
THE CLASSIC WORK
NEWLY UPDATED AND REVISED

The Art of
Computer
Programming

VOLUME 1

Fundamental Algorithms
Third Edition

DONALD E. KNUTH

DONALD E. KNUTH *Stanford University*



ADDISON-WESLEY
An Imprint of Addison Wesley Longman, Inc.

Volume 1 / **Fundamental Algorithms**

THE ART OF COMPUTER PROGRAMMING

THIRD EDITION

Reading, Massachusetts · Harlow, England · Menlo Park, California
Berkeley, California · Don Mills, Ontario · Sydney
Bonn · Amsterdam · Tokyo · Mexico City

T_EX is a trademark of the American Mathematical Society

METRFONT is a trademark of Addison-Wesley

Library of Congress Cataloging-in-Publication Data

Knuth, Donald Ervin, 1938-

The art of computer programming : fundamental algorithms / Donald Knuth. -- 3rd ed.

xx,650 p. 24 cm.

Includes bibliographical references and index.

ISBN 0-201-89683-4

1. Electronic digital computers--Programming. 2. Computer algorithms. I. Title.

QA76.6.K64 1997

005.1--dc21

97-2147

CIP

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

Copyright © 1997 by Addison Wesley Longman

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher. Printed in the United States of America. Published simultaneously in Canada.

ISBN 0-201-89683-4

Text printed on acid-free paper

1 2 3 4 5 6 7 8 9 MA 00999897

First printing, May 1997

write simulators for the hypothetical machine. Such simulators are ideal for instructional purposes, since they are even easier to use than a real computer would be.

I have attempted to cite the best early papers in each subject, together with a sampling of more recent work. When referring to the literature, I use standard abbreviations for the names of periodicals, except that the most commonly cited journals are abbreviated as follows:

CACM = Communications of the Association for Computing Machinery

JACM = Journal of the Association for Computing Machinery

Comp. J. = The Computer Journal (British Computer Society)

Math. Comp. = Mathematics of Computation

AMM = American Mathematical Monthly

SICOMP = SIAM Journal on Computing

FOCS = IEEE Symposium on Foundations of Computer Science

SODA = ACM-SIAM Symposium on Discrete Algorithms

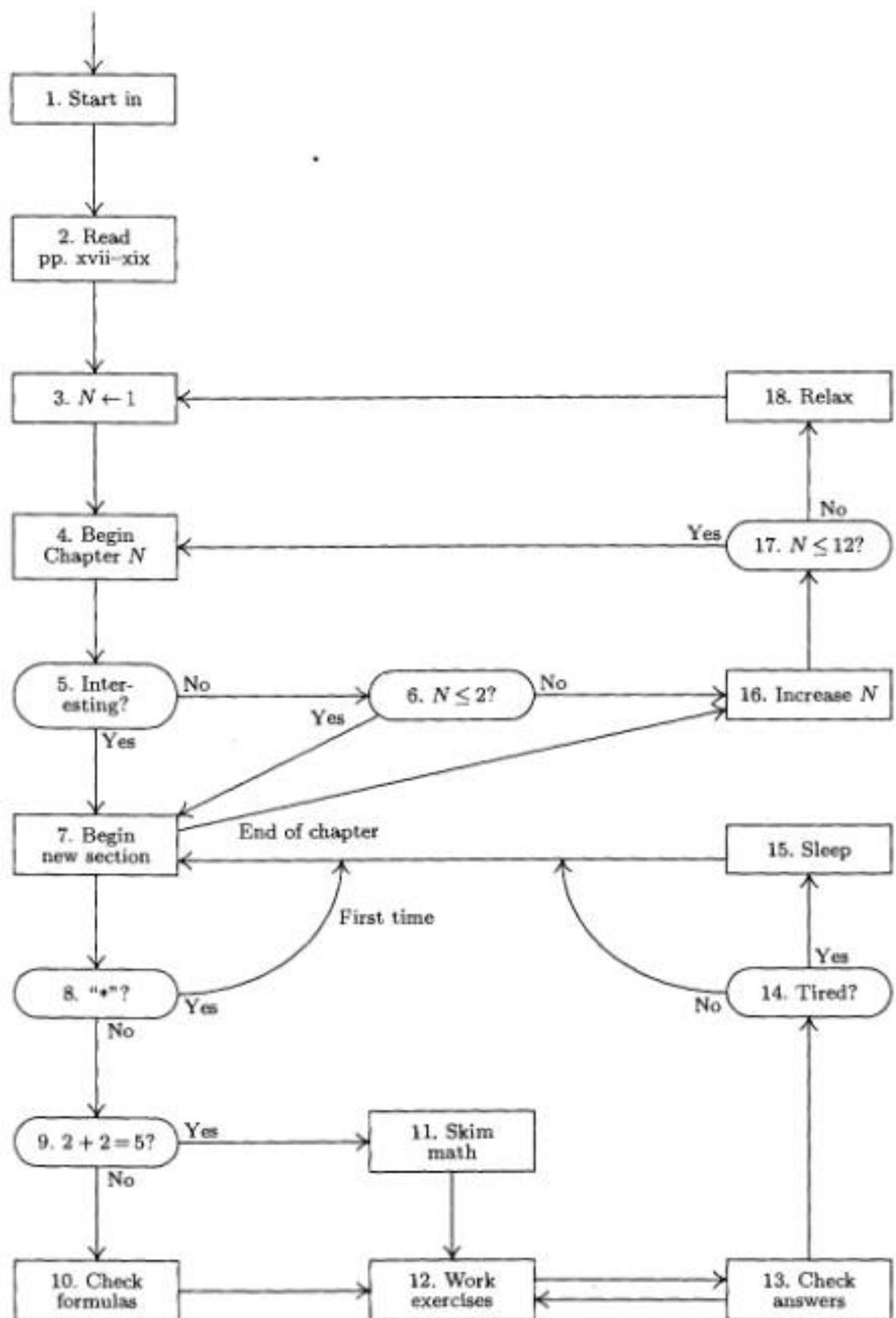
STOC = ACM Symposium on Theory of Computing

Crell = Journal für die reine und angewandte Mathematik

As an example, "CACM 6 (1963), 555–563" stands for the reference given in a preceding paragraph of this preface. I also use "CMath" to stand for the book *Concrete Mathematics*, which is cited in the introduction to Section 1.2.

Much of the technical content of these books appears in the exercises. When the idea behind a nontrivial exercise is not my own, I have attempted to give credit to the person who originated that idea. Corresponding references to the literature are usually given in the accompanying text of that section, or in the answer to that exercise, but in many cases the exercises are based on unpublished material for which no further reference can be given.

I have, of course, received assistance from a great many people during the years I have been preparing these books, and for this I am extremely thankful. Acknowledgments are due, first, to my wife, Jill, for her infinite patience, for preparing several of the illustrations, and for untold further assistance of all kinds; secondly, to Robert W. Floyd, who contributed a great deal of his time towards the enhancement of this material during the 1960s. Thousands of other people have also provided significant help — it would take another book just to list their names! Many of them have kindly allowed me to make use of hitherto unpublished work. My research at Caltech and Stanford was generously supported for many years by the National Science Foundation and the Office of Naval Research. Addison-Wesley has provided excellent assistance and cooperation ever since I began this project in 1962. The best way I know how to thank everyone is to demonstrate by this publication that their input has led to books that resemble what I think they wanted me to write.



Flow chart for reading this set of books.

Procedure for Reading This Set of Books

1. Begin reading this procedure, unless you have already begun to read it. *Continue to follow the steps faithfully.* (The general form of this procedure and its accompanying flow chart will be used throughout this book.)
2. Read the Notes on the Exercises, on pages xv–xvii.
3. Set N equal to 1.
4. Begin reading Chapter N . Do *not* read the quotations that appear at the beginning of the chapter.
5. Is the subject of the chapter interesting to you? If so, go to step 7; if not, go to step 6.
6. Is $N \leq 2$? If not, go to step 16; if so, scan through the chapter anyway. (Chapters 1 and 2 contain important introductory material and also a review of basic programming techniques. You should at least skim over the sections on notation and about MIX.)
7. Begin reading the next section of the chapter; if you have already reached the end of the chapter, however, go to step 16.
8. Is section number marked with "*"? If so, you may omit this section on first reading (it covers a rather specialized topic that is interesting but not essential); go back to step 7.
9. Are you mathematically inclined? If math is all Greek to you, go to step 11; otherwise proceed to step 10.
10. Check the mathematical derivations made in this section (and report errors to the author). Go to step 12.
11. If the current section is full of mathematical computations, you had better omit reading the derivations. However, you should become familiar with the basic results of the section; they are usually stated near the beginning, or in slanted type right at the very end of the hard parts.
12. Work the recommended exercises in this section in accordance with the hints given in the Notes on the Exercises (which you read in step 2).
13. After you have worked on the exercises to your satisfaction, check your answers with the answer printed in the corresponding answer section at the

rear of the book (if any answer appears for that problem). Also read the answers to the exercises you did not have time to work. Note: In most cases it is reasonable to read the answer to exercise n before working on exercise $n + 1$, so steps 12–13 are usually done simultaneously.

14. Are you tired? If not, go back to step 7.
15. Go to sleep. Then, wake up, and go back to step 7.
16. Increase N by one. If $N = 3, 5, 7, 9, 11$, or 12, begin the next volume of this set of books.
17. If N is less than or equal to 12, go back to step 4.
18. Congratulations. Now try to get your friends to purchase a copy of Volume 1 and to start reading it. Also, go back to step 3.

Woe be to him that reads but one book,

— GEORGE HERBERT, *Jacula Prudentum*, 1144 (1640)

*Le défaut unique de tous les ouvrages
c'est d'être trop longs.*

— VAUVENARGUES, *Réflexions*, 628 (1746)

Books are a triviality. Life alone is great.

— THOMAS CARLYLE, *Journal* (1839)

CONTENTS

Chapter 1—Basic Concepts	1
1.1. Algorithms	1
1.2. Mathematical Preliminaries	10
1.2.1. Mathematical Induction	11
1.2.2. Numbers, Powers, and Logarithms	21
1.2.3. Sums and Products	27
1.2.4. Integer Functions and Elementary Number Theory	39
1.2.5. Permutations and Factorials	45
1.2.6. Binomial Coefficients	52
1.2.7. Harmonic Numbers	75
1.2.8. Fibonacci Numbers	79
1.2.9. Generating Functions	87
1.2.10. Analysis of an Algorithm	96
*1.2.11. Asymptotic Representations	107
*1.2.11.1 The O -notation	107
*1.2.11.2 Euler's summation formula	111
*1.2.11.3 Some asymptotic calculations	116
1.3. MIX	124
1.3.1. Description of MIX	124
1.3.2. The MIX Assembly Language	144
1.3.3. Applications to Permutations	164
1.4. Some Fundamental Programming Techniques	186
1.4.1. Subroutines	186
1.4.2. Coroutines	193
1.4.3. Interpretive Routines	200
1.4.3.1. A MIX simulator	202
*1.4.3.2. Trace routines	212
1.4.4. Input and Output	215
1.4.5. History and Bibliography	229
Chapter 2—Information Structures	232
2.1. Introduction	232
2.2. Linear Lists	238
2.2.1. Stacks, Queues, and Deques	238
2.2.2. Sequential Allocation	244
2.2.3. Linked Allocation	254

2.2.4. Circular Lists	273
2.2.5. Doubly Linked Lists	280
2.2.6. Arrays and Orthogonal Lists	298
2.3. Trees	308
2.3.1. Traversing Binary Trees	318
2.3.2. Binary Tree Representation of Trees	334
2.3.3. Other Representations of Trees	348
2.3.4. Basic Mathematical Properties of Trees	362
2.3.4.1. Free trees	363
2.3.4.2. Oriented trees	372
*2.3.4.3. The "infinity lemma"	382
*2.3.4.4. Enumeration of trees	386
2.3.4.5. Path length	399
*2.3.4.6. History and bibliography	406
2.3.5. Lists and Garbage Collection	408
2.4. Multilinked Structures	424
2.5. Dynamic Storage Allocation	435
2.6. History and Bibliography	457
Answers to Exercises	466
Appendix A—Tables of Numerical Quantities	619
1. Fundamental Constants (decimal)	619
2. Fundamental Constants (octal)	620
3. Harmonic Numbers, Bernoulli Numbers, Fibonacci Numbers	621
Appendix B—Index to Notations	623
Index and Glossary	628

d) *Manipulating the domain.* If $R(j)$ and $S(j)$ are arbitrary relations, we have

$$\sum_{R(j)} a_j + \sum_{S(j)} a_j = \sum_{R(j) \text{ or } S(j)} a_j + \sum_{R(j) \text{ and } S(j)} a_j. \quad (11)$$

For example,

$$\sum_{1 \leq j \leq m} a_j + \sum_{m \leq j \leq n} a_j = \left(\sum_{1 \leq j \leq n} a_j \right) + a_m, \quad (12)$$

assuming that $1 \leq m \leq n$. In this case " $R(j)$ and $S(j)$ " is simply " $j = m$," so we have reduced the second sum to simply " a_m ." In most applications of Eq. (11), either $R(j)$ and $S(j)$ are simultaneously satisfied for only one or two values of j , or else it is impossible to have both $R(j)$ and $S(j)$ true for the same j . In the latter case, the second sum on the right-hand side of Eq. (11) simply disappears.

Now that we have seen the four basic rules for manipulating sums, let's study some further illustrations of how to apply these techniques.

Example 1.

$$\begin{aligned} \sum_{0 \leq j \leq n} a_j &= \sum_{\substack{0 \leq j \leq n \\ j \text{ even}}} a_j + \sum_{\substack{0 \leq j \leq n \\ j \text{ odd}}} a_j && \text{by rule (d)} \\ &= \sum_{\substack{0 \leq 2j \leq n \\ 2j \text{ even}}} a_{2j} + \sum_{\substack{0 \leq 2j+1 \leq n \\ 2j+1 \text{ odd}}} a_{2j+1} && \text{by rule (b)} \\ &= \sum_{0 \leq j \leq n/2} a_{2j} + \sum_{0 \leq j < n/2} a_{2j+1}. \end{aligned}$$

The last step merely consists of simplifying the relations below the \sum 's.

Example 2. Let

$$\begin{aligned} S_1 &= \sum_{i=0}^n \sum_{j=0}^i a_i a_j = \sum_{j=0}^n \sum_{i=j}^n a_i a_j && \text{by rule (c) [see Eq. (10)]} \\ &= \sum_{i=0}^n \sum_{j=i}^n a_i a_j && \text{by rule (b),} \end{aligned}$$

interchanging the names i and j and recognizing that $a_j a_i = a_i a_j$. If we denote the latter sum by S_2 , we have

$$\begin{aligned} 2S_1 &= S_1 + S_2 = \sum_{i=0}^n \left(\sum_{j=0}^i a_i a_j + \sum_{j=i}^n a_i a_j \right) && \text{by Eq. (8)} \\ &= \sum_{i=0}^n \left(\left(\sum_{j=0}^n a_i a_j \right) + a_i a_i \right) && \text{by rule (d) [see Eq. (12)]} \end{aligned}$$

$$\begin{aligned}
 &= \sum_{i=0}^n \sum_{j=0}^n a_i a_j + \sum_{i=0}^n a_i a_i && \text{by Eq. (8)} \\
 &= \left(\sum_{i=0}^n a_i \right) \left(\sum_{j=0}^n a_j \right) + \left(\sum_{i=0}^n a_i^2 \right) && \text{by rule (a)} \\
 &= \left(\sum_{i=0}^n a_i \right)^2 + \left(\sum_{i=0}^n a_i^2 \right) && \text{by rule (b).}
 \end{aligned}$$

Thus we have derived the important identity

$$\sum_{i=0}^n \sum_{j=0}^i a_i a_j = \frac{1}{2} \left(\left(\sum_{i=0}^n a_i \right)^2 + \left(\sum_{i=0}^n a_i^2 \right) \right). \quad (13)$$

Example 3 (*The sum of a geometric progression*). Assume that $x \neq 1$, $n \geq 0$. Then

$$\begin{aligned}
 a + ax + \cdots + ax^n &= \sum_{0 \leq j \leq n} ax^j && \text{by definition (2)} \\
 &= a + \sum_{1 \leq j \leq n} ax^j && \text{by rule (d)} \\
 &= a + x \sum_{1 \leq j \leq n} ax^{j-1} && \text{by a very special case of (a)} \\
 &= a + x \sum_{0 \leq j \leq n-1} ax^j && \text{by rule (b) [see Eq. (6)]} \\
 &= a + x \sum_{0 \leq j \leq n} ax^j - ax^{n+1} && \text{by rule (d).}
 \end{aligned}$$

Comparing the first relation with the last, we have

$$(1-x) \sum_{0 \leq j \leq n} ax^j = a - ax^{n+1};$$

hence we obtain the basic formula

$$\sum_{0 \leq j \leq n} ax^j = a \left(\frac{1-x^{n+1}}{1-x} \right). \quad (14)$$

Example 4 (*The sum of an arithmetic progression*). Assume that $n \geq 0$. Then

$$\begin{aligned}
 a + (a+b) + \cdots + (a+nb) &= \sum_{0 \leq j \leq n} (a+bj) && \text{by definition (2)}
 \end{aligned}$$

- 30. [M23] (J. Binet, 1812.) Without using induction, prove the identity

$$\left(\sum_{j=1}^n a_j x_j \right) \left(\sum_{j=1}^n b_j y_j \right) = \left(\sum_{j=1}^n a_j y_j \right) \left(\sum_{j=1}^n b_j x_j \right) + \sum_{1 \leq j < k \leq n} (a_j b_k - a_k b_j)(x_j y_k - x_k y_j).$$

[An important special case arises when $w_1, \dots, w_n, z_1, \dots, z_n$ are arbitrary complex numbers and we set $a_j = w_j, b_j = \bar{z}_j, x_j = \bar{w}_j, y_j = z_j$:

$$\left(\sum_{j=1}^n |w_j|^2 \right) \left(\sum_{j=1}^n |z_j|^2 \right) = \left| \sum_{j=1}^n w_j z_j \right|^2 + \sum_{1 \leq j < k \leq n} |w_j \bar{z}_k - w_k \bar{z}_j|^2.$$

The terms $|w_j \bar{z}_k - w_k \bar{z}_j|^2$ are nonnegative, so the famous *Cauchy-Schwarz inequality*

$$\left(\sum_{j=1}^n |w_j|^2 \right) \left(\sum_{j=1}^n |z_j|^2 \right) \geq \left| \sum_{j=1}^n w_j z_j \right|^2$$

is a consequence of Binet's formula.]

