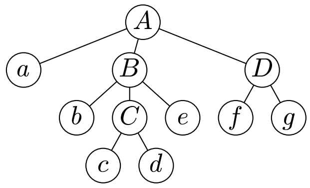
S9. [Undo level l deletions.] Set $e \leftarrow s_l$. While $e \neq 0$, set $u \leftarrow t_e$, $v \leftarrow t_{e\oplus 1}$, $d_u \leftarrow d_u + 1$, $d_v \leftarrow d_v + 1$, undelete($e \oplus 1$), undelete(e), and $e \leftarrow l_e$. Return to S6. ▌

The reader is encouraged to play through the steps of this algorithm on a small graph such as (48). Notice that a subtle case arises in steps S3 and S7, if u's adjacency list happens to become empty. Notice also that several shortcuts would be possible, at the expense of a more complicated algorithm; we will discuss such improvements later in this section.

***Series-parallel graphs.** The task of finding all spanning trees becomes especially simple when the given graph has a serial and/or parallel decomposition. A *series-parallel graph between s and t* is a graph G with two designated vertices, s and t, whose edges can be built up recursively as follows: Either G consists of a single edge, $s — t$; or G is a *serial superedge* consisting of $k \geq 2$ series-parallel subgraphs G_j between s_j and t_j, joined in series with $s = s_1$ and $t_j = s_{j+1}$ for

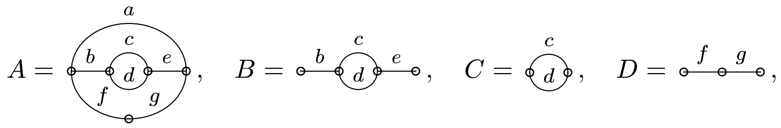
$1 \le j < k$ and $t_k = t$; or G is a *parallel superedge* consisting of $k \ge 2$ series-parallel subgraphs G_j between s and t joined in parallel. This decomposition is essentially unique, given s and t, if we require that the subgraphs G_j for serial superedges are not themselves serial superedges, and that the subgraphs G_j for parallel superedges are not themselves parallel.

Any series-parallel graph can be represented conveniently as a tree, with no nodes of degree 1. The leaf nodes of this tree represent edges, and the branch nodes represent superedges, alternating between serial and parallel from level to level. For example, the tree



(53)

corresponds to the series-parallel graphs and subgraphs



(54)

if the top node A is taken to be parallel. Edges are named in (54), but not vertices, because edges are of prime importance with respect to spanning trees.

Let's say that a *near tree* of a series-parallel graph between s and t is a set of $n-2$ cycle-free edges that do not connect s to t. The spanning trees and near trees of a series-parallel graph are easy to describe recursively, as follows: (1) A spanning tree of a serial superedge corresponds to spanning trees of all its principal subgraphs G_j; a near tree corresponds to spanning trees in all but one of the G_j, and a near tree in the other. (2) A near tree of a parallel superedge corresponds to near trees of all its principal subgraphs G_j; a spanning tree corresponds to near trees in all but one of the G_j, and a spanning tree in the other.

Rules (1) and (2) suggest the following data structures for listing the spanning trees and/or near trees of series-parallel graphs. Let p point to a node in a tree like (53). Then we define

$t_p = 1$ for serial superedges, 0 otherwise (the "type" of p);
$v_p = 1$ if we have a spanning tree for p, 0 if we have a near tree;
$l_p =$ pointer to p's leftmost child, or 0 if p is a leaf;
$r_p =$ pointer to p's right sibling, wrapping around cyclically;
$d_p =$ pointer to a designated child of p, or 0 if p is a leaf.

If q points to the rightmost child of p, its "right sibling" r_q equals l_p. And if q points to *any* child of p, rules (1) and (2) state that

$$v_q = \begin{cases} v_p, & \text{if } q = d_p; \\ t_p, & \text{if } q \ne d_p. \end{cases} \tag{55}$$

(For example, if p is a branch node that represents a serial superedge, we must have $v_q = 1$ for all but one of p's children; the only exception is the designated child d_p. Thus we must have a spanning tree for all of the subgraphs that were joined serially to form p, except for one designated subgraph in the case that we have a near tree for p.)

Given any setting of the designated-child pointers d_p, and given any value 0 or 1 for v_p at the root of the tree, Eq. (55) tells us how to propagate values down to all of the leaves. For example, if we set $v_A \leftarrow 1$ in the tree (53), and if we designate the leftmost child of each branch node (so that $d_A = a$, $d_B = b$, $d_C = c$, and $d_D = f$), we find successively

$$v_a = 1,\ v_B = 0,\ v_b = 0,\ v_C = 1,\ v_c = 1,\ v_d = 0,\ v_e = 1,\ v_D = 0,\ v_f = 0,\ v_g = 1. \qquad (56)$$

A leaf node q is present in the spanning tree if and only if $v_q = 1$; hence (56) specifies the spanning tree $aceg$ of the series-parallel graph A in (54).

For convenience, let's say that the *configs* of p are its spanning trees if $v_p = 1$, its near trees if $v_p = 0$. We would like to generate all configs of the root. A branch node p is called "easy" if $v_p = t_p$; that is, a serial node is easy if its configs are spanning trees, and a parallel node is easy if its configs are near trees. If p is easy, its configs are the Cartesian product of the configs of its children, namely all k-tuples of the children's configs, varying independently; the designated child d_p is immaterial in the easy case. But if p is uneasy, its configs are the union of such Cartesian k-tuples, taken over all possible choices of d_p.

As luck would have it, easy nodes are relatively rare: At most one child of an uneasy node (namely the designated child) can be easy, and all children of an easy node are uneasy unless they are leaves.

Even so, the tree representation of a series-parallel graph makes the recursive generation of all its spanning trees and/or near trees quite straightforward and efficient. The operations of Algorithm S — shrinking and unshrinking, deleting and undeleting, bridge detection — are not needed when we deal with series-parallel graphs. Furthermore, exercise 99 shows that there is a pleasant way to obtain the spanning trees or near trees in a revolving-door Gray code order, by using focus pointers as in several algorithms that we've seen earlier.

***Refinements of Algorithm S.** Although Algorithm S provides us with a simple and reasonably effective way to visit all spanning trees of a general graph, its author Malcolm Smith realized that the properties of series-parallel graphs can be used to make it even better. For example, if a graph has two or more edges that run between the same vertices u and v, we can combine them into a superedge; the spanning trees of the original graph can then be obtained readily from those of the simpler, reduced graph. And if a graph has a vertex v of degree 2, so that the only edges touching v are $u — v$ and $v — w$, we can eliminate v and replace those edges by a single superedge between u and w. Furthermore, any vertex of degree 1 can effectively be eliminated, together with its adjacent edge, by simply including that edge in every spanning tree.

After the reductions in the preceding paragraph have been applied to a given graph G, we obtain a reduced graph $\hat{G}$ having no parallel edges and no vertices of degrees 1 or 2, together with a set of $m \geq 0$ series-parallel graphs $S_1, \ldots, S_m$, representing edges (or superedges) that must be included in all spanning trees of G. Every remaining edge $u — v$ of $\hat{G}$ corresponds, in fact, to a series-parallel graph S_{uv} between vertices u and v. *The spanning trees of G are then obtained as the union, taken over all spanning trees T of $\hat{G}$, of the Cartesian product of the spanning trees of $S_1, \ldots, S_m$ and the spanning trees of all S_{uv} for edges $u — v$ in T, together with the near trees of all S_{uv} for edges $u — v$ that are in $\hat{G}$ but not in T.* And all spanning trees T of $\hat{G}$ can be obtained by using the strategy of Algorithm S.

In fact, when Algorithm S is extended in this way, its operations of replacing the current graph G by G / e or $G \setminus e$ typically trigger further reductions, as new parallel edges appear or as the degree of a vertex drops below 3. Therefore it turns out that the "stopping state" of the implicit recursion in Algorithm S, namely the case when only two vertices are left (step S5), never actually arises: A reduced graph $\hat{G}$ either has only a single vertex and no edges, or it has at least four vertices and six edges.

The resulting algorithm retains the desirable revolving-door property of Algorithm S, and it is quite pretty (although about four times as long as the original); see exercise 100. Smith proved that it has the best possible asymptotic running time: If G has n vertices, m edges, and N spanning trees, the algorithm visits them all in $O(m + n + N)$ steps.

The performance of Algorithm S and of its souped-up version Algorithm S$'$ can best be appreciated by considering the number of memory accesses that those algorithms actually make when they generate the spanning trees of typical graphs, as shown in Table 5. The bottom line of that table corresponds to the graph *plane_miles*$(16, 0, 0, 1, 0, 0, 0)$ from the Stanford GraphBase, which serves as an "organic" antidote to the purely mathematical examples on the previous lines. The random multigraph on the penultimate line, also from the Stanford GraphBase, can be described more precisely by its official name *random_graph*$(16, 37, 1, 0, 0, 0, 0, 0, 0, 0)$. Although the 4×4 torus is isomorphic to the 4-cube (see exercise 7.2.1.1–17), those isomorphic graphs yield slightly different running times because their vertices and edges are encountered differently when the algorithms are run.

In general we can say that Algorithm S is not too bad on small examples, except when the graph is quite sparse; but Algorithm S$'$ begins to shine when many spanning trees are present. Once Algorithm S$'$ gets warmed up, it tends to crank out a new tree after every 18 or 19 mems go by.

Table 5 also indicates that a mathematically-defined graph often has a surprisingly "round" number of spanning trees. For example, D. M. Cvetković [*Srpska Akademija Nauka, Matematicheski Institut* **11** (Belgrade: 1971), 135–141] discovered, among other things, that the n-cube has exactly

$$2^{2^n-n-1}\, 1^{\binom{n}{1}}\, 2^{\binom{n}{2}} \ldots n^{\binom{n}{n}} \tag{57}$$

of them. Exercises 104–109 explore some of the reasons why that happens.

Table 5

RUNNING TIME IN MEMS NEEDED TO GENERATE ALL SPANNING TREES

	m	n	N	Algorithm S	Algorithm S′	μ per tree	
path P_{10}	9	10	1	794 μ	473 μ	794.0	473.0
path P_{100}	99	100	1	9,974 μ	5,063 μ	9974.0	5063.0
cycle C_{10}	10	10	10	3,480 μ	998 μ	348.0	99.8
cycle C_{100}	100	100	100	355,605 μ	10,538 μ	3556.1	105.4
complete graph K_4	6	4	16	1,213 μ	1,336 μ	75.8	83.5
complete graph K_{10}	45	10	100,000,000	3,759.58 Mμ	1,860.95 Mμ	37.6	18.6
complete bigraph $K_{5,5}$	25	10	390,625	23.43 Mμ	8.88 Mμ	60.0	22.7
4×4 grid $P_4\times P_4$	24	16	100,352	12.01 Mμ	1.87 Mμ	119.7	18.7
5×5 grid $P_5\times P_5$	40	25	557,568,000	54.68 Gμ	10.20 Gμ	98.1	18.3
4×4 cylinder $P_4\times C_4$	28	16	2,558,976	230.96 Mμ	49.09 Kμ	90.3	19.2
5×5 cylinder $P_5\times C_5$	45	25	38,720,000,000	3,165.31 Gμ	711.69 Gμ	81.7	18.4
4×4 torus $C_4\times C_4$	32	16	42,467,328	3,168.15 Mμ	823.08 Mμ	74.6	19.4
4-cube $P_2\times P_2\times P_2\times P_2$	32	16	42,467,328	3,168.16 Mμ	823.38 Mμ	74.7	19.4
random multigraph	37	16	59,933,756	3,818.19 Mμ	995.91 Mμ	63.7	16.6
16 cities	37	16	179,678,881	11,772.11 Mμ	3,267.43 Mμ	65.5	18.2

A general quasi-Gray code. Let's close this section by discussing something completely different, yet still related to trees. Consider the following hybrid variants of the two standard ways to traverse a nonempty forest:

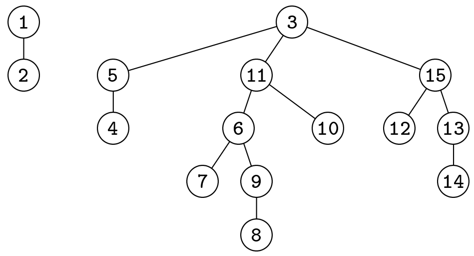
Prepostorder traversal

Visit the root of the first tree
Traverse the subtrees of the first tree, in postpreorder
Traverse the remaining trees, in prepostorder

Postpreorder traversal

Traverse the subtrees of the first tree, in prepostorder
Visit the root of the first tree
Traverse the remaining trees, in postpreorder

In the first case, every tree of the forest is traversed in prepostorder, with its root first; but the subtrees of those roots are traversed in postpreorder, with roots coming last. The second variant is similar but with 'pre' and 'post' interchanged. And in general, prepostorder visits roots first on every even-numbered level of the forest, but visits them last on the odd-numbered levels. For example, the forest in (2) becomes



(58)

when we label its nodes in prepostorder.

Prepostorder and postpreorder are not merely curiosities; they're actually useful. The reason is that adjacent nodes, in either of these orders, are always near each other in the forest. For example, nodes k and $k+1$ are adjacent in (58) for $k = 1, 4, 6, 8, 10, 13$; they are separated by only one node when $k = 3, 12, 14$; and they're three steps apart when $k = 2, 5, 7, 9, 11$ (if we imagine an invisible super-parent at the top of the forest). A moment's thought proves inductively that at most two nodes can possibly intervene between prepostorder neighbors or postpreorder neighbors — because postpreorder(F) always begins with the root of the first tree or its leftmost child, and prepostorder(F) always ends with the root of the last tree or its rightmost child.

Suppose we want to generate all combinatorial patterns of some kind, and we want to visit them in a Gray-code-like manner so that consecutive patterns are always "close" to each other. We can form, at least conceptually, the graph of all possible patterns p, with edges $p \longrightarrow q$ for all pairs of patterns that are close to each other. The following theorem, due to Milan Sekanina [*Spisy Přírodovědecké Fakulty University v Brně*, No. 412 (1960), 137–142], proves that a pretty good Gray code is always possible, provided only that we can get from any pattern to any other in a sequence of short steps:

Theorem S. The vertices of any connected graph can be listed in a cyclic order $(v_0, v_1, \ldots, v_{n-1})$ so that the distance between v_k and $v_{(k+1) \bmod n}$ is at most 3, for $0 \le k < n$.

Proof. Find a spanning tree in the graph, and traverse it in prepostorder. ▮

Graph theorists traditionally say that the kth power of a graph G is the graph G^k whose vertices are those of G, with $u \longrightarrow v$ in G^k if and only if there's a path of length k or less from u to v in G. Thus they can state Theorem S much more succinctly, when $n > 2$: *The cube of a connected graph is Hamiltonian.*

Prepostorder traversal is also useful when we want to visit the nodes of a tree in loopless fashion, with a bounded number of steps between stops:

Algorithm Q (*Prepostorder successor in a triply linked forest*). If `P` points to a node in a forest represented by links `PARENT`, `CHILD`, and `SIB`, corresponding to each node's parent, leftmost child, and right sibling, this algorithm computes `P`'s successor node, `Q`, in prepostorder. We assume that we know the level `L` at which `P` appears in the forest; the value of `L` is updated to be the level of `Q`. If `P` happens to be the final node in prepostorder, the algorithm sets $\mathtt{Q} \leftarrow \Lambda$ and $\mathtt{L} \leftarrow -1$.

Q1. [Pre or post?] If `L` is even, go to step Q4.

Q2. [Continue postpreorder.] Set $\mathtt{Q} \leftarrow \mathtt{SIB(P)}$. Go to Q6 if $\mathtt{Q} \ne \Lambda$.

Q3. [Move up.] Set $\mathtt{P} \leftarrow \mathtt{PARENT(P)}$ and $\mathtt{L} \leftarrow \mathtt{L} - 1$. Go to Q7.

Q4. [Continue prepostorder.] If $\mathtt{CHILD(P)} = \Lambda$, go to Q7.

Q5. [Move down.] Set $\mathtt{Q} \leftarrow \mathtt{CHILD(P)}$ and $\mathtt{L} \leftarrow \mathtt{L} + 1$.

Q6. [Move down if possible.] If $\mathtt{CHILD(Q)} \ne \Lambda$, set $\mathtt{Q} \leftarrow \mathtt{CHILD(Q)}$ and $\mathtt{L} \leftarrow \mathtt{L}+1$. Terminate the algorithm.

Q7. [Move right or up.] If `SIB(P)` $\neq \Lambda$, set `Q` $\leftarrow$ `SIB(P)`; otherwise set `Q` $\leftarrow$ `PARENT(P)` and `L` $\leftarrow$ `L` $- 1$. Terminate the algorithm. ▮

Notice that, as in Algorithm 2.4C, the link `PARENT(P)` is examined only if `SIB(P)` $= \Lambda$. A complete traversal is really a worm walk around the forest, like (3): The worm "sees" the nodes on even-numbered levels when it passes them on the left, and it sees the odd-level nodes when it passes them on the right.

EXERCISES

1. [*15*] If a worm crawls around the binary tree (4), how could it easily reconstruct the parentheses of (1)?

2. [*20*] (S. Zaks, 1980.) Modify Algorithm P so that it produces the combinations $z_1 z_2 \ldots z_n$ of (8) instead of the parenthesis strings $a_1 a_2 \ldots a_{2n}$.

▸ **3.** [*23*] Prove that (11) converts $z_1 z_2 \ldots z_n$ to the inversion table $c_1 c_2 \ldots c_n$.

4. [*20*] True or false: If the strings $a_1 \ldots a_{2n}$ are generated in lexicographic order, so are the corresponding strings $d_1 \ldots d_n$, $z_1 \ldots z_n$, $p_1 \ldots p_n$, and $c_1 \ldots c_n$.

5. [*15*] What tables $d_1 \ldots d_n$, $z_1 \ldots z_n$, $p_1 \ldots p_n$, and $c_1 \ldots c_n$ correspond to the nested parenthesis string (1)?

▸ **6.** [*20*] What *matching* corresponds to (1)? (See the final column of Table 1.)

7. [*16*] (a) What is the state of the string $a_1 a_2 \ldots a_{2n}$ when Algorithm P terminates? (b) What do the arrays $l_1 l_2 \ldots l_n$ and $r_1 r_2 \ldots r_n$ contain when Algorithm B terminates?

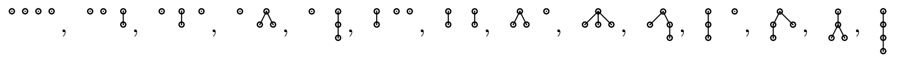
8. [*15*] What tables $l_1 \ldots l_n$, $r_1 \ldots r_n$, $e_1 \ldots e_n$, and $s_1 \ldots s_n$ correspond to the example forest (2)?

9. [*M20*] Show that the tables $c_1 \ldots c_n$ and $s_1 \ldots s_n$ are related by the law

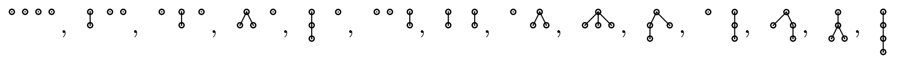
$$c_k = [s_1 \geq k-1] + [s_2 \geq k-2] + \cdots + [s_{k-1} \geq 1].$$

10. [*M20*] (*Worm walks.*) Given a string of nested parentheses $a_1 a_2 \ldots a_{2n}$, let w_j be the excess of left parentheses over right parentheses in $a_1 a_2 \ldots a_j$, for $0 \leq j \leq 2n$. Prove that $w_0 + w_1 + \cdots + w_{2n} = 2(c_1 + \cdots + c_n) + n$.

11. [*11*] If F is a forest, its *conjugate* F^R is obtained by left-to-right mirror reflection. For example, the fourteen forests in Table 1 are

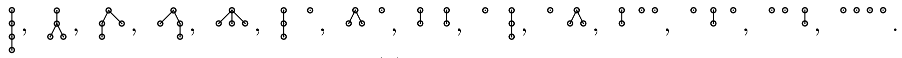


and their conjugates are respectively



as in the colex forests of Table 2. If F corresponds to the nested parentheses $a_1 a_2 \ldots a_{2n}$, what string of parentheses corresponds to F^R?

12. [*15*] If F is a forest, its *transpose* F^T is the forest whose binary tree is obtained by interchanging left and right links in the binary tree representing F. For example, the transposes of the fourteen forests in Table 1 are respectively



What is the transpose of the forest (2)?

13. [*20*] Continuing exercises 11 and 12, how do the preorder and postorder of a labeled forest F relate to the preorder and postorder of (a) F^R? (b) F^T?

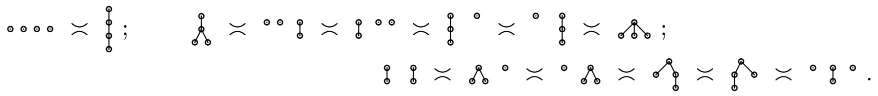
▸ **14.** [*21*] Find all labeled forests F such that $F^{RT} = F^{TR}$.

15. [*20*] Suppose B is the binary tree obtained from a forest F by linking each node to its left sibling and its rightmost child, as in exercise 2.3.2–5 and the last column of Table 2. Let F' be the forest that corresponds to B in the normal way, via left-child and right-sibling links. Prove that $F' = F^{RT}$, in the notation of exercises 11 and 12.

16. [*20*] If F and G are forests, let FG be the forest obtained by placing the trees of F to the left of the trees of G; also let $F \mid G = (G^T F^T)^T$. Give an intuitive explanation of the operator $\mid$, and prove that it is associative.

17. [*M46*] Characterize all *unlabeled* forests F such that $F^{RT} = F^{TR}$. (See exercise 14.)

18. [*30*] Two forests are said to be *cognate* if one can be obtained from the other by repeated operations of taking the conjugate and/or the transpose. The examples in exercises 11 and 12 show that all forests on 4 nodes belong to one of three cognate classes:



Study the set of all forests with 15 nodes. How many equivalence classes of cognate forests do they form? What is the largest class? What is the smallest class? What is the size of the class containing (2)?

19. [*28*] Let $F_1, F_2, \ldots, F_N$ be the sequence of unlabeled forests that correspond to the nested parentheses generated by Algorithm P, and let $G_1, G_2, \ldots, G_N$ be the sequence of unlabeled forests that correspond to the binary trees generated by Algorithm B. Prove that $G_k = F_k^{RTR}$, in the notation of exercises 11 and 12. (The forest F^{RTR} is called the *dual* of F; it is denoted by F^D in several exercises below.)

20. [*25*] Recall from Section 2.3 that the *degree* of a node in a tree is the number of children it has, and that an extended binary tree is characterized by the property that every node has degree either 0 or 2. In the extended binary tree (4), the sequence of node degrees is 2200222002220220002002202200000 in preorder; this string of 0s and 2s is identical to the sequence of parentheses in (1), except that each '(' has been replaced by 2, each ')' has been replaced by 0, and an additional 0 has been appended.

a) Prove that a sequence of nonnegative integers $b_1 b_2 \ldots b_N$ is the preorder degree sequence of a forest if and only if it satisfies the following property for $1 \le k \le N$:

$$b_1 + b_2 + \cdots + b_k + f \; > \; k \qquad \text{if and only if} \qquad k < N.$$

Here $f = N - b_1 - b_2 - \cdots - b_N$ is the number of trees in the forest.

b) Recall from exercise 2.3.4.5–6 that an *extended ternary tree* is characterized by the property that every node has degree 0 or 3; an extended ternary tree with n internal nodes has $2n + 1$ external nodes, hence $N = 3n + 1$ nodes altogether. Design an algorithm to generate all ternary trees with n internal nodes, by generating the associated sequences $b_1 b_2 \ldots b_N$ in lexicographic order.

▸ **21.** [*26*] (S. Zaks and D. Richards, 1979.) Continuing exercise 20, explain how to generate the preorder degree sequences of all forests that have $N = n_0 + \cdots + n_t$ nodes, with exactly n_j nodes of degree j. *Example:* When $n_0 = 4$, $n_1 = n_2 = n_3 = 1$, and $t = 3$, and the valid sequences $b_1 b_2 b_3 b_4 b_5 b_6 b_7$ are

1203000, 1230000, 1300200, 1302000, 1320000, 2013000, 2030010, 2030100, 2031000, 2103000, 2130000, 2300010, 2300100, 2301000, 2310000, 3001200, 3002010, 3002100, 3010200, 3012000, 3020010, 3020100, 3021000, 3100200, 3102000, 3120000, 3200010, 3200100, 3201000, 3210000.

▶ **22.** [*30*] (J. Korsh, 2004.) As an alternative to Algorithm B, show that binary trees can also be generated directly and efficiently in linked form if we produce them in *colex* order of the numbers $d_1 \ldots d_{n-1}$ defined in (9). (The actual values of $d_1 \ldots d_{n-1}$ should not be computed explicitly; but the links $l_1 \ldots l_n$ and $r_1 \ldots r_n$ should be manipulated in such a way that we get the binary trees corresponding successively to $d_1 d_2 \ldots d_{n-1} = 000\ldots0,\ 100\ldots0,\ 010\ldots0,\ 110\ldots0,\ 020\ldots0,\ 001\ldots0,\ \ldots,\ 000\ldots(n{-}1)$.)

▶ **23.** [*25*] (a) What is the last string visited by Algorithm N? (b) What is the last binary tree or forest visited by Algorithm L? *Hint:* See exercise 40 below.

24. [*22*] Using the notation of Table 3, what sequences $l_0 l_1 \ldots l_{15}$, $r_1 \ldots r_{15}$, $k_1 \ldots k_{15}$, $q_1 \ldots q_{15}$, and $u_1 \ldots u_{15}$ correspond to the binary tree (4) and the forest (2)?

▶ **25.** [*30*] (*Pruning and grafting.*) Representing binary trees as in Algorithm B, design an algorithm that visits all link tables $l_1 \ldots l_n$ and $r_1 \ldots r_n$ in such a way that, between visits, exactly one link changes from j to 0 and another from 0 to j, for some index j. (In other words, every step removes some subtree j from the binary tree and places it elsewhere, preserving preorder.)

26. [*M31*] (*The Kreweras lattice.*) Let F and F' be n-node forests with their nodes numbered 1 to n in preorder. We write $F \ltimes F'$ ("F coalesces F'") if j and k are siblings in F whenever they are siblings in F', for $1 \le j < k \le n$. Figure 39 illustrates this partial ordering in the case $n = 4$; each forest is encoded by the sequence $c_1 \ldots c_n$ of (9) and (10), which specifies the depth of each node. (With this encoding, j and k are siblings if and only if $c_j = c_k \le c_{j+1}, \ldots, c_{k-1}$.)