31. [M20] Use Binet's formula to express the sum $\sum_{1 \leq j < k \leq n} (u_j - u_k)(v_j - v_k)$ in terms of $\sum_{j=1}^n u_j v_j$, $\sum_{j=1}^n u_j$, and $\sum_{j=1}^n v_j$.

32. [M20] Prove that

$$\prod_{j=1}^n \sum_{i=1}^m a_{ij} = \sum_{1 \leq i_1, \dots, i_n \leq m} a_{i_1 1} \dots a_{i_n n}.$$

- 33. [M30] One evening Dr. Matrix discovered some formulas that might even be classed as more remarkable than those of exercise 20:

$$\frac{1}{(a-b)(a-c)} + \frac{1}{(b-a)(b-c)} + \frac{1}{(c-a)(c-b)} = 0,$$

$$\frac{a}{(a-b)(a-c)} + \frac{b}{(b-a)(b-c)} + \frac{c}{(c-a)(c-b)} = 0,$$

$$\frac{a^2}{(a-b)(a-c)} + \frac{b^2}{(b-a)(b-c)} + \frac{c^2}{(c-a)(c-b)} = 1,$$

$$\frac{a^3}{(a-b)(a-c)} + \frac{b^3}{(b-a)(b-c)} + \frac{c^3}{(c-a)(c-b)} = a+b+c.$$

Prove that these formulas are a special case of a general law; let x_1, x_2, \dots, x_n be distinct numbers, and show that

$$\sum_{j=1}^n \left(x_j^r \middle/ \prod_{\substack{1 \leq k \leq n \\ k \neq j}} (x_j - x_k) \right) = \begin{cases} 0, & \text{if } 0 \leq r < n-1; \\ 1, & \text{if } r = n-1; \\ \sum_{j=1}^n x_j, & \text{if } r = n. \end{cases}$$

34. [M25] Prove that

$$\sum_{k=1}^n \frac{\prod_{1 \leq r \leq n, r \neq k} (x+k-r)}{\prod_{1 \leq r \leq n, r \neq k} (k-r)} = 1,$$

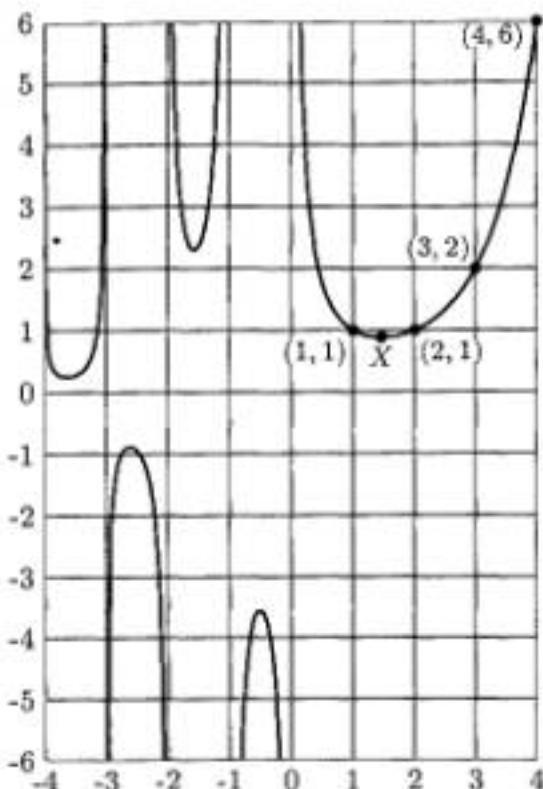


Fig. 7. The function $\Gamma(x) = (x - 1)!$. The local minimum at X has the coordinates (1.46163 21449 68362 34126 26595, 0.88560 31944 10888 70027 88159).

Many formulas of discrete mathematics involve factorial-like products known as *factorial powers*. The quantities x^k and $x^{\bar{k}}$ (read, “ x to the k falling” and “ x to the k rising”) are defined as follows, when k is a positive integer:

$$x^k = x(x - 1) \dots (x - k + 1) = \prod_{j=0}^{k-1} (x - j); \quad (18)$$

$$x^{\bar{k}} = x(x + 1) \dots (x + k - 1) = \prod_{j=0}^{k-1} (x + j); \quad (19)$$

Thus, for example, the number p_{nk} of (2) is just n^k . Notice that we have

$$x^{\bar{k}} = (x + k - 1)^{\bar{k}} = (-1)^k (-x)^k. \quad (20)$$

The general formulas

$$x^k = \frac{x!}{(x - k)!}, \quad x^{\bar{k}} = \frac{\Gamma(x + k)}{\Gamma(x)} \quad (21)$$

can be used to define factorial powers for other values of k . [This notation $x^{\bar{k}}$ is due to A. Capelli, *Giornale di Matematiche di Battaglini* **31** (1893), 291–313.]

The interesting history of factorials from the time of Stirling to the present day is traced in an article by P. J. Davis, “Leonhard Euler’s integral: A historical profile of the gamma function,” *AMM* **66** (1959), 849–869. See also J. Dutka, *Archive for History of Exact Sciences* **31** (1984), 15–34.

D. Addition formula. The basic relation

$$\binom{r}{k} = \binom{r-1}{k} + \binom{r-1}{k-1}, \quad \text{integer } k, \quad (9)$$

is clearly valid in Table 1 (every value is the sum of the two values above and to the left) and we may easily verify it in general from Eq. (3). Alternatively, Eqs. (7) and (8) tell us that

$$r \binom{r-1}{k} + r \binom{r-1}{k-1} = (r-k) \binom{r}{k} + k \binom{r}{k} = r \binom{r}{k}.$$

Equation (9) is often useful in obtaining proofs by induction on r , when r is an integer.

E. Summation formulas. Repeated application of (9) gives

$$\binom{r}{k} = \binom{r-1}{k} + \binom{r-1}{k-1} = \binom{r-1}{k} + \binom{r-2}{k-1} + \binom{r-2}{k-2} = \dots;$$

or

$$\binom{r}{k} = \binom{r-1}{k-1} + \binom{r-1}{k} = \binom{r-1}{k-1} + \binom{r-2}{k-1} + \binom{r-2}{k} = \dots.$$

Thus we are led to two important summation formulas that can be expressed as follows:

$$\sum_{k=0}^n \binom{r+k}{k} = \binom{r}{0} + \binom{r+1}{1} + \dots + \binom{r+n}{n} = \binom{r+n+1}{n}, \quad \text{integer } n \geq 0. \quad (10)$$

$$\sum_{k=0}^n \binom{k}{m} = \binom{0}{m} + \binom{1}{m} + \dots + \binom{n}{m} = \binom{n+1}{m+1}, \quad \text{integer } m \geq 0, \text{ integer } n \geq 0. \quad (11)$$

Equation (11) can easily be proved by induction on n , but it is interesting to see how it can also be derived from Eq. (10) with two applications of Eq. (6):

$$\begin{aligned} \sum_{0 \leq k \leq n} \binom{k}{m} &= \sum_{0 \leq m+k \leq n} \binom{m+k}{m} = \sum_{-m \leq k < 0} \binom{m+k}{m} + \sum_{0 \leq k \leq n-m} \binom{m+k}{k} \\ &= 0 + \binom{m+(n-m)+1}{n-m} = \binom{n+1}{m+1}, \end{aligned}$$

assuming that $n \geq m$. If $n < m$, Eq. (11) is obvious.

Equation (11) occurs very frequently in applications; in fact, we have already derived special cases of it in previous sections. For example, when $m = 1$, we have our old friend, the sum of an arithmetic progression:

$$\binom{0}{1} + \binom{1}{1} + \dots + \binom{n}{1} = 0 + 1 + \dots + n = \binom{n+1}{2} = \frac{(n+1)n}{2}.$$

Stirling numbers of the first kind are used to convert from factorial powers to ordinary powers:

$$\begin{aligned} x^n &= x(x-1)\dots(x-n+1) \\ &= \begin{bmatrix} n \\ n \end{bmatrix} x^n - \begin{bmatrix} n \\ n-1 \end{bmatrix} x^{n-1} + \dots + (-1)^n \begin{bmatrix} n \\ 0 \end{bmatrix} \\ &= \sum_k (-1)^{n-k} \begin{bmatrix} n \\ k \end{bmatrix} x^k. \end{aligned} \quad (44)$$

For example, from Table 2,

$$\binom{x}{5} = \frac{x^5}{5!} = \frac{1}{120}(x^5 - 10x^4 + 35x^3 - 50x^2 + 24x).$$

Stirling numbers of the second kind are used to convert from ordinary powers to factorial powers:

$$x^n = \left\{ \begin{array}{l} n \\ n \end{array} \right\} x^n + \dots + \left\{ \begin{array}{l} n \\ 1 \end{array} \right\} x^1 + \left\{ \begin{array}{l} n \\ 0 \end{array} \right\} x^0 = \sum_k \left\{ \begin{array}{l} n \\ k \end{array} \right\} x^k. \quad (45)$$

This formula was, in fact, Stirling's original reason for studying the numbers $\left\{ \begin{array}{l} n \\ k \end{array} \right\}$ in his *Methodus Differentialis* (London: 1730). From Table 2 we have, for example,

$$\begin{aligned} x^5 &= x^5 + 10x^4 + 25x^3 + 15x^2 + x^1 \\ &= 120 \binom{x}{5} + 240 \binom{x}{4} + 150 \binom{x}{3} + 30 \binom{x}{2} + \binom{x}{1}. \end{aligned}$$

We shall now list the most important identities involving Stirling numbers. In these equations, the variables m and n always denote nonnegative integers.

Addition formulas:

$$\begin{aligned} \begin{bmatrix} n+1 \\ m \end{bmatrix} &= n \begin{bmatrix} n \\ m \end{bmatrix} + \begin{bmatrix} n \\ m-1 \end{bmatrix}; \\ \left\{ \begin{array}{l} n+1 \\ m \end{array} \right\} &= m \left\{ \begin{array}{l} n \\ m \end{array} \right\} + \left\{ \begin{array}{l} n \\ m-1 \end{array} \right\}. \end{aligned} \quad (46)$$

Inversion formulas (compare with Eq. (33)):

$$\sum_k \begin{bmatrix} n \\ k \end{bmatrix} \left\{ \begin{array}{l} k \\ m \end{array} \right\} (-1)^{n-k} = \delta_{mn}, \quad \sum_k \left\{ \begin{array}{l} n \\ k \end{array} \right\} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{n-k} = \delta_{mn}. \quad (47)$$

Special values:

$$\binom{0}{n} = \begin{bmatrix} 0 \\ n \end{bmatrix} = \left\{ \begin{array}{l} 0 \\ n \end{array} \right\} = \delta_{n0}, \quad \binom{n}{n} = \begin{bmatrix} n \\ n \end{bmatrix} = \left\{ \begin{array}{l} n \\ n \end{array} \right\} = 1; \quad (48)$$

$$\begin{bmatrix} n \\ n-1 \end{bmatrix} = \left\{ \begin{array}{l} n \\ n-1 \end{array} \right\} = \binom{n}{2}; \quad (49)$$

$$\left[\begin{matrix} n+1 \\ 0 \end{matrix} \right] = \left\{ \begin{matrix} n+1 \\ 0 \end{matrix} \right\} = 0, \quad \left[\begin{matrix} n+1 \\ 1 \end{matrix} \right] = n!, \quad \left\{ \begin{matrix} n+1 \\ 1 \end{matrix} \right\} = 1, \quad \left\{ \begin{matrix} n+1 \\ 2 \end{matrix} \right\} = 2^n - 1. \quad (50)$$

Expansion formulas:

$$\sum_k \left[\begin{matrix} n \\ k \end{matrix} \right] \binom{k}{m} = \left[\begin{matrix} n+1 \\ m+1 \end{matrix} \right], \quad \sum_k \left[\begin{matrix} n+1 \\ k+1 \end{matrix} \right] \binom{k}{m} (-1)^{k-m} = \left[\begin{matrix} n \\ m \end{matrix} \right]; \quad (51)$$

$$\sum_k \left\{ \begin{matrix} k \\ m \end{matrix} \right\} \binom{n}{k} = \left\{ \begin{matrix} n+1 \\ m+1 \end{matrix} \right\}, \quad \sum_k \left\{ \begin{matrix} k+1 \\ m+1 \end{matrix} \right\} \binom{n}{k} (-1)^{n-k} = \left\{ \begin{matrix} n \\ m \end{matrix} \right\}; \quad (52)$$

$$\sum_k \binom{m}{k} (-1)^{m-k} k^n = m! \left\{ \begin{matrix} n \\ m \end{matrix} \right\}; \quad (53)$$

$$\begin{aligned} \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left\{ \begin{matrix} m+k \\ k \end{matrix} \right\} &= \left[\begin{matrix} n \\ n-m \end{matrix} \right], \\ \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left[\begin{matrix} m+k \\ k \end{matrix} \right] &= \left\{ \begin{matrix} n \\ n-m \end{matrix} \right\}; \end{aligned} \quad (54)$$

$$\sum_k \left\{ \begin{matrix} n+1 \\ k+1 \end{matrix} \right\} \left[\begin{matrix} k \\ m \end{matrix} \right] (-1)^{k-m} = \left(\begin{matrix} n \\ m \end{matrix} \right); \quad (55)$$

$$\sum_{k \leq n} \left[\begin{matrix} k \\ m \end{matrix} \right] \frac{n!}{k!} = \left[\begin{matrix} n+1 \\ m+1 \end{matrix} \right], \quad \sum_{k \leq n} \left\{ \begin{matrix} k \\ m \end{matrix} \right\} (m+1)^{n-k} = \left\{ \begin{matrix} n+1 \\ m+1 \end{matrix} \right\}. \quad (56)$$

Some other fundamental Stirling number identities appear in exercises 1.2.6–61 and 1.2.7–6, and in Eqs. (23), (26), (27), and (28) of Section 1.2.9.

Eq. (49) is just one instance of a general phenomenon: The Stirling numbers $\left[\begin{smallmatrix} n \\ n-m \end{smallmatrix} \right]$ and $\left\{ \begin{smallmatrix} n \\ n-m \end{smallmatrix} \right\}$ are polynomials in n of degree $2m$, whenever m is a nonnegative integer. For example, the formulas for $m = 2$ and $m = 3$ are

$$\begin{aligned} \left[\begin{matrix} n \\ n-2 \end{matrix} \right] &= \binom{n}{4} + 2 \binom{n+1}{4}, & \left\{ \begin{matrix} n \\ n-2 \end{matrix} \right\} &= \binom{n+1}{4} + 2 \binom{n}{4}, \\ \left[\begin{matrix} n \\ n-3 \end{matrix} \right] &= \binom{n}{6} + 8 \binom{n+1}{6} + 6 \binom{n+2}{6}, & \left\{ \begin{matrix} n \\ n-3 \end{matrix} \right\} &= \binom{n+2}{6} + 8 \binom{n+1}{6} + 6 \binom{n}{6}. \end{aligned} \quad (57)$$

Therefore it makes sense to define the numbers $\left[\begin{smallmatrix} r \\ r-m \end{smallmatrix} \right]$ and $\left\{ \begin{smallmatrix} r \\ r-m \end{smallmatrix} \right\}$ for arbitrary real (or complex) values of r . With this generalization, the two kinds of Stirling numbers are united by an interesting duality law

$$\left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \left[\begin{matrix} -m \\ -n \end{matrix} \right], \quad (58)$$

well known in mathematics as *Riemann's zeta function*:

$$H_{\infty}^{(r)} = \zeta(r) = \sum_{k \geq 1} \frac{1}{k^r}. \quad (5)$$

If r is an even integer, the value of $\zeta(r)$ is known to be equal to

$$H_{\infty}^{(r)} = \frac{1}{2} |B_r| \frac{(2\pi)^r}{r!}, \quad \text{integer } r/2 \geq 1, \quad (6)$$

where B_r is a Bernoulli number (see Section 1.2.11.2 and Appendix A). In particular,

$$H_{\infty}^{(2)} = \frac{\pi^2}{6}, \quad H_{\infty}^{(4)} = \frac{\pi^4}{90}, \quad H_{\infty}^{(6)} = \frac{\pi^6}{945}, \quad H_{\infty}^{(8)} = \frac{\pi^8}{9450}. \quad (7)$$

These results are due to Euler; for discussion and proof, see CMath, §6.5.

Now we will consider a few important sums that involve harmonic numbers. First,

$$\sum_{k=1}^n H_k = (n+1)H_n - n. \quad (8)$$

This follows from a simple interchange of summation:

$$\sum_{k=1}^n \sum_{j=1}^k \frac{1}{j} = \sum_{j=1}^n \sum_{k=j}^n \frac{1}{j} = \sum_{j=1}^n \frac{n+1-j}{j}.$$

Formula (8) is a special case of the sum $\sum_{k=1}^n \binom{k}{m} H_k$, which we will now determine using an important technique called summation by parts (see exercise 10). Summation by parts is a useful way to evaluate $\sum a_k b_k$ whenever the quantities $\sum a_k$ and $(b_{k+1} - b_k)$ have simple forms. We observe in this case that

$$\binom{k}{m} = \binom{k+1}{m+1} - \binom{k}{m+1},$$

and therefore

$$\binom{k}{m} H_k = \binom{k+1}{m+1} \left(H_{k+1} - \frac{1}{k+1} \right) - \binom{k}{m+1} H_k;$$

hence

$$\begin{aligned} \sum_{k=1}^n \binom{k}{m} H_k &= \left(\binom{2}{m+1} H_2 - \binom{1}{m+1} H_1 \right) + \dots \\ &\quad + \left(\binom{n+1}{m+1} H_{n+1} - \binom{n}{m+1} H_n \right) - \sum_{k=1}^n \binom{k+1}{m+1} \frac{1}{k+1} \\ &= \binom{n+1}{m+1} H_{n+1} - \binom{1}{m+1} H_1 - \frac{1}{m+1} \sum_{k=0}^n \binom{k}{m} + \frac{1}{m+1} \binom{0}{m}. \end{aligned}$$

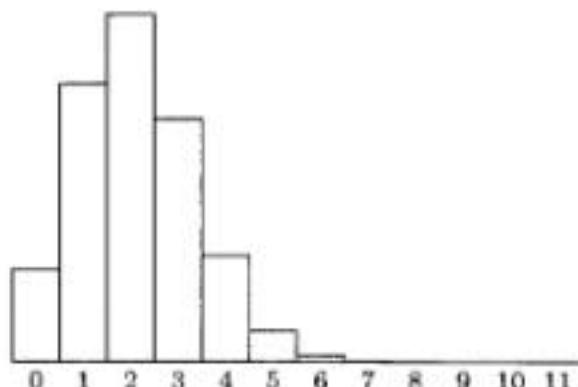


Fig. 10. Probability distribution for step M4, when $n = 12$. The mean is $58301/27720$, or approximately 2.10. The variance is approximately 1.54.

This equation will determine p_{nk} if we provide the initial conditions

$$p_{1k} = \delta_{0k}; \quad p_{nk} = 0 \quad \text{if } k < 0. \quad (5)$$

We can now get information about the quantities p_{nk} by using generating functions. Let

$$G_n(z) = p_{n0} + p_{n1}z + \dots = \sum_k p_{nk} z^k. \quad (6)$$

We know that $A \leq n - 1$, so $p_{nk} = 0$ for large values of k ; thus $G_n(z)$ is actually a polynomial, even though an infinite sum has been specified for convenience.

From Eq. (5) we have $G_1(z) = 1$; and from Eq. (4) we have

$$G_n(z) = \frac{z}{n} G_{n-1}(z) + \frac{n-1}{n} G_{n-1}(z) = \frac{z+n-1}{n} G_{n-1}(z). \quad (7)$$

(The reader should study the relation between Eqs. (4) and (7) carefully.) We can now see that

$$\begin{aligned} G_n(z) &= \frac{z+n-1}{n} G_{n-1}(z) = \frac{z+n-1}{n} \frac{z+n-2}{n-1} G_{n-2}(z) = \dots \\ &= \frac{1}{n!} (z+n-1)(z+n-2)\dots(z+1) \\ &= \frac{1}{z+n} \binom{z+n}{n}. \end{aligned} \quad (8)$$

So $G_n(z)$ is essentially a binomial coefficient!

This function appears in the previous section, Eq. 1.2.9-(27), where we have

$$G_n(z) = \frac{1}{n!} \sum_k \binom{n}{k} z^{k-1}.$$

Therefore p_{nk} can be expressed in terms of Stirling numbers:

$$p_{nk} = \binom{n}{k+1} / n!. \quad (9)$$

Figure 10 shows the approximate sizes of p_{nk} when $n = 12$.

$G_1(z) = q + pz$ we have

$$G_n(z) = (q + pz)^n. \quad (18)$$

Hence, by Theorem A,

$$\text{mean}(G_n) = n \text{ mean}(G_1) = pn;$$

$$\text{var}(G_n) = n \text{ var}(G_1) = (p - p^2)n = pqn.$$

For the number of heads, we have therefore

$$(\min 0, \text{ ave } pn, \text{ max } n, \text{ dev } \sqrt{pqn}). \quad (19)$$

Figure 11 shows the values of p_{nk} when $p = \frac{3}{5}$, $n = 12$. When the standard deviation is proportional to \sqrt{n} and the difference between maximum and minimum is proportional to n , we may consider the situation "stable" about the average.

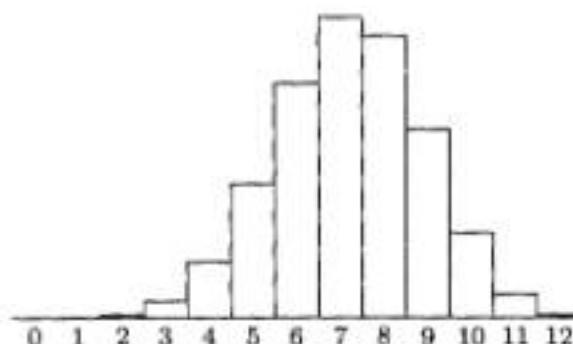


Fig. 11. Probability distribution for coin-tossing: 12 independent tosses with a chance of success equal to $3/5$ at each toss.

Let us work one more simple problem. Suppose that in some process there is *equal* probability of obtaining the values $1, 2, \dots, n$. The generating function for this situation is

$$G(z) = \frac{1}{n}z + \frac{1}{n}z^2 + \dots + \frac{1}{n}z^n = \frac{1}{n} \frac{z^{n+1} - z}{z - 1}. \quad (20)$$

We find after some rather laborious calculation that

$$G'(z) = \frac{nz^{n+1} - (n+1)z^n + 1}{n(z-1)^2};$$

$$G''(z) = \frac{n(n-1)z^{n+1} - 2(n+1)(n-1)z^n + n(n+1)z^{n-1} - 2}{n(z-1)^3}.$$

Now to calculate the mean and variance, we need to know $G'(1)$ and $G''(1)$; but the form in which we have expressed these equations reduces to 0/0 when we substitute $z = 1$. This makes it necessary to find the limit as z approaches unity, and that is a nontrivial task.

Fortunately there is a much simpler way to proceed. By Taylor's theorem we have

$$G(1+z) = G(1) + G'(1)z + \frac{G''(1)}{2!}z^2 + \dots; \quad (21)$$

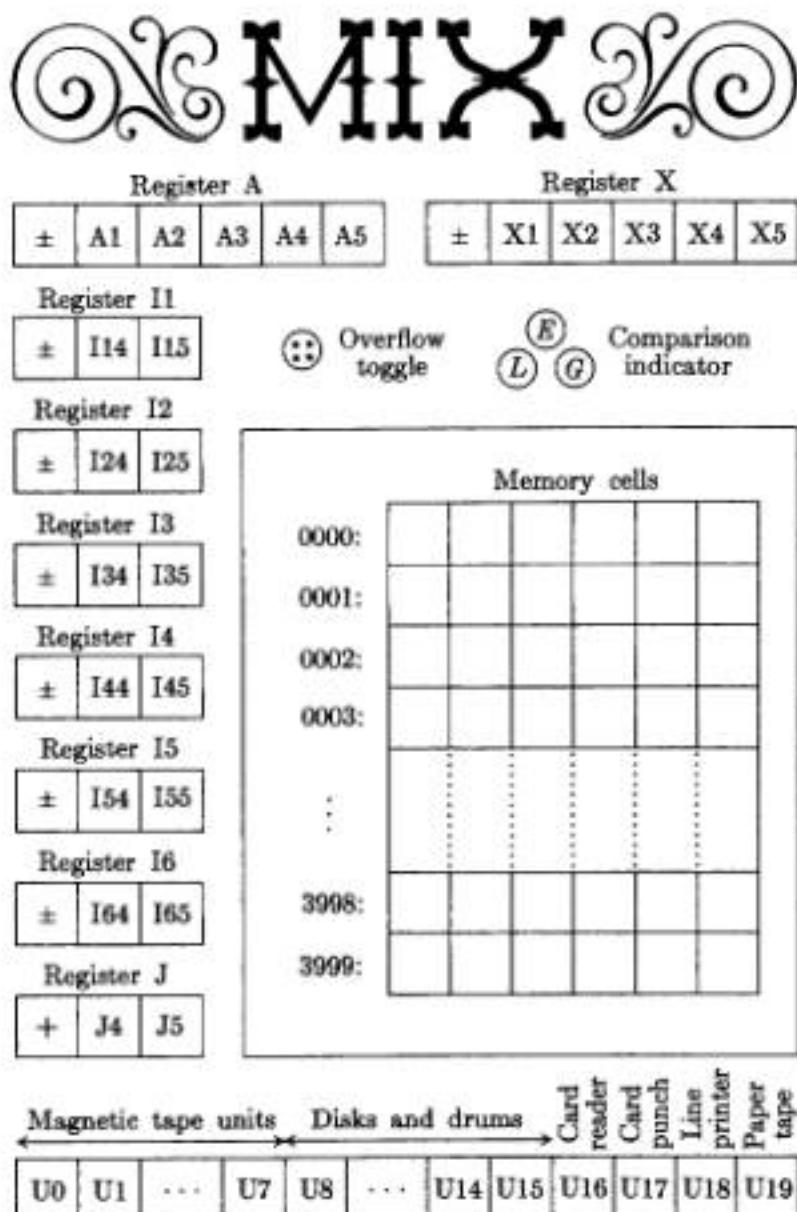


Fig. 13. The MIX computer.

Besides its registers, MIX contains

- an *overflow toggle* (a single bit that is either “on” or “off”);
- a *comparison indicator* (having three values: LESS, EQUAL, or GREATER);
- memory* (4000 words of storage, each word with five bytes and a sign);
- and *input-output devices* (cards, tapes, disks, etc.).

Partial fields of words. The five bytes and sign of a computer word are numbered as follows:

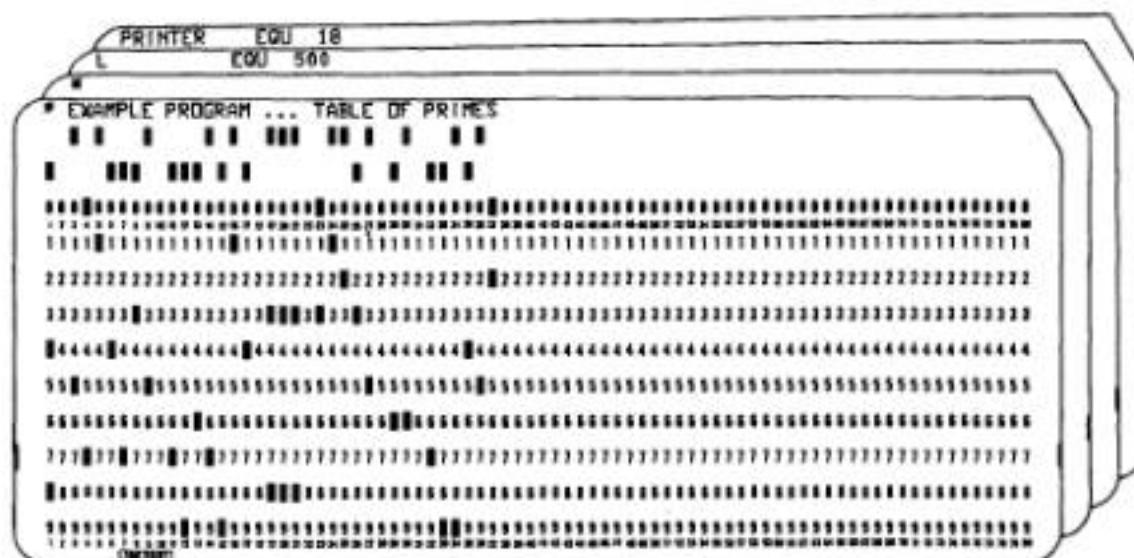
0	1	2	3	4	5
±	Byte	Byte	Byte	Byte	Byte

(2)

(namely, + if the signs of V and rA were the same, - if they were different). The sign of rX afterwards is the previous sign of rA.

Examples of arithmetic instructions: In most cases, arithmetic is done only with MIX words that are single five-byte numbers, not packed with several fields. It is, however, possible to operate arithmetically on packed MIX words, if some caution is used. The following examples should be studied carefully. (As before, ? designates an unknown value.)

		+ 1234 1 150	rA before
		+ 100 5 50	Cell 1000
ADD	1000	+ 1334 6 200	rA after
		- 1234 0 0 9	rA before
		- 2000 150 0	Cell 1000
SUB	1000	+ 766 149 ?	rA after
		+ 1 1 1 1 1	rA before
		+ 1 1 1 1 1	Cell 1000
MUL	1000	+ 0 1 2 3 4	rA after
		+ 5 4 3 2 1	rX after
		- 112	rA before
		? 2 ? ? ? ?	Cell 1000
MUL	1000(1:1)	- 0	rA after
		- 224	rX after
		- 50 0 112 4	rA before
		- 2 0 0 0 0	Cell 1000
MUL	1000	+ 100 0 224	rA after
		+ 8 0 0 0 0	rX after
		+ 0	rA before
		? 17	rX before
DIV	1000	+ 3	Cell 1000
		+ 5	rA after
		+ 2	rX after



```

* EXAMPLE PROGRAM ... TABLE OF PRIMES
*
L EQU 500
PRINTER EQU 18
PRIME EQU -1
BUFO EQU 2000
BUF1 EQU BUFO+25
ORIG 3000
START IOC 0(PRINTER)
LD1 =1-L=

```

Fig. 15. The first lines of Program P punched onto cards, or typed on a terminal.

It may be of interest to note a few of the statistics observed when Program P was actually run. The division instruction in line 19 was executed 9538 times; the time to perform lines 10-24 was 182144u.

MIXAL programs can be punched onto cards or typed on a computer terminal, as shown in Fig. 15. The following format is used in the case of punched cards:

Columns 1-10	LOC (location) field;
Columns 12-15	OP field;
Columns 17-80	ADDRESS field and optional remarks;
Columns 11, 16	blank.

However, if column 1 contains an asterisk, the entire card is treated as a comment. The ADDRESS field ends with the first blank column following column 16; any explanatory information may be punched to the right of this first blank column with no effect on the assembled program. (*Exception:* When the OP field is ALF, the remarks always start in column 22.)