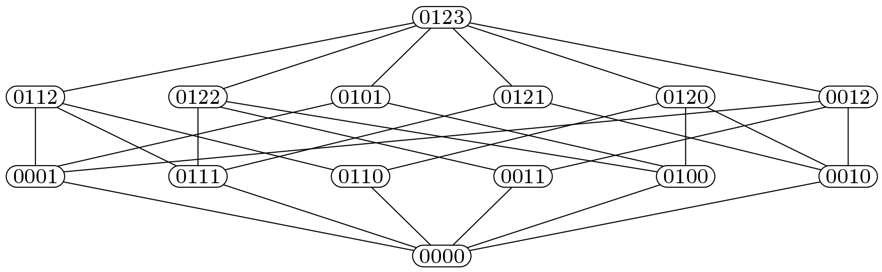


Fig. 39. The Kreweras lattice of order 4. Each forest is represented by its sequence of node depths $c_1 c_2 c_3 c_4$ in preorder. (See exercises 26–28.)

a) Let Π be a partition of $\{1, \ldots, n\}$. Show that there exists a forest F, with nodes labeled $(1, \ldots, n)$ in preorder and with

$$j \equiv k \text{ (modulo } \Pi) \iff j \text{ is a sibling of } k \text{ in } F,$$

if and only if Π satisfies the *noncrossing* property

$$i < j < k < l \text{ and } i \equiv k \text{ and } j \equiv l \text{ (modulo } \Pi) \quad \text{implies} \quad i \equiv j \equiv k \equiv l \text{ (modulo } \Pi).$$

b) Given any two n-node forests F and F', explain how to compute their least upper bound $F \veebar F'$, the element such that $F \ltimes G$ and $F' \ltimes G$ if and only if $F \veebar F' \ltimes G$.

c) When does F' cover F with respect to the relation $\ltimes$? (See exercise 7.2.1.4–55.)

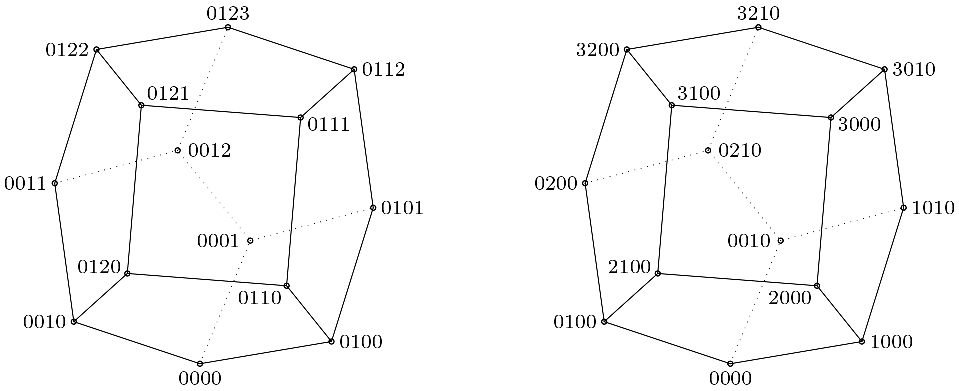
d) Show that if F' covers F, it has exactly one less leaf than F.

e) How many forests cover F, when node k has e_k children for $1 \le k \le n$?

f) Using the definition of duality in exercise 19, what is the dual of the forest (2)?

g) Prove that $F \ltimes F'$ holds if and only if $F'^D \ltimes F^D$. (Because of this property, dual elements have been placed symmetrically about the center of Fig. 39.)
h) Given any two n-node forests F and F', explain how to compute their greatest lower bound $F \barwedge F'$; that is, $G \ltimes F$ and $G \ltimes F'$ if and only if $G \ltimes F \barwedge F'$.
i) Does this lattice satisfy a semimodular law analogous to exercise 7.2.1.5–12(f)?

▶ **27.** [*M33*] (*The Tamari lattice.*) Continuing exercise 26, let us write $F \dashv F'$ if the jth node in preorder has at least as many descendants in F' as it does in F, for all j. In other words, if F and F' are characterized by their scope sequences $s_1 \ldots s_n$ and $s'_1 \ldots s'_n$ as in Table 2, we have $F \dashv F'$ if and only $s_j \le s'_j$ for $1 \le j \le n$. (See Fig. 40.)



(a) (b)

Fig. 40. The Tamari lattice of order 4. Each forest is represented by its sequences of (a) node depths and (b) descendant counts, in preorder. (See exercises 26–28.)

a) Show that the scope coordinates $\min(s_1, s'_1)\min(s_2, s'_2)\ldots\min(s_n, s'_n)$ define a forest that is the greatest lower bound of F and F'. (We denote it by $F \perp F'$.) *Hint:* Prove that $s_1 \ldots s_n$ corresponds to a forest if and only if $0 \le k \le s_j$ implies $s_{j+k} + k \le s_j$, for $0 \le j \le n$, if we define $s_0 = n$.
b) When does F' cover F in this partial ordering?
c) Prove that $F \dashv F'$ if and only if $F'^D \dashv F^D$. (Compare with exercise 26(g).)
d) Explain how to compute a least upper bound, $F \top F'$, given F and F'.
e) Prove that $F \ltimes F'$ in the Kreweras lattice implies $F \dashv F'$ in the Tamari lattice.
f) True or false: $F \barwedge F' \dashv F \perp F'$.
g) True or false: $F \veebar F' \ltimes F \top F'$.
h) What are the longest and shortest paths from the top of the Tamari lattice to the bottom, when each forest of the path covers its successor? (Such paths are called *maximal chains* in the lattice; compare with exercise 7.2.1.4–55(h).)

28. [*M26*] (*The Stanley lattice.*) Continuing exercises 26 and 27, let us define yet another partial ordering on n-node forests, saying that $F \subseteq F'$ whenever the depth coordinates $c_1 \ldots c_n$ and $c'_1 \ldots c'_n$ satisfy $c_j \le c'_j$ for $1 \le j \le n$. (See Fig. 41).
a) Prove that this partial ordering is a lattice, by explaining how to compute the greatest lower bound $F \cap F'$ and least upper bound $F \cup F'$ of any two given forests.
b) Show that Stanley's lattice satisfies the distributive laws

$$F \cap (G \cup H) = (F \cap G) \cup (F \cap H), \qquad F \cup (G \cap H) = (F \cup G) \cap (F \cup H).$$

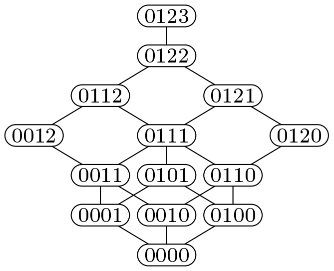


Fig. 41. The Stanley lattice of order 4. Each forest is represented by its sequence of node depths in preorder. (See exercises 26–28.)

c) When does F' cover F in this lattice?
d) True or false: $F \subseteq G$ if and only if $F^R \subseteq G^R$.
e) Prove that $F \subseteq F'$ in the Stanley lattice whenever $F \dashv F'$ in the Tamari lattice.

29. [*HM31*] The covering graph of a Tamari lattice is sometimes known as an "associahedron," because of its connection with the associative law (14), proved in exercise 27(b). The associahedron of order 4, depicted in Fig. 40, looks like it has three square faces and six faces that are regular pentagons. (Compare with Fig. 23 in exercise 7.2.1.2–60, which shows the "permutahedron" of order 4, a well-known Archimedean solid.) Why doesn't Fig. 40 show up in classical lists of uniform polyhedra?

30. [*M26*] The *footprint* of a forest is the bit string $f_1 \ldots f_n$ defined by

$$f_j \;=\; [\text{node } j \text{ in preorder is not a leaf}].$$

a) If F has footprint $f_1 \ldots f_n$, what is the footprint of F^D? (See exercise 27.)
b) How many forests have the footprint 1010110111111000010101000101011000?
c) Prove that $f_j = [d_j = 0]$, for $1 \le j < n$, in the notation of (6).
d) Two elements of a lattice are called *complementary* if their greatest lower bound is the bottom element while their least upper bound is the top element. Show that F and F' are complementary in the Tamari lattice if and only if their footprints are complementary, in the sense that $f'_1 \ldots f'_{n-1} = \bar{f}_1 \ldots \bar{f}_{n-1}$.

▸ **31.** [*M28*] A binary tree with n internal nodes is called *degenerate* if it has height $n-1$.
a) How many n-node binary trees are degenerate?
b) We've seen in Tables 1, 2, and 3 that binary trees and forests can be encoded by various n-tuples of numbers. For each of the encodings $c_1 \ldots c_n$, $d_1 \ldots d_n$, $e_1 \ldots e_n$, $k_1 \ldots k_n$, $p_1 \ldots p_n$, $s_1 \ldots s_n$, $u_1 \ldots u_n$, and $z_1 \ldots z_n$, explain how to see at a glance if the corresponding binary tree is degenerate.
c) True or false: If F is degenerate, so is F^D.
d) Prove that if F and F' are degenerate, so are $F \barwedge F' = F \perp F'$ and $F \veebar F' = F \top F'$.

▸ **32.** [*M30*] Prove that if $F \dashv F'$, there is a forest F'' such that for all G we have

$$F' \perp G \;=\; F \quad \text{if and only if} \quad F \dashv G \dashv F''.$$

Consequently the *semidistributive laws* hold in the Tamari lattice:

$$F \perp G = F \perp H \quad \text{implies} \quad F \perp (G \top H) = F \perp G;$$
$$F \top G = F \top H \quad \text{implies} \quad F \top (G \perp H) = F \top G.$$

▸ **33.** [*M27*] (*Permutation representation of trees.*) Let σ be the cycle $(1\ 2\ \ldots\ n)$.

a) Given any binary tree whose nodes are numbered 1 to n in symmetric order, prove that there is a unique permutation λ of $\{1,\ldots,n\}$ such that, for $1 \le k \le n$,

$$\mathtt{LLINK[}k\mathtt{]} = \begin{cases} k\lambda, & \text{if } k\lambda < k; \\ 0, & \text{otherwise;} \end{cases} \qquad \mathtt{RLINK[}k\mathtt{]} = \begin{cases} k\sigma\lambda, & \text{if } k\sigma\lambda > k; \\ 0, & \text{otherwise.} \end{cases}$$

Thus λ neatly packs $2n$ link fields into a single n-element array.

b) Show that this permutation λ is particularly easy to describe in cycle form when the binary tree is the left-sibling/right-child representation of a forest F. What is the cycle form of $\lambda(F)$ when F is the forest in (2)?

c) Find a simple relation between $\lambda(F)$ and the dual permutation $\lambda(F^D)$.

d) Prove that, in exercise 26, F' covers F if and only if $\lambda(F') = (j\,k)\,\lambda(F)$, where j and k are siblings in F.

e) Consequently the number of maximal chains in the Kreweras lattice of order n is the number of ways to factor an n-cycle as a product of $n-1$ transpositions. Evaluate this number. *Hint:* See Eq. 1.2.6–(16).

34. [*M25*] (R. P. Stanley.) Show that the number of maximal chains in the Stanley lattice of order n is $(n(n-1)/2)!/(1^{n-1}3^{n-2}\ldots(2n-5)^2(2n-3)^1)$.

35. [*HM37*] (D. B. Tyler and D. R. Hickerson.) Explain why the denominators of the asymptotic formula (16) are all powers of 2.

▸ **36.** [*M25*] Analyze the ternary tree generation algorithm of exercise 20(b). *Hint:* There are $(2n+1)^{-1}\binom{3n}{n}$ ternary trees with n internal nodes, by exercise 2.3.4.4–11.

▸ **37.** [*M40*] Analyze the Zaks–Richards algorithm for generating all trees with a given distribution n_0, n_1, n_2, …, n_t of degrees (exercise 21). *Hint:* See exercise 2.3.4.4–32.

38. [*M22*] What is the total number of memory references performed by Algorithm L, as a function of n?

39. [*22*] Prove formula (23) by showing that the elements of A_{pq} in (5) correspond to Young tableaux with two rows.

40. [*M22*] (a) Prove that C_{pq} is odd if and only if $p \wedge (q+1) = 0$, in the sense that the binary representations of p and $q+1$ have no bits in common. (b) Therefore C_n is odd if and only if $n+1$ is a power of 2.

41. [*M21*] Show that the ballot numbers have a simple generating function $\sum C_{pq} w^p z^q$.

▸ **42.** [*M22*] How many unlabeled forests with n nodes are (a) self-conjugate? (b) self-transpose? (c) self-dual? (See exercises 11, 12, 19, and 26.)

43. [*M21*] Express C_{pq} in terms of the Catalan numbers $\langle C_0, C_1, C_2, \ldots\rangle$, aiming for a formula that is simple when $q-p$ is small. (For example, $C_{(q-2)q} = C_q - C_{q-1}$.)

▸ **44.** [*M27*] Prove that Algorithm B makes only $8\frac{2}{3} + O(n^{-1})$ references to memory per binary tree visited.

45. [*M26*] Analyze the memory references made by the algorithm in exercise 22. How does it compare to Algorithm B?

46. [*M30*] (*Generalized Catalan numbers.*) Generalize (21) by defining

$$C_{pq}(x) = C_{p(q-1)}(x) + x^{q-p}C_{(p-1)q}(x), \quad \text{if } 0 \le p \le q \ne 0; \qquad C_{00}(x) = 1;$$

and $C_{pq}(x) = 0$ if $p < 0$ or $p > q$; thus $C_{pq} = C_{pq}(1)$. Also let $C_n(x) = C_{nn}(x)$, so that $\langle C_0(x), C_1(x), \ldots\rangle = \langle 1, 1, 1+x, 1+2x+x^2+x^3, 1+3x+3x^2+3x^3+2x^4+x^5+x^6, \ldots\rangle$.

a) Show that $[x^k]\,C_{pq}(x)$ is the number of paths from (pq) to (00) in (28) that have area k, where the “area” of a path is the number of rectangular cells above it. (Thus an L-shaped path has the maximum possible area, $p(q-p)+\binom{p}{2}$.)

b) Prove that $C_n(x) = \sum_F x^{c_1+\cdots+c_n} = \sum_F x^{\text{internal path length}(F)}$, summed over all n-node forests F.

c) If $C(x,z) = \sum_{n=0}^{\infty} C_n(x)z^n$, show that $C(x,z) = 1 + zC(x,z)C(x,xz)$.

d) Furthermore, $C(x,z)C(x,xz)\ldots C(x,x^r z) = \sum_{p=0}^{\infty} C_{p(p+r)}(x)z^p$.

47. [*M27*] Continuing the previous exercise, generalize the identity (27).

48. [*M28*] (F. Ruskey and A. Proskurowski.) Evaluate $C_{pq}(x)$ when $x=-1$, and use this result to show that no “perfect” Gray code for nested parentheses is possible when $n \ge 5$ is odd.

49. [*17*] What is the lexicographically millionth string of 15 nested parenthesis pairs?

50. [*20*] Design the inverse of Algorithm U: Given a string $a_1 \ldots a_{2n}$ of nested parentheses, determine its rank $N-1$ in lexicographic order. What is the rank of (1)?

51. [*M22*] Let $\bar z_1 \bar z_2 \ldots \bar z_n$ be the complement of $z_1 z_2 \ldots z_n$ with respect to $2n$; in other words, $\bar z_j = 2n - z_j$, where z_j is defined in (8). Show that if $\bar z_1 \bar z_2 \ldots \bar z_n$ is the $(N+1)$st n-combination of $\{0,1,\ldots,2n-1\}$ generated by Algorithm 7.2.1.3L, then $z_1 z_2 \ldots z_n$ is the $(N - \kappa_n N + 1)$st n-combination of $\{1,2,\ldots,2n\}$ generated by the algorithm of exercise 2. (Here κ_n denotes the nth Kruskal function, defined in 7.2.1.3–(60).)

52. [*M23*] Find the mean and variance of the quantity d_n in Table 1, when nested parentheses $a_1 \ldots a_{2n}$ are chosen at random.

53. [*M28*] Let X be the distance from the root of an extended binary tree to the leftmost external node. (a) What is the expected value of X, when all binary trees with n nodes are equally likely? (b) What is the expected value of X in a random *binary search tree*, constructed by Algorithm 6.2.2T from a random permutation $K_1 \ldots K_n$? (c) What is the expected value of X in a random *degenerate* binary tree, in the sense of exercise 31? (d) What is the expected value of 2^X in all three cases?

54. [*HM29*] What are the mean and variance of $c_1 + \cdots + c_n$? (See exercise 46.)

55. [*HM33*] Evaluate $C'_{pq}(1)$, the total area of all the paths in exercise 46(a).

56. [*M23*] (Renzo Sprugnoli, 1990.) Prove the summation formula

$$\sum_{k=0}^{m-1} C_k C_{n-1-k} \;=\; \frac{1}{2}C_n + \frac{2m-n}{2n(n+1)}\binom{2m}{m}\binom{2n-2m}{n-m}, \qquad \text{for } 0 \le m \le n.$$

57. [*M28*] Express the sums $S_p(a,b) = \sum_{k\ge 0}\binom{2a}{a-k}\binom{2b}{b-k}k^p$ in closed form for $p = 0$, 1, 2, 3, and use these formulas to prove (30).

58. [*HM34*] Let t_{lmn} be the number of n-node binary trees in which external node m appears at level l when the external nodes are numbered from 0 to n in symmetric order. Also let $t_{mn} = \sum_{l=1}^{n} l t_{lmn}$, so that t_{mn}/C_n is the average level of external node m; and let $t(w,z)$ be the super generating function

$$\sum_{m,n} t_{mn} w^m z^n = (1{+}w)z + (3{+}4w{+}3w^2)z^2 + (9{+}13w{+}13w^2{+}9w^3)z^3 + \cdots.$$

Prove that $t(w,z) = \bigl(C(z) - wC(wz)\bigr)/(1-w) - 1 + zC(z)t(w,z) + wzC(wz)t(w,z)$, and deduce a simple formula for the numbers t_{mn}.

59. [*HM29*] Similarly, let T_{lmn} count all n-node binary trees in which *internal* node m appears at level l. Find a simple formula for $T_{mn} = \sum_{l=1}^{n} l\,T_{lmn}$.

▸ **60.** [*M26*] (*Balanced strings.*) A string α of nested parentheses is *atomic* if it has the form (α') where α' is nested; every nested string can be represented uniquely as a product of atoms $\alpha_1 \ldots \alpha_r$. A string with equal numbers of left and right parentheses is called *balanced*; every balanced string can be represented uniquely as $\beta_1 \ldots \beta_r$ where each β_j is either an atom or a *co-atom* (the reverse of an atom). The *defect* of a balanced string is half the length of its co-atoms. For example, the balanced string

(( ))) ((( )))))) (( ) ((( )) ((( )) (( )

has the factored form $\beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7\beta_8 = \alpha_1\alpha_2^R\alpha_3\alpha_4^R\alpha_5^R\alpha_6\alpha_7^R\alpha_8$, with four atoms and four co-atoms; its defect is $|\alpha_2\alpha_4\alpha_5\alpha_7|/2 = 9$.

a) Prove that the defect of a balanced string is the number of indices k for which the kth right parenthesis *precedes* the kth left parenthesis.

b) If $\beta_1 \ldots \beta_r$ is balanced, we can map it into a nested string by simply reversing its co-atoms. But the following mapping is more interesting, because it produces unbiased (uniformly random) nested strings from unbiased balanced strings: Let there be s co-atoms $\beta_{i_1} = \alpha_{i_1}^R, \ldots, \beta_{i_s} = \alpha_{i_s}^R$. Replace each co-atom by `(`; then append the string $)\alpha'_{i_s} \ldots)\alpha'_{i_1}$, where $\alpha_j = (\alpha'_j)$. For example, the string above is mapped into $\alpha_1(\alpha_3((\alpha_6(\alpha_8)\alpha'_7)\alpha'_5)\alpha'_4)\alpha'_2$, which just happens to equal the string (1) illustrated at the beginning of this section.
Design an algorithm that applies this mapping to a given balanced string $b_1 \ldots b_{2n}$.

c) Also design an algorithm for the inverse mapping: Given a nested string $\alpha = a_1 \ldots a_{2n}$ and an integer l with $0 \le l \le n$, compute a balanced string $\beta = b_1 \ldots b_{2n}$ of defect l for which $\beta \mapsto \alpha$. What balanced string of defect 11 maps into (1)?

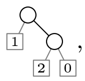
▸ **61.** [*M26*] (*Raney's Cycle Lemma.*) Let $b_1b_2 \ldots b_N$ be a string of nonnegative integers such that $f = N - b_1 - b_2 - \cdots - b_N > 0$.

a) Prove that exactly f of the cyclic shifts $b_{j+1} \ldots b_N b_1 \ldots b_j$ for $1 \le j \le N$ satisfy the preorder degree sequence property in exercise 20.

b) Design an efficient algorithm to determine all such j, given $b_1b_2 \ldots b_N$.

c) Explain how to generate a random forest that has $N = n_0 + \cdots + n_t$ nodes, with exactly n_j nodes of degree j. (For example, we obtain random n-node t-ary trees as a special case of this general procedure when $N = tn + 1$, $n_0 = (t-1)n + 1$, $n_1 = \cdots = n_{t-1} = 0$, and $n_t = n$.)

62. [*22*] A binary tree can also be represented by bit strings $(l_1 \ldots l_n, r_1 \ldots r_n)$, where l_j and r_j tell whether the left and right subtrees of node j in preorder are nonempty. (See Theorem 2.3.1A.) Prove that if $l_1 \ldots l_n$ and $r_1 \ldots r_n$ are arbitrary bit strings with $l_1 + \cdots + l_n + r_1 + \cdots + r_n = n - 1$, exactly one cyclic shift $(l_{j+1} \ldots l_n l_1 \ldots l_j, r_{j+1} \ldots r_n r_1 \ldots r_j)$ yields a valid binary tree representation, and explain how to find it.

63. [*16*] If the first two iterations of Rémy's algorithm have produced 

, what decorated binary trees are possible after the next iteration?

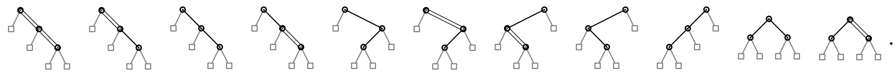
64. [*20*] What sequence of X values in Algorithm R corresponds to the decorated trees of (34), and what are the final values of $L_0L_1 \ldots L_{12}$?

65. [*38*] Generalize Rémy's algorithm (Algorithm R) to t-ary trees.

66. [*21*] A *Schröder tree* is a binary tree in which every nonnull right link is colored either white or black. The number S_n of n-node Schröder trees is

n =	0	1	2	3	4	5	6	7	8	9	10	11	12
S_n =	1	1	3	11	45	197	903	4279	20793	103049	518859	2646723	13648869

for small n. For example, $S_3 = 11$ because the possibilities are



(White links are "hollow"; external nodes have also been attached.)

a) Find a simple correspondence between Schröder trees with n internal nodes and ordinary trees with $n+1$ leaves and no nodes of degree one.

b) Devise a Gray code for Schröder trees.

67. [*M22*] What is the generating function $S(z) = \sum_n S_n z^n$ for Schröder numbers?

68. [*10*] What is the Christmas tree pattern of order 0?

69. [*16*] Are the Christmas tree patterns of orders 6 and 7 visible in Table 4, possibly in slight disguise?

▶ **70.** [*20*] Find a simple rule that defines, for every bit string σ, another bit string σ' called its *mate*, with the following properties: (i) $\sigma'' = \sigma$; (ii) $|\sigma'| = |\sigma|$; (iii) either $\sigma \subseteq \sigma'$ or $\sigma' \subseteq \sigma$; (iv) $\nu(\sigma) + \nu(\sigma') = |\sigma|$.

71. [*M21*] Let M_{tn} be the size of the largest possible set S of n-bit strings with the property that, if σ and τ are members of S with $\sigma \subseteq \tau$, then $\nu(\tau) < \nu(\sigma) + t$. (Thus, for example, $M_{1n} = M_n$ by Sperner's theorem.) Find a formula for M_{tn}.

▶ **72.** [*M28*] If you start with a single row $\sigma_1\ \sigma_2\ \ldots\ \sigma_s$ of length s and apply the growth rule (36) repeatedly n times, how many rows do you obtain?

73. [*15*] In the Christmas tree pattern of order 30, what are the first and last elements of the row that contains the bit string 011001001000011111101101011100?

74. [*M26*] Continuing the previous exercise, how many rows precede that row?

▶ **75.** [*HM23*] Let $(r_1^{(n)}, r_2^{(n)}, \ldots, r_{n-1}^{(n)})$ be the row numbers in which the Christmas tree pattern of order n has $n-1$ entries; for example, Table 4 tells us that $(r_1^{(8)}, \ldots, r_7^{(8)}) = (20, 40, 54, 62, 66, 68, 69)$. Find formulas for $r_{j+1}^{(n)} - r_j^{(n)}$ and for $\lim_{n\to\infty} r_j^{(n)}/M_n$.

76. [*HM46*] Study the limiting shape of the Christmas tree patterns as $n \to \infty$. Does it, for example, have a fractal dimension under some appropriate scaling?

77. [*21*] Design an algorithm to generate the sequence of rightmost elements $a_1 \ldots a_n$ in the rows of the Christmas tree pattern, given n. *Hint:* These bit strings are characterized by the property that $a_1 + \cdots + a_k \geq k/2$ for $0 \leq k \leq n$.

78. [*20*] True or false: If $\sigma_1\ \ldots\ \sigma_s$ is a row of the Christmas tree pattern, so is $\bar\sigma_s^R\ \ldots\ \bar\sigma_1^R$ (the reverse sequence of reverse complements).

79. [*M26*] The number of permutations $p_1 \ldots p_n$ that have exactly one "descent" where $p_k > p_{k+1}$ is the Eulerian number $\left\langle {n \atop 1} \right\rangle = 2^n - n - 1$, according to Eq. 5.1.3–(12). The number of entries in the Christmas tree pattern, above the bottom row, is the same.

a) Find a combinatorial explanation of this coincidence, by giving a one-to-one correspondence between one-descent permutations and unsorted bit strings.

b) Show that two unsorted bit strings belong to the same row of the Christmas tree pattern if and only if they correspond to permutations that define the same P tableau under the Robinson–Schensted correspondence (Theorem 5.1.4A).