When the input comes from a terminal, a less restrictive format is used: The LOC field ends with the first blank space, while the OP and ADDRESS fields (if present) begin with a nonblank character and continue to the next blank; the special OP-code ALF is, however, followed either by two blank spaces and five characters of alphabetic data, or by a single blank space and five alphabetic

In other words, if $x_0x_1\dots x_{m-1} = \alpha$ and $x_mx_{m+1}\dots x_{m+n-1} = \beta$, we want to change the array $x_0x_1\dots x_{m+n-1} = \alpha\beta$ to the array $x_mx_{m+1}\dots x_{m+n-1}x_0x_1\dots x_{m-1} = \beta\alpha$; this is the permutation on $\{0, 1, \dots, m+n-1\}$ that takes k into $(k+m) \bmod (m+n)$. Show that every such “cyclic-shift” permutation has a simple cycle structure, and exploit that structure to devise a simple algorithm for the desired rearrangement.

35. [M30] Continuing the previous exercise, let $x_0x_1\dots x_{l+m+n-1} = \alpha\beta\gamma$ where α , β , and γ are strings of respective lengths l , m , and n , and suppose that we want to change $\alpha\beta\gamma$ to $\gamma\beta\alpha$. Show that the corresponding permutation has a convenient cycle structure that leads to an efficient algorithm. [Exercise 34 considered the special case $m = 0$.] *Hint:* Consider changing $(\alpha\beta)(\gamma\beta)$ to $(\gamma\beta)(\alpha\beta)$.

36. [27] Write a MIX subroutine for the algorithm in the answer to exercise 35, and analyze its running time. Compare it with the simpler method that goes from $\alpha\beta\gamma$ to $(\alpha\beta\gamma)^R = \gamma^R\beta^R\alpha^R$ to $\gamma\beta\alpha$, where σ^R denotes the left-right reversal of the string σ .

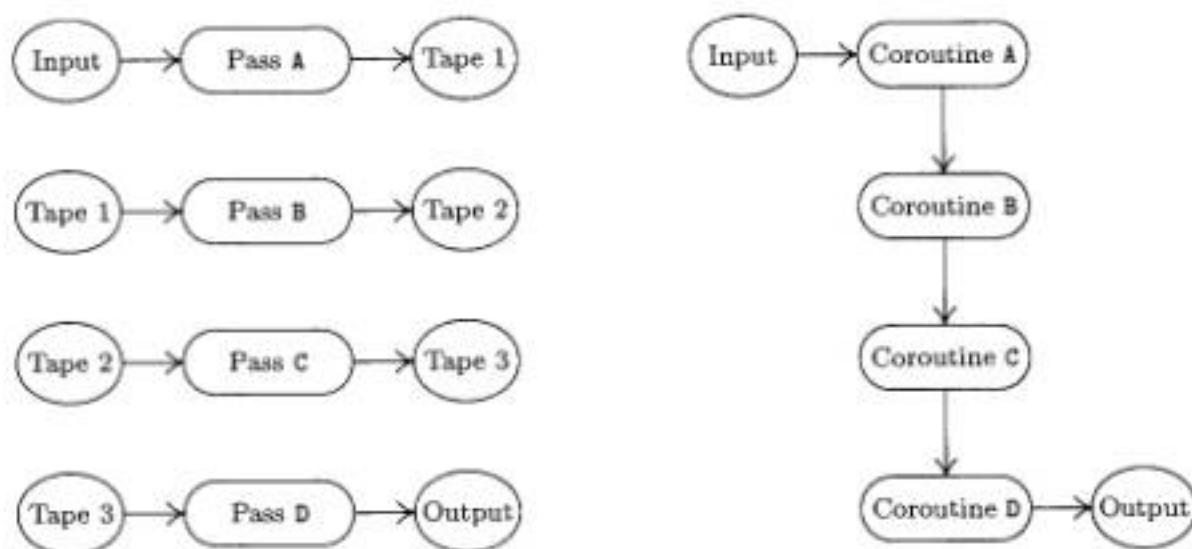


Fig. 22. Passes: (a) a four-pass algorithm, and (b) a one-pass algorithm.

b) *Time difference.* The time required to pack, write, read, and unpack the intermediate data that flows between passes (for example, the information on tapes in Fig. 22) is avoided in a one-pass algorithm. For this reason, a one-pass algorithm will be faster.

c) *Space difference.* The one-pass algorithm requires space to hold all the programs in memory simultaneously, while a multipass algorithm requires space for only one at a time. This requirement may affect the speed, even to a greater extent than indicated in statement (b). For example, many computers have a limited amount of "fast memory" and a larger amount of slower memory; if each pass just barely fits into the fast memory, the result will be considerably faster than if we use coroutines in a single pass (since the use of coroutines would presumably force most of the program to appear in the slower memory or to be repeatedly swapped in and out of fast memory).

Occasionally there is a need to design algorithms for several computer configurations at once, some of which have larger memory capacity than others. In such cases it is possible to write the program in terms of coroutines, and to let the memory size govern the number of passes: Load together as many coroutines as feasible, and supply input or output subroutines for the missing links.

Although this relationship between coroutines and passes is important, we should keep in mind that coroutine applications cannot always be split into multipass algorithms. If coroutine B gets input from A and also sends back crucial information to A, as in the example of chess play mentioned earlier, the sequence of actions can't be converted into pass A followed by pass B.

Conversely, it is clear that some multipass algorithms cannot be converted to coroutines. Some algorithms are inherently multipass; for example, the second pass may require cumulative information from the first pass (like the total

- Step 1. Read five blocks in rapid succession.
- Step 2. Perform a fairly long calculation based on this data.
- Step 3. Return to step 1.

Here five or six buffers would be desirable, so that the next batch of five blocks could be read during step 2. This tendency for I/O activity to be "bunched" makes multiple buffering an improvement over buffer swapping.

Suppose we have N buffers for some input or output process using a single I/O device; we will imagine that the buffers are arranged in a circle, as in Fig. 23. The program external to the buffering process can be assumed to have the following general form with respect to the I/O unit of interest:

```
⋮  
ASSIGN  
⋮  
RELEASE  
⋮  
ASSIGN  
⋮  
RELEASE  
⋮
```

in other words, we can assume that the program alternates between an action called "ASSIGN" and an action called "RELEASE", separated by other computations that do not affect the allocation of buffers.

ASSIGN means that the program acquires the address of the next buffer area; this address is assigned as the value of some program variable.

RELEASE means that the program is done with the current buffer area.

Between **ASSIGN** and **RELEASE** the program is communicating with one of the buffers, called the *current* buffer area; between **RELEASE** and **ASSIGN**, the program makes no reference to any buffer area.

Conceivably, **ASSIGN** could immediately follow **RELEASE**, and discussions of buffering have often been based on this assumption. However, if **RELEASE** is done as soon as possible, the buffering process has more freedom and will be more effective; by separating the two essentially different functions of **ASSIGN** and **RELEASE** we will find that the buffering technique remains easy to understand, and our discussion will be meaningful even if $N = 1$.

To be more explicit, let us consider the cases of input and output separately. For input, suppose we are dealing with a card reader. The action **ASSIGN** means that the program needs to see information from a new card; we would like to set an index register to the memory address at which the next card image is located. The action **RELEASE** occurs when the information in the current card image is no longer needed—it has somehow been digested by the program, perhaps copied

remove the bottom card and make NEWCARD link to it. (This algorithm is sometimes called "cheating" in solitaire games.)

6. [06] In the playing card example, suppose that CARD is the name of a variable whose value is an entire node as in (6). The operation $CARD \leftarrow NODE(TOP)$ sets the fields of CARD respectively equal to those of the top of the pile. After this operation, which of the following notations stands for the suit of the top card? (a) SUIT(CARD); b) SUIT(LOC(CARD)); (c) SUIT(CONTENTS(CARD)); (d) SUIT(TOP) ?

7. [04] In the text's example MIX program, (5), the link variable TOP is stored in the MIX computer word whose assembly language name is TOP. Given the field structure (1), which of the following sequences of code brings the quantity NEXT(TOP) into register A? Explain why the other sequence is incorrect.

a) LDA TOP(NEXT)

b) LD1 TOP

LDA 0,1(NEXT)

8. [18] Write a MIX program corresponding to Algorithm B.

9. [29] Write a MIX program that prints out the alphabetic names of the current contents of the card pile, starting at the top card, with one card per line, and with parentheses around cards that are face down.

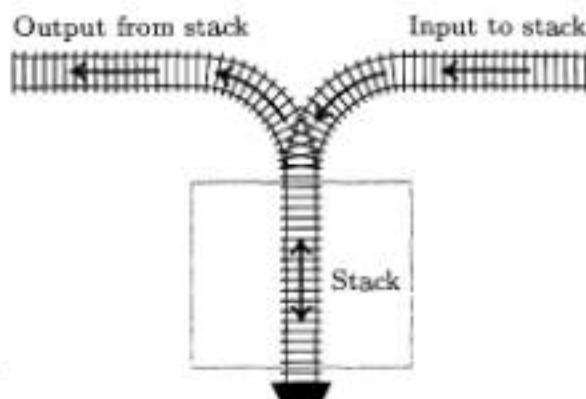


Fig. 1. A stack represented as a railway switching network.

Sometimes it helps to understand the mechanism of a stack in terms of an analogy from the switching of railroad cars, as suggested by E. W. Dijkstra (see Fig. 1). A corresponding picture for deques is shown in Fig. 2.

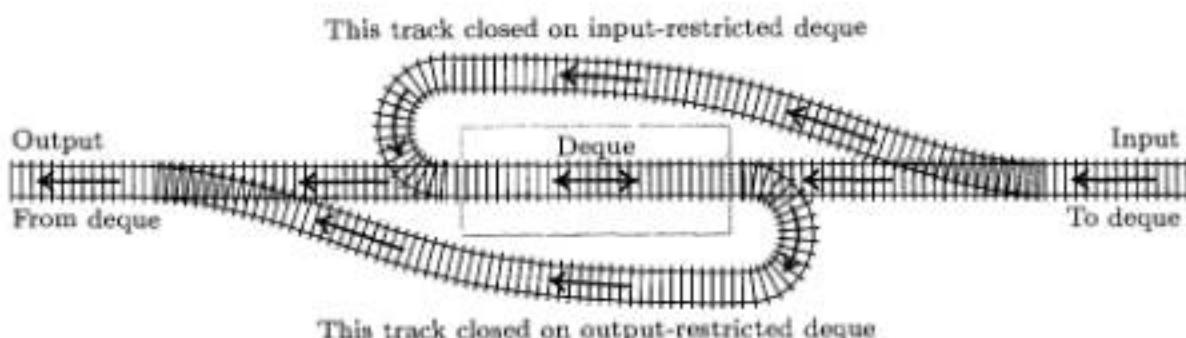


Fig. 2. A deque represented as a railway switching network.

With a stack we always remove the "youngest" item currently in the list, namely the one that has been inserted more recently than any other. With a queue just the opposite is true: The "oldest" item is always removed; the nodes leave the list in the same order as they entered it.

Many people who have independently realized the importance of stacks and queues have given them other names: Stacks have been called push-down lists, reversion storages, cellars, nesting stores, piles, last-in-first-out ("LIFO") lists, and even yo-yo lists. Queues are sometimes called circular stores or first-in-first-out ("FIFO") lists. The terms LIFO and FIFO have been used for many years by accountants, as names of methods for pricing inventories. Still another term, "shelf," has been applied to output-restricted deques, and input-restricted deques have been called "scrolls" or "rolls." This multiplicity of names is interesting in itself, since it is evidence for the importance of the concepts. The words stack and queue are gradually becoming standard terminology; of all the other words listed above, only "push-down list" is still reasonably common, particularly in connection with automata theory.

Stacks arise quite frequently in practice. We might, for example, go through a set of data and keep a list of exceptional conditions or things to do later; after we're done with the original set, we can then do the rest of the processing by

As an example of the input, we might have the pairs

$$9 \prec 2, 3 \prec 7, 7 \prec 5, 5 \prec 8, 8 \prec 6, 4 \prec 6, 1 \prec 3, 7 \prec 4, 9 \prec 5, 2 \prec 8. \quad (18)$$

It is not necessary to give any more pairs than are needed to characterize the desired partial ordering. Thus, additional relations like $9 \prec 8$ (which can be deduced from $9 \prec 5$ and $5 \prec 8$) may be omitted from or added to the input without harm. In general, it is necessary to give only the pairs corresponding to arrows on a diagram such as Fig. 6.

The algorithm that follows uses a sequential table $X[1], X[2], \dots, X[n]$, and each node $X[k]$ has the form

+	0	COUNT[k]	TOP[k]	.
---	---	----------	--------	---

Here COUNT[k] is the *number of direct predecessors* of object k (the number of pairs $j \prec k$ that have appeared in the input), and TOP[k] is a link to the beginning of the *list of direct successors* of object k . The latter list contains entries in the format

+	0	SUC	NEXT	,
---	---	-----	------	---

where SUC is a direct successor of k and NEXT is the next item of the list. As an example of these conventions, Fig. 8 shows the schematic contents of memory corresponding to the input (18).

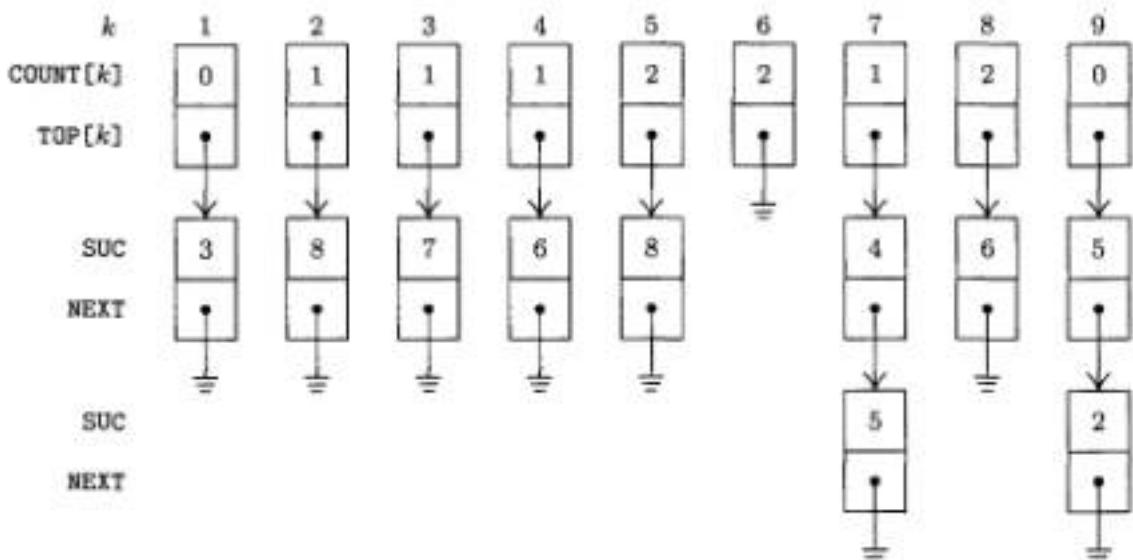


Fig. 8. Computer representation of Fig. 6 corresponding to the relations (18).

Using this memory layout, it is not difficult to work out the algorithm. We want to output the nodes whose COUNT field is zero, then to decrease the COUNT fields of all successors of those nodes by one. The trick is to avoid doing any "searching" for nodes whose COUNT field is zero, and this can be done by maintaining a queue containing those nodes. The links for this queue are kept in the COUNT field, which by now has served its previous purpose; for clarity in the algorithm below, we use the notation QLINK[k] to stand for COUNT[k] when that field is no longer being used to keep a count.

Subroutine D (DECISION subroutine). This subroutine is performed at certain critical times, as specified in the coroutines above, when a decision about the elevator's next direction is to be made.

- D1. [Decision necessary?] If STATE \neq NEUTRAL, exit from this subroutine.
- D2. [Should door open?] If the elevator is positioned at E1 and if CALLUP[2], CALLCAR[2], and CALLDOWN[2] are not all zero, cause the elevator to start its activity E3 after 20 units of time, and exit from this subroutine. (If the DECISION subroutine is currently being invoked by the independent activity E9, it is possible for the elevator coroutine to be positioned at E1.)
- D3. [Any calls?] Find the smallest $j \neq$ FLOOR for which CALLUP[j], CALLCAR[j], or CALLDOWN[j] is nonzero, and go on to step D4. But if no such j exists, then set $j \leftarrow 2$ if the DECISION subroutine is currently being invoked by step E6; otherwise exit from this subroutine.
- D4. [Set STATE.] If FLOOR $>$ j , set STATE \leftarrow GOINGDOWN; if FLOOR $<$ j , set STATE \leftarrow GOINGUP.
- D5. [Elevator dormant?] If the elevator coroutine is positioned at step E1, and if $j \neq 2$, set the elevator to perform step E6 after 20 units of time. Exit from the subroutine. ■

The elevator system described above is quite complicated by comparison with other algorithms we have seen in this book, but the choice of a real-life system is more typical of a simulation problem than any cooked-up "textbook example" would ever be.

To help understand the system, consider Table 1, which gives part of the history of one simulation. It is perhaps best to start by examining the simple case starting at time 4257: The elevator is sitting idly at floor 2 with its doors shut, when a user arrives (time 4384); let's say the user's name is Don. Two seconds later, the doors open, and Don gets in after two more seconds. By pushing button "3" he starts the elevator moving up; ultimately he gets off at floor 3 and the elevator returns to floor 2.

The first entries in Table 1 show a much more dramatic scenario: A user calls the elevator to floor 0, but loses patience and gives up after 15.2 sec. The elevator stops at floor 0 but finds nobody there; then it heads to floor 4, since there are several calls wanting to go downward; etc.

The programming of this system for a computer (in our case, MIX) merits careful study. At any given time during the simulation, we may have many simulated users in the system (in various queues and ready to "give up" at various times), and there is also the possibility of essentially simultaneous execution of steps E4, E5, and E9 if many people are trying to get out as the elevator is trying to close its doors. The passing of simulated time and the handling of "simultaneity" may be programmed by having each entity represented by a node that includes a NEXTTIME field (denoting the time when the next action for this entity is to take place) and a NEXTINST field (denoting the memory address where this entity is to start executing instructions, analogous to ordinary coroutine

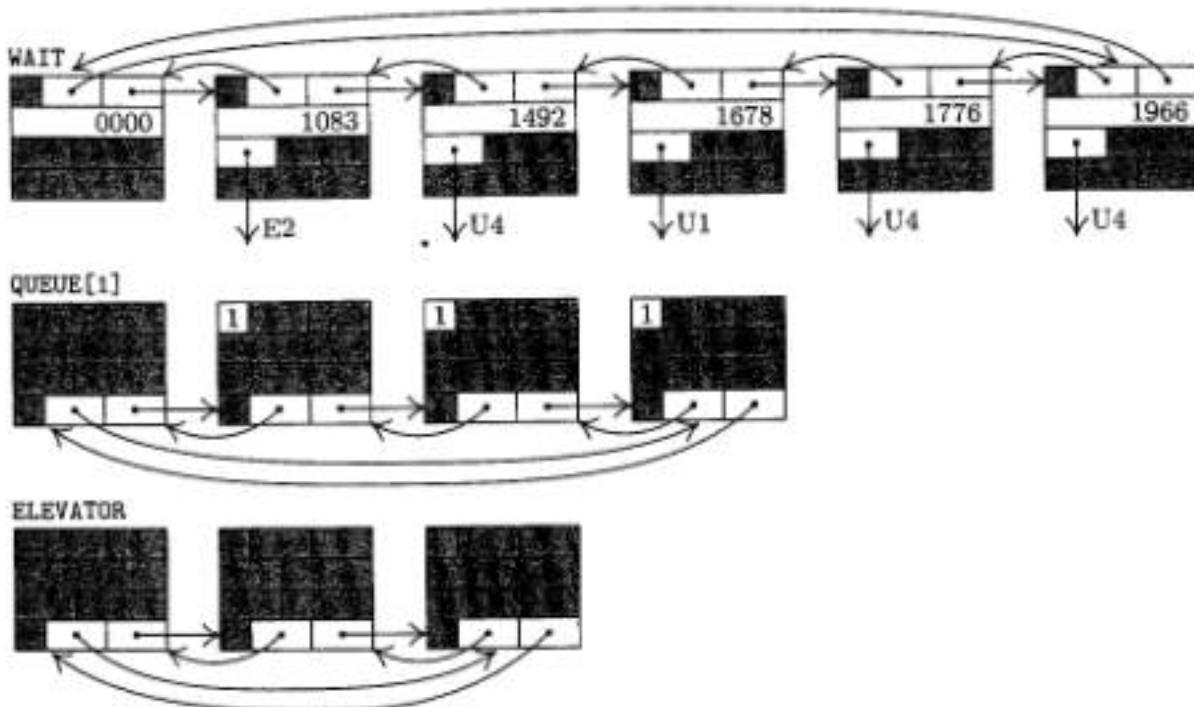


Fig. 12. Some lists used in the elevator simulation program. (List heads appear at the left.)

linkage). Each entity waiting for time to pass is placed in a doubly linked list called the WAIT list; this "agenda" is sorted on the NEXTTIME fields of its nodes, so that the actions may be processed in the correct sequence of simulated times. The program also uses doubly linked lists for the ELEVATOR and for the QUEUE lists.

Each node representing an activity (whether a user or an elevator action) has the form

+	IN	LLINK1	RLINK1	
+	NEXTTIME			
+	NEXTINST	0	0	39
+	OUT	LLINK2	RLINK2	

(6)

Here LLINK1 and RLINK1 are the links for the WAIT list; LLINK2 and RLINK2 are used as links in the QUEUE lists or the ELEVATOR. The latter two fields and the IN and OUT field are relevant when node (6) represents a user, but they are not relevant for nodes that represent elevator actions. The third word of the node is actually a MIX "JMP" instruction.

Figure 12 shows typical contents of the WAIT list, ELEVATOR list, and one of the QUEUE lists; each node in the QUEUE list is simultaneously in the WAIT list with NEXTINST = U4, but this has not been indicated in the figure, since the complexity of the linking would obscure the basic idea.

Now let us consider the program itself. It is quite long, although (as with all long programs) it divides into small parts each of which is quite simple in itself.

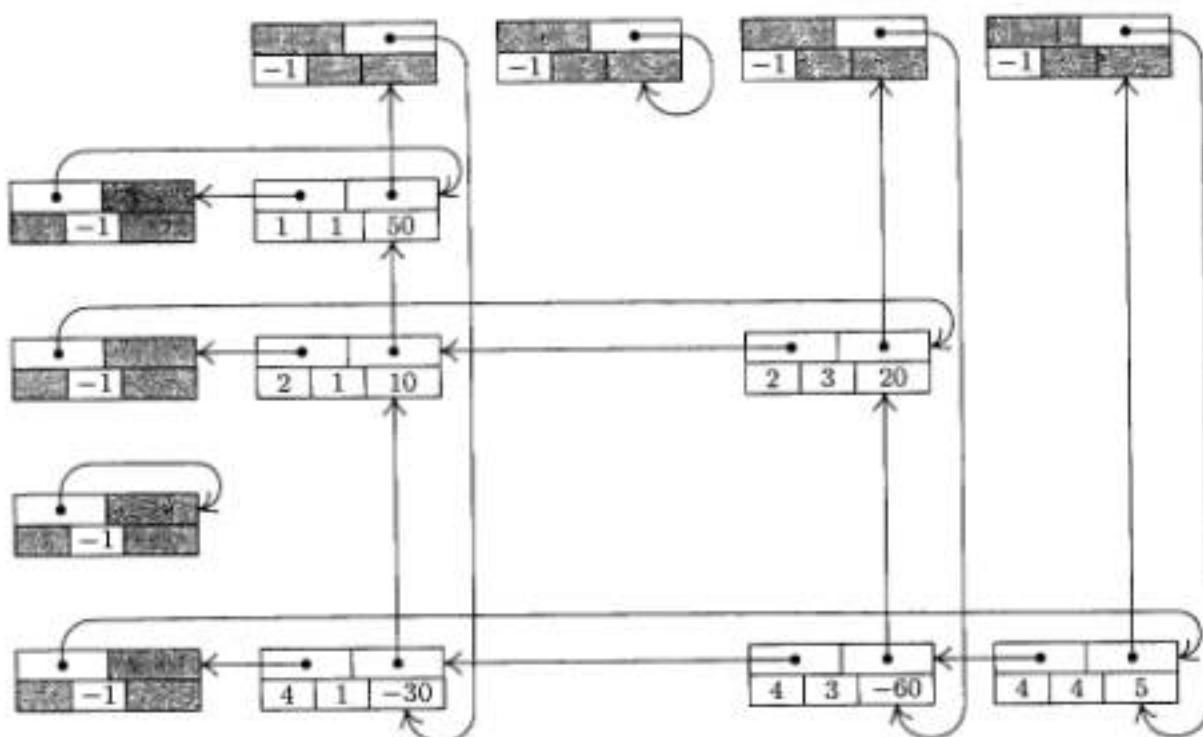


Fig. 14. Representation of matrix (12), with nodes in the format

LEFT	UP
ROW	COL

List heads appear at the left and at the top.

solving linear programming problems by the simplex method. A pivot step is the following matrix transformation:

Before pivot step	After pivot step
Any Pivot other column column	Any Pivot other column column
Pivot row $\begin{pmatrix} \vdots & & \vdots & & \vdots \\ \dots & a & \dots & b & \dots \\ \vdots & & \vdots & & \vdots \\ \dots & c & \dots & d & \dots \\ \vdots & & \vdots & & \vdots \end{pmatrix}$	$\begin{pmatrix} \vdots & & \vdots & & \vdots \\ \dots & 1/a & \dots & b/a & \dots \\ \vdots & & \vdots & & \vdots \\ \dots & -c/a & \dots & d - bc/a & \dots \\ \vdots & & \vdots & & \vdots \end{pmatrix}$
Any other row	

(13)

It is assumed that the *pivot element*, a , is nonzero. For example, a pivot step applied to matrix (12), with the element 10 in row 2 column 1 as pivot, leads to

$$\begin{pmatrix} -5 & 0 & -100 & 0 \\ 0.1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 5 \end{pmatrix}. \quad (14)$$

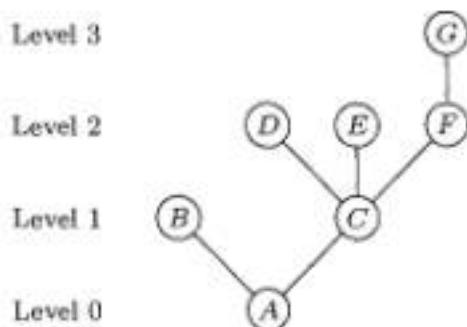


Fig. 15. A tree.

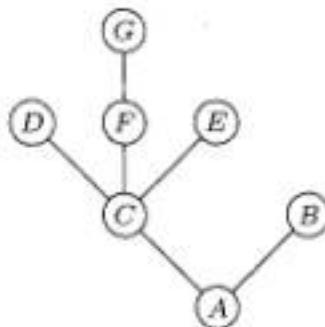


Fig. 16. Another tree.

order, is being considered. The very nature of computer representation defines an implicit ordering for any tree, so in most cases ordered trees are of greatest interest to us. We will therefore tacitly assume that *all trees we discuss are ordered, unless explicitly stated otherwise*. Accordingly, the trees of Figs. 15 and 16 will generally be considered to be different, although they would be the same as oriented trees.

A *forest* is a set (usually an ordered set) of zero or more disjoint trees. Another way to phrase part (b) of the definition of tree would be to say that *the nodes of a tree excluding the root form a forest*.

There is very little distinction between abstract forests and trees. If we delete the root of a tree, we have a forest; conversely, if we add just one node to any forest and regard the trees of the forest as subtrees of the new node, we get a tree. Therefore the words tree and forest are often used almost interchangeably during informal discussions about data structures.

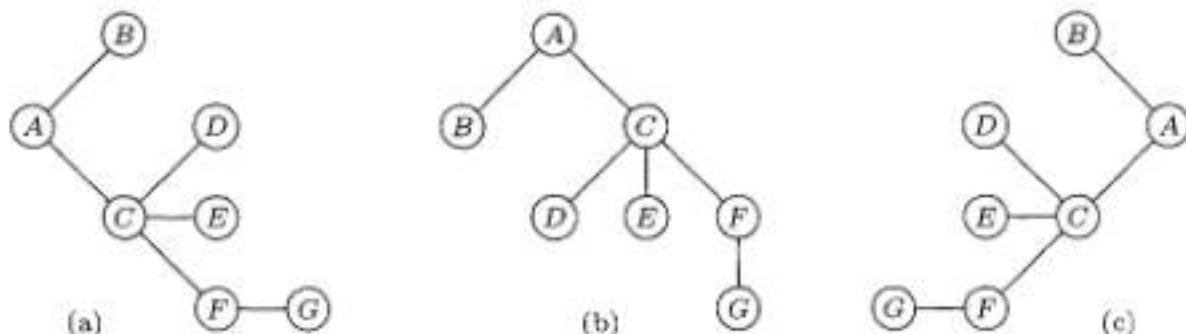


Fig. 17. How shall we draw a tree?

Trees can be drawn in many ways. Besides the diagram of Fig. 15, three of the principal alternatives are shown in Fig. 17, depending on where the root is placed. It is not a frivolous joke to worry about how tree structures are drawn in diagrams, since there are many occasions in which we want to say that one node is "above" or "higher than" another node, or to refer to the "rightmost" element, etc. Certain algorithms for dealing with tree structures have become known as "top down" methods, as opposed to "bottom up." Such terminology leads to confusion unless we adhere to a uniform convention for drawing trees.

It may seem that the form of Fig. 15 would be preferable simply because that is how trees grow in nature; in the absence of any compelling reason to adopt

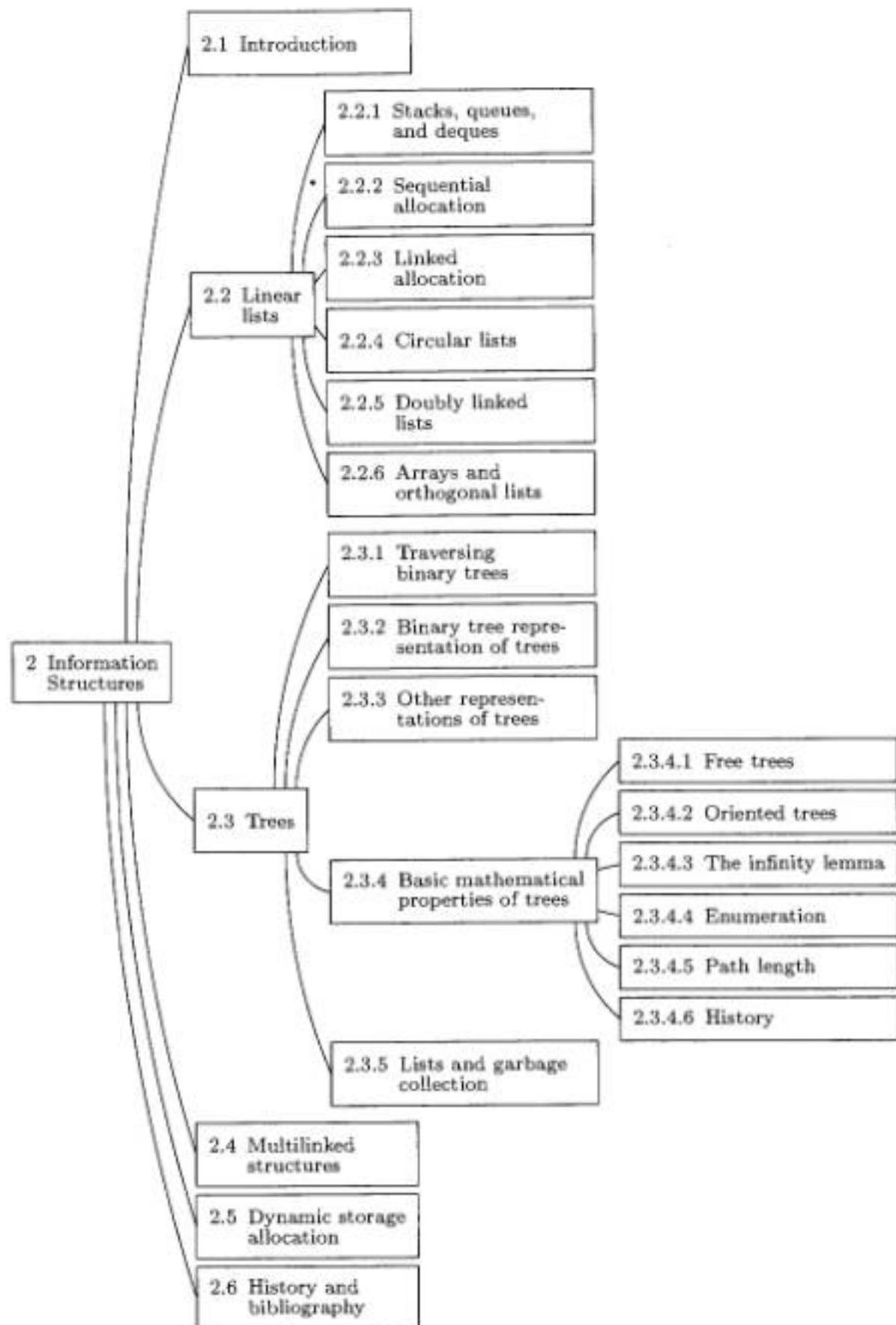


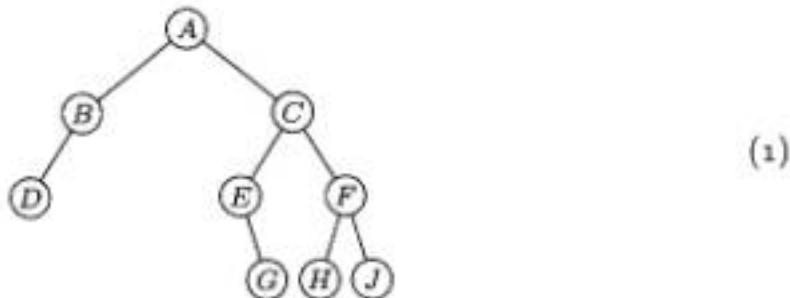
Fig. 22. The structure of Chapter 2.