80. [*30*] Say that two bit strings are *concordant* if we can obtain one from the other via the transformations $010 \leftrightarrow 100$ or $101 \leftrightarrow 110$ on substrings. For example, the strings

$$\begin{array}{ccccccccc}
011100 \leftrightarrow 011010 \leftrightarrow & 010110 & \leftrightarrow & 010101 & \leftrightarrow 011001 & & & & \\
& \updownarrow & & \updownarrow & & & & & \\
& 100110 & \leftrightarrow & 100101 & \leftrightarrow 101001 \leftrightarrow 110001 & & & &
\end{array}$$

are mutually concordant, but no other string is concordant with any of them.

Prove that strings are concordant if and only if they belong to the same column of the Christmas tree pattern and to rows of the same length in that pattern.

81. [*M30*] A *biclutter* of order (n, n') is a family S of bit string pairs (σ, σ'), where $|\sigma| = n$ and $|\sigma'| = n'$, with the property that distinct members (σ, σ') and (τ, τ') of S are allowed to satisfy $\sigma \subseteq \tau$ and $\sigma' \subseteq \tau'$ only if $\sigma \neq \tau$ and $\sigma' \neq \tau'$.

Use Christmas tree patterns to prove that S contains at most $M_{n+n'}$ string pairs.

▶ **82.** [*M26*] Let $E(f)$ be the number of times Algorithm H evaluates the function f.

a) Show that $M_n \le E(f) \le M_{n+1}$, with equality when f is constant.

b) Among all f such that $E(f) = M_n$, which one minimizes $\sum_\sigma f(\sigma)$?

c) Among all f such that $E(f) = M_{n+1}$, which one maximizes $\sum_\sigma f(\sigma)$?

83. [*M20*] (G. Hansel.) Show that there are at most 3^{M_n} monotone Boolean functions $f(x_1, \ldots, x_n)$ of n Boolean variables.

▶ **84.** [*HM27*] (D. Kleitman.) Let A be an $m \times n$ matrix of real numbers in which every column v has length $\|v\| \ge 1$, and let b be an m-dimensional column vector. Prove that at most M_n column vectors $x = (a_1, \ldots, a_n)^T$, with components $a_j = 0$ or 1, satisfy $\|Ax - b\| < \frac{1}{2}$. *Hint:* Use a construction analogous to the Christmas tree pattern.

85. [*HM35*] (Philippe Golle.) Let V be any vector space contained in the set of all real n-dimensional vectors, but containing none of the unit vectors $(1, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0)$, $\ldots$, $(0, \ldots, 0, 1)$. Prove that V contains at most M_n vectors whose components are all 0 or 1; furthermore the upper bound M_n is achievable.

86. [*15*] If (2) is regarded as an *oriented forest* instead of an ordered forest, what canonical forest corresponds to it? Specify that forest both by its level codes $c_1 \ldots c_{15}$ and its parent pointers $p_1 \ldots p_{15}$.

87. [*M20*] Let F be an ordered forest in which the kth node in preorder appears on level c_k and has parent p_k, where $p_k = 0$ if that node is a root.

a) How many forests satisfy the condition $c_k = p_k$ for $1 \le k \le n$?

b) Suppose F and F' have level codes $c_1 \ldots c_n$ and $c'_1 \ldots c'_n$, respectively, as well as parent links $p_1 \ldots p_n$ and $p'_1 \ldots p'_n$. Prove that, lexicographically, $c_1 \ldots c_n \le c'_1 \ldots c'_n$ if and only if $p_1 \ldots p_n \le p'_1 \ldots p'_n$.

88. [*M20*] Analyze Algorithm O: How often is step O4 performed? What is the total number of times p_k is changed in step O5?

89. [*M46*] How often does step O5 set $p_k \leftarrow p_j$?

▶ **90.** [*M27*] If $p_1 \ldots p_n$ is a canonical sequence of parent pointers for an oriented forest, the graph with vertices $\{0, 1, \ldots, n\}$ and edges $\{k \text{ — } p_k \mid 1 \le k \le n\}$ is a *free tree*, namely a connected graph with no cycles. (See Theorem 2.3.4.1A.) Conversely, every free tree corresponds to at least one oriented forest in this way. But the parent pointers 011 and 000 both yield the same free tree ; similarly, 012 and 010 both yield .

The purpose of this exercise is to restrict the sequences $p_1 \ldots p_n$ further so that each free tree is obtained exactly once. We proved in 2.3.4.4–(9) that the number of structurally different free trees on $n+1$ vertices has a fairly simple generating function, by showing that a free tree always has at least one *centroid*.

a) Show that a canonical n-node forest corresponds to a free tree with a single centroid if and only if no tree in the forest has more than $\lfloor n/2 \rfloor$ nodes.
b) Modify Algorithm O so that it generates all sequences $p_1 \ldots p_n$ that satisfy (a).
c) Explain how to find all $p_1 \ldots p_n$ for free trees that have *two* centroids.

91. [*M37*] (Nijenhuis and Wilf.) Show that a random oriented tree can be generated with a procedure analogous to the random partition algorithm of exercise 7.2.1.4–47.

92. [*15*] Are the first and last spanning trees visited by Algorithm S adjacent, in the sense that they have $n-2$ edges in common?

93. [*20*] When Algorithm S terminates, has it restored the graph to its original state?

94. [*22*] Algorithm S needs to "prime the pump" by finding an initial spanning tree in step S1. Explain how to do that task.

95. [*26*] Complete Algorithm S by implementing the bridge test in step S8.

▶ **96.** [*28*] Analyze the approximate running time of Algorithm S when the given graph is simply (a) a path P_n of length $n-1$; (b) a cycle C_n of length n.

97. [*15*] Is (48) a series-parallel graph?

98. [*16*] What series-parallel graph corresponds to (53) if A is taken to be *serial*?

▶ **99.** [*30*] Consider a series-parallel graph represented by a tree as in (53), together with node values that satisfy (55). These values define a spanning tree or a near tree, according as v_p is 1 or 0 at the root p. Show that the following method will generate all of the other configs of the root:

i) Begin with all uneasy nodes active, other nodes passive.
ii) Select the rightmost active node, p, in preorder; but terminate if all nodes are passive.
iii) Change $d_p \leftarrow r_{d_p}$, update all values in the tree, and visit the new config.
iv) Activate all uneasy nodes to the right of p.
v) If d_p has run through all children of p since p last became active, make node p passive. Return to (ii).

Also explain how to perform these steps efficiently. *Hints:* To implement step (v), introduce a pointer z_p; make node p passive when d_p becomes equal to z_p, and at such times also reset z_p to the previous value of d_p. To implement steps (ii) and (iv), use focus pointers f_p analogous to those in Algorithms 7.2.1.1L and 7.2.1.1K.

100. [*40*] Implement the text's "Algorithm S$'$" for revolving-door generation of all spanning trees, by combining Algorithm S with the ideas of exercise 99.

101. [*46*] Is there a simple revolving-door way to list all n^{n-2} spanning trees of the complete graph K_n? (The order produced by Algorithm S is quite complicated.)

102. [*46*] An *oriented spanning tree* of a directed graph D on n vertices, also known as a "spanning arborescence," is an oriented subtree of D containing $n-1$ arcs. The matrix tree theorem (exercise 2.3.4.2–19) tells us that the oriented subtrees having a given root can readily be counted by evaluating an $(n-1)\times(n-1)$ determinant.

Can those oriented subtrees be listed in a revolving-door order, always removing one arc and replacing it with another?

▸ **103.** [*HM39*] (*Sandpiles.*) Consider any digraph D on vertices $V_0, V_1, \ldots, V_n$ with e_{ij} arcs from V_i to V_j, where $e_{ii} = 0$. Assume that D has at least one oriented spanning tree rooted at V_0; this assumption means that, if we number the vertices appropriately, we have $e_{i0} + \cdots + e_{i(i-1)} > 0$ for $1 \le i \le n$. Let $d_i = e_{i0} + \cdots + e_{in}$ be the total out-degree of V_i. Put x_i grains of sand on vertex V_i for $0 \le i \le n$, and play the following game: If $x_i \ge d_i$ for any $i \ge 1$, decrease x_i by d_i and set $x_j \leftarrow x_j + e_{ij}$ for all $j \ne i$. (In other words, pass one grain of sand from V_i through each of its outgoing arcs, whenever possible, except when $i = 0$. This operation is called "toppling" V_i, and a sequence of topplings is called an "avalanche." Vertex V_0 is special; instead of toppling, it collects particles of sand that essentially leave the system.) Continue until $x_i < d_i$ for $1 \le i \le n$. Such a state $x = (x_1, \ldots, x_n)$ is called *stable*.

a) Prove that every avalanche terminates in a stable state after a finite number of topplings. Furthermore, the final state depends only on the initial state, not on the order in which toppling is performed.

b) Let $\sigma(x)$ be the stable state that results from initial state x. A stable state is called *recurrent* if it is $\sigma(x)$ for some x with $x_i \ge d_i$ for $1 \le i \le n$. (Recurrent states correspond to sandpiles that have evolved over a long period of time, after new grains of sand are repeatedly introduced at random.) Find the recurrent states in the special case when $n = 4$ and when the only arcs of D are

$$V_1 \to V_0,\ V_1 \to V_2,\ V_2 \to V_0,\ V_2 \to V_1,\ V_3 \to V_0,\ V_3 \to V_4,\ V_4 \to V_0,\ V_4 \to V_3.$$

c) Let $d = (d_1, \ldots, d_n)$. Prove that x is recurrent if and only if $x = \sigma(x + t)$, where t is the vector $d - \sigma(d)$.

d) Let a_i be the vector $(-e_{i1}, \ldots, -e_{i(i-1)}, d_i, -e_{i(i+1)}, \ldots, -e_{in})$, for $1 \le i \le n$; thus, toppling V_i corresponds to changing the state vector $x = (x_1, \ldots, x_n)$ to $x - a_i$. Say that two states x and x' are *congruent*, written $x \equiv x'$, if $x - x' = m_1 a_1 + \cdots + m_n a_n$ for some integers $m_1, \ldots, m_n$. Prove that there are exactly as many equivalence classes of congruent states as there are oriented spanning trees in D, rooted at V_0. *Hint:* See the matrix tree theorem, exercise 2.3.4.2–19.

e) If $x \equiv x'$ and if both x and x' are recurrent, prove that $x = x'$.

f) Prove that every congruence class contains a unique recurrent state.

g) If D is *balanced*, in the sense that the in-degree of each vertex equals its out-degree, prove that x is recurrent if and only if $x = \sigma(x + a)$, where $a = (e_{01}, \ldots, e_{0n})$.

h) Illustrate these concepts when D is a "wheel" with n spokes: Let there be $3n$ arcs, $V_j \to V_0$ and $V_j \leftrightarrow V_{j+1}$ for $1 \le j \le n$, regarding V_{n+1} as identical to V_1. Find a one-to-one correspondence between the oriented spanning trees of this digraph and the recurrent states of its sandpiles.

i) Similarly, analyze the recurrent sandpiles when D is the *complete* graph on $n+1$ vertices, namely when $e_{ij} = [i \ne j]$ for $0 \le i, j \le n$. *Hint:* See exercise 6.4–31.

▸ **104.** [*HM21*] If G is a graph on n vertices $\{V_1, \ldots, V_n\}$, with e_{ij} edges between V_i and V_j, let $C(G)$ be the matrix with entries $c_{ij} = -e_{ij} + \delta_{ij} d_i$, where $d_i = e_{i1} + \cdots + e_{in}$ is the degree of V_i. Let us say that the *aspects* of G are the eigenvalues of $C(G)$, namely the roots $\alpha_0, \ldots, \alpha_{n-1}$ of the equation $\det(\alpha I - C(G)) = 0$. Since $C(G)$ is a symmetric matrix, its eigenvalues are real numbers, and we can assume that $\alpha_0 \le \alpha_1 \le \cdots \le \alpha_{n-1}$.

a) Prove that $\alpha_0 = 0$.

b) Prove that G has exactly $c(G) = \alpha_1 \ldots \alpha_{n-1}/n$ spanning trees.

c) What are the aspects of the complete graph K_n?

105. [*HM37*] Continuing exercise 104, we wish to prove that there is often an easy way to determine the aspects of G when G has been constructed from other graphs whose aspects are known. Suppose G' has aspects $\alpha'_0, \ldots, \alpha'_{n'-1}$ and G'' has aspects $\alpha''_0, \ldots, \alpha''_{n''-1}$; what are the aspects of G in the following cases?

a) $G = \overline{G'}$ is the complement of G'. (Assume that $e'_{ij} \le [i \ne j]$ in this case.)

b) $G = G' + G''$ is the sum (juxtaposition) of G' and G''.

c) $G = G' \mathbin{\overline{+}} G''$ is the cosum (join) of G' and G''.

d) $G = G' \times G''$ is the Cartesian product of G' and G''.

e) $G = L(G')$ is the line graph of G', when G' is a regular graph of degree d (namely when all vertices of G' have exactly d neighbors, and there are no self-loops).

f) $G = G' \circ G''$ is the direct product (conjunction) of G' and G'', when G' is regular of degree d' and G'' is regular of degree d''.

g) $G = G' \otimes G''$ is the strong product of regular graphs G' and G''.

▶ **106.** [*HM37*] Find the total number of spanning trees in (a) an $m \times n$ grid $P_m \times P_n$; (b) an $m \times n$ cylinder $P_m \times C_n$; (c) an $m \times n$ torus $C_m \times C_n$. Why do these numbers tend to have only small prime factors? *Hint:* Show that the aspects of P_n and C_n can be expressed in terms of the numbers $\sigma_{kn} = 4\sin^2 \frac{k\pi}{2n}$.

107. [*M24*] Determine the aspects of all connected graphs that have $n \le 5$ vertices and no self-loops or parallel edges.

108. [*HM40*] Extend the results of exercises 104–106 to directed graphs.

109. [*M46*] Find a combinatorial explanation for the fact that (57) is the number of spanning trees in the n-cube.

▶ **110.** [*M27*] Prove that if G is any connected multigraph without self-loops, it has

$$c(G) > \sqrt{(d_1 - 1) \ldots (d_n - 1)}$$

spanning trees, where d_j is the degree of vertex j.

111. [*05*] List the nodes of the tree (58) in postpreorder.

112. [*15*] If node p of a forest precedes node q in prepostorder and follows it in postpreorder, what can you say about p and q?

▶ **113.** [*20*] How do prepostorder and postpreorder of a forest F relate to prepostorder and postpreorder of the conjugate forest F^R? (See exercise 13.)

114. [*15*] If we want to traverse an entire forest in prepostorder using Algorithm Q, how should we begin the process?

115. [*20*] Analyze Algorithm Q: How often is each step performed, during the complete traversal of a forest?

▶ **116.** [*28*] If the nodes of a forest F are labeled 1 to n in prepostorder, say that node k is *lucky* if it is adjacent to node $k+1$ in F, *unlucky* if it is three steps away, and *ordinary* otherwise, for $1 \le k \le n$; in this definition, node $n+1$ is an imaginary super-root considered to be the parent of each root.

a) Prove that lucky nodes occur only on even-numbered levels; unlucky nodes occur only on odd-numbered levels.

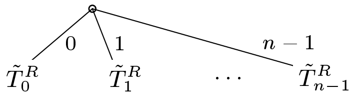
b) Show that the number of lucky nodes is exactly one greater than the number of unlucky nodes, unless $n = 0$.

117. [*21*] How many n-node forests contain no unlucky nodes?

118. [*M28*] How many lucky nodes are present in (a) the complete t-ary tree with $(t^k-1)/(t-1)$ internal nodes? (b) the Fibonacci tree of order k, with $F_{k+1}-1$ internal nodes? (See 2.3.4.5–(6) and Fig. 8 in Section 6.2.1.)

119. [*21*] The *twisted binomial tree* $\tilde{T}_n$ of order n is defined recursively by the rules

$\tilde{T}_0 = \circ\,,\qquad \tilde{T}_n =$



for $n>0$.

(Compare with 7.2.1.3–(21); we reverse the order of children on alternate levels.) Show that prepostorder traversal of $\tilde{T}_n$ has a simple connection with Gray binary code.

120. [*22*] True or false: The square of a graph is Hamiltonian if the graph is connected and has no bridges.

121. [*M32*] (F. Neuman, 1964.) The *derivative* of a graph G is the graph $G^{(\prime)}$ obtained by removing all vertices of degree 1 and the edges touching them. Prove that, when T is a free tree, its square T^2 contains a Hamiltonian path if and only if its derivative has no vertex of degree greater than 4 and the following two additional conditions hold:

i) All vertices of degree 3 or 4 in $T^{(\prime)}$ lie on a single path.

ii) Between any two vertices of degree 4 in $T^{(\prime)}$, there is at least one vertex that has degree 2 in T.

▸ **122.** [*31*] (*Dudeney's Digital Century puzzle.*) There are many curious ways to obtain the number 100 by inserting arithmetical operators and possibly also parentheses into the sequence 123456789. For example,

$$\begin{aligned}100 = 1+2\times 3+4\times 5-6+7+8\times 9 &= (1+2-3-4)\times(5-6-7-8-9)\\ &= \bigl((1/((2+3)/4-5+6))\times 7+8\bigr)\times 9\,.\end{aligned}$$

a) How many such representations of 100 are possible? To make this question precise, in view of the associative law and other algebraic properties, assume that expressions are written in canonical form according to the following syntax:

$\langle\,\text{expression}\,\rangle \to \langle\,\text{number}\,\rangle \mid \langle\,\text{sum}\,\rangle \mid \langle\,\text{product}\,\rangle \mid \langle\,\text{quotient}\,\rangle$
$\langle\,\text{sum}\,\rangle \to \langle\,\text{term}\,\rangle + \langle\,\text{term}\,\rangle \mid \langle\,\text{term}\,\rangle - \langle\,\text{term}\,\rangle \mid \langle\,\text{sum}\,\rangle + \langle\,\text{term}\,\rangle \mid \langle\,\text{sum}\,\rangle - \langle\,\text{term}\,\rangle$
$\langle\,\text{term}\,\rangle \to \langle\,\text{number}\,\rangle \mid \langle\,\text{product}\,\rangle \mid \langle\,\text{quotient}\,\rangle$
$\langle\,\text{product}\,\rangle \to \langle\,\text{factor}\,\rangle \times \langle\,\text{factor}\,\rangle \mid \langle\,\text{product}\,\rangle \times \langle\,\text{factor}\,\rangle \mid (\langle\,\text{quotient}\,\rangle) \times \langle\,\text{factor}\,\rangle$
$\langle\,\text{quotient}\,\rangle \to \langle\,\text{factor}\,\rangle/\langle\,\text{factor}\,\rangle \mid \langle\,\text{product}\,\rangle/\langle\,\text{factor}\,\rangle \mid (\langle\,\text{quotient}\,\rangle)/\langle\,\text{factor}\,\rangle$
$\langle\,\text{factor}\,\rangle \to \langle\,\text{number}\,\rangle \mid (\langle\,\text{sum}\,\rangle)$
$\langle\,\text{number}\,\rangle \to \langle\,\text{digit}\,\rangle$

The digits used must be 1 through 9, in that order.

b) Extend problem (a) by allowing multidigit numbers, with the syntax

$$\langle\,\text{number}\,\rangle \to \langle\,\text{digit}\,\rangle \mid \langle\,\text{number}\,\rangle\langle\,\text{digit}\,\rangle$$

For example, $100 = \bigl(1/(2-3+4)\bigr)\times 567-89$. What is the shortest such representation? What is the longest?

c) Extend problem (b) by also allowing decimal points:

$$\begin{aligned}\langle\,\text{number}\,\rangle &\to \langle\,\text{digit string}\,\rangle \mid .\langle\,\text{digit string}\,\rangle\\ \langle\,\text{digit string}\,\rangle &\to \langle\,\text{digit}\,\rangle \mid \langle\,\text{digit string}\,\rangle\langle\,\text{digit}\,\rangle\end{aligned}$$

For example, $100 = (.1-2-34\times .5)/(.6-.789)$, amazingly enough.

123. [*21*] Continuing the previous exercise, what are the smallest positive integers that *cannot* be represented using conventions (a), (b), (c)?

Fig. 42. "Organic" illustrations of binary trees.

▶ **124.** [*40*] Experiment with methods for drawing extended binary trees that are inspired by simple models from nature. For example, we can assign a value $v(x)$ to each node x, called its *Horton–Strahler number*, as follows: Each external (leaf) node has $v(x) = 0$; an internal node with children (l, r) has $v(x) = \max(v(l), v(r)) + [v(l) = v(r)]$. The edge from internal node x to its parent can be drawn as a rectangle with height $h(v(x))$ and width $w(v(x))$, and the edge rectangles with children (l, r) can be offset by angles $\theta(v(l(x)), v(r(x)))$, $-\theta(v(r(x)), v(l(x)))$, for certain functions h, w, and θ. The examples in Fig. 42 show typical results when we choose $w(k) = 3 + k$, $h(k) = 18k$, $\theta(k, k) = 30^\circ$, $\theta(j, k) = ((k + 1)/j) \times 20^\circ$ for $0 \le k < j$, and $\theta(j, k) = ((k - j)/k) \times 30^\circ$ for $0 \le j < k$; the roots appear at the bottom. Part (a) of Fig. 42 is the binary tree (4); part (b) is a random 100-node tree generated by Algorithm R; part (c) is the Fibonacci tree of order 11, which has 143 nodes; and part (d) is a random 100-node binary search tree. (The trees in parts (b), (c), and (d) clearly belong to different species.)

SECTION 7.2.1.6

1. It could "see" a left parenthesis at the left of every internal node and a right parenthesis at the bottom of every internal node. Alternatively, it could associate right parentheses with the *external* nodes that it encounters — except for the very last $\square$; see exercise 20.

2. Z1. [Initialize.] Set $z_k \leftarrow 2k-1$ for $0 \le k \le n$. (Assume that $n \ge 2$.)

Z2. [Visit.] Visit the tree-combination $z_1 z_2 \ldots z_n$.

Z3. [Easy case?] If $z_{n-1} < z_n - 1$, set $z_n \leftarrow z_n - 1$ and return to Z2.

Z4. [Find j.] Set $j \leftarrow n-1$ and $z_n \leftarrow 2n-1$. While $z_{j-1} = z_j - 1$, set $z_j \leftarrow 2j-1$ and $j \leftarrow j-1$.

Z5. [Decrease z_j]. Terminate the algorithm if $j = 1$. Otherwise set $z_j \leftarrow z_j - 1$ and go back to Z2. ▌

3. Label the nodes of the forest in preorder. The first $z_k - 1$ elements of $a_1 \ldots a_{2n}$ contain $k-1$ left parentheses and $z_k - k$ right parentheses. So there is an excess of $2k-1-z_k$ left parentheses over right parentheses when the "worm" first reaches node k; and $2k-1-z_k$ is the level (or depth) of that node.

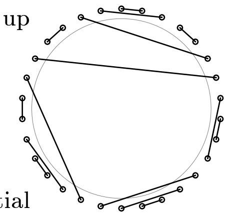
Let $q_1 \ldots q_n$ be the inverse of $p_1 \ldots p_n$, so that node k is the q_kth node in postorder. Since k occurs to the left of j in $p_1 \ldots p_n$ if and only if $q_k < q_j$, we see that c_k is the number of nodes j that precede k in preorder but follow it in postorder, namely the number of ancestors of k; again, this is the level of k.

Alternative proof: We can also show that both sequences $z_1 \ldots z_n$ and $c_1 \ldots c_n$ have essentially the same recursive structure as (5): $Z_{pq} = (Z_{p(q-1)} + 1^p),\ 1(Z_{(p-1)q} + 1^{p-1})$ when $0 \le p \le q$; and $C_{pq} = C_{p(q-1)},\ (q-p)C_{(p-1)q}$. (Consider the mate of the last, next-to-last, etc., left parenthesis.)

Incidentally, the formula '$c_{k+1} + d_k = c_k + 1$' is equivalent to (11).

4. Almost true; but $d_1 \ldots d_n$ and $z_1 \ldots z_n$ occur in *decreasing* order, while $p_1 \ldots p_n$ and $c_1 \ldots c_n$ are increasing. (This lexicographic property for a sequence of permutations $p_1 \ldots p_n$ is not automatically inherited from lexicographic order of the corresponding inversion tables $c_1 \ldots c_n$; but the result does hold for this particular class of $p_1 \ldots p_n$.)

5. $d_1 \ldots d_{15} = 0\,2\,0\,0\,2\,0\,0\,1\,0\,3\,2\,0\,1\,0\,4$; $z_1 \ldots z_{15} = 1\,2\,5\,6\,7\,10\,11\,12\,14\,15\,19\,22\,23\,25\,26$; $p_1 \ldots p_{15} = 2\,1\,5\,4\,8\,10\,9\,7\,11\,6\,13\,15\,14\,12\,3$; $c_1 \ldots c_{15} = 0\,1\,0\,1\,2\,1\,2\,3\,3\,4\,2\,1\,2\,2\,3$.



6. Match up the parentheses as usual; then simply curl the string up and around until a_{2n} becomes adjacent to a_1, and notice that the distinction between left and right parentheses can be reconstructed from the context. Letting a_1 correspond to the bottom of the circle, as in Table 1, yields the diagram shown. [A. Errera, *Mémoires de la Classe Sci. 8°, Acad. Royale de Belgique* (2) **11**, 6 (1931), 26 pp.]

7. (a) It equals `))()...()`; setting $a_1 \leftarrow$ '`(`' will restore the initial string. (b) The initial binary tree (from step B1) will have been restored, except that $l_n = n+1$.

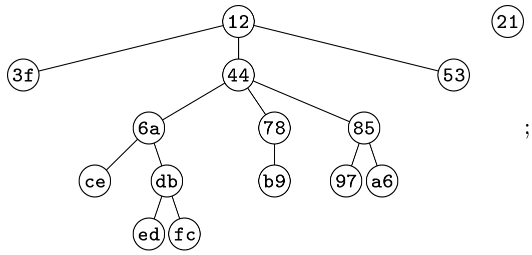
8. $l_1 \ldots l_{15} = 2\,0\,4\,5\,0\,7\,8\,0\,10\,0\,0\,13\,0\,15\,0$; $r_1 \ldots r_{15} = 3\,0\,0\,6\,0\,12\,11\,9\,0\,0\,0\,0\,14\,0\,0$; $e_1 \ldots e_{15} = 1\,0\,3\,1\,0\,2\,2\,0\,1\,0\,0\,2\,0\,1\,0$; $s_1 \ldots s_{15} = 1\,0\,12\,1\,0\,5\,3\,0\,1\,0\,0\,3\,0\,1\,0$.

9. Node j is an ancestor of node k if and only if $s_j + j \ge k$. (As a consequence, we have $c_1 + \cdots + c_n = s_1 + \cdots + s_n$.)