21. [M22] If a tree has n_1 nodes of degree 1, n_2 nodes of degree 2, ..., and n_m nodes of degree m , how many terminal nodes does it have?
- 22. [21] Standard European paper sizes A0, A1, A2, ... are rectangles whose sides are in the ratio $\sqrt{2}$ to 1. Therefore if we cut a sheet of An paper in half, we get two sheets of A($n+1$) paper. Use this principle to design a graphic representation of binary trees, and illustrate your idea by drawing the representation of 2.3.1-(1) below.

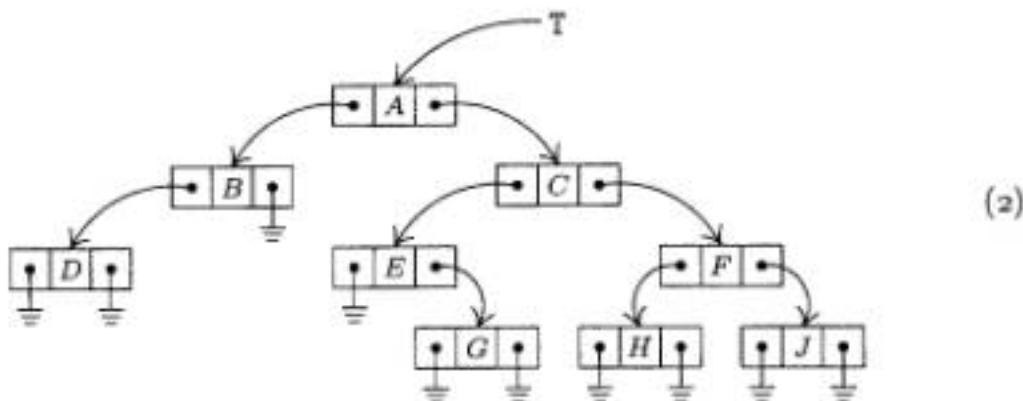
2.3.1. Traversing Binary Trees

It is important to acquire a good understanding of the properties of binary trees before making further investigations of trees, since general trees are usually represented in terms of some equivalent binary tree inside a computer.

We have defined a binary tree as a finite set of nodes that either is empty, or consists of a root together with two binary trees. This definition suggests a natural way to represent binary trees within a computer: We can have two links, LLINK and RLINK, within each node, and a link variable T that is a “pointer to the tree.” If the tree is empty, $T = \Lambda$; otherwise T is the address of the root node of the tree, and LLINK(T), RLINK(T) are pointers to the left and right subtrees of the root, respectively. These rules recursively define the memory representation of any binary tree; for example,



is represented by



This simple and natural memory representation accounts for the special importance of binary tree structures. We will see in Section 2.3.2 that general trees can conveniently be represented as binary trees. Moreover, many trees that arise in applications are themselves inherently binary, so binary trees are of interest in their own right.

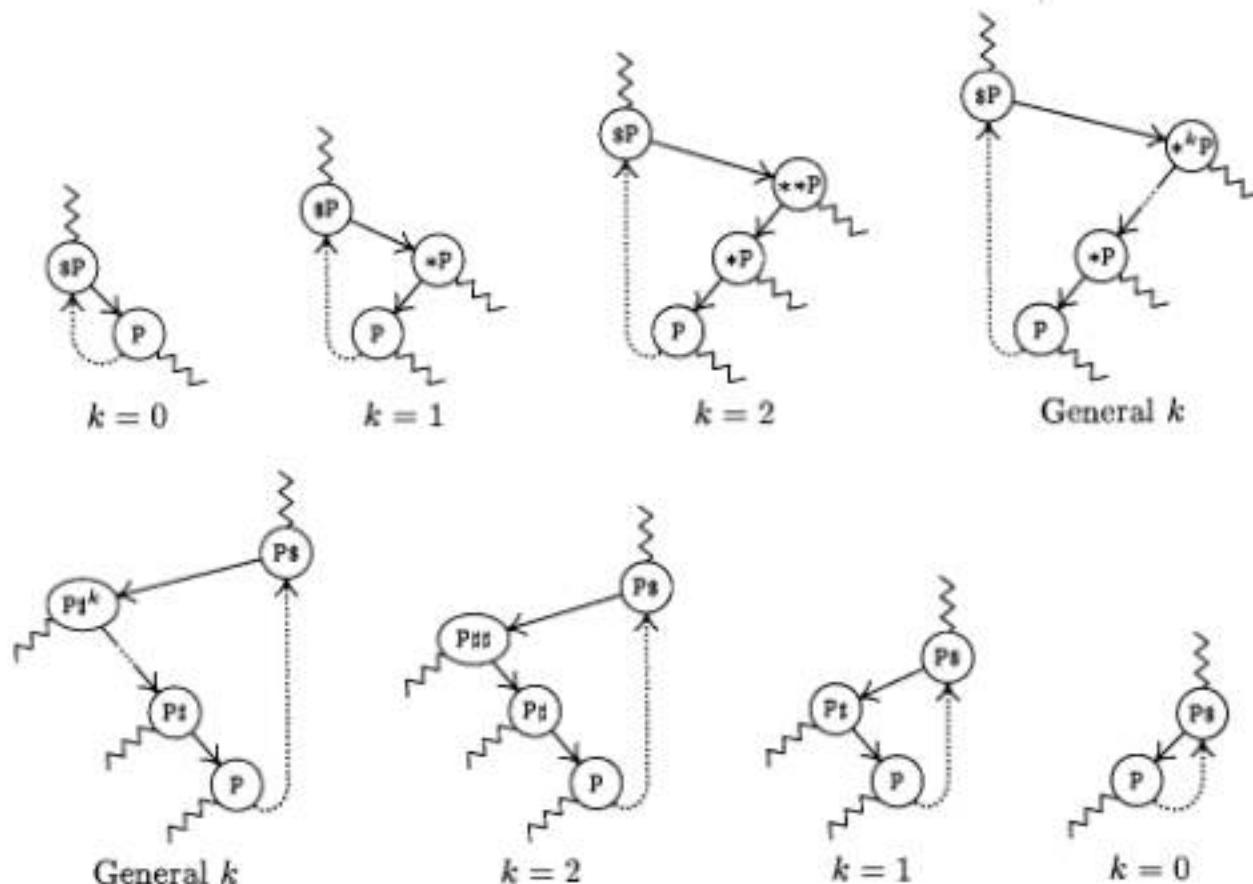


Fig. 24. General orientation of left and right thread links in a threaded binary tree. Wavy lines indicate links or threads to other parts of the tree.

The great advantage of threaded trees is that traversal algorithms become simpler. For example, the following algorithm calculates P_s , given P :

Algorithm S (*Symmetric (inorder) successor in a threaded binary tree*). If P points to a node of a threaded binary tree, this algorithm sets $Q \leftarrow P_s$.

- S1.** [RLINK(P) a thread?] Set $Q \leftarrow \text{RLINK}(P)$. If $\text{RTAG}(P) = 1$, terminate the algorithm.
- S2.** [Search to left.] If $\text{LTAG}(Q) = 0$, set $Q \leftarrow \text{LLINK}(Q)$ and repeat this step. Otherwise the algorithm terminates. ■

Notice that no stack is needed here to accomplish what was done using a stack in Algorithm T. In fact, the ordinary representation (2) makes it impossible to find P_s efficiently, given only the address of a random point P in the tree. Since no links point upward in an unthreaded representation, there is no clue to what nodes are above a given node, unless we retain a history of how we reached that point. The stack in Algorithm T provides the necessary history when threads are absent.

We claim that Algorithm S is "efficient," although this property is not immediately obvious, since step S2 can be executed any number of times. In view of the loop in step S2, would it perhaps be faster to use a stack after all,

as Algorithm T does? To investigate this question, we will consider the average number of times that step S2 must be performed if P is a "random" point in the tree; or what is the same, we will determine the total number of times that step S2 is performed if Algorithm S is used repeatedly to traverse an entire tree.

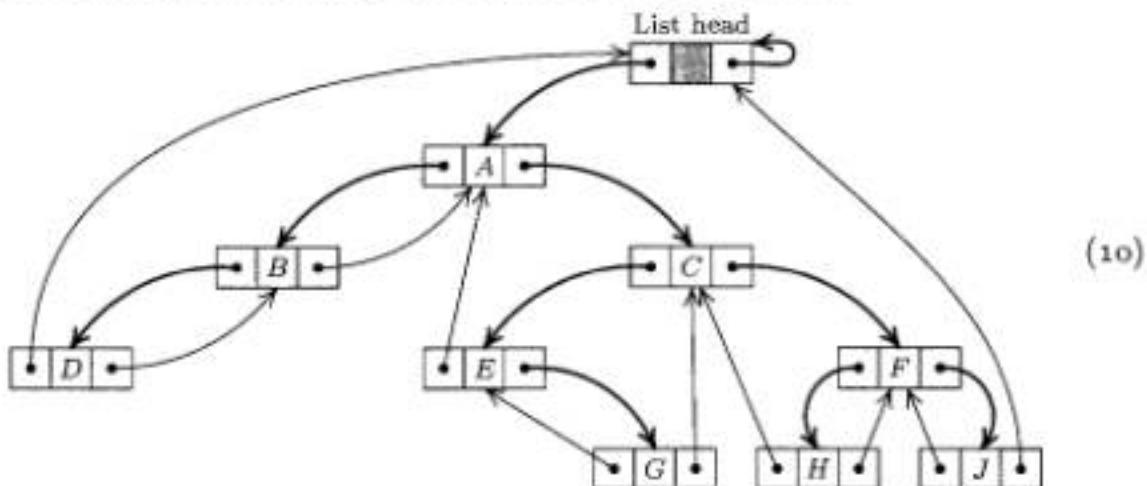
At the same time as this analysis is being carried out, it will be instructive to study complete programs for both Algorithms S and T. As usual, we should be careful to set all of our algorithms up so that they work properly with empty binary trees; and if T is the pointer to the tree, we would like to have $\text{LOC}(T)^*$ and $\text{LOC}(T)_\$$ be the first nodes in preorder or symmetric order, respectively. For threaded trees, it turns out that things will work nicely if $\text{NODE}(\text{LOC}(T))$ is made into a "list head" for the tree, with

$$\begin{array}{ll} \text{LLINK(HEAD)} = T, & \text{LTAG(HEAD)} = 0, \\ \text{RLINK(HEAD)} = \text{HEAD}, & \text{RTAG(HEAD)} = 0. \end{array} \quad (8)$$

(Here HEAD denotes $\text{LOC}(T)$, the address of the list head.) An empty threaded tree will satisfy the conditions

$$\text{LLINK(HEAD)} = \text{HEAD}, \quad \text{LTAG(HEAD)} = 1. \quad (9)$$

The tree grows by having nodes inserted to the left of the list head. (These initial conditions are primarily dictated by the algorithm to compute P^* , which appears in exercise 17.) In accordance with these conventions, the computer representation for the binary tree (1), as a threaded tree, is

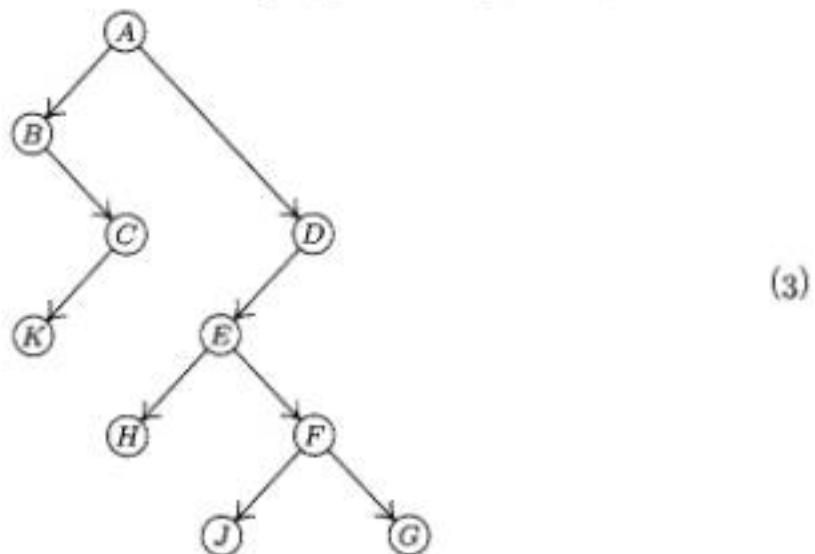


With these preliminaries out of the way, we are now ready to consider MIX versions of Algorithms S and T. The following programs assume that binary tree nodes have the two-word form

LTAG	LLINK	INFO1
RTAG	RLINK	INFO2

In an unthreaded tree, LTAG and RTAG will always be "+" and terminal links will be represented by zero. In a threaded tree, we will use "+" for tags that are 0 and "-" for tags that are 1. The abbreviations LLINKT and RLINKT will be used to stand for the combined LTAG-LLINK and RTAG-RLINK fields, respectively.

Then, tilt the diagram 45° and tweak it slightly, obtaining a binary tree:



Conversely, it is easy to see that any binary tree corresponds to a unique forest of trees by reversing the process.

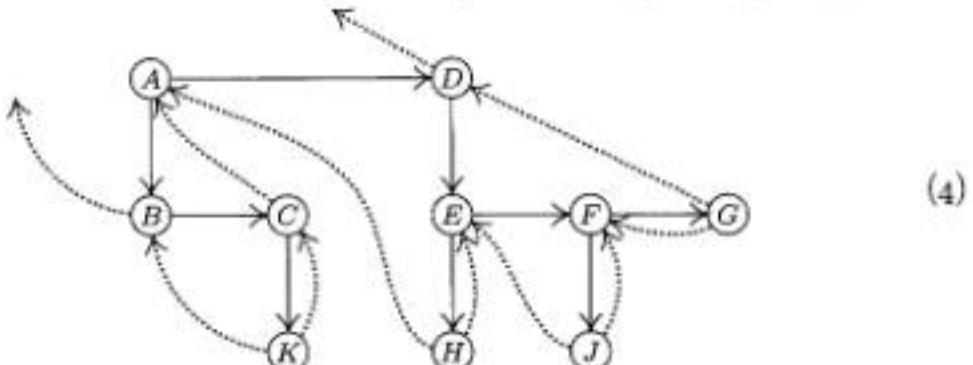
The transformation from (1) to (3) is extremely important; it is called the *natural correspondence* between forests and binary trees. In particular, it gives a correspondence between trees and a special class of binary trees, namely the binary trees that have a root but no right subtree. (We might also change our viewpoint slightly and let the root of a tree correspond to the list head of a binary tree, thus obtaining a one-to-one correspondence between trees with $n+1$ nodes and binary trees with n nodes.)

Let $F = (T_1, T_2, \dots, T_n)$ be a forest of trees. The binary tree $B(F)$ corresponding to F can be defined rigorously as follows:

- If $n = 0$, $B(F)$ is empty.
- If $n > 0$, the root of $B(F)$ is $\text{root}(T_1)$; the left subtree of $B(F)$ is $B(T_{11}, T_{12}, \dots, T_{1m})$, where $T_{11}, T_{12}, \dots, T_{1m}$ are the subtrees of $\text{root}(T_1)$; and the right subtree of $B(F)$ is $B(T_2, \dots, T_n)$.

These rules specify the transformation from (1) to (3) precisely.

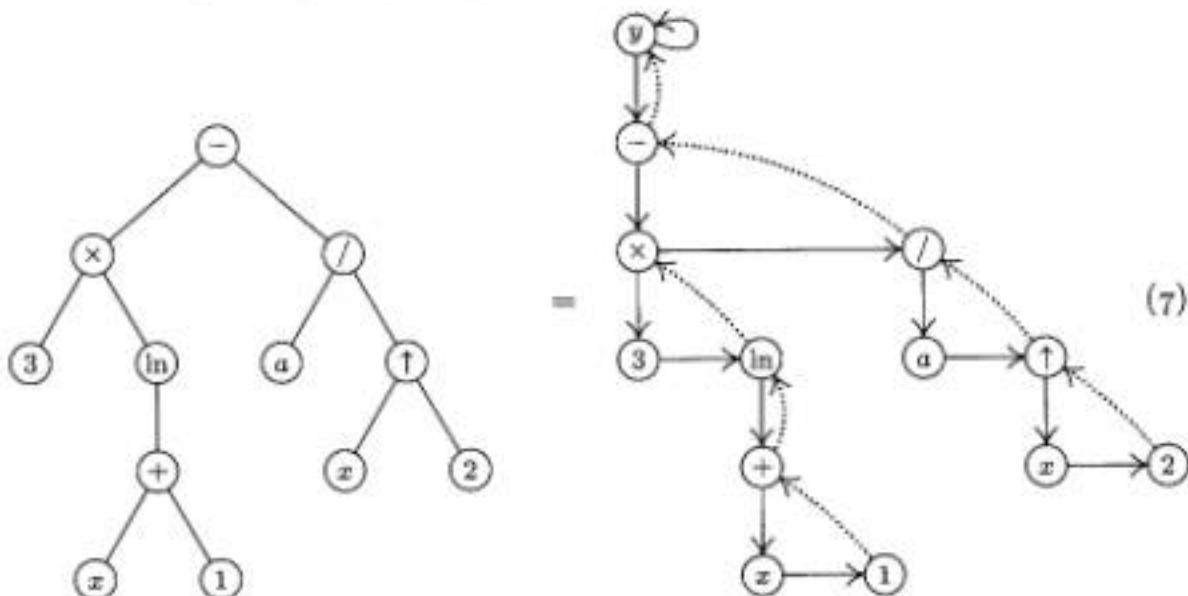
It will occasionally be convenient to draw our binary tree diagram as in (2), without the 45° rotation. The *threaded binary tree* corresponding to (1) is



(compare with Fig. 24, giving the latter a 45° change in orientation). Notice that *right thread links go from the rightmost child of a family to the parent*.

definitions on page 319, we find that traversing a forest in preorder is *exactly the same* as traversing the corresponding binary tree in preorder. Traversing a forest in postorder is exactly the same as traversing the corresponding binary tree in *inorder*. The algorithms developed in Section 2.3.1 may therefore be used without change. (Note that postorder for trees corresponds to inorder, *not* postorder, for binary trees. This is fortunate, since we have seen that it is comparatively hard to traverse binary trees in postorder.) Because of this equivalence, we use the notation P_s for the *postorder* successor of node P in a tree, while it denotes the *inorder* successor in a binary tree.

As an example of the application of these methods to a practical problem, we will consider the manipulation of algebraic formulas. Such formulas are most properly regarded as representations of tree structures, not as one- or two-dimensional configurations of symbols, nor even as binary trees. For example, the formula $y = 3 \ln(x + 1) - a/x^2$ has the tree representation



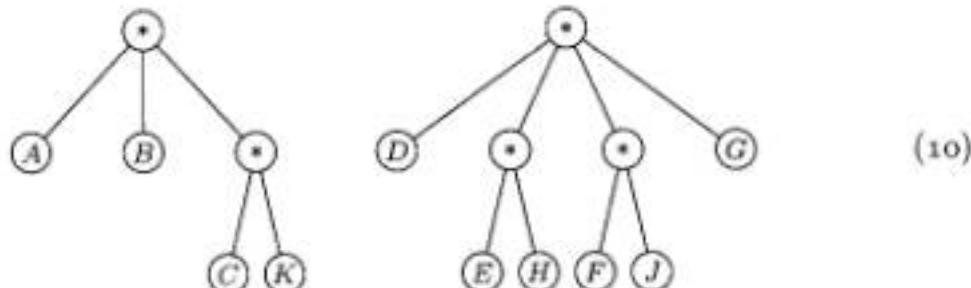
Here the illustration on the left is a conventional tree diagram like Fig. 21, in which the binary operators $+$, $-$, \times , $/$, and \uparrow (the latter denotes exponentiation) have two subtrees corresponding to their operands; the unary operator "ln" has one subtree; variables and constants are terminal nodes. The illustration on the right shows the equivalent right-threaded binary tree, including an additional node y that is a list head for the tree. The list head has the form described in 2.3.1-(8).

It is important to note that, even though the left-hand tree in (7) bears a superficial resemblance to a binary tree, we are treating it here as a *tree*, and representing it by a quite different binary tree, shown at the right in (7). Although we could develop routines for algebraic manipulations based directly on binary tree structures—the so-called "three-address code" representations of algebraic formulas—several simplifications occur in practice if we use the general tree representation of algebraic formulas, as in (7), because postorder traversal is easier in a tree.

The validity of Algorithm F follows by induction on the size of the trees processed (see exercise 16). This algorithm bears a striking similarity to the differentiation procedure of the previous section (Algorithm 2.3.2D), which evaluates a function of a closely related type; see exercise 3. The same idea is used in many interpretive routines in connection with the evaluation of arithmetic expressions in postfix notation; we will return to this topic in Chapter 8. See also exercise 17, which gives another important procedure similar to Algorithm F.

Thus we have seen various sequential representations of trees and forests. There are also a number of *linked* forms of representation, which we shall now consider.

The first idea is related to the transformation that takes (3) into (6): We remove the INFO fields from all nonterminal nodes and put this information as a new terminal node below the previous node. For example, the trees (2) would become



This new form shows that we may assume (without loss of generality) that all INFO in a tree structure appears in its terminal nodes. Therefore in the natural binary tree representation of Section 2.3.2, the LLINK and INFO fields are mutually exclusive and they can share the same field in each node. A node might have the fields

LTAG	LLINK or INFO	RLINK
------	---------------	-------

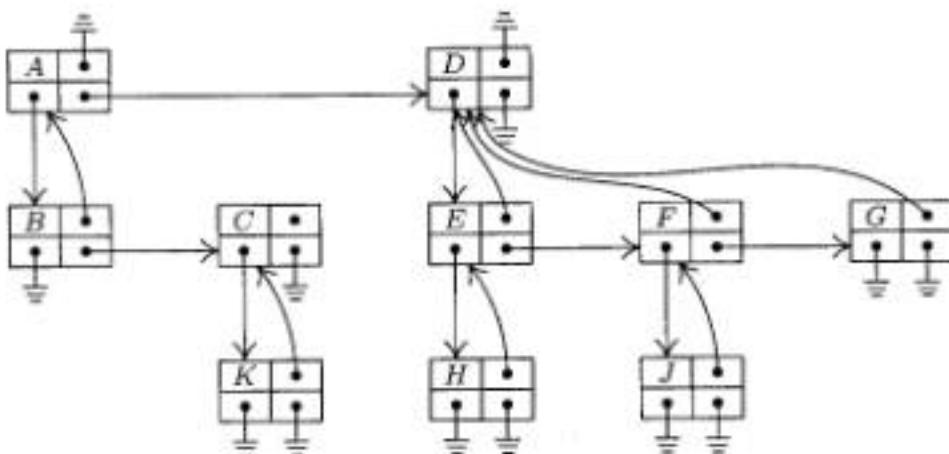
where the sign LTAG tells whether the second field is a link or not. (Compare this representation with, for example, the two-word format of (10) in Section 2.3.2.) By cutting INFO down from 6 bytes to 3, we can fit each node into one word. However, notice that there are now 15 nodes instead of 10; the forest (10) takes 15 words of memory while (2) takes 20, yet the latter has 60 bytes of INFO compared to 30 in the other. There is no real gain in memory space in (10) unless the excess INFO space was going to waste; the LLINKs replaced in (10) are removed at the expense of about the same number of new RLINKs in the added nodes. Precise details of the differences between the two representations are discussed in exercise 4.

In the standard binary tree representation of a tree, the LLINK field might be more accurately called the LCHILD field, since it points from a parent node to its leftmost child. The leftmost child is usually the "youngest" of the children in the tree, since it is easier to insert a node at the left of a family than at the right; so the abbreviation LCHILD may also be thought of as the "last child" or "least child."

Many applications of tree structures require rather frequent references upward in the tree as well as downward. A threaded tree gives us the ability to go upward, but not with great speed; we can sometimes do better if we have a third link, PARENT, in each node. This leads to a *triply linked tree*, where each node has LCHILD, RLINK, and PARENT links. Figure 26 shows a triply linked tree representation of (2). For an example of the use of triply linked trees, see Section 2.4.

INFO	PARENT
LCHILD	RLINK

Fig. 26. A triply linked tree.



It is clear that the PARENT link all by itself is enough to specify any *oriented* tree (or forest) completely. For we can draw the diagram of the tree if we know all the upward links. Every node except the root has just one parent, but there may be several children; so it is simpler to give upward links than downward ones. Why then haven't we considered upward links much earlier in our discussion? The answer, of course, is that upward links by themselves are hardly adequate in most situations, since it is very difficult to tell quickly if a node is terminal or not, or to locate any of its children, etc. There is, however, a very important application in which upward links are sufficient by themselves: We now turn to a brief study of an elegant algorithm for dealing with equivalence relations, due to M. J. Fischer and B. A. Galler.

An *equivalence relation* " \equiv " is a relation between the elements of a set of objects S satisfying the following three properties for any objects x , y , and z (not necessarily distinct) in S :

- i) If $x \equiv y$ and $y \equiv z$, then $x \equiv z$. (Transitivity.)
- ii) If $x \equiv y$, then $y \equiv x$. (Symmetry.)
- iii) $x \equiv x$. (Reflexivity.)

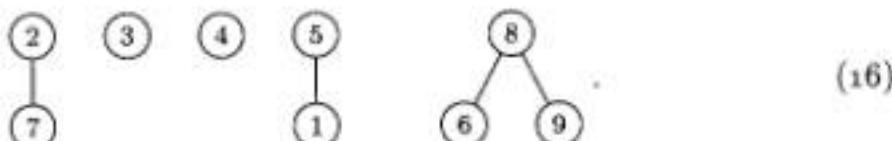
(Compare this with the definition of a partial ordering relation in Section 2.2.3; equivalence relations are quite different from partial orderings, in spite of the fact that two of the three defining properties are the same.) Examples of equivalence relations are the relation " $=$ ", the relation of congruence (modulo m) for integers, the relation of similarity between trees as defined in Section 2.3.1, etc.

- E1.** [Initialize.] Set $\text{PARENT}[k] \leftarrow 0$ for $1 \leq k \leq n$. (This means that all trees initially consist of a root alone, as in (13).)
- E2.** [Input new pair.] Get the next pair of equivalent elements " $j \equiv k$ " from the input. If the input is exhausted, the algorithm terminates.
- E3.** [Find roots.] If $\text{PARENT}[j] > 0$, set $j \leftarrow \text{PARENT}[j]$ and repeat this step. If $\text{PARENT}[k] > 0$, set $k \leftarrow \text{PARENT}[k]$ and repeat this step. (After this operation, j and k have moved up to the roots to two trees that are to be made equivalent. The input relation $j \equiv k$ was redundant if and only if we now have $j = k$.)
- E4.** [Merge trees.] If $j \neq k$, set $\text{PARENT}[j] \leftarrow k$. Go back to step E2. \blacksquare

The reader should try this algorithm on the input (11). After processing $1 \equiv 5, 6 \equiv 8, 7 \equiv 2$, and $9 \equiv 8$, we will have

$$\begin{array}{ll} \text{PARENT}[k]: & 5 \ 0 \ 0 \ 0 \ 0 \ 8 \ 2 \ 0 \ 8 \\ k : & 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \end{array} \quad (15)$$

which represents the trees



After this point, the remaining relations of (11) are somewhat more interesting; see exercise 9.

This equivalence problem arises in many applications. We will discuss significant refinements of Algorithm E in Section 7.4.1, when we study the connectivity of graphs. A more general version of the problem, which arises when a compiler processes "equivalence declarations" in languages like FORTRAN, is discussed in exercise 11.

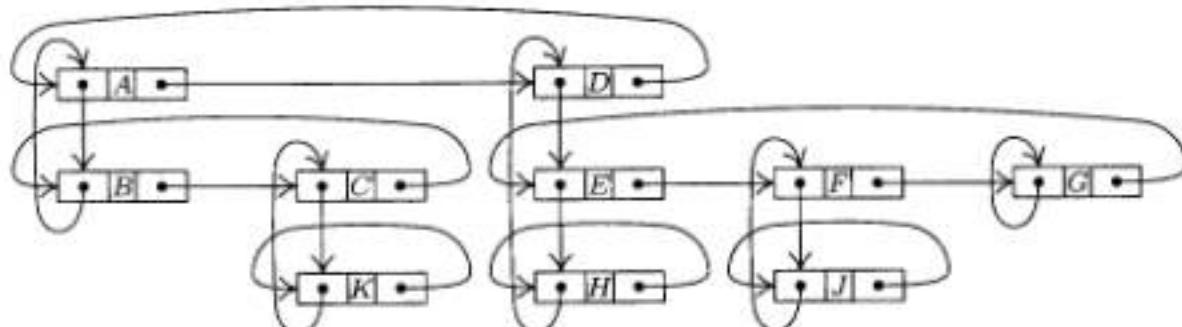


Fig. 27. A ring structure.

There are still more ways to represent trees in computer memory. Recall that we discussed three principal methods for representing linear lists in Section 2.2: the straight representation with terminal link Λ , the circularly linked lists, and the doubly linked lists. The representation of unthreaded binary trees described in Section 2.3.1 corresponds to a straight representation in both LLINKs and

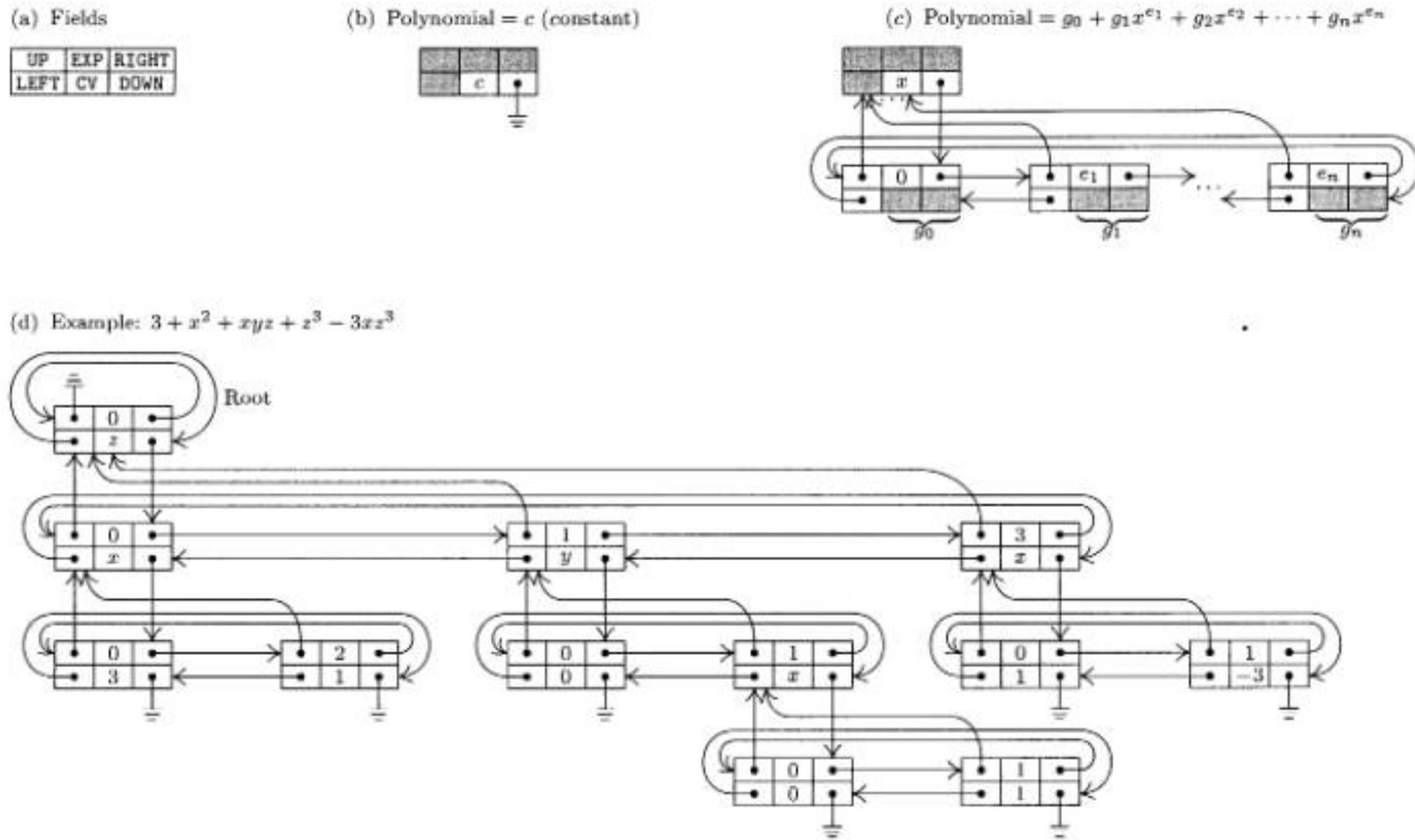


Fig. 28. Representation of polynomials using four-directional links. Shaded areas of nodes indicate information that is irrelevant in the context considered.