10. If j is the index z_k of the kth left parenthesis, we have $w_j = c_k + 1$ and $w_{j'} = c_k$, where j' is the index of the matching right parenthesis.

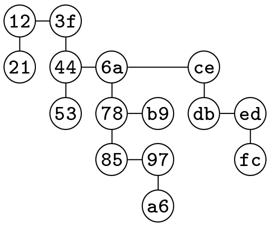
11. Swap left and right parentheses in $a_{2n} \ldots a_1$ to get the mirror image of $a_1 \ldots a_{2n}$.

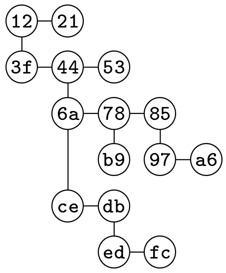
12. The mirror reflection of (4) corresponds to the forest



;

but the significance of transposition is clearer, forest-wise, if we draw right-sibling and left-child links horizontally and vertically, then do a matrix-like transposition:





13. (a) By induction on the number of nodes, we have preorder$(F^R) =$ postorder$(F)^R$ and postorder$(F^R) =$ preorder$(F)^R$.

(b) Let F correspond to the binary tree B; then preorder$(F) =$ preorder(B) and postorder$(F) =$ inorder(B), as noted after 2.3.2–(6). Therefore preorder$(F^T) =$ preorder$(B^R) =$ postorder$(B)^R$ has no simple relationship to either preorder(F) or postorder(F). But postorder$(F^T) =$ inorder$(B^R) =$ inorder$(B)^R =$ postorder$(F)^R$.

14. According to answer 13, postorder$(F^{RT}) =$ preorder$(F) =$ preorder(B) when F corresponds naturally to B; and postorder$(F^{TR}) =$ preorder$(F^T)^R =$ postorder(B). Therefore the equation $F^{RT} = F^{TR}$ holds if and only if F has at most one node.

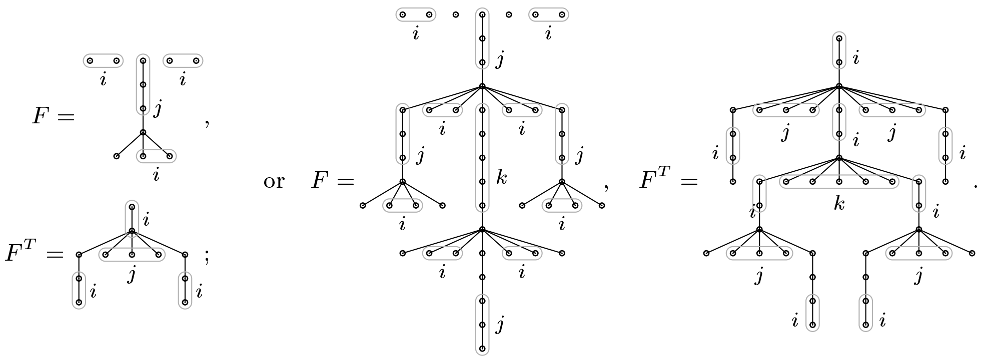
15. If F^R corresponds naturally to the binary tree B', the root of B' is the root of F's rightmost tree. The left link of node x in B' is to the leftmost child of x in F^R, which is the rightmost child of x in F; similarly, the right link is to x's left sibling in F.

Note: Since B corresponds naturally to F^{RT}, answer 13 tells us that inorder$(B) =$ postorder$(F^{RT}) =$ postorder$(F^R)^R =$ preorder(F).

16. The forest $F \mid G$ is obtained by placing the trees of F below the first node of G in postorder. Associativity follows because $F|(G|H) = (H^T G^T F^T)^T = (F|G)|H$. Notice, incidentally, that postorder$(F \mid G) =$ postorder(F) postorder(G), and that $F \mid (GH) = (F \mid G)H$ when G is nonnull.

17. Any nonnull forest can be written $F = (G \mid \cdot)H$, where $\cdot$ denotes the 1-node forest; then $F^R = H^R(G^R \mid \cdot)$ and $F^T = (H^T \mid \cdot)G^T$. In particular we cannot have $F^R = F^T$ unless H is the null forest Λ, since the first tree of H^R can't be $H^T \mid \cdot$; and G must then also be Λ. Furthermore $F = F^T$ if and only if $G = H^T$. In that case we cannot also have $F^R = F^{RT}$ unless $G = \Lambda$; the first tree of G^{TR} would otherwise have more nodes than G itself.

It appears to be true that we cannot have $F^{RT} = F^{TR}$ unless $F = F^R$. Under that assumption, $F^{RT} = F^{TR}$ if and only if F and F^T are both self-conjugate. David Callan has discovered two infinite families of such forests, with parameters $i, j, k \geq 0$:



(In these examples, $i = 2$, $j = 3$, and $k = 5$.) Are there any other cases?

18. The $C_{15} = 9{,}694{,}845$ forests are partitioned into 20,982 classes. The largest is a cycle of length 58,968, one of whose elements is ((()(()))())()((()(()))())()(). The shortest are six two-element classes (corresponding to exercise 17), consisting of

()()()()()()()()()()()()()()(),   ()()()()((()()()()()))()()()(),
()()()(((((()()()())))))()()(),   ()()((((((((()()()))))))))()(),
()((()())((()(())()))(()()))(),   ()(((((((((((()()))))))))))))(),

and their transposes. The somewhat strange strings (((((((())))))))()()()()()()(), ()()()()()()()(((((((()))))))), and ((((((()()()()()()()()))))))) each have wedge-shaped binary trees and form a unique class of size 3. The path that runs from ()((()(()()))(())((()())()))() to ((()())(()())(())(()())(()())) has 3120 elements, one of which is (2). According to the conjecture in answer 19, the shortest possible cycle has length 6; when $n = 15$ there are 66 such cycles. (The next-shortest cycle, which is unique, has length 10 and includes ()(()()())()((((())()))((()))).)

19. The transformation from F_j to F_{j+1} by Algorithm P can be paraphrased as follows: "Find the last node in preorder, say x, that has a left sibling, say y. Remove x from its family and make it the new rightmost child of y. And if $x < n$, change all of x's descendants $x + 1, \ldots, n$ into trivial one-node trees."

The transformation that takes F_j^R into F_{j+1}^R can therefore be stated as follows, if we recall that the kth node of F_j in preorder is the kth-from-last node of F_j^R in postorder: "Find the first node in preorder, say x, that has a right sibling, say y. Remove x from its family and make it the new leftmost child of y. And if $x < n$, change all of x's descendants $x + 1, \ldots, n$ into trivial one-node trees."

Similarly, we can paraphrase the transformation from G_j to G_{j+1} that is specified by Algorithm B: "Find j, the root of the leftmost nontrivial tree; then find k, its rightmost child. Remove k and its descendants from j's family, and insert them between j and j's right sibling. Finally, if $j > 1$, make j and its siblings all children of $j - 1$, and $j - 1$ a child of $j - 2$, etc."

When this transformation changes the left-sibling/right-child representation from G_j^{RT} to G_{j+1}^{RT} (see exercise 15), it turns out to be identical to the transformation that takes F_j^R to F_{j+1}^R in the left-child/right-sibling representation. Therefore $G_j^{RT} = F_j^R$, because this identity clearly holds when $j = 1$.

(It follows that the sequence of tables $e_1 \ldots e_{n-1}$ for the binary trees generated by Algorithm B is exactly the sequence of tables $d_{n-1} \ldots d_1$ for the parenthesis strings generated by Algorithm P; this phenomenon is illustrated in Tables 1 and 2.)

The forest F^{RTR} is called the *dual* of F; see exercise 26(f). Several symmetries between lists of forests have been explored by M. C. Er, *Comp. J.* **32** (1989), 76–85.

20. (a) This assertion, which generalizes Lemma 2.3.1P, is readily proved by induction.
(b) The following procedure is, in fact, almost identical to Algorithm P:

T1. [Initialize.] Set $b_{3k-2} \leftarrow 3$ and $b_{3k-1} \leftarrow b_{3k} \leftarrow 0$ for $1 \le k \le n$; also set $b_0 \leftarrow b_N \leftarrow 0$ and $m \leftarrow N-3$, where $N = 3n+1$.

T2. [Visit.] Visit $b_1 \ldots b_N$. (Now $b_m = 3$ and $b_{m+1} \ldots b_N = 0 \ldots 0$.)

T3. [Easy case?] Set $b_m \leftarrow 0$. If $b_{m-1} = 0$, set $b_{m-1} \leftarrow 3$, $m \leftarrow m-1$, and go to T2.

T4. [Find j.] Set $j \leftarrow m-1$ and $k \leftarrow N-3$. While $b_j = 3$, set $b_j \leftarrow 0$, $b_k \leftarrow 3$, $j \leftarrow j-1$, and $k \leftarrow k-3$.

T5. [Increase b_j.] Terminate the algorithm if $j = 0$. Otherwise set $b_j \leftarrow 3$, $m \leftarrow N-3$, and return to T2. ▮

[See S. Zaks, *Theoretical Comp. Sci.* **10** (1980), 63–82. In that article, Zaks pointed out that it is even easier to generate the sequence $z_1 \ldots z_n$ of indices j such that $b_j = 3$, using an algorithm virtually identical to the answer to exercise 2, because a valid ternary tree combination $z_1 \ldots z_n$ is characterized by the inequalities $z_{k-1} < z_k \le 3k-2$.]

21. For this problem we can essentially combine Algorithm P with Algorithm 7.2.1.2L. We shall assume for convenience that $n_t > 0$ and $n_1 + \cdots + n_t > 1$.

G1. [Initialize.] Set $l \leftarrow N$. Then for $j = t, \ldots, 2, 1$ (in this order), do the following operations n_j times: Set $b_{l-j} \leftarrow j$, $b_{l-j+1} \leftarrow \cdots \leftarrow b_{l-1} \leftarrow 0$, and $l \leftarrow l-j$. Finally set $b_0 \leftarrow b_N \leftarrow c_0 \leftarrow 0$ and $m \leftarrow N-t$.

G2. [Visit.] Visit $b_1 \ldots b_N$. (At this point $b_m > 0$ and $b_{m+1} = \cdots = b_N = 0$.)

G3. [Easy case?] If $b_{m-1} = 0$, set $b_{m-1} \leftarrow b_m$, $b_m \leftarrow 0$, $m \leftarrow m-1$, and return to G2.

G4. [Find j.] Set $c_1 \leftarrow b_m$, $b_m \leftarrow 0$, $j \leftarrow m-1$, and $k \leftarrow 1$. While $b_j \ge c_k$, set $k \leftarrow k+1$, $c_k \leftarrow b_j$, $b_j \leftarrow 0$, and $j \leftarrow j-1$.

G5. [Increase b_j.] If $b_j > 0$, find the smallest $l \ge 1$ such that $b_j < c_l$, and interchange $b_j \leftrightarrow c_l$. Otherwise, if $j > 0$, set $b_j \leftarrow c_1$ and $c_1 \leftarrow 0$. Otherwise terminate.

G6. [Reverse and spread out.] Set $j \leftarrow k$ and $l \leftarrow N$. While $c_j > 0$, set $b_{l-c_j} \leftarrow c_j$, $l \leftarrow l - c_j$, and $j \leftarrow j-1$. Then set $m \leftarrow N - c_k$ and go back to G2. ▮

This algorithm assumes that $N > n_1 + 2n_2 + \cdots + tn_t$. [See *SICOMP* **8** (1979), 73–81.]

22. Note first that d_1 can be increased if and only if $r_1 = 0$ in the linked representation. Otherwise the successor of $d_1 \ldots d_{n-1}$ is obtained by finding the smallest j with $d_j > 0$ and setting $d_j \leftarrow 0$, $d_{j+1} \leftarrow d_{j+1} + 1$. We may assume that $n > 2$.

K1. [Initialize.] Set $l_k \leftarrow k+1$ and $r_k \leftarrow 0$ for $1 \le k < n$; also set $l_n \leftarrow r_n \leftarrow 0$.

K2. [Visit.] Visit the binary tree represented by $l_1 l_2 \ldots l_n$ and $r_1 r_2 \ldots r_n$.

K3. [Easy cases?] Set $y \leftarrow r_1$. If $y = 0$, set $r_1 \leftarrow 2$, $l_1 \leftarrow 0$, and return to K2. Otherwise if $l_1 = 0$, set $l_1 \leftarrow 2$, $r_1 \leftarrow r_2$, $r_2 \leftarrow l_2$, $l_2 \leftarrow 0$, and return to K2. Otherwise set $j \leftarrow 2$ and $k \leftarrow 1$.

K4. [Find j and k.] If $r_j > 0$, set $k \leftarrow j$ and $y \leftarrow r_j$. Then if $j \ne y-1$, set $j \leftarrow j+1$ and repeat this step.

K5. [Shuffle subtrees.] Set $l_j \leftarrow y$, $r_j \leftarrow r_y$, $r_y \leftarrow l_y$, and $l_y \leftarrow 0$. If $j = k$, go to K2.

K6. [Shift subtrees.] Terminate if $y = n$. Otherwise, while $k > 1$, set $k \leftarrow k - 1$, $j \leftarrow j - 1$, and $r_j \leftarrow r_k$. Then while $j > 1$, set $j \leftarrow j - 1$ and $r_j \leftarrow 0$. Return to K2. ▌

(See the analysis in exercise 45. Korsh [to appear] has shown that this algorithm can be extended in an interesting way to t-ary trees; and he has also found an efficient t-ary generalization of Algorithm B.)

23. (a) Since z_n begins at $2n - 1$ and goes back and forth C_{n-1} times, it ends at $2n - 1 - (C_{n-1} \bmod 2)$, when $n > 1$. Furthermore the final value of z_j is constant for all $n \geq j$. Thus the final string $z_1 z_2 \ldots$ is 1 2 5 6 9 11 13 14 17 19 …, containing all odd numbers $< 2n$ except 3, 7, 15, 31, … .

(b) Similarly, the preorder permutation that characterizes the final tree is $2^k\ 2^{k-1} \ldots 1\ 3\ 5\ 6\ 7\ 9\ 10 \ldots$, where $k = \lfloor \lg n \rfloor$. Forestwise, node 2^j is the parent of 2^{j-1} nodes $\{2^{j-1}, 2^{j-1}+1, \ldots, 2^j - 1\}$, for $1 < j \leq k$, and the trees $\{2^k + 1, \ldots, n\}$ are trivial.

Note: If Algorithm N is restarted at step N2 after it has terminated, it will generate the same sequence, but backwards. Algorithm L has the same property.

24. $l_0 l_1 \ldots l_{15} = 2\,0\,1\,0\,3\,0\,0\,6\,5\,0\,8\,0\,0\,12\,11\,4$; $r_1 \ldots r_{15} = 0\,15\,0\,10\,7\,0\,0\,9\,0\,14\,13\,0\,0\,0\,0$; $k_1 \ldots k_{15} = 0\,0\,2\,2\,4\,5\,5\,4\,8\,4\,10\,11\,11\,10\,2$; $q_1 \ldots q_{15} = 2\,1\,15\,4\,3\,10\,8\,5\,7\,6\,9\,14\,11\,13\,12$; and $u_1 \ldots u_{15} = 1\,2\,3\,1\,0\,0\,5\,0\,3\,1\,0\,0\,1\,0\,1\,0$. (If nodes of the forest F are numbered in postorder, k_j is the left sibling of j; or, if j is the leftmost child of p, $k_j = k_p$. Stated another way, k_j is the parent of j in the forest F^{TR}. And k_j is also $j - 1 - u_{n+1-j}$, the number of elements to the left of j in $q_1 \ldots q_n$ that are less than j.)

25. Taking a cue from Algorithms N and R, we want to extend each $(n-1)$-node tree to a list of two or more n-node trees. The idea in this case is to make n a child of $n - 1$ in the binary tree at the beginning and the end of every such list. The following algorithm uses additional link fields p_j and s_j, where p_j points to the parent of j in the forest, and s_j points to j's left sibling or to j's rightmost sibling if j is the leftmost in its family. (These pointers p_j and s_j are, of course, not the same as the permutations $p_1 \ldots p_n$ in Table 1 or the scope coordinates $s_1 \ldots s_n$ in Table 2. In fact $s_1 \ldots s_n$ is the permutation λ of exercise 33 below.)

M1. [Initialize.] Set $l_j \leftarrow j+1$, $r_j \leftarrow 0$, $s_j \leftarrow j$, $p_j \leftarrow j-1$, and $o_j \leftarrow -1$ for $1 \leq j \leq n$, except that $l_n \leftarrow 0$.

M2. [Visit.] Visit $l_1 \ldots l_n$ and $r_1 \ldots r_n$. Then set $j \leftarrow n$.

M3. [Find j.] If $o_j > 0$, set $k \leftarrow p_j$ and go to M5 if $k \neq j - 1$. If $o_j < 0$, set $k \leftarrow s_j$ and go to M4 if $k \neq j - 1$. If $k = j - 1$ in either case, set $o_j \leftarrow -o_j$, $j \leftarrow j - 1$, and repeat this step.

M4. [Transfer down.] (At this point k is j's left sibling, or the rightmost member of j's family.) If $k \geq j$, terminate if $j = 1$, otherwise set $x \leftarrow p_j$, $l_x \leftarrow 0$, $z \leftarrow k$, and $k \leftarrow 0$ (thereby detaching node j from its parent and heading for the top level). But if $k < j$, set $x \leftarrow p_j + 1$, $z \leftarrow s_x$, $r_k \leftarrow 0$, and $s_x \leftarrow k$ (thereby detaching node j from k and going down a level). Then set $x \leftarrow k + 1$, $y \leftarrow s_x$, $s_x \leftarrow z$, $s_j \leftarrow y$, $r_y \leftarrow j$, and $x \leftarrow j$. While $x \neq 0$, set $p_x \leftarrow k$ and $x \leftarrow r_x$. Return to M2.

M5. [Transfer up.] (At this point k is j's parent.) Set $x \leftarrow k + 1$, $y \leftarrow s_j$, $z \leftarrow s_x$, $s_x \leftarrow y$, and $r_y \leftarrow 0$. If $k \neq 0$, set $y \leftarrow p_k$, $r_k \leftarrow j$, $s_j \leftarrow k$, $s_{y+1} \leftarrow z$, and $x \leftarrow j$; otherwise set $y \leftarrow j - 1$, $l_y \leftarrow j$, $s_j \leftarrow z$, and $x \leftarrow j$. While $x \neq 0$, set $p_x \leftarrow y$ and $x \leftarrow r_x$. Return to M2. ▌

Running time notes: We can argue as in exercise 44 that step M3 costs $2C_n+3(C_{n-1}+\cdots+C_1)$ mems, and that steps M4 and M5 together cost $8C_n-2(C_{n-1}+\cdots+C_1)$, plus twice the number of times $x \leftarrow r_x$. The latter quantity is difficult to analyze precisely; for example, when $n=15$ and $j=6$, the algorithm sets $x \leftarrow r_x$ exactly $(1,2,3,4,5,6)$ times in respectively $(45,23,7,9,2,4)$ cases. But heuristically the average number of times $x \leftarrow r_x$ should be approximately $2-2^{j-n}$ when j is given, therefore about $(2C_n-(C_n-C_{n-1})-(C_{n-1}-C_{n-2})/2-(C_{n-2}-C_{n-3})/4-\cdots)/C_n \approx 8/7$ overall. Empirical tests confirm this predicted behavior, showing that the total cost per tree approaches $265/21 \approx 12.6$ mems as $n \to \infty$.

26. (a) The condition is clearly necessary. And if it holds, we can uniquely construct F: Node 1 and its siblings are the roots of the forest, and their descendants are defined inductively by noncrossing partitions. (In fact, we can compute the depth coordinates $c_1 \ldots c_n$ directly from Π's restricted growth sequence $a_1 \ldots a_n$: Set $c_1 \leftarrow 0$ and $i_0 \leftarrow 0$. For $2 \le j \le n$, if $a_j > \max(a_1,\ldots,a_{j-1})$, set $c_j \leftarrow c_{j-1}+1$ and $i_{a_j} \leftarrow c_j$, otherwise set $c_j \leftarrow i_{a_j}$.)

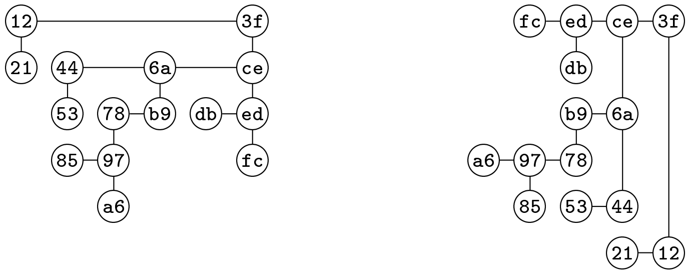
(b) If Π and Π' satisfy the noncrossing condition, so does their greatest common refinement $\Pi \vee \Pi'$, so we can proceed as in exercise 7.2.1.5–12(a).

(c) Let $x_1, \ldots, x_m$ be the children of some node in F, and let $1 \le j < k \le m$. Form F' by removing $x_{j+1}, \ldots, x_k$ from their family and reattaching them as children of $x_{j+1}-1$, the rightmost descendant of x_j.

(d) Obvious, by (c). Thus the forests are ranked from bottom to top by the number of nonleaf nodes they contain (which is one less than the number of blocks in Π).

(e) Exactly $\sum_{k=0}^{n} e_k(e_k-1)/2$, where $e_0 = n-e_1-\cdots-e_n$ is the number of roots.

(f) Dualization is similar to the transposition operation in exercise 12, but we use left-sibling and right-child links instead of left-child and right-sibling, and we transpose about the *minor* diagonal:



("Right" links now point downward. Notice that j is the rightmost child of k in F if and only if j is the left sibling of k in F^D. Preorder of F^D reverses the preorder of F, just as postorder of F^T reverses postorder of F.)

(g) From (f) we can see that F' covers F if and only if F^D covers F'^D. (Therefore F^D has $n+1-k$ leaves if F has k.)

(h) $F \barwedge F' = (F^D \veebar F'^D)^D$.

(i) No. If it did, equality would necessarily hold, by duality. But, for example, $0101 \barwedge 0121 = 0000$ and $0101 \veebar 0121 = 0123$, while $\text{leaves}(0101)+\text{leaves}(0121) \ne \text{leaves}(0000)+\text{leaves}(0123)$.

[Noncrossing partitions were first considered by H. W. Becker in *Math. Mag.* **22** (1948), 23–26. G. Kreweras proved in 1971 that they form a lattice; see the references in answer 2.3.4.6–3.]

27. (a) This assertion is equivalent to exercise 2.3.3–19.

(b) If we represent a forest by right-child and left-sibling links, preorder corresponds to inorder of the binary tree (see exercise 2.3.2–5), and s_j is the size of node j's right subtree. Rotation to the left at any nonleaf of this binary tree decreases exactly one of the scope coordinates, and the amount of decrease is as small as possible consistent with a valid table $s_1 \ldots s_n$. Therefore F' covers F if and only if F is obtained from F' by such a rotation. (Rotation in the left-child/right-sibling representation is similar, but with respect to postorder.)

(c) Dualization preserves the covering relation but exchanges left with right.

(d) $F \top F' = (F^D \perp F'^D)^D$. Equivalently, as noted in exercise 6.2.3–32, we can independently minimize the left-subtree sizes.

(e) The covering transformation in answer 26(c) obviously makes $s_j \le s'_j$ for all j.

(f) True, because $F \barwedge F' \ltimes F \dashv F \perp F'$ and $F \barwedge F' \ltimes F' \dashv F \perp F'$.

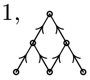
(g) False; for example, $0121 \veebar 0122 = 0123$ and $0121 \top 0122 = 0122$. (But we do have $F \top F' \dashv F \veebar F'$, by taking duals in (f).)

(h) The longest path, of length $\binom{n}{2}$, repeatedly decreases the rightmost nonzero s_j by 1. The shortest, of length $n-1$, repeatedly sets the leftmost nonzero s_j to 0.

Answer 6.2.3–32 gives many references to the literature of Tamari lattices.

28. (a) Just compute $\min(c_1, c'_1) \ldots \min(c_n, c'_n)$ and $\max(c_1, c'_1) \ldots \max(c_n, c'_n)$, because $c_1 \ldots c_n$ is a valid sequence if and only if $c_1 = 0$ and $c_j \le c_{j-1} + 1$ for $1 < j \le n$.

(b) Obvious because of (a). *Note:* The elements of any distributive lattice can be represented as the order ideals of some partial ordering. In the case of Fig. 41, that partial ordering is shown at the right, and a similar triangular grid with sides of length $n-2$ yields Stanley's lattice of order n.



(c) Take a node k of F that has a left sibling, j. Remove k from its family and place it as a new right child of j, followed by its former children as new children of j; the former children of k retain their own descendants. (This operation corresponds to changing)(to () in a nested parenthesis string. Thus a "perfect" Gray code for parentheses corresponds to a Hamiltonian path in the cover graph of Stanley's lattice. Exactly 38 such paths exist when $n = 4$, namely $(8, 6, 6, 8, 4, 6)$ from 0123 to $(1001, 0010, 0012, 0100, 0111, 0120)$ respectively.)

(d) True, because the cover relation in (c) is left-right symmetric. (We have $F \subseteq F'$ if and only if $w_j \le w'_j$ for $0 \le j \le 2n$, where the worm depths w_j are defined in exercise 10. If $w_0 \ldots w_{2n}$ is the worm walk of F, its reverse $w_{2n} \ldots w_0$ is the worm walk of F^R. Notice that the cover relation changes just one coordinate w_j. One can compute $F \cap F'$ and $F \cup F'$ by taking min and max of the w's instead of the c's.)

(e) See exercise 9. (Thus $F \perp F' \subseteq F \cap F'$, etc., as in exercise 27(f).)

Notes: Stanley introduced this lattice in *Fibonacci Quarterly* **13** (1975), 222–223. Since three important lattices are defined on the same elements, we need three notations for the different orderings; the symbols $\ltimes$, $\dashv$, and $\subseteq$ adopted here are intended to be reminiscent of the names of Kreweras, Tamari, and Stanley (who is Стенли in Russia).

29. If we paste six regular pentagons together, we get 14 vertices whose coordinates after suitable rotation and scaling are respectively

$$p_{1010} = p^{-}_{0000} = p^{*}_{3000} = p^{*-}_{2100} = (-1, \sqrt{3}, 2/\phi);$$
$$p_{0010} = p^{*}_{3100} = (\phi^{-2}, \sqrt{3}\,\phi, 0); \quad p_{3010} = p^{-}_{0100} = (0, 0, 2); \quad p_{3210} = p^{-}_{0200} = (2, 0, 2/\phi);$$
$$p_{0210} = p^{*}_{3200} = (\sqrt{5}, \sqrt{3}, 0); \quad p_{1000} = p^{*}_{2000} = (-\phi^2, \sqrt{3}/\phi, 0);$$

here $(x,y,z)^*$ means $(x,-y,z)$ and $(x,y,z)^-$ means $(x,y,-z)$. But then the three 4-edged "faces" are not squares; in fact, they don't even lie in a plane.

(One can however get a similar-looking solid, with true squares but irregular pentagons, by gluing together two suitable tetrahedra and lopping off the three glued-together corners. Alternative sets of coordinates for the associahedron, of substantial mathematical interest but less appealing to the eye, are discussed by Günter Ziegler in his *Lectures on Polytopes* (New York: Springer, 1995), example 9.11.)

30. (a) $\bar f_{n-1}\ldots \bar f_1\,0$, because internal node j in symmetric order has a nonempty right subtree if and only if internal node $j+1$ in symmetric order has an empty left subtree.

(b) In general if the footprint were $1^{p_1}0^{q_1+1}1^{p_2+1}0^{q_2+1}\ldots 1^{p_k+1}0^{q_k+1}$, we would want to count all binary trees whose nodes in symmetric order have the specification $R^{p_1}NL^{q_1}BR^{p_2}NL^{q_2}B\ldots R^{p_k}NL^{q_k}$, where B means "both subtrees are nonempty," R means "the right subtree is nonempty but not the left," L means "the left subtree is nonempty but not the right," and N means "neither subtree is nonempty." This number in general is

$$\binom{p_1+q_1}{p_1}\binom{p_2+q_2}{p_2}\ldots\binom{p_k+q_k}{p_k}C_{k-1},$$

and in particular it is $\binom{1+0}{1}\binom{0+0}{0}\binom{1+0}{1}\binom{5+3}{5}\binom{0+0}{0}\binom{0+0}{0}\binom{0+2}{0}\binom{0+0}{0}\binom{1+2}{1}C_8 = 240240$.

(c) $d_j = 0$ if and only if $c_{j+1} > c_j$, by exercise 3.

(d) In general, the footprint of $F \perp F'$ is $f_1\ldots f_n \wedge f'_1\ldots f'_n$, by exercise 27(a); the footprint of $F \top F'$ is $f_1\ldots f_n \vee f'_1\ldots f'_n$, by (a) and exercise 27(d).

[The fact that complements always exist in the Tamari lattice is due to H. Lakser; see G. Grätzer, *General Lattice Theory* (1978), exercise I.6.30.]

31. (a) 2^{n-1}; see exercise 6.2.2–5.

(b) $c_1 \le \cdots \le c_n$; $d_1, \ldots, d_{n-1} \le 1$; $e_j > 0$ implies $e_j + \cdots + e_n = n-j$; $k_{j+1} \le k_j + 1$; $p_1 \le \cdots \le p_j \ge \cdots \ge p_n$ for some j; $s_j > 0$ implies $s_j = n-j$; $u_1 \ge \cdots \ge u_n$; $z_{j+1} \le z_j + 2$. (Other constraints, which apply in general, whittle down the number of possibilities to 2^{n-1} in each case. For example, $u_1\ldots u_n$ must be a valid sequence of scope coordinates.)

(c) True in only n cases out of 2^{n-1}. (But F^T *is* degenerate.)

(d) The degenerate forest with footprint $f_1\ldots f_n$ has $c_{j+1} = c_j + f_j$. Elements $j < k$ are siblings if and only if $f_j = f_{j+1} = \cdots = f_{k-1} = 0$. Thus if F'' is the degenerate forest with footprint $f_1\ldots f_n \wedge f'_1\ldots f'_n$, then $F'' \ltimes F$ and $F'' \ltimes F'$; hence $F'' \ltimes F \barwedge F' \dashv F \perp F'$. And we also have $F \perp F' \dashv F''$ by (b). A similar argument proves that $F \veebar F' = F \top F'$ is the degenerate forest with footprint $f_1\ldots f_n \vee f'_1\ldots f'_n$.

Thus, when the Kreweras and Tamari lattices are restricted to degenerate forests, they become identical to the Boolean lattice of subsets of $\{1,\ldots,n-1\}$. [This result, in the case of Tamari lattices, is due to George Markowsky, *Order* **9** (1992), 265–290, whose paper also shows that Tamari lattices enjoy many further properties.]

32. Suppose F and F' have scope coordinates $s_1\ldots s_n$ and $s'_1\ldots s'_n$. Call index j *frozen* if $s_j < s'_j$ or $j = 0$. We want to specify the values of the frozen coordinates and maximize the others. Let $s_0 = n$, and for $0 \le k \le n$ let

$$s''_k = s_j - k + j, \quad \text{where } j = \max\{i \mid 0 \le i \le k,\ i \text{ is frozen, and } i + s_i \ge k\}.$$

Since $s_k \le s_j - (k-j)$ whenever $0 \le k - j \le s_j$, we have $s''_k \ge s_k$, with equality when k is frozen.

The scopes $s''_0 s''_1 \ldots s''_n$ correspond to a valid forest according to the condition of exercise 27(a). For if $k \geq 0$ and $0 \leq l \leq s''_k = s_j - k + j$ and $s''_{k+l} = s_{j'} - k - l + j'$, we have $s''_{k+l} + l \leq s''_k$ if $0 \leq j' - j \leq s_j$, because $s_{j'} + j' - j \leq s_j$ in that case. And we can't have $j > j'$ or $j' > j + s_j$, because $j + s_j \geq k + l \geq j'$.

Let F''' be a forest with scopes satisfying $s_k \leq s'''_k \leq s''_k$. Then $\min(s'_k, s'''_k) = s_k$, because $s_k = s''_k$ when k is frozen, otherwise $s_k = s'_k$.

Conversely, if F''' is a forest with $F' \perp F''' = F$, we must have $s_k \leq s'''_k \leq s''_k$. For $s'''_k < s_k$ would imply $s'''_k < s'_k$. And if k is minimal with $s'''_k > s''_k$, we have $s''_k = s_j - k + j$ for some frozen j with $0 \leq j \leq k$ and $j + s_j \geq k$. Then $s'''_j \geq s_j$ implies $k - j \leq s'''_j$, hence $s'''_k + k - j \leq s'''_j$. If $j < k$ we have $s'''_j \leq s''_j = s_j$, a contradiction. But $j = k$ implies $\min(s'''_k, s'_k) > s_k$.

To get the first semidistributive law, apply this principle with F replaced by $F \perp G$ and F' replaced by F; then the hypotheses $F \dashv G \dashv F''$ and $F \dashv H \dashv F''$ imply that $F \dashv G \top H \dashv F''$. The second semidistributive law follows by taking duals in the first.

(Ralph Freese suggests calling F'' the *pseudo-complement* of F' over F.)

33. (a) Let $k\lambda =$ `LLINK[`k`]` if `LLINK[`k`]` $\neq 0$, otherwise `RLINK[`$k-1$`]` if $k \neq 1$, otherwise the root of the binary tree. This rule defines a permutation because $k\lambda = j$ if and only if $k = \mathrm{parent}(j) + [j \text{ is a right child}]$, or $k = 1$ and j is the root. Also $k\lambda \geq k$ when `LLINK[`k`]` $= 0$ and $k\sigma\lambda \leq k$ when `RLINK[`k`]` $= 0$. [For a generalization to t-ary trees, see P. H. Edelman, *Discrete Math.* **40** (1982), 171–179.]

(b) Using the representation of (2) in answer 26(f), we see that $\lambda(F)$ is $(3\,1)(2)$ $(12\,6\,4)(5)(11\,7)(14\,13)(9\,8)(15)(10)$ in that case. In general the cycles are the families of the forest, in decreasing order within each cycle; nodes are numbered in preorder. [See Dershowitz and Zaks, *Discrete Math.* **62** (1986), 215–218.]

(c) $\lambda(F^D) = \rho\sigma\lambda\rho$, where ρ is the "flip" permutation $(1\,n)(2\,n{-}1)\ldots$, because the dual forest interchanges `LLINK` $\leftrightarrow$ `RLINK` and flips the preorder numbering.

(d) The cycle breakup $(x_j\, x_k)(x_1 \ldots x_m) = (x_1 \ldots x_j x_{k+1} \ldots x_m)(x_{j+1} \ldots x_k)$ corresponds to answer 26(c).

(e) By (d), each covering path corresponds to a factorization of $(n\, \ldots\, 2\,1)$. Let q_n denote the number of such factorizations. Then we have the recurrence $q_1 = 1$ and $q_n = \sum_{l=1}^{n-1}(n-l)\binom{n-2}{l-1} q_l q_{n-l}$, because there are $n - l$ choices with $k - j = l$ by which the first transposition breaks the cycle into parts of sizes l and $n - l$, then $\binom{n-2}{l-1}$ ways to interleave the subsequent factors. The solution is $q_n = n^{n-2}$, because

$$\sum_{l=1}^{n-1}\binom{n-1}{l} l^{l-1}(y-l)^{n-1-l} = \lim_{x\to 0}\sum_{l=1}^{n-1}\binom{n-1}{l}(x+l)^{l-1}(y-l)^{n-1-l}$$
$$= \lim_{x\to 0}\frac{(x+y)^{n-1} - y^{n-1}}{x} = (n-1)y^{n-1}.$$

[See J. Dénes, *Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei* **4** (1959), 63–70. It is natural to seek a correspondence between factorizations and labeled free trees, since there also happen to be n^{n-2} of the latter. Perhaps the simplest is the following, given $(1\,2\ldots n) = (x_1\, y_1)\ldots(x_{n-1}\, y_{n-1})$ where $x_j < y_j$: Suppose the cycle containing x_j and y_j in $(x_j\, y_j)\ldots(x_{n-1}\, y_{n-1})$ is $(z_1\, \ldots\, z_m)$, where $z_1 < \cdots < z_m$. If $y_j = z_m$, let $a_j = z_1$, otherwise let $a_j = \min\{z_i \mid z_i > x_j\}$. Then one can show that $a_1 \ldots a_{n-1}$ is a "wake-up sequence" for parking $n - 1$ cars, and exercise 6.4–31 connects it to free trees.]

34. Each covering path from bottom to top is equivalent to a Young tableau of shape $(n-1, n-2, \ldots, 1)$, so we can use Theorem 5.1.4H. (See exercise 5.3.4–38.)

[The enumeration of such paths in Tamari lattices remains mysterious; the relevant sequence is 1, 1, 2, 9, 98, 2981, 340549, ...]

35. Multiply by $n+1$, then see *AMM* **97** (1990), 626–630.

36. We might as well generalize to t-ary trees for arbitrary $t \ge 1$, by making obvious amendments to steps T1–T5. Let $C_n^{(t)}$ be the number of t-ary trees with n internal nodes; thus $C_n = C_n^{(2)}$ and $C_n^{(t)} = ((t-1)n+1)^{-1}\binom{tn}{n}$. If h of the degrees b_j are changed between visits, we have $h \ge x$ in $C_{n-x}^{(t)}$ cases. So the easy case occurs with probability $1 - C_{n-1}^{(t)}/C_n^{(t)} \approx 1-(t-1)^{t-1}/t^t$, and the average number of times $b_j \leftarrow 0$ in step T4 is $(C_{n-1}^{(t)} + \cdots + C_1^{(t)})/C_n^{(t)} \approx (t-1)^{t-1}/(t^t-(t-1)^{t-1})$, or $4/23$ when $t=3$.

Indeed, we can also study the t-ary recursive structure $A_{pq}^{(t)} = 0\,A_{p(q-1)}^{(t)}, t\,A_{(p-1)q}^{(t)}$ when $0 \le (t-1)p \le q \ne 0$, generalizing (5). The number of such degree sequences, $C_{pq}^{(t)}$, satisfies the recurrence (21) except that $C_{pq}^{(t)} = 0$ when $p<0$ or $(t-1)p > q$. The general solution is

$$C_{pq}^{(t)} \;=\; \frac{q-(t-1)p+1}{q+1}\binom{p+q}{p} \;=\; \binom{p+q}{p} - (t-1)\binom{p+q}{p-1},$$

and we have $C_n^{(t)} = C_{n((t-1)n)}^{(t)}$. The triangle for $t=3$ begins as shown at the right.

1				
1				
1	1			
1	2			
1	3	3		
1	4	7		
1	5	12	12	
1	6	18	30	
1	7	25	55	55
1	8	33	88	143

37. The basic lexicographic recursion for all such forests is

$$\begin{aligned} A(n_0, n_1, \ldots, n_t) \;=\; & 0\,A(n_0-1, n_1, \ldots, n_t), \\ & 1\,A(n_0, n_1-1, \ldots, n_t),\; \ldots,\; t\,A(n_0, n_1, \ldots, n_t-1) \end{aligned}$$

when $n_0 > n_2 + 2n_3 + \cdots + (t-1)n_t$ and $n_1, \ldots, n_t \ge 0$; otherwise $A(n_0, n_1, \ldots, n_t)$ is empty, except that $A(0, \ldots, 0) = \epsilon$ is the sequence consisting of the empty string alone. Step G1 computes the first entry of $A(n_0, \ldots, n_t)$. We want to analyze five quantities:

- C, the number of times G2 is executed (the total number of forests);
- E, the number of times G3 goes to G2 (the number of easy cases);
- K, the number of times G4 moves some b_i into list c;
- L, the number of times G5 compares b_j with some c_i;
- Z, the number of times G5 sets $c_1 \leftarrow 0$.

Then the loop in step G6 sets $b_{l-c_j} \leftarrow c_j$ a total of $K - Z - n_1 - \cdots - n_t$ times.

Let n be the vector $(n_0, n_1, \ldots, n_t)$, and let e_j be the unit vector with 1 in coordinate position j. Let $|n| = n_0 + n_1 + \cdots + n_t$ and $\|n\| = n_1 + 2n_2 + \cdots + tn_t$. Using this notation we can rewrite the basic recurrence above in the convenient form

$$A(n) \;=\; 0\,A(n-e_0),\; 1\,A(n-e_1),\; \ldots,\; t\,A(n-e_t) \qquad \text{when } |n| > \|n\|.$$

Consider the general recurrence relation

$$F(n) \;=\; f(n) + \Bigl(\sum_{j=0}^{t} F(n-e_j)\Bigr)\bigl[|n| > \|n\|\bigr],$$

with $F(n) = 0$ whenever the vector n has a negative component. If $f(n) = [|n| = 0]$, then $F(n) = C(n)$ is the total number of forests. Answer 2.3.4.4–32 tells us that

$$C(n) \;=\; \frac{(|n|-1)!\,(|n|-\|n\|)}{n_0!\,n_1!\ldots n_t!} \;=\; \sum_{j=0}^{t}(1-j)\binom{|n|-1}{n_0,\ldots,n_{j-1},n_j-1,n_{j+1},\ldots,n_t},$$

generalizing the formula for $C_{pq}^{(t)}$ in answer 26 (which is the case $n_0 = (t-1)q+1$ and $n_t = p$). Similarly, we obtain recurrences for the other quantities $E(n)$, $K(n)$, $L(n)$, and $Z(n)$ needed in our analysis by choosing other kernel functions $f(n)$:

$f(n) = \big[|n| = n_0+1 \text{ and } n_0 > \|n\|\big]$	yields	$F(n) = E(n)$;
$f(n) = \big[|n| > n_0\big]$	yields	$F(n) = E(n) + K(n)$;
$f(n) = \big[|n| = \|n\|+1\big]$	yields	$F(n) = C(n) + K(n) - Z(n)$;
$f(n) = \sum_{1\le j<k\le t} n_j[n_k>0]$	yields	$F(n) = L(n)$.

The symbolic methods of exercise 2.3.4.4–32 do not seem to yield quick solutions to these more general recurrences, but we can readily establish the value of $C - E$ by noting that $b_m + m < N$ in step G2 if and only if the previous step was G3. Therefore

$$C(n) - E(n) \;=\sum_{j=1}^{t} C(n-f_j), \qquad \text{where } f_j = e_j - (j-1)e_0;$$

this sum counts the subforests in which $n_1 + \cdots + n_t$, the number of internal (nonleaf) nodes, has decreased by 1. Similarly we can let

$$C^{(x)}(n) = \sum\{C(n - i_1f_1 - \cdots - i_tf_t) \mid i_1 + \cdots + i_t = x\}$$

be the number of subforests having $n_1 + \cdots + n_t - x$ internal nodes. Then we have

$$K(n) - Z(n) = \sum_{x=1}^{|n|} C^{(x)}(n),$$

a formula analogous to (20), because $k - [b_j = 0] \ge x \ge 1$ in step G5 if and only if $b_{m-x} > 0$ and $b_{m-x+1} \ge \cdots \ge b_m$. Such preorder degree strings are in one-to-one correspondence with the forests of $C^{(x)}(n)$ if we remove $b_{m-x+1}\ldots b_m$ and an appropriate number of trailing 0s from the string $b_1 \ldots b_N$.

From these formulas we can conclude that the Zaks–Richards algorithm needs only $O(1)$ operations per forest visited, whenever $n_1 = n_2 + \cdots + n_t + O(1)$, because $C(n-f_j)/C(n) = n_j n_0^{\underline{j-1}}/(|n|-1)^{\underline{j}} \le 1/4 + O(|n|^{-1})$ when $j > 1$. Indeed, the value of K is quite small in nearly all cases of practical interest. However, the algorithm can be slow when n_1 is large. For example, if $t = 1$, $n_0 = m + r + 1$, and $n_1 = m$, the algorithm essentially computes all r-combinations of $m + r$ things; then $C(n) = \binom{m+r}{r}$ and $K(n) - Z(n) = \binom{m+r}{r+1} = \Omega(mC(n))$ when r is fixed. [To ensure efficiency in all cases, we can keep track of trailing 1s; see Ruskey and Roelants van Baronaigien, *Congressus Numerantium* **41** (1984), 53–62.]

Exact formulas for K, Z, and (especially) L do not seem to be simple, but we can compute those quantities as follows. Say that the "active block" of a forest is the right-most substring of nonzero degrees; for example, the active block of 302102021230000000 is 2123. All permutations of the active block occur equally often. Indeed, let $D(n)$ denote the sum of "trailing zeros$(\beta) - 1$" over all preorder degree strings β for forests of specification n. Then a block with n'_j occurrences of j for $1 \le j \le t$ is active in exactly $D(n - n'_1 f_1 - \cdots - n'_t f_t) + [n'_1 + \cdots + n'_t = n_1 + \cdots + n_t]$ cases. For example, given the string 3021020000, we can insert 21230000 in three places to obtain a forest with active block 2123. The contributions to K and L when the active block is flush left (not preceded by any 0s) can be computed as in exercise 7.2.1.2–6, namely

$$k(n) = w(e_{n_1}(z) \ldots e_{n_t}(z)), \qquad l(n) = w\Bigl(e_{n_1}(z) \ldots e_{n_t}(z) \sum_{1 \le i < j \le t} (n_i - z r_i(z)) r_j(z)\Bigr)$$

in the notation of that answer. Analogous contributions occur in general; therefore

$$K(n) = k(n) + \sum D(n-n')k(n'),\ L(n) = l(n) + \sum D(n-n')l(n'),\ Z(n) = \sum D(n-n'),$$

summed over all vectors n' such that $n'_j \le n_j$ for $1 \le j \le t$ and $|n'| - \|n'\| = |n| - \|n\|$ and $n'_1 + \cdots + n'_t \le n_1 + \cdots + n_t - 2$.

It remains to determine $D(n)$. Let $C(n;j)$ be the number of forests of specification $n = (n_0, \ldots, n_t)$ in which the last internal node in preorder has degree j. Then we have

$$C(n) = \sum_{j=1}^{t} C(n;j) \text{ and } C(n+e_1;1) = C(n+e_2;2) = \cdots = C(n+e_t;t) = C(n) + D(n).$$

From this infinite system of linear equations we can deduce that $C(n) + D(n)$ is

$$\sum_{i_2=0}^{n_2} \ldots \sum_{i_t=0}^{n_t} (-1)^{i_2+\cdots+i_t} \binom{i_2 + \cdots + i_t}{i_2, \ldots, i_t} C(n + (1+i_2+\cdots+i_t)e_1 - i_2 f_2 - \cdots - i_t f_t).$$

Simpler expressions would of course be desirable, if they exist.

38. Step L1 obviously uses $4n+2$ mems. Step L3 exits to L4 or L5 exactly $C_j - C_{j-1}$ times with a particular value of j; therefore it costs $2C_n + 3\sum_{j=0}^{n}(n-j)(C_j - C_{j-1}) = 2C_n + 3(C_{n-1} + \cdots + C_1 + C_0)$ mems. Steps L4 and L5 jointly cost a total of $6C_n - 6$. Therefore the entire process involves $9 + O(n^{-1/2})$ mems per visit.

39. A Young tableau of shape (q,p) and entries y_{ij} corresponds to an element of A_{pq} that has left parens in positions $p+q+1-y_{21}, \ldots, p+q+1-y_{2p}$ and right parens in positions $p+q+1-y_{11}, \ldots, p+q+1-y_{1q}$. The hook lengths are $\{q+1, q, \ldots, 1, p, p-1, \ldots, 1\} \setminus \{q-p+1\}$; so $C_{pq} = (p+q)!(q-p+1)/(p!(q+1)!)$ by Theorem 5.1.4H.

40. (a) $C_{pq} = \binom{p+q}{p} - \binom{p+q}{p-1} \equiv \binom{p+q}{p} + \binom{p+q}{p-1} = \binom{p+q+1}{p}$ (modulo 2); now use exercise 1.2.6–11. (b) By Eq. 7.1–(oo) we know that $\nu(n \wedge (n+1)) = \nu(n+1) - 1$.

41. It equals $C(wz)/(1 - zC(wz)) = 1/(1 - z - wzC(wz)) = (1 - wC(wz))/(1 - w - z)$, where $C(z)$ is the Catalan generating function (18). The first of these formulas, $C(wz) + zC(wz)^2 + z^2C(wz)^3 + \cdots$, is easily seen to be equivalent to (24). [See P. A. MacMahon, *Combinatory Analysis* **1** (Cambridge Univ. Press, 1915), 128–130.]

42. (a) Elements $a_1 \ldots a_n$ determine an entire self-conjugate nested string $a_1 \ldots a_{2n}$, and there are $C_{q(n-q)}$ possibilities for $a_1 \ldots a_n$ having exactly q right parentheses. So the answer is

$$\sum_{q=0}^{\lfloor n/2 \rfloor} C_{q(n-q)} = \sum_{q=0}^{\lfloor n/2 \rfloor} \left(\binom{n}{q} - \binom{n}{q-1}\right) = \binom{n}{\lfloor n/2 \rfloor}.$$

(b) Exactly $C_{(n-1)/2}\,[n \text{ odd}]$, because a self-transpose binary tree is determined by its left subtree. And (c) has the same answer, because F is self-dual if and only if F^R is self-transpose.

43. $C_{pq} = C_q - \binom{q-p-1}{1}C_{q-1} + \cdots = \sum_{r=0}^{q-p}(-1)^r\binom{q-p-r}{r}C_{q-r}$, by induction on $q-p$.

44. The number of mems between visits is $3j-2$ in step B3, $h+1$ in step B4, and 4 in step B5, where h is the number of times $y \leftarrow r_y$. The number of binary trees with $h \geq x$, given j and x, is $[z^{n-j-x-1}]\,C(z)^{x+3}$ when $j<n$, because we get such trees by attaching $x{+}3$ subtrees below $j{+}x{+}1$ internal nodes. Setting $x=0$ tells us that a given value of j occurs $C_{(n-j-1)(n-j+1)} = C_{n+1-j} - C_{n-j}$ times, using (24) and exercise 43. Thus $\sum j$ over all binary trees is $n + \sum_{j=1}^{n}(C_{n+1-j} - C_{n-j})j = C_n + C_{n-1} + \cdots + C_1$. Similarly, $\sum(h+1)$ is $\sum_{j=1}^{n-1}\sum_{x=0}^{n-j-1} C_{(n-j-x-1)(n-j+1)} = \sum_{j=1}^{n-1} C_{(n-j-1)(n-j+2)} = \sum_{j=1}^{n}(C_{n-j+2} - 2C_{n-j+1}) = C_{n+1} - (C_n + C_{n-1} + \cdots + C_0)$. So overall, the algorithm costs $C_{n+1} + 4C_n + 2(C_{n-1} + \cdots + C_1) + O(n) = \bigl(26/3 - 10/(3n) + O(n^{-2})\bigr)C_n$ mems.

45. Each of the easy cases in step K3 occurs C_{n-1} times, so the total cost of that step is $3C_{n-1} + 8C_{n-1} + 2(C_n - 2C_{n-1})$ mems. Step K4 fetches r_i a total of $[z^{n-i-1}]\,C(z)^{i+2} = C_{(n-i-1)n}$ times; summing for $i \geq 2$ gives $C_{(n-3)(n+1)} = C_{n+1} - 3C_n + C_{n-1}$ mems altogether in that loop. Step K5 costs $6C_n - 12C_{n-1}$. Step K6 is a bit more complicated, but one can show that the operation $r_j \leftarrow r_k$ is performed $C_n - 3C_{n-1} + 1$ times when $n > 2$, while the operation $r_j \leftarrow 0$ is performed $C_{n-1} - n + 1$ times. The total number of mems therefore comes to $C_{n+1} + 7C_n - 9C_{n-1} + n + 3 = \bigl(8.75 - 9.375/n + O(n^{-2})\bigr)C_n$.

Although this total is asymptotically worse than that of Algorithm B in answer 44, the large negative coefficient of n^{-1} means that Algorithm B actually wins only when $n \geq 58$; and n won't ever be that big.

46. (a) Going to the left from (pq) increases the area by $q-p$.

(b) The leftward steps on a path from (nn) to (00) correspond to the left parentheses in $a_1 \ldots a_{2n}$, and we have $q - p = c_k$ at the kth such step.

(c) Equivalently, $C_{n+1}(x) = \sum_{k=0}^{n} x^k C_k(x) C_{n-k}(x)$. This recurrence holds because an $(n+1)$-node forest F consists of the root of the leftmost tree together with a k-node forest F_l (the descendants of that root) and an $(n-k)$-node forest F_r (the remaining trees), and because we have

$$\text{internal path length}(F) \;=\; k + \text{internal path length}(F_l) + \text{internal path length}(F_r).$$

(d) The strings of $A_{p(p+r)}$ have the form $\alpha_0)\alpha_1)\ldots\alpha_{r-1})\alpha_r$ where each α_j is properly nested. The area of such a string is the sum over j of the area of α_j plus $r-j$ times the number of left parens in α_j.

Notes: The polynomials $C_{pq}(x)$ were introduced by L. Carlitz and J. Riordan in *Duke Math. J.* **31** (1964), 371–388; the identity in part (d) is equivalent to their formula (10.12). They also proved that

$$C_{pq}(x) \;=\; \sum_r (-1)^r x^{r(r-1)-\binom{q-p}{2}} \binom{q-p-r}{r}_{\!x} C_{q-r}(x),$$

generalizing the result of exercise 43. From part (c) we have the infinite continued fraction $C(x,z) = 1/(1 - z/(1 - xz/(1 - x^2z/(1 - \cdots))))$, which G. N. Watson proved is equal to $F(x,z)/F(x,z/x)$, where

$$F(x,z) = \sum_{n=0}^{\infty} \frac{(-1)^n\, x^{n^2}\, z^n}{(1-x)(1-x^2)\ldots(1-x^n)};$$

see *J. London Math. Soc.* **4** (1929), 39–48. We have already encountered the same generating function, slightly disguised, in exercise 5.2.1–15.

The internal path length of a forest is the "left path length" of the corresponding binary tree, namely the sum over all internal nodes of the number of left branches on the path from the root. The more general polynomial

$$C_n(x,y) = \sum x^{\text{left path length}(T)} y^{\text{right path length}(T)},$$

summed over all n-node binary trees T, seems to obey no simple additive recurrence like the one for $C_{nn}(x) = C_n(x,1)$ studied in this exercise; but we do have $C_{n+1}(x,y) = \sum_k x^k C_k(x,y) y^{n-k} C_{n-k}(x,y)$. Therefore the super generating function $C(x,y,z) = \sum_n C_n(x,y)z^n$ satisfies the functional equation $C(x,y,z) = 1 + zC(x,y,xz)C(x,y,yz)$. (The case $x = y$ was considered in exercise 2.3.4.5–5.)

47. $C_n(x) = \sum_q x^{\binom{q-p}{2}} C_{pq}(x) C_{(n-q)(n-1-p)}(x)$ for $0 \le p < n$.

48. Let $\bar{C}(z) = C(-1,z)$ in the notation of exercise 46, and let $\bar{C}(z)\bar{C}(-z) = F(z^2)$. Then $\bar{C}(z) = 1 + zF(z^2)$ and $\bar{C}(-z) = 1 - zF(z^2)$; so $F(z) = 1 - zF(z)^2$, and $F(z) = C(-z)$. It follows that $C_{pq}(-1) = [z^p]\, C(-z^2)^{\lceil (q-p)/2 \rceil}(1 + zC(-z^2))^{[q-p \text{ even}]}$, which is $(-1)^{(p/2)} C_{(p/2)(q/2-1)}[p \text{ even}]$ when q is even, $(-1)^{\lfloor p/2 \rfloor} C_{\lfloor p/2 \rfloor \lfloor q/2 \rfloor}$ when q is odd. A perfect Gray code through the strings of A_{pq} can exist only if $|C_{pq}(-1)| \le 1$, because the associated graph is bipartite (see Fig. 41); $|C_{pq}(-1)|$ is the difference between the sizes of the parts, because each perfect transposition changes $c_1 + \cdots + c_n$ by ± 1.

49. By Algorithm U with $n = 15$ and $N = 10^6$, it is `()(()())((()()))((((())()))`.

50. Make the following changes to Algorithm U: In step U1, also set $r \leftarrow 0$. In step U3, test if $a_m =$ '`)`' instead of testing if $N \le c'$. In step U4, set $r \leftarrow r + c'$ instead of $N \leftarrow N - c'$. And omit the assignments to a_m in steps U3 and U4.

The string in (1) turns out to have rank 3141592. (Who knew?)

51. By Theorem 7.2.1.3L, $N = \binom{\bar{z}_1}{n} + \binom{\bar{z}_2}{n-1} + \cdots + \binom{\bar{z}_n}{1}$; hence $\kappa_n N = \binom{\bar{z}_1}{n-1} + \binom{\bar{z}_2}{n-2} + \cdots + \binom{\bar{z}_n}{0}$, since $\bar{z}_n \ge 1$. Now note that $N - \kappa_n N$ is the rank of $z_1 z_2 \ldots z_n$, because of (23) and exercise 50. (For example, let $z_1 \ldots z_4 = 1256$, which has rank 6 in Table 1. Then $\bar{z}_1 \ldots \bar{z}_4 = 7632$, $N = 60$, and $\kappa_4 60 = 54$. Notice that N is fairly large, because $\bar{z}_1 = 2n - 1$; Fig. 27 shows that $\kappa_n N$ usually *exceeds* N when N is smaller.)

52. The number of trailing right parentheses has the same distribution as the number of leading left parentheses, and the sequence of nested strings that begin with '$(^k)$' is $(^k)A_{(n-k)(n-1)}$. Therefore the probability that $d_n = k$ is $C_{(n-k)(n-1)}/C_n$. We find

$$\begin{aligned}\sum_{k=0}^{n} \binom{k}{t} C_{(n-k)(n-1)} &= \sum_{k=0}^{n} \left(\binom{2n-1-k}{n-1} - \binom{2n-1-k}{n} \right) \binom{k}{t} \\ &= \binom{2n}{n+t} - \binom{2n}{n+t+1} = C_{(n-t)(n+t)}\end{aligned}$$

using Eq. 1.2.6–(25), and it follows that the mean and variance are respectively equal to $3n/(n+2) = 3 - 6/(n+2)$ and $2n(2n^2 - n - 1)/((n+2)^2(n+3)) = 4 + O(n^{-1})$. [The moments of this distribution were first calculated by R. Kemp in *Acta Informatica* **35** (1998), 17–89, Theorem 9. Notice that $c_n = d_n - 1$ has essentially the same behavior.]

53. (a) $3n/(n+2)$, by exercise 52. (b) H_n, by exercise 6.2.2–7. (c) $2 - 2^{-n}$, by induction.

(d) Any particular (but fixed) sequence of left or right branches has the same distribution of steps before a leaf is encountered. (In other words, the probability that a node with Dewey binary notation 01101 occurs is the same as the probability that 00000 occurs.) Thus if $X = k$ with probability p_k, each of the 2^k potential nodes on level k is external with probability p_k. The expected value $\sum_k 2^k p_k$ is therefore the expected number of external nodes, namely $n+1$ in all three cases. (One can of course also verify this result directly, with $p_k = C_{(n-k)(n-1)}/C_n$ in case (a), $p_k = {n \brack k}/n!$ in case (b), and $p_k = 2^{-k+[k=n]}$ in case (c).)

Notes: The average path length turns out to be $\Theta(\sqrt{n})$, $\Theta(\log n)$, and $\Theta(n)$ in these three cases; thus it is longer when the expected time to hit a leaf is shorter! The reason is that ubiquitous "holes" near the root force other paths to be long. Case (a) has an interesting generalization to t-ary trees, when $p_k = C^{(t)}_{(n-k)((t-1)n-1)}/C^{(t)}_n$ in the notation of answer 36. Then the mean distance to a leaf is $(t+1)n/((t-1)n+2)$, and it is instructive to prove via telescoping series that

$$\sum_k t^k C^{(t)}_{(n-k)((t-1)n-1)} = \binom{tn}{n}.$$

54. Differentiating with respect to x we have

$$C'(x,z) = zC'(x,z)C(x,xz) + zC(x,z)(C'(x,xz) + zC_\prime(x,xz)),$$

where $C_\prime(x,z)$ denotes the derivative of $C(x,z)$ with respect to z. Thus $C'(1,z) = 2zC'(1,z)C(z) + z^2C(z)C'(z)$; and since $C'(z) = C(z)^2 + 2zC(z)C'(z)$ we can solve for $C'(1,z)$, obtaining $z^2C(z)^3/(1-2zC(z))^2$. Therefore $\sum(c_1+\cdots+c_n) = [z^n]\,C'(1,z) = 2^{2n-1} - \frac{1}{2}(3n+1)C_n$, in agreement with exercise 2.3.4.5–5. Similarly we find

$$\sum(c_1+\cdots+c_n)^2 = [z^n]\,C''(1,z) = \Bigl(\frac{5n^2+19n+6}{6}\Bigr)\binom{2n}{n} - \Bigl(1+\frac{3n}{2}\Bigr)4^n.$$

Thus the mean and variance are $\frac{1}{2}\sqrt{\pi}\,n^{3/2}+O(n)$ and $(\frac{5}{6}-\frac{\pi}{4})n^{3/2}+O(n)$, respectively.

55. Differentiating as in answer 54, and using the formulas of exercises 46(d) and 5.2.1–14 together with $[z^n]\,C(z)^r/(1-4z) = 2^{2n+r} - \sum_{j=1}^{r} 2^{r-j}\binom{2n+j}{n}$, yields

$$\begin{aligned}
C'_{p(p+r)}(1) &= [z^p]\,\Bigl((r{+}1)\frac{z^2C(z)^{r+3}}{1-4z} + \binom{r{+}1}{2}\frac{zC(z)^{r+2}}{\sqrt{1-4z}}\Bigr)\\
&= [z^p]\,\Bigl((r{+}1)\frac{C(z)^{r+1}-2C(z)^r+C(z)^{r-1}}{1-4z} + \binom{r{+}1}{2}\frac{C(z)^{r+1}-C(z)^r}{\sqrt{1-4z}}\Bigr)\\
&= (r{+}1)\Bigl(2^{2p+r-1} - \binom{2p{+}r{+}1}{p} - \sum_{j=1}^{r-1} 2^{r-1-j}\binom{2p{+}j}{p}\Bigr) + \binom{r{+}1}{2}\binom{2p{+}r}{p{-}1}.
\end{aligned}$$

56. Use 1.2.6–53(b). [See *BIT* **30** (1990), 67–68.]

57. $2S_0(a,b) = \binom{2a}{a}\binom{2b}{b} + \binom{2a+2b}{a+b}$ by 1.2.6–(21). Exercise 1.2.6–53 tells us that

$$\sum_{k=a-m}^{a}\binom{2a}{a-k}\binom{2b}{b-k}k = (m+1)(a+b-m)\binom{2a}{m+1}\binom{2b}{a+b-m};$$

therefore $2S_1(a,b) = \binom{2a}{a}\binom{2b}{b}\frac{ab}{a+b}$. And since $b^2S_p(a,b) - S_{p+2}(a,b) = S_p(a,b-1)$, we find $2S_2(a,b) = \binom{2a+2b}{a+b}\frac{ab}{2a+2b-1}$; $2S_3(a,b) = \binom{2a}{a}\binom{2b}{b}a^2b^2/(a+b)^2$. Formula (30) follows by setting $a=m$, $b=n-m$, and $C_{(x-k)(x+k)} = \binom{2x}{x-k} - \binom{2x}{x-k-1}$.

Similarly, the average of w_{2m-1} is $\sum_{k\ge 0}(2k-1)C_{(m-k)(m+k-1)}C_{(n-m-k+1)(n-m+k)}$ divided by C_n, namely

$$\frac{2S_3(m,n{+}1{-}m) - S_2(m,n{+}1{-}m)}{m(n{+}1{-}m)C_n} = \frac{m(n{+}1{-}m)}{n}\binom{2m}{m}\binom{2n{+}2{-}2m}{n{+}1{-}m}\Big/\binom{2n}{n} - 1.$$

[R. Kemp, *BIT* **20** (1980), 157–163; H. Prodinger, *Soochow J. Math.* **9** (1983), 193–196.]

58. Summing over cases in which the left subtree has k internal nodes, we have

$$t_{lmn} = [l=m=n=0] + \sum_{k=0}^{m-1} C_k t_{(l-1)(m-k-1)(n-k-1)} + \sum_{k=m}^{n-1} C_{n-1-k} t_{(l-1)mk}.$$

Thus the triple generating function $t(v,w,z) = \sum_{l,m,n} t_{lmn} v^l w^m z^n$ satisfies

$$t(v,w,z) = 1 + vwzC(wz)t(v,w,z) + vzC(z)t(v,w,z);$$

and the analogous linear relation for $t(w,z) = \partial t(v,w,z)/\partial v\,|_{v=1}$ follows, because $t(1,w,z) = \sum_{n=0}^{\infty}\sum_{m=0}^{n} C_n w^m z^n = \bigl(C(z) - wC(wz)\bigr)/(1-w)$ and $zC(z)^2 = C(z) - 1$. Algebraic manipulation now yields

$$t(w,z) = \frac{C(z) + wC(wz) - (1+w)}{(1-w)^2 z} - \frac{2wC(z)C(wz)}{(1-w)^2} - \frac{C(z) - wC(wz)}{1-w},$$

and we obtain the formula $t_{mn} = (m+1)C_{n+1} - 2\sum_{k=0}^{m}(m-k)C_kC_{n-k} - C_n$. Now

$$\sum_{k=0}^{m-1}(k+1)C_kC_{n-1-k} \;=\; \frac{m}{2n}\binom{2m}{m}\binom{2n-2m}{n-m}$$

can be proved as in exercise 56, and it follows that

$$t_{mn} = 2\binom{2m}{m}\binom{2n-2m}{n-m}\frac{(2m+1)(2n-2m+1)}{(n+1)(n+2)} - C_n, \qquad \text{for } 0 \le m \le n.$$

[P. Kirschenhofer, *J. Combinatorics, Information and System Sciences* **8** (1983), 44–60. For higher moments and generalizations, see W. J. Gutjahr, *Random Structures and Algorithms* **3** (1992), 361–374; A. Panholzer and H. Prodinger, *J. Statistical Planning and Inference* **101** (2002), 267–279. Note that the generating function $t(v,w,z)$ yields

$$t_{lmn} = \sum_k \binom{l}{k} C_{(m-k)(m-1)} C_{(n-m-l+k)(n-m-1)}.$$

Using the fact that $\sum_k \binom{k}{r} C_{(n-k)(m-1)} = C_{(n-r)(m+r)}$ when $m \ge 1$, we obtain the formula $t_{mn} + C_n = \sum_k (k+1) C_{(m-k)(m-1)} C_{(n-m)(n-m+k+1)}$, a sum that can therefore (surprisingly) be expressed in closed form.]

59. $$T(w,z) = \frac{w\bigl(C(z) - C(wz)\bigr)}{(1-w)} - wzC(z)C(wz) + zC(z)T(w,z) + wzC(wz)T(wz)$$
$$= \frac{w\bigl((C(z)+C(wz)-2)/z - (1+w)C(z)C(wz) - (1-w)(C(z)-C(wz))\bigr)}{(1-w)^2}.$$

Hence $T_{mn} = t_{mn} - \sum_{k=m}^{n} C_kC_{n-k}$. [Is there a combinatorial proof?] And

$$T_{mn} = \binom{2m}{m}\binom{2n+2-2m}{n+1-m}\frac{4m(n+1-m)+n+1}{2(n+1)(n+2)} - \frac{1}{2}C_{n+1} - C_n, \quad \text{for } 1 \le m \le n.$$

60. (a) It is the number of right parentheses in co-atoms. (Therefore it is also the number of k for which $w_{2k-1} < 0$ in the associated “worm walk.”)

(b) For convenience let d(‘(’) $= +1$ and d(‘)’) $= -1$.

A1. [Initialize.] Set $i \leftarrow j \leftarrow 1$ and $k \leftarrow 2n$.

A2. [Done?] Terminate the algorithm if $j > k$. Otherwise set $a_j \leftarrow$ ‘(’, $j \leftarrow j+1$.

A3. [Atom?] If $b_i =$ ‘)’, set $s \leftarrow -1$, $i \leftarrow i+1$, and go to A4. Otherwise set $s \leftarrow 1$, $i \leftarrow i+1$, and while $s > 0$ set $a_j \leftarrow b_i$, $j \leftarrow j+1$, $s \leftarrow s + d(b_i)$, $i \leftarrow i+1$. Return to A2.

A4. [Co-atom.] Set $s \leftarrow s + d(b_i)$. Then if $s < 0$, set $a_k \leftarrow b_i$, $k \leftarrow k-1$, $i \leftarrow i+1$, and repeat step A4. Otherwise set $a_k \leftarrow$ ‘)’, $k \leftarrow k-1$, $i \leftarrow i+1$, and return to A2. ▌

(c) The defect-11 inverse of (1) is `(()))((()))))))(()((())(()))(((`. In general we find it by locating the subscript m just before the lth-from-last right parenthesis, and the indices (u_0, v_0), …, (u_{s-1}, v_{s-1}) of matching parentheses such that $u_j \le m < v_j$.

I1. [Initialize.] Set $c \leftarrow j \leftarrow s \leftarrow 0$, $k \leftarrow m \leftarrow 2n$, and $u_0 \leftarrow 2n+1$.

I2. [Scan right to left.] If $a_k =$ ‘)’, go to I3; if $a_k =$ ‘(’, go to I4; if $k = 0$, go to I5.

I3. [Process a ‘)’.] Set $r_j \leftarrow k$, $j \leftarrow j+1$, $c \leftarrow c+1$. If $c = l$, set $m \leftarrow k-1$, $s \leftarrow j$, and $u_s \leftarrow k$. Then decrease k by 1 and return to I2.

I4. [Process a ‘(’.] (At this point the left parenthesis a_k matches the right parenthesis $a_{r_{j-1}}$.) Set $j \leftarrow j-1$. If $r_j > m$, set $u_j \leftarrow k$ and $v_j \leftarrow r_j$. Then decrease k by 1 and return to I2.

I5. [Prepare to permute.] Set $i \leftarrow j \leftarrow 1$, $k \leftarrow 2n$, and $c \leftarrow 0$.

I6. [Permute.] While $j \ne u_c$, set $b_i \leftarrow a_j$, $i \leftarrow i+1$, $j \leftarrow j+1$. Then terminate if $c = s$; otherwise set $b_i \leftarrow$ ‘)’, $i \leftarrow i+1$, $j \leftarrow j+1$. While $k \ne v_c$, set $b_i \leftarrow a_k$, $i \leftarrow i+1$, $k \leftarrow k-1$. Then set $b_i \leftarrow$ ‘(’, $i \leftarrow i+1$, $k \leftarrow k-1$, $c \leftarrow c+1$, and repeat step I6. ▌

Notes: The fact that exactly C_n balanced strings of length $2n$ have defect l, for $0 \le l \le n$, was discovered by P. A. MacMahon [*Philosophical Transactions* **209** (1909), 153–175, §20], then rediscovered by K. L. Chung and W. Feller [*Proc. Nat. Acad. Sci.* **35** (1949), 605–608], using generating functions. A simple combinatorial explanation was found subsequently by J. L. Hodges, Jr. [*Biometrika* **42** (1955), 261–262], who observed that if $\beta_1 \ldots \beta_r$ has defect $l > 0$ and if $\beta_k = \alpha_k^R$ is its rightmost co-atom, the balanced string $\beta_1 \ldots \beta_{k-1}(\beta_{k+1} \ldots \beta_r)\alpha_k'^R$ has defect $l-1$ (and this transformation is reversible). The efficient mapping in the present exercise is similar to a construction of M. D. Atkinson and J.-R. Sack [*Information Processing Letters* **41** (1992), 21–23].

61. (a) Let $c_j = 1 - b_j$; thus $c_j \le 1$, $c_1 + \cdots + c_N = f$, and we must prove that

$$c_1 + c_2 + \cdots + c_k \; < \; f \qquad \text{if and only if} \qquad k < N$$

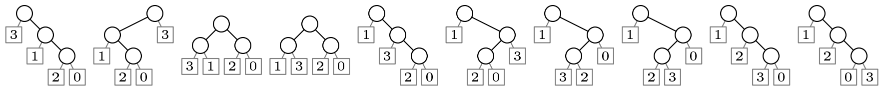
holds for exactly f cyclic shifts. We can define c_j for all integers j by letting $c_{j \pm N} = c_j$. Let us also define Σ_j for all j by letting $\Sigma_0 = 0$ and $\Sigma_j = \Sigma_{j-1} + c_j$; then $\Sigma_{j+Nt} = \Sigma_j + ft$, and $\Sigma_{j+1} \le \Sigma_j + 1$. It follows that for each integer x there is a smallest integer $j = j(x)$ such that $\Sigma_j = x$. Moreover, $j(x) < j(x+1)$; and $j(x+f) = j(x) + N$. Thus the desired condition holds if and only if we shift by $j(x) \bmod N$ for $x = 1, 2, \ldots,$ or f. (The history of this important lemma is discussed in answer 2.3.4.4–32.)

(b) Start with $l \leftarrow m \leftarrow s \leftarrow 0$. Then for $k = 1, 2, \ldots, N$ (in this order) do the following: Set $s \leftarrow s + 1 - b_k$; and if $s > m$, set $m \leftarrow s$, $j_l \leftarrow k$, and $l \leftarrow (l+1) \bmod f$. The answers are $j_0, \ldots, j_{f-1}$, by the proof in part (a).

(c) Start with any string $b_1 b_2 \ldots b_N$ containing n_j occurrences of j for $0 \le j \le t$. Apply a random permutation to this string, then apply the algorithm of part (b). Choose randomly between $(j_0, \ldots, j_{f-1})$ and use the resulting cyclic shift as a preorder sequence to define the forest.

[See L. Alonso, J. L. Rémy, and R. Schott, *Algorithmica* **17** (1997), 162–182, for an even more general algorithm.]

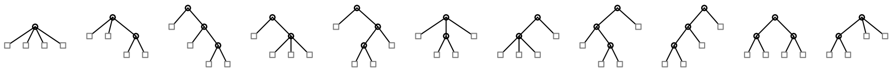
62. Bit strings $(l_1 \ldots l_n, r_1 \ldots r_n)$ are valid if and only if $b_1 \ldots b_n$ is valid in exercise 20, where $b_j = l_j + r_j$. Therefore we can use exercise 61. [See J. F. Korsh, *Information Processing Letters* **45** (1993), 291–294.]

63. 

64. $X = 2k + b$ where $(k, b) = (0,1)$, $(2,1)$, $(0,0)$, $(5,1)$, $(6,0)$, $(1,1)$; eventually $L_0 L_1 \ldots L_{12} = 5\ 11\ 3\ 4\ 0\ 7\ 9\ 8\ 1\ 6\ 10\ 12\ 2$.

65. See A. Panholzer and H. Prodinger, *Discrete Mathematics* **250** (2002), 181–195; M. Luczak and P. Winkler, *Random Structures and Algorithms* **24** (2004), 420–443.

66. (a) "Shrink" the white edges, merging the nodes that they connect. For example,



are the ordinary trees that correspond to the eleven Schröder trees depicted for $n = 3$. Under this correspondence a left link means, "here is a child"; a white right link means, "look here for more children"; a black right link means, "here's the last child."

(b) Mimic Algorithm L, but between rotations use an ordinary Gray binary code to run through all color patterns of whatever right links are present. (The case $n = 3$ has, in fact, been illustrated in the example.)

Note that Schröder trees also correspond to series-parallel graphs, as in (53). They do, however, impose an order on the edges and/or superedges that are joined in parallel; so they correspond more precisely to series-parallel graphs *as embedded in the plane* (and with edges and vertices unlabeled, except for s and t).

67. $S(z) = 1 + zS(z)\bigl(1 + 2(S(z) - 1)\bigr)$, because $1 + 2(S(z) - 1)$ enumerates the right subtrees; therefore $S(z) = (1 + z - \sqrt{1 - 6z + z^2}\,)/(4z)$.

Notes: We've seen Schröder numbers in exercise 2.3.4.4–31, where $G(z) = zS(z)$; and in exercise 2.2.1–11, where $b_n = 2S_{n-1}$ for $n \ge 2$ and where we found the recurrence $(n-1)S_n = (6n-3)S_{n-1} - (n-2)S_{n-2}$. They grow asymptotically as explored in exercise 2.2.1–12. A triangle of numbers S_{pq}, analogous to (22), can be used to generate random Schröder trees. These numbers satisfy

$$\begin{aligned} S_{pq} &= S_{p(q-1)} + S_{(p-1)q} + S_{(p-2)q} + \cdots + S_{0q} = S_{p(q-1)} + 2S_{(p-1)q} - S_{(p-1)(q-1)} \\ &= \frac{q-p+1}{q+1} \sum_{k=0}^{p} \binom{q+1}{p-k}\binom{p-1}{k} 2^k = \sum_{k=0}^{p} \left(\binom{q}{p-k}\binom{p-1}{k} - \binom{q}{p-k-1}\binom{p-1}{k-1}\right) 2^k \\ &= [w^p z^q]\, S(wz)/(1 - zS(wz)); \end{aligned}$$

the double generating function on the last line is due to Emeric Deutsch. Many other properties of Schröder trees are discussed in Richard Stanley's *Enumerative Combinatorics* **2** (1999), exercise 6.39.

68. A single row that contains only the empty string ϵ. (The general rule (36) for going from $n-1$ to n converts this row into '0 1', the pattern of order 1.)

69. The first $\binom{6}{3} = 20$ rows are the Christmas tree pattern of order 6, if we ignore the '10' at the beginning of each string. The pattern of order 7 is a bit more difficult to see; but there are $\binom{7}{3} = 35$ rows in which the leftmost entry begins with 0. Disregard the rightmost string in all such rows, and ignore the 0 at the beginning of each remaining string. (Other answers are also possible.)

70. If σ appears in column k of the Christmas tree pattern, let σ' be the string in column $n-k$ of the same row. (If we think of parentheses instead of bits, this rule takes the mirror reflection of the free parentheses in the sense of answer 11, by (39).)

71. M_{tn} is the sum of the t largest binomial coefficients $\binom{n}{k}$, because each row of the Christmas tree pattern can contain at most t elements of S, and because we do get such a set S by choosing all strings σ with $(n-t)/2 \le \nu(\sigma) \le (n+t-1)/2$. (The formula

$$M_{tn} = \sum_{n-t\le 2k\le n+t-1} \binom{n}{k}$$

is about as simple as possible; however, special formulas like $M_{(2)n} = M_{n+1}$ hold for small t, and we also have $M_{tn} = 2^n$ for $t > n$.)

72. You get M_{sn}, the same number as in the previous exercise. In fact, one can prove by induction that there are exactly $\binom{n}{n-k} - \binom{n}{k-s}$ rows of length $s+n-2k \ge 0$.

73. 011001001000000000100101001100, 111001011011111111101101011100; see (38).

74. By the lexicographic property, we want to count the number of rows whose rightmost elements have the respective forms $0{*}^{29}$, $10{*}^{28}$, $110{*}^{27}$, $111000{*}^{24}$, $11100100{*}^{22}$, $111001010{*}^{21}$, $11100101100{*}^{19}$, $111001011010{*}^{18}$, $1110010110110{*}^{17}$, $\ldots$, namely all 30-bit strings that precede $\tau = 111001011011111111101101011100$.

If θ has p more 1s than 0s, the number of Christmas tree rows ending with $\theta{*}^n$ is the same as the number of rows ending with $1^p{*}^n$; and this is $M_{(p+1)n}$, by exercise 71, because all such rows are the n-step descendants of the starting row '$0^p\ 0^{p-1}1\ \ldots\ 1^p$'.

Consequently the answer is $M_{0(29)} + M_{1(28)} + M_{2(27)} + M_{1(24)} + \cdots + M_{(12)3} + M_{(13)2} = \sum_{k=1}^{21} M_{(2k-1-z_k)(n-z_k)} = 0 + \binom{28}{14} + \binom{27}{14} + \binom{27}{13} + \binom{24}{12} + \cdots + 8 + 4 = 84867708$, where $(z_1, \ldots, z_{21}) = (1, 2, 3, 6, \ldots, 27, 28)$ is the sequence of places where 1s occur in τ.

75. We have $r_1^{(n)} = M_{n-2}$, because row $r_1^{(n)}$ is the bottom descendant of the first row in (33). We also have $r_{j+1}^{(n)} - r_j^{(n)} = M_{j(n-1-j)} - M_{(j-1)(n-2-j)} = M_{(j+1)(n-2-j)}$ by the formula in answer 74, because the relevant sequence $z_1 \ldots z_{n-1}$ for row $r_j^{(n)}$ is 1^j01^{n-1-j}. Therefore, since $M_{jn}/M_n \to j$ for fixed j as $n \to \infty$, we have

$$\lim_{n\to\infty} \frac{r_j^{(n)}}{M_n} \;=\; \sum_{k=1}^{j} \frac{k}{2^{k+1}} \;=\; 1 - \frac{j+2}{2^{j+1}}.$$

And we've also implicitly proved that $\sum_{k=0}^{n} M_{k(n-k)} = M_{n+1} - 1$.

76. The first $\binom{2n}{n}$ elements of the infinite sequence

$$Q = 13133513133513353557131335131335133535571313351335355713353557355757779\ldots$$

are the row sizes in the pattern of order $2n$; this sequence $Q = q_1q_2q_3\ldots$ is the unique fixed point of the transformation that maps $1 \mapsto 13$ and $n \mapsto (n{-}2)\,n\,n\,(n{+}2)$ for odd $n > 1$, representing two steps of (36).

Let $f(x) = \limsup_{n\to\infty} s(\lceil xM_n\rceil)/n$ for $0 < x \le 1$. This function apparently vanishes almost everywhere; but it equals 1 when x has the form $(q_1 + \cdots + q_j)/2^n$, because of answer 72. On the other hand if we define $g(x) = \lim_{n\to\infty} s(\lceil xM_n\rceil)/\sqrt{n}$, the function $g(x)$ appears to be measurable, with $\int_0^1 g(x)\,dx = \sqrt{\pi}$, although $g(x)$ is infinite when $f(x) > 0$. (Rigorous proofs or disproofs of these conjectures are solicited.)

77. The hint follows from (39), by considering worm walks; so we can proceed thus:

X1. [Initialize.] Set $a_j \leftarrow 0$ for $0 \le j \le n$; also set $x \leftarrow 1$. (In the following steps we will have $x = 1 + 2(a_1 + \cdots + a_n)$.)

X2. [Correct the tail.] While $x \le n$, set $a_x \leftarrow 1$ and $x \leftarrow x + 2$.

X3. [Visit.] Visit the bit string $a_1 \ldots a_n$.

X4. [Easy case?] If $a_n = 0$, set $a_n \leftarrow 1$, $x \leftarrow x + 2$, and return to X3.

X5. [Find and advance a_j.] Set $a_n \leftarrow 0$ and $j \leftarrow n - 1$. Then while $a_j = 1$, set $a_j \leftarrow 0$, $x \leftarrow x - 2$, and $j \leftarrow j - 1$. Stop if $j = 0$; otherwise set $a_j \leftarrow 1$ and go back to X2. ▌

78. True, by (39) and exercise 11.

79. (a) List the indices of the 0s, then the indices of the 1s; for instance, the bit string in exercise 73 corresponds to the permutation 1 4 5 7 8 10 11 12 13 20 23 25 29 30 2 3 6 9 14 15 16 17 18 19 21 22 24 26 27 28.

(b) Using the conventions of (39), the P tableau has the indices of left parentheses and free parentheses in its top row, other indices in the second row. Thus, from (38),

$$P \;=\; \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1&2&3&6&8&9&11&12&13&14&15&16&17&18&19&21&22&24&26&27&28\\ \hline 4&5&7&10&20&23&25&29&30\\ \cline{1-9}\end{array}\,.$$

[See K.-P. Vo, *SIAM J. Algebraic and Discrete Methods* **2** (1981), 324–332, for a generalization to chains of submultisets.]

80. This curious fact is a consequence of exercise 79 together with Theorem 6 in the author's paper on tableaux; see *Pacific J. Math.* **34** (1970), 709–727.

81. Suppose σ and σ' belong respectively to chains of length s and s' in the Christmas tree patterns of order n and n'. At most $\min(s, s')$ of the ss' pairs of strings in those chains can be in the biclutter. Furthermore, because of (39), those ss' pairs of strings actually constitute exactly $\min(s, s')$ chains in the Christmas tree pattern of order $n + n'$, when they are concatenated. Therefore the sum of $\min(s, s')$ over all pairs of chains is $M_{n+n'}$, and the result follows. We have incidentally proved the nonobvious identity

$$\sum_{j,\,k} \min(m+1-2j, n+1-2k)\, C_{j(m-j)} C_{k(n-k)} \;=\; M_{m+n}.$$

Notes: This extension of Sperner's theorem was proved independently by G. Katona [*Studia Sci. Math. Hungar.* **1** (1966), 59–63] and D. J. Kleitman [*Math. Zeitschrift* **90** (1965), 251–259]. See Greene and Kleitman, *J. Combinatorial Theory* **A20** (1976), 80–88, for the proof given here and for further results.

82. (a) There is at least one evaluation in each row m; there are two if and only if $s(m) > 1$ and the first evaluation yields 0. Thus if f is identically 1, we get the minimum, M_n; if f is identically 0, we get the maximum, $M_n + \sum_m [s(m) > 1] = M_{n+1}$.

(b) Let $f(\chi(m, n/2)) = 0$ in the $C_{n/2}$ cases where $s(m) = 1$; otherwise let $f(\chi(m, a)) = 1$, where a is defined by the algorithm. When n is odd, this rule implies that $f(\sigma)$ is always 1; but when n is even, $f(\sigma) = 0$ if and only if σ is first in its row. (To see why, use the fact that the row containing σ'_j in (41) always has size $s - 2$.) This function f is indeed monotonic; for if $\sigma \le \tau$ and if σ has a free left parenthesis, so does τ. For example, in the case $n = 8$ we have

$$f(x_1, \ldots, x_8) = x_8 \vee x_6x_7 \vee x_4x_5(x_6 \vee x_7) \vee x_2x_3(x_4(x_5 \vee x_6 \vee x_7) \vee x_5(x_6 \vee x_7)).$$

(c) In these circumstances (45) is the solution for all n.

83. At most 3 outcomes are possible in step H4 — in fact, at most 2 when $s(m) = 1$. [See exercise 5.3.4–31 for sharper bounds; in the notation of that exercise, there are exactly $\delta_n + 2$ monotone Boolean functions of n Boolean variables.]

84. For this problem we partition the 2^n bit strings into M_n blocks instead of chains, where the strings $\{\sigma_1, \ldots, \sigma_s\}$ of each block satisfy $\|A\sigma_i^T - A\sigma_j^T\| \ge 1$ for $i \ne j$; then at most one bit string per block can satisfy $\|A\sigma^T - b\| < \frac{1}{2}$.

Let A' denote the first $n-1$ columns of A, and let v be the nth column. Suppose $\{\sigma_1, \ldots, \sigma_s\}$ is a block for A', and number the subscripts so that $v^T A' \sigma_1^T$ is the minimum of $v^T A' \sigma_j^T$. Then rule (36) defines appropriate blocks for A, because we have $\|A(\sigma_i 0)^T - A(\sigma_j 0)^T\| = \|A(\sigma_i 1)^T - A(\sigma_j 1)^T\| = \|A'\sigma_i^T - A'\sigma_j^T\|$ and

$$\begin{aligned}\|A(\sigma_j 1)^T - A(\sigma_1 0)^T\|^2 &= \|A'\sigma_j^T + v - A'\sigma_1^T\|^2 \\ &= \|A'(\sigma_j - \sigma_1)^T\|^2 + \|v\|^2 + 2v^T A'(\sigma_j - \sigma_1)^T \ge \|v\|^2 \ge 1.\end{aligned}$$

[And more is true; see *Advances in Math.* **5** (1970), 155–157. This result extends a theorem of J. E. Littlewood and A. C. Offord, *Mat. Sbornik* **54** (1943), 277–285, who considered the case $m = 2$.]

85. If V has dimension $n - m$, we can renumber the coordinates so that

$$\begin{array}{l}(1,\ 0,\ \ldots,\ 0,\quad x_{11},\quad \ldots,\quad x_{1m})\\ (0,\ 1,\ \ldots,\ 0,\quad x_{21},\quad \ldots,\quad x_{2m})\\ \ \ \vdots\quad \vdots\quad \ddots\quad \vdots\qquad \vdots\qquad\qquad \vdots\\ (0,\ 0,\ \ldots,\ 1,\ x_{(n-m)1},\ \ldots,\ x_{(n-m)m})\end{array}$$

is a basis, with none of the row vectors $v_j = (x_{j1}, \ldots, x_{jm})$ entirely zero. Let $v_{n-m+1} = (-1, 0, \ldots, 0), \ldots, v_n = (0, 0, \ldots, -1)$. Then the number of 0–1 vectors in V is the number of 0–1 solutions to $Ax = 0$, where A is the $m \times n$ matrix with columns $v_1, \ldots, v_n$. But this quantity is at most the number of solutions to $\|Ax\| < \frac{1}{2}\min(\|v_1\|, \ldots, \|v_n\|)$, which is at most M_n by exercise 84.

Conversely, the basis with $m = 1$ and $x_{j1} = (-1)^{j-1}$ yields M_n solutions. [This result has application to electronic voting; see Golle's Ph.D. thesis (Stanford, 2004).]

86. First reorder the 4-node subtrees so that their level codes are 0121 (plus a constant); then sort larger and larger subtrees until everything is canonical. The resulting level codes are 0 1 2 3 4 3 2 1 2 3 2 1 2 0 1, and the parent pointers are 0 1 2 3 4 3 2 1 8 9 8 1 12 0 14.

87. (a) The condition holds if and only if $c_1 < \cdots < c_k \ge c_{k+1} \ge \cdots \ge c_n$ for some k, so the total number of cases is $\sum_k \binom{n-1}{n-k} = 2^{n-1}$.

(b) Note that $c_1 \ldots c_k = c'_1 \ldots c'_k$ if and only if $p_1 \ldots p_k = p'_1 \ldots p'_k$; and in such cases, $c_{k+1} < c'_{k+1}$ if and only if $p_{k+1} < p'_{k+1}$.

88. Exactly A_{n+1} forests are visited, and A_k of them have $p_k = \cdots = p_n = 0$. Therefore O4 is performed A_n times; and p_k is changed $A_{k+1} - 1$ times in step O5, for $1 \le k < n$. Step O5 also changes p_n a total of $A_n - 1$ times. The average number of mems per visit is therefore only $2 + 3/(\alpha - 1) + O(1/n) \approx 3.534$, if we keep p_n in a register. [See E. Kubicka, *Combinatorics, Probability and Computing* **5** (1996), 403–417.]

89. If step O5 sets $p_n \leftarrow p_j$ exactly Q_n times, it sets $p_k \leftarrow p_j$ exactly $Q_k + A_{k+1} - A_k$ times, for $1 < k < n$, because every prefix of a canonical $p_1 \ldots p_n$ is canonical. We have $(Q_1, Q_2, \ldots) = (0, 0, 1, 2, 5, 9, 22, 48, 118, 288, \ldots)$; and one can show that $Q_n = \sum_{d\ge 1} \sum_{1 \le c < n/d - 1} a_{(n-cd)(n-cd-d)}$, where a_{nk} is the number of canonical parent sequences $p_1 \ldots p_n$ with $p_n = k$. But these numbers a_{nk} remain mysterious.

90. (a) This property is equivalent to 2.3.4.4–(7); vertex 0 is the centroid.

(b) Let $m = \lfloor n/2 \rfloor$. At the end of step O1, set $p_{m+1} \leftarrow 0$, and also $p_{2m+1} \leftarrow 0$ if n is odd. At the end of step O4, set $i \leftarrow j$ and while $p_i \neq 0$ set $i \leftarrow p_i$. (Then i is the root of the tree containing j and k.) At the beginning of step O5, if $k = i + m$ and $i < j$, set $j \leftarrow i$ and $d \leftarrow m$.

(c) If n is even, there are no bicentroidal trees with $n+1$ vertices. Otherwise find all pairs $(p'_1 \ldots p'_m, p''_1 \ldots p''_m)$ of canonical forests on $m = \lfloor n/2 \rfloor$ nodes, with $p'_1 \ldots p'_m \geq p''_1 \ldots p''_m$; let $p_1 = 0$, $p_{j+1} = p'_j + 1$, and $p_{m+j+1} = (p''_j + m + 1)[p''_j > 0]$ for $1 \leq j \leq m$. (Two incarnations of Algorithm O will generate all such sequences. This algorithm for free trees is due to F. Ruskey and G. Li; see *SODA* **10** (1999), S939–S940.)

91. Use the following recursive procedure $W(n)$: If $n \leq 2$, return the unique n-node oriented tree. Otherwise choose positive integers j and d so that a given pair (j, d) is obtained with probability $dA_d A_{n-jd}/((n-1)A_n)$. Compute random oriented trees $T' \leftarrow W(n-jd)$ and $T'' \leftarrow W(d)$. Return the tree T obtained by linking j clones of T'' to the root of T'. [*Combinatorial Algorithms* (Academic Press, 1975), Chapter 25.]

92. Not always. [R. L. Cummins, in *IEEE Trans.* **CT-13** (1966), 82–90, proved that the graph of $S(G)$ always contains a cycle; see also C. A. Holzmann and F. Harary, *SIAM J. Applied Math.* **22** (1972), 187–193. But their constructions are unsuitable for efficient computation, because they require foreknowledge of the parity of the sizes of intermediate results.]

93. Yes. Step S7 undoes step S3; step S9 undoes the deletions of step S8.

94. For example, we can use depth-first search, with an auxiliary table $b_1 \ldots b_n$:

i) Set $b_1 \ldots b_n \leftarrow 0 \ldots 0$, then $v \leftarrow 1$, $w \leftarrow 1$, $b_1 \leftarrow 1$, and $k \leftarrow n-1$.

ii) Set $e \leftarrow n_{v-1}$. While $t_e \neq 0$, do the following substeps:
  a) Set $u \leftarrow t_e$. If $b_u \neq 0$, go to substep (c).
  b) Set $b_u \leftarrow w$, $w \leftarrow u$, $a_k \leftarrow e$, $k \leftarrow k-1$. Terminate if $k = 0$.
  c) Set $e \leftarrow n_e$.

iii) If $w \neq 1$, set $v \leftarrow w$, $w \leftarrow b_w$, and return to (ii). Otherwise report an error: The given graph was not connected.

We could actually terminate as soon as substep (b) reduces k to 1, since Algorithm S never looks at the initial value of a_1. But we might as well test for connectivity.

95. The following steps perform a breadth-first search from u, to see if v is reachable without using edge e. An auxiliary array $b_1 \ldots b_n$ of arc pointers is used, which should be initialized to $0 \ldots 0$ at the end of step S1; we will reset it to $0 \ldots 0$ again.

i) Set $w \leftarrow u$ and $b_w \leftarrow v$.

ii) Set $f \leftarrow n_{u-1}$. While $t_f \neq 0$, do the following substeps:
  a) Set $v' \leftarrow t_f$. If $b_{v'} \neq 0$, go to substep (d).
  b) If $v' \neq v$, set $b_{v'} \leftarrow v$, $b_w \leftarrow v'$, $w \leftarrow v'$, and go to substep (d).
  c) If $f \neq e \oplus 1$, go to step (v).
  d) Set $f \leftarrow n_f$.

iii) Set $u \leftarrow b_u$. If $u \neq v$, return to step (ii).

iv) Set $u \leftarrow t_e$. While $u \neq v$, set $w \leftarrow b_u$, $b_u \leftarrow 0$, $u \leftarrow w$. Go to S9 (e is a bridge).

v) Set $u \leftarrow t_e$. While $u \neq v$, set $w \leftarrow b_u$, $b_u \leftarrow 0$, $u \leftarrow w$. Then set $u \leftarrow t_e$ again and continue step S8 (e is not a bridge).

Two quick heuristics can be used before starting this calculation: If $d_u = 1$, then e is obviously a bridge; and if $l_{l_e} \neq 0$, then e is obviously a nonbridge (because there's another edge between u and v). Such special cases are detected readily by the breadth-first search, yet experiments by the author indicate that both heuristics are definitely worthwhile. For example, the test on l_{l_e} typically saves 3% or so of the total running time.

96. (a) Let e_k be the arc $k-1 \to k$. The steps in answer 94 set $a_k \leftarrow e_{n+1-k}$ for $n > k \geq 1$. Then at level k we shrink e_{n-k}, for $1 \leq k < n-1$. After visiting the (unique) spanning tree $e_{n-1} \ldots e_2 e_n$, we unshrink e_{n-k} and discover quickly that it is a bridge, for $n-1 > k \geq 1$. Thus the running time is linear in n; in the author's implementation it turns out to be exactly $102n - 226$ mems for $n \geq 3$.

However, this result depends critically on the order of the edges in the initial spanning tree. If step S1 had produced "organ-pipe order" such as

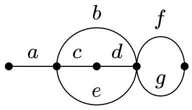
$$e_{n/2+1}\, e_{n/2}\, e_{n/2+2}\, e_{n/2-1} \ldots e_{n-1}\, e_2$$

in positions $a_2 \ldots a_{n-1}$ when n is even, the running time would have been $\Omega(n^2)$, because $\Omega(n)$ of the bridge tests would each have taken $\Omega(n)$ steps.

(b) Now a_k is initially e_{n-k} for $n > k \geq 1$, where e_1 is the arc $n \to 1$. The spanning trees visited, when $n \geq 4$, are respectively $e_{n-2} \ldots e_1 e_n$, $e_{n-2} \ldots e_1 e_{n-1}$, $e_{n-2} \ldots e_2 e_{n-1} e_n$, $e_{n-2} \ldots e_3 e_{n-1} e_n e_1$, $\ldots$, $e_{n-1} e_n e_1 \ldots e_{n-3}$. Following the tree $e_{n-2} \ldots e_{k+2} e_{n-1} e_n e_1 \ldots e_k$ the computations move down to level $n-k-3$ and up again, for $0 \leq k \leq n-4$; the bridge tests are all efficient. Thus the total running time is quadratic (in the author's version, exactly $35.5n^2 + 7.5n - 145$ mems, for $n \geq 5$).

Incidentally, P_n is *board*$(n,0,0,0,1,0,0)$ in the notation of the Stanford GraphBase, and C_n is *board*$(n,0,0,0,1,1,0)$; the SGB vertices are named 0 through $n-1$.

97. Yes, when $\{s,t\}$ is $\{1,2\}$, $\{1,3\}$, $\{2,3\}$, $\{2,4\}$, or $\{3,4\}$, but not $\{1,4\}$.

98. $A' =$ 

; this is the "dual planar graph" of the planar graph A.

(The near trees of A' are complements of the spanning trees of A, and vice versa.)

99. The stated method works, by induction on the size of the tree, for essentially the same reasons that it worked for n-tuples in Section 7.2.1.1 — but with the additional proviso that we must successively designate each child of an uneasy node.

Leaf nodes are always passive, and they are neither easy nor uneasy; so we will assume that the branch nodes are numbered 1 to m in preorder. Let $f_p = p$ for all branch nodes, except when p is a passive uneasy node for which the nearest uneasy node to its right is active; in the latter case, f_p should point to the nearest active uneasy node to its left. (For purposes of this definition, we imagine that artificial nodes 0 and $m+1$ are present at the left and right, both of which are uneasy and active.)

F1. [Initialize.] Set $f_p \leftarrow p$ for $0 \leq p \leq m$; also set $t_0 \leftarrow 1$, $v_0 \leftarrow 0$, and set each z_p so that $r_{z_p} = d_p$.

F2. [Select node p.] Set $q \leftarrow m$; then while $t_q = v_q$ set $q \leftarrow q-1$. Set $p \leftarrow f_q$ and $f_q \leftarrow q$; terminate the algorithm if $p = 0$.

F3. [Change d_p.] Set $s \leftarrow d_p$, $s' \leftarrow r_s$, $k \leftarrow v_p$, and $d_p \leftarrow s'$. (Now $k = v_s \neq v_{s'}$.)

F4. [Update the values.] Set $q \leftarrow s$ and $v_q \leftarrow k \oplus 1$. While $d_q \neq 0$, set $q \leftarrow d_q$ and $v_q \leftarrow k \oplus 1$. (Now q is a leaf that has entered the config if $k = 0$, left it if $k = 1$.) Similarly, set $q \leftarrow s'$ and $v_q \leftarrow k$. While $d_q \neq 0$, set $q \leftarrow d_q$ and $v_q \leftarrow k \oplus 1$. (Now q is a leaf that has left the config if $k = 0$, entered it if $k = 1$.)

F5. [Visit.] Visit the current config, represented by all the leaf values.

F6. [Passivate p?] (All uneasy nodes to p's right are now active.) If $d_p \neq z_p$, return to step F2. Otherwise set $z_p \leftarrow s$, $q \leftarrow p - 1$; while $t_q = v_q$, set $q \leftarrow q - 1$. (Now q is the first uneasy node to the left of p; we will make p passive.) Set $f_p \leftarrow f_q$, $f_q \leftarrow q$, and return to F2. ▌

Although step F4 may change uneasy nodes to easy nodes and vice versa, the focus pointers need not be updated, because they're still set correctly.

100. A complete program, called GRAYSPSPAN, appears on the author's website. Its asymptotic efficiency can be proved by using the result of exercise 110 below.

102. If so, ordinary spanning trees can be listed in a *strong revolving-door order*, where the edges that enter and leave at each step are adjacent.

Interesting algorithms to generate all the oriented spanning trees with a given root have been developed by Harold N. Gabow and Eugene W. Myers, *SICOMP* **7** (1978), 280–287; S. Kapoor and H. Ramesh, *Algorithmica* **27** (2000), 120–130.

103. (a) Toppling increases $(x_0, x_1, \ldots, x_n)$ lexicographically, but does not change $x_0 + \cdots + x_n$. If we can topple at both V_i and V_j, either order gives the same result.

(b) Adding a grain of sand changes the 16 stable states as follows:

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

The recurrent states are the nine cases with $x_1 + x_2 > 0$ and $x_3 + x_4 > 0$. Notice that repeated addition of 0001 leads to the infinite cycle $0000 \to 0001 \to 0010 \to 0011 \to 0001 \to 0010 \to \cdots$; but the states 0001, 0010, and 0011 are *not* recurrent.

(c) If $x = \sigma(x + t)$ then also $x = \sigma(x + kt)$ for all $k \geq 0$. All components of t are positive; thus $x = \sigma(x + \max(d_1, \ldots, d_n)t)$ is recurrent. Conversely, suppose $x = \sigma(d + y)$, where all $y_i \geq 0$; then $d + y + t$ topples to $x + t$ and it also topples to $\sigma(d) + y + t = d + y$. Therefore $\sigma(x + t) = \sigma(d + y) = x$.

(d) There are $N = \det(a_{ij})$ classes, because elementary row operations (exercise 4.6.1–19) triangularize the matrix while preserving congruence.

(e) There are nonnegative integers $m_1, \ldots, m_n, m'_1, \ldots, m'_n$ such that

$$x + m_1 a_1 + \cdots + m_n a_n \;=\; x' + m'_1 a_1 + \cdots + m'_n a_n \;=\; y, \text{ say.}$$

For sufficiently large k, the vector $y + kt$ topples in $m_1 + \cdots + m_n$ steps to $x + kt$, and in $m'_1 + \cdots + m'_n$ steps to $x' + kt$. Therefore $x = \sigma(x + kt) = \sigma(x' + kt) = x'$.

(f) The triangularization in (d) shows that $x \equiv x + Ny$ for arbitrary vectors y. And toppling preserves congruence; hence every class contains a recurrent state.

(g) Since $a = a_1 + \cdots + a_n$ in a balanced digraph, we have $x \equiv x + a$. If x is recurrent, we see in fact that every vertex topples exactly once when $x + a$ reduces to x, because the vectors $\{a_1, \ldots, a_n\}$ are linearly independent.

Conversely, if $\sigma(x + a) = x$ we must prove that x is recurrent. Let $z_m = \sigma(ma)$; there must be some positive k and m with $z_{m+k} = z_m$. Then every vertex topples k times when $z_m + ka$ reduces to z_m; hence there are vectors $y_j = (y_{j1}, \ldots, y_{jn})$ with $y_{jj} \geq d_j$ such that $(m + k)a$ topples to y_j. It follows that $x + n(m + k)a$ topples to $x + y_1 + \cdots + y_n$, and $\sigma(x + y_1 + \cdots + y_n) = \sigma(x + n(m + k)a) = x$.

(h) Treating subscripts cyclically, the spanning trees with arcs $V_j \to V_0$ for $j = i_1, \ldots, i_k$ have $n - k$ other arcs: $V_j \to V_{j-1}$ for $i_l < j \leq i_l + q_l$ and $V_j \to V_{j+1}$ for

$i_l + q_l < j < i_{l+1}$. The recurrent states, similarly, have $x_j = 2$ for $j = i_1, \ldots, i_k$, and $x_j = 1$ for $i_l < j < i_{l+1}$, except that $x_j = 0$ when $j = i_l + q_l$ and $q_l > 0$.

(i) In this case state $x = (x_1, \ldots, x_n)$ is recurrent if and only if $(n - x_1, \ldots, n - x_n)$ solves the parking problem in the hint, because $t = (1, \ldots, 1)$, and a sequence that doesn't get parked leaves a "hole" that stops $x + t$ from toppling to x.

Notes: This sandpile model, introduced by Deepak Dhar [*Phys. Review Letters* **64** (1990), 1613–1616], has led to many papers in the physics literature. Dhar noted that, if M grains of sand are introduced at random, each recurrent state is equally probable as $M \to \infty$. The present exercise was inspired by the work of R. Cori and D. Rossin, *European J. Combinatorics* **21** (2000), 447–459.

Sandpile theory proves that every digraph D yields an abelian group whose elements correspond somehow to the oriented spanning trees of D with root V_0. In particular, the same is true when D is an ordinary graph, with arcs $u \to v$ and $v \to u$ whenever u and v are adjacent. Thus, for example, we can "add" two spanning trees; and some spanning tree can be regarded as "zero." An elegant correspondence between spanning trees and recurrent states, in the special case when D is an ordinary graph, has been found by R. Cori and Y. Le Borgne, *Advances in Applied Math.* **30** (2003), 44–52. But no simple correspondence is known for general digraphs D. For example, suppose $n = 2$ and $(e_{10}, e_{12}, e_{20}, e_{21}) = (p, q, r, s)$; then there are $pr + ps + qr$ oriented trees, and the recurrent states correspond to generalized two-dimensional toruses as in exercise 7–00. Yet even in the "balanced" case, when $p + q \geq s$ and $r + s \geq q$, no easy mapping between spanning trees and recurrent states is apparent.

104. (a) If $\det(\alpha I - C) = 0$, there is a vector $x = (x_1, \ldots, x_n)^T$ such that $Cx = \alpha x$ and $\max(x_1, \ldots, x_n) = x_m = 1$ for some m. Then $\alpha = \alpha x_m = c_{mm} - \sum_{j \neq m} e_{mj} x_j \geq c_{mm} - \sum_{j \neq m} e_{mj} = 0$. (Incidentally, a real symmetric matrix whose eigenvalues are nonnegative is called *positive semidefinite*. Our proof establishes the well-known result that any real symmetric matrix with $c_{mm} \geq \left|\sum_{j \neq m} c_{mj}\right|$ for $1 \leq m \leq n$ has this property.)

(b) $\det\bigl(xI - C(G)\bigr) = x(x - \alpha_1) \ldots (x - \alpha_{n-1})$; and the coefficient of x is $(-1)^{n-1} n$ times the number of spanning trees, by the matrix tree theorem.

(c) $\det\bigl(\alpha I - C(K_n)\bigr) = \det\bigl((\alpha - n)I + J\bigr) = (\alpha - n)^{n-1} \alpha$ by exercise 1.2.3–36; here J is the matrix of all 1s. The aspects are therefore $0, n, \ldots, n$.

105. (a) If $e_{ij} = a + b e'_{ij}$ we have $C(G) = naI - aJ + bC(G')$. And if C is any matrix whose row sums are zero, the identity

$$\det(xI + yJ - zC) = \frac{x + ny}{x}\, z^n \det\bigl((x/z)I - C\bigr)$$

can be proved by adding columns 2 through n to column 1, factoring out $(x + ny)/x$, subtracting y/x times column 1 from columns 2 through n, then subtracting columns 2 through n from column 1. Therefore, by setting $x = \alpha - na$, $y = a$, $z = b$, $a = 1$, and $b = -1$, we find that G has the aspects $0, n - \alpha_{n-1}, \ldots, n - \alpha_1$. (In particular, this result agrees with exercise 104(c) when G' is the empty graph $\overline{K_n}$.)

(b) Sort $\{\alpha'_0, \ldots, \alpha'_{n'-1}, \alpha''_0, \ldots, \alpha''_{n''-1}\}$ into order. (An easy case, for variety.)

(c) Here $\overline{G} = \overline{G'} + \overline{G''}$, so G's aspects are $\{0, n' + n'', n'' + \alpha'_1, \ldots, n'' + \alpha'_{n'-1}, n' + \alpha''_1, \ldots, n' + \alpha''_{n''-1}\}$ by (a) and (b). (In particular, G is $K_{m,n}$ when $G' = \overline{K_m}$ and $G'' = \overline{K_n}$, hence the aspects of $K_{m,n}$ are $\{0, (n-1) \cdot m, (m-1) \cdot n, m+n\}$.)

(d) $C(G) = I_{n'} \otimes C(G'') + C(G') \otimes I_{n''}$, where I_n denotes the $n \times n$ identity matrix and $\otimes$ denotes the Kronecker product of matrices. The aspects of $C(G)$ are $\{\alpha'_j + \alpha''_k \mid 0 \leq j < n', 0 \leq k < n''\}$; for if A and B are arbitrary matrices whose eigenvalues are $\{\lambda_1,\ldots,\lambda_m\}$ and $\{\mu_1,\ldots,\mu_n\}$, respectively, the eigenvalues of $A\otimes I_n + I_m\otimes B$ are the mn sums $\lambda_j+\mu_k$. *Proof:* Choose S and T so that S^-AS and T^-BT are triangular. Then use the matrix identity $(A\otimes B)(C\otimes D) = AC\otimes BD$ to show that $(S\otimes T)^-(A\otimes I_n+I_m\otimes B)(S\otimes T) = (S^-AS)\otimes I_n+I_m\otimes(T^-BT)$. (In particular, repeated use of this formula shows that the aspects of the n-cube are $\{\binom{n}{0}\cdot 0,\ \binom{n}{1}\cdot 2,\ \ldots,\ \binom{n}{n}\cdot 2n\}$, and Eq. (57) follows from exercise 104(b).)

(e) When G is a regular graph of degree d, its aspects are $\alpha_j = d-\lambda_{j+1}$, where $\lambda_1\ge\cdots\ge\lambda_n$ are the eigenvalues of the adjacency matrix $A=(e_{ij})$. The adjacency matrix of G' is $A' = B^TB - dI_{n'}$, where $B=(b_{ij})$ is the $n\times n'$ incidence matrix with entries $b_{ij} = [\text{edge } i \text{ touches vertex } j]$, and where $n = n'd/2$ is the number of edges. The adjacency matrix of G is $A = BB^T - 2I_n$. Now we have

$$x^n\det(xI_{n'}-B^TB) \;=\; x^{n'}\det(xI_n - BB^T);$$

this identity follows from the fact that the coefficients of $\det(xI-A)$ can be expressed in terms of $\mathrm{trace}(A^k)$ for $k=1,\,2,\,\ldots$, via Newton's identities (exercise 1.2.9–10). So the aspects of G are the same as those of G', plus $n-n'$ aspects equal to $2d$. [This result is due to E. B. Vakhovsky, *Sibirskiĭ Mat. Zhurnal* **6** (1965), 44–49; see also H. Sachs, *Wissenschaftliche Zeitschrift der Technischen Hochschule Ilmenau* **13** (1967), 405–412.]

(f) $A = A'\otimes A''$, so the aspects are $\{d''\alpha'_j+d'\alpha''_k-\alpha'_j\alpha''_k \mid 0\le j<n',0\le k<n''\}$.

(g) $A(G) = I_{n'}\otimes A'' + A'\otimes I_{n''} + A'\otimes A'' = (I_{n'}+A')\otimes(I_{n''}+A'') - I_n$ yields the aspects $\{(d''+1)\alpha'_j + (d'+1)\alpha''_k - \alpha'_j\alpha''_k \mid 0\le j<n', 0\le k<n''\}$.

106. (a) If α is an aspect of the path P_n, there's a nonzero solution $(x_0,x_1,\ldots,x_{n+1})$ to the equations $\alpha x_k = 2x_k - x_{k-1} - x_{k+1}$ for $1\le k\le n$, with $x_0=x_1$ and $x_n=x_{n+1}$. If we set $x_k=\cos(2k-1)\theta$, we find $x_0=x_1$ and $2x_k-x_{k-1}-x_{k+1} = 2x_k-(2\cos 2\theta)x_k$; hence $2-2\cos 2\theta = 4\sin^2\theta$ will be an aspect if we choose θ so that $x_n=x_{n+1}$ and so that the x's are not all zero. Thus the aspects of P_n turn out to be $\sigma_{0n},\ \ldots,\ \sigma_{(n-1)n}$.

We must have $\alpha_1\ldots\alpha_{n-1}=n$, by exercise 104(b), since $c(P_n)=1$; therefore

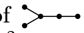
$$c(P_m\times P_n) \;=\; \prod_{j=1}^{m-1}\prod_{k=1}^{n-1}(\sigma_{jm}+\sigma_{kn}).$$

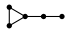
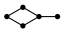
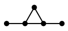
(b, c) Similarly, if α is an aspect of the cycle C_n, there's a nonzero solution to the stated equations with $x_n=x_0$. For this case we try $x_k=\cos 2k\theta$ and find solutions when $\theta=j\pi/n$ for $0\le j<\lceil n/2\rceil$. And $x_k=\sin k\theta$ gives further, linearly independent solutions for $\lceil n/2\rceil\le j<n$. The aspects of C_n are therefore $\sigma_{0n},\ \sigma_{2n},\ \ldots,\ \sigma_{(2n-2)n}$; and we have

$$c(P_m\times C_n) = n\prod_{j=1}^{m-1}\prod_{k=1}^{n-1}(\sigma_{jm}+\sigma_{(2k)n}),\qquad c(C_m\times C_n) = mn\prod_{j=1}^{m-1}\prod_{k=1}^{n-1}(\sigma_{(2j)m}+\sigma_{(2k)n}).$$

Let $f_n(x) = (x+\sigma_{1n})\ldots(x+\sigma_{(n-1)n})$ and $g_n(x) = (x+\sigma_{2n})\ldots(x+\sigma_{(2n-2)n})$. These polynomials have integer coefficients; indeed, $f_n(x) = U_{n-1}(x/2+1)$ and $g_n(x) = 2(T_n(x/2+1)-1)/x$, where $T_n(x)$ and $U_n(x)$ are the Chebyshev polynomials defined by $T_n(\cos\theta)=\cos n\theta$ and $U_n(\cos\theta) = (\sin(n+1)\theta)/\sin\theta$. The calculation of $c(P_m\times P_n)$ can be reduced to the evaluation of an $m\times m$ determinant, because it is the resultant of $f_m(x)$ with $f_n(-x)$; see exercise 4.6.1–12. Similarly, $\frac{1}{n}c(P_m\times C_n)$ and $\frac{1}{mn}c(C_m\times C_n)$ are the respective resultants of $f_m(x)$ with $g_n(-x)$ and of $g_m(x)$ with $g_n(-x)$.

Let $\alpha_n(x) = \prod_{d\backslash n} f_d(x)^{\mu(n/d)}$; thus $\alpha_1(x)=1$, $\alpha_2(x)=x+2$, $\alpha_3(x)=(x+3)\times(x+1)$, $\alpha_4(x)=x^2+4x+2$, $\alpha_5(x)=(x^2+5x+5)(x^2+3x+1)$, $\alpha_6(x)=x^2+4x+1$, etc. By considering so-called field polynomials one can show that $\alpha_n(x)$ is irreducible over the integers when n is even, otherwise it is the product of two irreducible factors of the same degree. Similarly, if $\beta_n(x) = \prod_{d\backslash n} g_d(x)^{\mu(n/d)}$, it turns out that $\beta_n(x)$ is the square of an irreducible polynomial when $n \geq 3$. These facts account for the presence of fairly small prime factors in the results. For example, the largest prime factor in $c(P_m \times P_n)$ for $m \leq n \leq 10$ is 1009; it occurs only in the resultant of $\alpha_6(x)$ with $\alpha_9(-x)$, which is $662913 = 3^2 \cdot 73 \cdot 1009$.

107. There are $(1, 1, 2, 6, 21)$ nonisomorphic graphs for $n = (1, \ldots, 5)$; but we need consider only cases with $\leq \frac{1}{2}\binom{n}{2}$ edges, because of exercise 105(a). The surviving cases when $n = 4$ are free trees: The star is the complement of $K_1 + K_3$, with aspects 0, 1, 1, 4; and P_4 has aspects 0, $2 - \sqrt{2}$, 2, $2 + \sqrt{2}$ by exercise 106. There are three free trees when $n = 5$: The star has aspects 0, 1, 1, 1, 5; P_5's aspects are 0, $2 - \phi$, $3 - \phi$, $1 + \phi$, $2 + \phi$; and the aspects of  are 0, r_1, 1, r_2, r_3, where $(r_1, r_2, r_3) \approx (0.52, 2.31, 4.17)$ are the roots of $x^3 - 7x^2 + 13x - 5 = 0$.

Finally, there are five cases with a single cycle:  is $K_1 \mathbin{\bar{+}} (K_2 + \overline{K_2})$, so its aspects are 0, 1, 1, 3, 5; C_5 has aspects 0, $3 - \phi$, $3 - \phi$, $2 + \phi$, $2 + \phi$;  has aspects 0, r_1, r_2, 3, r_3; its complement  has aspects 0, $5 - r_3$, 2, $5 - r_2$, $5 - r_1$; and the aspects of  turn out to be 0, $(5 - \sqrt{13})/2$, $3 - \phi$, $2 + \phi$, $(5 + \sqrt{13})/2$.

108. Given a digraph D on vertices $\{V_1, \ldots, V_n\}$, let e_{ij} be the number of arcs from V_i to V_j. Define $C(D)$ and its aspects as before. Since $C(D)$ is not necessarily symmetric, the aspects are no longer guaranteed to be real. But if α is an aspect, so is its complex conjugate $\bar{\alpha}$; and if we order the aspects by their real parts, again we find $\alpha_0 = 0$. The formula $c(D) = \alpha_1 \ldots \alpha_{n-1}/n$ remains valid if we now interpret $c(D)$ as the *average* number of *oriented* spanning trees, taken over all n possible roots V_j. The aspects of the transitive tournament T_n, whose arcs are $V_i \to V_j$ for $1 \leq i < j \leq n$, are obviously 0, 1, ..., $n - 1$; and those of its subgraphs are equally obvious.

The derivations in parts (a)–(d) of answer 105 carry over without change. For example, consider $K_1 \mathbin{\bar{+}} T_3$, which has aspects 0, 2, 3, 4; this digraph D has $(2, 4, 6, 12)$ oriented spanning trees with the four possible roots, and $c(D)$ is indeed equal to $2\cdot3\cdot4/4$. Notice also that the digraph  is its own complement, and that it has the same aspects as T_3.

Directed graphs also admit another family of interesting operations: If D' and D'' are digraphs on disjoint sets of vertices V' and V'', consider adding a arcs $v' \to v''$ and b arcs $v'' \to v'$ whenever $v' \in V'$ and $v'' \in V''$. By manipulating determinants as in answer 105(a), we can show that the resulting digraph has aspects $\{0, an'' + bn', an'' + \alpha'_1, \ldots, an'' + \alpha'_{n'-1}, bn' + \alpha''_1, \ldots, bn' + \alpha''_{n''-1}\}$. In the special case $a = 1$ and $b = 0$, we can conveniently denote the new digraph by $D' \to D''$; thus, for example, $T_n = K_1 \to T_{n-1}$. The digraph $K_{n_1} \to K_{n_2} \to \cdots \to K_{n_m}$ on $n_1 + n_2 + \cdots + n_m$ vertices has aspects $\{0,\ n_m \cdot s_m,\ \ldots,\ n_2 \cdot s_2,\ (n_1 - 1) \cdot s_1\}$, where $s_k = n_k + \cdots + n_m$.

The aspects of the oriented path Q_n from V_1 to V_n are obviously 0, 1, ..., 1. The oriented cycle O_n has aspects $\{0, 1 - \omega, \ldots, 1 - \omega^{n-1}\}$, where $\omega = e^{2\pi i/n}$.

There is also a nice result for arc digraphs: The aspects of D^* are obtained from those of D by simply adding $\tau_k - 1$ copies of the number σ_k, for $1 \leq k \leq n$, where τ_k is the in-degree of V_k and σ_k is its out-degree. (If $\tau_k = 0$, we *remove* one aspect equal to σ_k.) The proof is similar to, but simpler than, the derivation in answer 2.3.4.2–21.

Historical remarks: The results in exercises 104(b) and 105(a) are due to A. K. Kelmans, *Avtomatika i Telemekhanika* **26** (1965), 2194–2204; **27**, 2 (February 1966), 56–65; English translation in *Automation and Remote Control* **26** (1965), 2118–2129;

27 (1966), 233–241. Miroslav Fiedler [*Czech. Math. J.* **23** (1973), 298–305] introduced exercise 105(d), and proved interesting results about the aspect α_1, which he called the "algebraic connectivity" of G. Germain Kreweras, in *J. Combinatorial Theory* **B24** (1978), 202–212, enumerated spanning trees on grids, cylinders, and toruses, as well as oriented spanning trees on directed toruses such as $O_m \times O_n$. An excellent survey of graph aspects was published by Bojan Mohar in *Graph Theory, Combinatorics and Applications* (Wiley, 1991), 871–898; *Discrete Math.* **109** (1992), 171–183. For a thorough discussion of important families of graph eigenvalues and their properties, including a comprehensive bibliography, see *Spectra of Graphs* by D. M. Cvetković, M. Doob, and H. Sachs, third edition (1995).

109. Perhaps there is also a sandpile-related reason; see exercise 103.

110. By induction: Suppose there are $k \geq 1$ parallel edges between u and v. Then $c(G) = kc(G_1) + c(G_2)$, where G_1 is G with u and v identified, and G_2 is G with those k edges removed. Let $d_u = k + a$ and $d_v = k + b$.

Case 1: G_2 is connected. Then $ab > 0$, so we can write $a = x + 1$ and $b = y + 1$. We have $c(G_1) > \alpha\sqrt{x+y+1}$ and $c(G_2) > \alpha\sqrt{xy}$, where α is a product over the other $n - 2$ vertices; and it is easy to verify that

$$k\sqrt{x+y+1} + \sqrt{xy} \quad \geq \quad \sqrt{(x+k)(y+k)}.$$

Case 2: There are no such u and v for which G_2 is connected. Then every multi-edge of G is a bridge; in other words, G is a free tree except for parallel edges. In this case the result is trivial if there's a vertex of degree 1. Otherwise suppose u is an endpoint, with $d_u = k$ edges $u — v$. If $d_v > k + 1$, we have $c(G) = kc(G_1) > \alpha k\sqrt{x}$ where $d_v = k + 1 + x$, and it is easy to check that $k\sqrt{x} > \sqrt{(k-1)(k+x)}$ when $x > 0$. If $d_v = k$ we have $c(G) = k > \sqrt{(k-1)^2}$. Finally if $d_v = k + 1$, let $v_0 = u$, $v_1 = v$, and consider the unique path $v_1 — v_2 — \cdots — v_r$ where $r > 1$ and v_r has degree greater than 2; only one edge joins v_j to v_{j+1} for $1 \leq j < r$. Again the induction goes through.

[Other lower bounds on the number of spanning trees have been derived by A. V. Kostochka, *Random Structures and Algorithms* **6** (1995), 269–274.]

111. 2 1 5 4 11 7 9 8 6 10 15 12 14 13 3.

112. Either p appears on an even level and is an ancestor of q, or q appears on an odd level and is an ancestor of p.

113. prepostorder(F^R)=postpreorder$(F)^R$ and postpreorder(F^R)=prepostorder$(F)^R$.

114. The most elegant approach, considering that the forest might be empty, is to set things up so that `CHILD(Λ)` points to the root of the leftmost tree, if any. Then initiate the first visit by setting `Q` $\leftarrow \Lambda$, `L` $\leftarrow -1$, and going to step Q6.

115. Suppose there are n_e nodes on even levels and n_o nodes on odd levels, and that n'_e of the even-level nodes are nonleaves. Then steps (Q1, ..., Q7) are performed respectively $(n_e + n_o, n_o, n'_e, n_e, n'_e, n_o + 1, n_e)$ times, including one execution of Q6 because of answer 114.

116. (a) This result follows from Algorithm Q.

(b) In fact, non-ordinary nodes strictly alternate between lucky and unlucky, beginning and ending with a lucky one. *Proof:* Consider the forest F' obtained by deleting the leftmost leaf of F, and use induction on n.

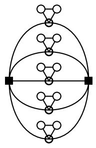
117. Such forests are precisely those whose left-child/right-sibling representation is a degenerate binary tree (exercise 31). So the answer is 2^{n-1}.

118. (a) t^{k-2}, for $k > 1$; luckiness occurs only near extreme leaves.

(b) An interesting recurrence leads to the solution $\bigl(F_k + 1 - (k+1) \bmod 3\bigr)/2$.

119. Label each node x with the value $v(x) = \sum\{\, 2^k \mid k$ is an arc label on the path from the root to $x\}$. Then the node values in prepostorder are exactly the Gray binary code Γ_n, because exercise 113 shows that they satisfy recurrence 7.2.1.1–5.

(If we apply the same value labeling to the ordinary binomial tree T_n and traverse it in preorder, we simply get the integers 0, 1, ..., $2^n - 1$.)



120. False: Only four of the "hollow" vertices in the illustration can appear next to the two "square" vertices, in a Hamiltonian cycle; one hollow pair is therefore out of luck. [See H. Fleischner and H. V. Kronk, *Monatshefte für Mathematik* **76** (1972), 112–117.]

121. Furthermore, there is a Hamiltonian path from u to v in T^2 if and only if similar conditions hold; but we retain u and/or v in $T^{(\prime)}$ if they have degree 1, and we require that the path in (i) be inside the path from u to v (excluding u and v themselves). Condition (ii) is also strengthened by changing 'vertices of degree 4' to 'dangerous vertices', where a vertex of $T^{(\prime)}$ is called dangerous if it either has degree 4 or has degree 2 and equals u or v. The smallest impossible case is $T = P_4$, with u and v chosen to be the non-endpoints. [*Časopis pro Pěstování Matematiky* **89** (1964), 323–338.]

Consequently T^2 contains a Hamiltonian cycle if and only if T is a *caterpillar*, namely a free tree whose derivative is a path. [See Frank Harary and A. J. Schwenk, *Mathematika* **18** (1971), 138-140.]

122. (a) We can represent an expression by a binary tree, with operators at the internal nodes and digits at the external nodes. If binary trees are implemented as in Algorithm B, the essential constraint imposed by the given grammar is that, if $r_j = k > 0$, then the operator at node j is $+$ or $-$ if and only if the operator at node k is $\times$ or $/$. Therefore the total number of possibilities for a tree with n leaves is $2^n S_{n-1}$, where S_n is a Schröder number; namely 10,646,016 when $n = 9$. (See exercise 66, but interchange left with right.) We can rather quickly generate them all, encountering exactly 1640 solutions. Only one expression, namely $1 + 2/\bigl((3-4)/(5+6) - (7-8)/9\bigr)$, does the job with no multiplications; twenty of them, such as $\bigl(\bigl((1-2)/\bigl((3/4) \times 5 - 6\bigr)\bigr) \times 7 + 8\bigr) \times 9$, require five pairs of parentheses; only 15 require no parentheses whatever.

(b) Now there are $1 + \sum_{k=1}^{8} \binom{8}{k} 2^{k+1} S_k = 23{,}463{,}169$ cases, and 3365 solutions. The shortest, of length 12, was found by Dudeney [*The Weekly Dispatch* (18 June 1899)], namely $123 - 45 - 67 + 89$; but he wasn't sure at the time that it was best. The longest solutions have length 27; there are twenty of them, as mentioned above.

(c) The number of cases rises dramatically to $2 + \sum_{k=1}^{8} \binom{8}{k} 4^{k+1} S_k = 8{,}157{,}017{,}474$, and there now are 96,504 solutions. The longest, which is unique, has length 40: $\bigl(\bigl(\bigl((.1/(.2+.3))/.4\bigr)/.5\bigr)/(.6-.7)\bigr)/(.8-.9)$. There are five amusing examples such as $.1 + (2+3+4+5) \times 6 + 7 + 8 + .9$, with seven $+$'s; furthermore, there are ten like $(1 - .2 - .3 - 4 - .5 - 6) \times (7 - 8 - 9)$, with seven $-$'s.

There is in fact very little principle in the thing,
and there is no certain way of demonstrating
that we have got the best possible solution.

— HENRY E. DUDENEY (1899)

Notes: Marie Leske's *Illustriertes Spielbuch für Mädchen*, first published in 1864, contained the earliest known appearance of such a problem; in the eleventh edition

(1889), the fact that $100 = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 \times 9$ was the solution to puzzle 16 in section 553. See also the references in exercise 7.2.1.1–111.

Richard Bellman explained in *AMM* **69** (1962), 640–643, how to handle the special case of part (a) in which the operators are restricted to be either $+$ and $\times$, without parentheses. His technique of dynamic programming can be used also in this more general problem to reduce the number of cases being considered. The idea is to determine the rational numbers obtainable from every subinterval of the digits $\{1, \ldots, n\}$, having a given operator at the root of the tree. We can also save a good deal of computation by discarding cases for the subintervals $\{1, \ldots, 8\}$ and $\{2, \ldots, 9\}$ that cannot lead to integer solutions. In this way the number of essentially different trees to consider is reduced to (a) 2,747,275 cases; (b) 6,834,708; (c) 741,167,401.

Floating point arithmetic is unreliable in this application. But the exact rational arithmetic routines of Section 4.5.1 do the job nicely, never needing to work with an integer greater than 10^9 in absolute value.

123. (a) 2284; but $2284 = (1 + 2 \times 3) \times (4 + 5 \times 67) - 89$. (b) 6964; but $6964 = (1/.2) \times 34 + 5 + 6789$. (c) 14786; but $14786 = -1 + 2 \times (.3 + 4 + 5) \times (6 + 789)$. [If we allow also a minus sign at the left of the expression, as Dudeney did, we actually obtain 1361, 2758, and 85054 additional solutions to problems 120(a), (b), and (c), including nineteen longer expressions in case (a) such as $-(1-2) \times ((3+4) \times (5-(6-7) \times 8)+9)$. With such an extension, the smallest unreachable numbers in the present problem become (a) 3802, (b) 8312, and (c) 17722.] The total number of representable integers (positive, negative, or zero) turns out to be (a) 27,666; (b) 136,607; (c) 200,765.

124. Horton–Strahler numbers originated in studies of river flows: R. E. Horton, *Bull. Geol. Soc. Amer.* **56** (1945), 275–370; A. N. Strahler, *Bull. Geol. Soc. Amer.* **63** (1952), 1117–1142. Many tree-drawing ideas are explored and illustrated in a classic paper by Viennot, Eyrolles, Janey, and Arquès, *Computer Graphics* **23**, 3 (July 1989), 31–40.

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.