For example, if G is the graph of Fig. 29 on page 363, with $(V_0, V_1, V_2, V_3, V_4) = (A, B, C, D, E)$, we find that

$$B = \begin{pmatrix} 3 & 0 & -1 & -1 \\ 0 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{pmatrix}.$$

Show that the number of free subtrees of G is $\det B$. [Hint: Use exercise 18 or 19.]

21. [HM38] (T. van Aardenne-Ehrenfest and N. G. de Bruijn.) Fig. 36 is an example of a directed graph that is not only balanced, it is *regular*, which means that every vertex has the same in-degree and out-degree as every other vertex. Let G be a regular digraph with $n + 1$ vertices V_0, V_1, \dots, V_n , in which every vertex has in-degree and out-degree equal to m . (Hence there are $(n + 1)m$ arcs in all.) Let G^* be the graph with $(n + 1)m$ vertices corresponding to the arcs of G ; let a vertex of G^* corresponding to an arc from V_j to V_k in G be denoted by V_{jk} . An arc goes from V_{jk} to $V_{j'k'}$ in G^* if and only if $k = j'$. For example, if G is the directed graph of Fig. 36, G^* is shown in Fig. 37. An Eulerian circuit in G is a Hamiltonian circuit in G^* and conversely.

Prove that the number of oriented subtrees of G^* is $m^{(n+1)(m-1)}$ times the number of oriented subtrees of G . [Hint: Use exercise 19.]

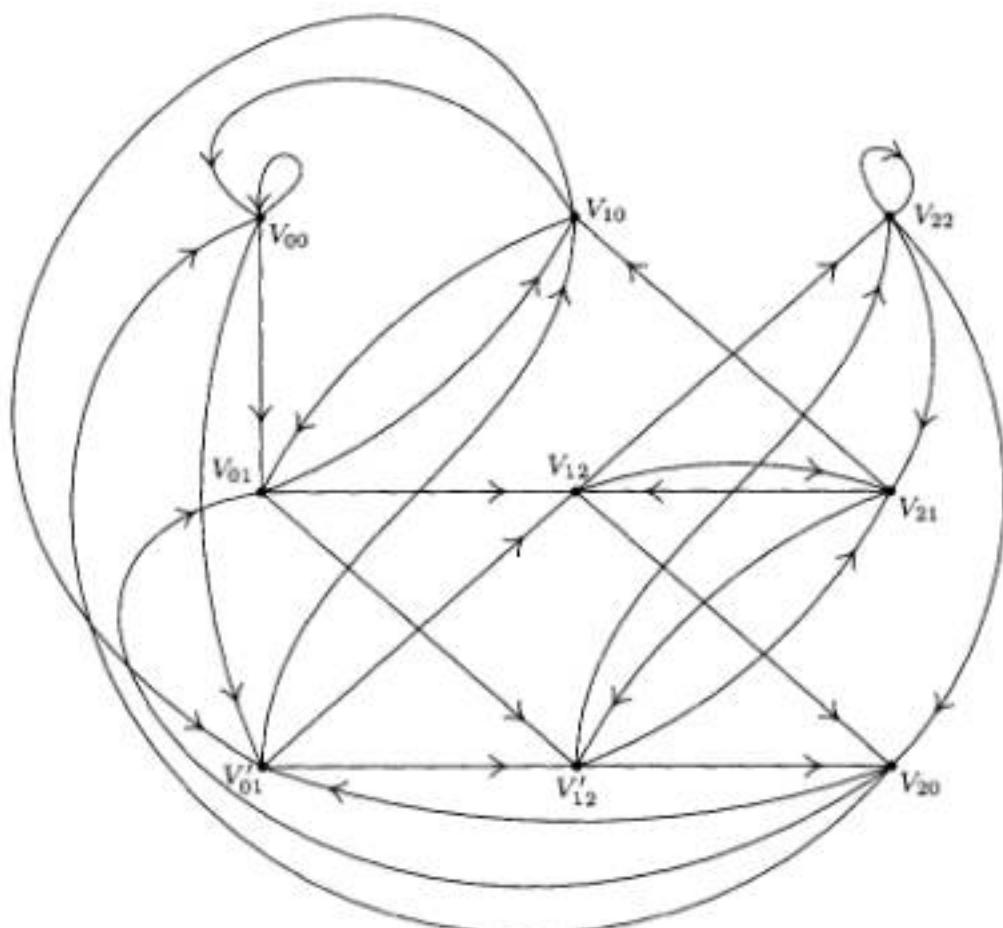


Fig. 37. Arc digraph corresponding to Fig. 36. (See exercise 21.)

Now by (26),

$$\begin{aligned} w &= \sum_{i=1}^s \sum_k \sum_{\substack{\mathbf{n}, \mathbf{q} \\ \sum(\mathbf{n}-\mathbf{q})=1}} \frac{r(\mathbf{n}-\mathbf{e}_i, \mathbf{q}-k\mathbf{e}_i)}{k! (\sum \mathbf{q}-k)!} y_1^{n_1} \dots y_s^{n_s} z_1^{q_1} \dots z_s^{q_s} \\ &= \sum_{i=1}^s \sum_k \frac{1}{k!} y_i z_i^k \sum_{\substack{\mathbf{n}, \mathbf{q} \\ \sum(\mathbf{n}-\mathbf{q})=k}} \frac{r(\mathbf{n}, \mathbf{q})}{(\sum \mathbf{q})!} y_1^{n_1} \dots y_s^{n_s} z_1^{q_1} \dots z_s^{q_s} \\ &= \sum_{i=1}^s \sum_k \frac{1}{k!} y_i z_i^k w^k. \quad \blacksquare \end{aligned}$$

The special case where $s = 1$ and $z_1 = 1$ in (27) and (28) is especially important in applications, so it has become known as the "tree function"

$$T(y) = \sum_{n \geq 1} \frac{n^{n-1}}{n!} y^n = y e^{T(y)}. \quad (30)$$

See Corless, Gonnet, Hare, Jeffrey, and Knuth, *Advances in Computational Math.* 5 (1996), 329–359, for a discussion of some of this function's remarkable properties.

A survey of enumeration formulas for trees, based on skillful manipulations of generating functions, has been given by I. J. Good [*Proc. Cambridge Philos. Soc.* 61 (1965), 499–517; 64 (1968), 489]. More recently, a mathematical *theory of species* developed by André Joyal [*Advances in Math.* 42 (1981), 1–82] has led to a high-level viewpoint in which algebraic operations on generating functions correspond directly to combinatorial properties of structures. The book *Combinatorial Species and Tree-like Structures* by F. Bergeron, G. Labelle, and P. Leroux (Cambridge Univ. Press, 1997), presents numerous examples of this beautiful and instructive theory, generalizing many of the formulas derived above.

EXERCISES

1. [M20] (G. Pólya.) Show that

$$A(z) = z \cdot \exp(A(z) + \frac{1}{2} A(z^2) + \frac{1}{3} A(z^3) + \dots).$$

[Hint: Take logarithms of (3).]

2. [HM24] (R. Otter.) Show that the numbers a_n satisfy the following condition:

$$na_{n+1} = a_1 s_{n1} + 2a_2 s_{n2} + \dots + na_n s_{nn},$$

where

$$s_{nk} = \sum_{1 \leq j \leq n/k} a_{n+1-jk}.$$

(These formulas are useful for the calculation of the a_n , since $s_{nk} = s_{(n-k)k} + a_{n+1-k}$.)

Several of the 40-digit values in Table 1 were computed on a desk calculator by John W. Wrench, Jr., for the first edition of this book. When computer software for such calculations became available during the 1970s, all of his contributions proved to be correct. See the answer to exercise 1.3.3-23 for the 0-digit value of another fundamental constant.

Table 3
VALUES OF HARMONIC NUMBERS, BERNOULLI NUMBERS,
AND FIBONACCI NUMBERS, FOR SMALL VALUES OF n

n	H_n	B_n	F_n	n
0	0	1	0	0
1	1	-1/2	1	1
2	3/2	1/6	1	2
3	11/6	0	2	3
4	25/12	-1/30	3	4
5	137/60	0	5	5
6	49/20	1/42	8	6
7	363/140	0	13	7
8	761/280	-1/30	21	8
9	7129/2520	0	34	9
10	7381/2520	5/66	55	10
11	83711/27720	0	89	11
12	86021/27720	-691/2730	144	12
13	1145993/360360	0	233	13
14	1171733/360360	7/6	377	14
15	1195757/360360	0	610	15
16	2436559/720720	-3617/510	987	16
17	42142223/12252240	0	1597	17
18	14274301/4084080	43867/798	2584	18
19	275295799/77597520	0	4181	19
20	55835135/15519504	-174611/330	6765	20
21	18858053/5173168	0	10946	21
22	19093197/5173168	854513/138	17711	22
23	444316699/118982864	0	28657	23
24	1347822955/356948592	-236364091/2730	46368	24
25	34052522467/8923714800	0	75025	25
26	34395742267/8923714800	8553103/6	121393	26
27	312536252003/80313433200	0	196418	27
28	315404588903/80313433200	-23749461029/870	317811	28
29	9227046511387/2329089562800	0	514229	29
30	9304682830147/2329089562800	8615841276005/14322	832040	30

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

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

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

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

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

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

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

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

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

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

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

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

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

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

APPENDIX B

INDEX TO NOTATIONS

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

j, k	integer-valued arithmetic expression
m, n	nonnegative integer-valued arithmetic expression
x, y	real-valued arithmetic expression
f	real-valued or complex-valued function
P	pointer-valued expression (either Λ or a computer address)
S, T	set or multiset
α	string of symbols

Formal symbolism	Meaning	Where defined
$V \leftarrow E$	give variable V the value of expression E	1.1
$U \leftrightarrow V$	interchange the values of variables U and V	1.1
A_n or $A[n]$	the n th element of linear array A	1.1
A_{mn} or $A[m, n]$	the element in row m and column n of rectangular array A	1.1
$\text{NODE}(P)$	the node (group of variables that are individually distinguished by their field names) whose address is P , assuming that $P \neq \Lambda$	2.1
$F(P)$	the variable in $\text{NODE}(P)$ whose field name is F	2.1
$\text{CONTENTS}(P)$	contents of computer word whose address is P	2.1
$\text{LOC}(V)$	address of variable V within a computer	2.1
$P \Leftarrow \text{AVAIL}$	set the value of pointer variable P to the address of a new node	2.2.3
$\text{AVAIL} \Leftarrow P$	return $\text{NODE}(P)$ to free storage; all its fields lose their identity	2.2.3
$\text{top}(S)$	node at the top of a nonempty stack S	2.2.1
$X \Leftarrow S$	pop up S to X : set $X \leftarrow \text{top}(S)$; then delete $\text{top}(S)$ from nonempty stack S	2.2.1
$S \Leftarrow X$	push down X onto S : insert the value X as a new entry on top of stack S	2.2.1

Formal symbolism	Meaning	Where defined
$\Pr(S(X))$	probability that statement $S(X)$ is true, for random values of X	1.2.10
$E X$	expected value of X : $\sum_x x \Pr(X = x)$	1.2.10
$\text{mean}(g)$	mean value of the probability distribution represented by generating function g : $g'(1)$	1.2.10
$\text{var}(g)$	variance of the probability distribution represented by generating function g : $g''(1) + g'(1) - g'(1)^2$	1.2.10
$(\min x_1, \text{ave } x_2,$ $\max x_3, \text{dev } x_4)$	a random variable having minimum value x_1 , average (expected) value x_2 , maximum value x_3 , standard deviation x_4	1.2.10
P^*	address of preorder successor of $\text{NODE}(P)$ in a binary tree or tree	2.3.1, 2.3.2
$P\$$	address of inorder successor of $\text{NODE}(P)$ in a binary tree, postorder successor in a tree	2.3.1, 2.3.2
$P\#$	address of postorder successor of $\text{NODE}(P)$ in a binary tree	2.3.1
$*P$	address of preorder predecessor of $\text{NODE}(P)$ in a binary tree or tree	2.3.1, 2.3.2
sP	address of inorder predecessor of $\text{NODE}(P)$ in a binary tree, postorder predecessor in a tree	2.3.1, 2.3.2
tP	address of postorder predecessor of $\text{NODE}(P)$ in a binary tree	2.3.1
I	end of algorithm, program, or proof	1.1
 	one blank space	1.3.1
rA	register A (accumulator) of MIX	1.3.1
rX	register X (extension) of MIX	1.3.1
rI1, ..., rI6	(index) registers I1, ..., I6 of MIX	1.3.1
rJ	(jump) register J of MIX	1.3.1
(L:R)	partial field of MIX word, $0 \leq L \leq R \leq 5$	1.3.1
OP ADDRESS, I(F)	notation for MIX instruction	1.3.1, 1.3.2
u	unit of time in MIX	1.3.1
*	"self" in MIXAL	1.3.2
OF, 1F, 2F, ..., 9F	"forward" local symbol in MIXAL	1.3.2
OB, 1B, 2B, ..., 9B	"backward" local symbol in MIXAL	1.3.2
OH, 1H, 2H, ..., 9H	"here" local symbol in MIXAL	1.3.2

- Nicolau, Alexandru, 614.
- Nicomachus of Gerasa (Νικόμαχος ὁ εκ Γεράση), 19.
- Nielsen, Norman Russell, 450.
- Nil link, *see* Null link.
- Niven, Ivan Morton, 87.
- Noah ben Lamech (נוֹחַ בֶּן לָמֶךְ), 310.
- Node: Basic component of data structures, 233, 462–464.
address of, 233.
diagram of, 234.
link to, 233.
notations for fields, 235–237, 458.
size of, 257, 299, 435, 453.
- NODE, 236.
- Node variable, 236.
- Noncrossing partitions of a polygon, 408.
- Nonnegative: Zero or positive.
- NOP (no operation), 136.
- Normal distribution, 104, 122.
approximately, 105–106.
- Notations, index to, 623–627.
- Null link (Λ), 234–235.
in binary trees, 322, 331.
in diagrams, 234.
in trees, 318.
- NUM (convert to numeric), 138.
- Number definitions, 21.
- Number system: A language for representing numbers.
binary, 24–26.
combinatorial, 73, 560.
decimal, 21, 691.
Fibonacci, 86, 495.
mixed-radix, 300.
octal, 620.
phi, 86.
- Number theory, elementary, 40–45.
- Nygaard, Kirsten, 229, 461.
- O-notation, 107–111, 118.
- O'Beirne, Thomas Hay, 518.
- Octal values of constants, 620.
- Odlyzko, Andrew Michael, 121, 565.
- Oettinger, Anthony Gervin, 460.
- Office of Naval Research, x, 230.
- Okada, Satio (岡田 幸雄, later 岡田 幸千生), 582.
- Oldenburg, Henry, 57.
- Oldham, Jeffrey David, xi.
- Omphaloskepsis, 214.
- One-address computer, 127, 350.
- One-way equalities, 108.
- One-way linkage, *see* Circular linkage,
Straight linkage.
- Onodera, Rikio (小野寺力夫), 582.
- Open subroutine, 229, *see* Macro instruction.
- Operation code field, of MIX instruction, 127.
of MIXAL line, 145, 155.
- Optimal search procedure, 402.
- Order of succession to the throne, 336.
- Ordered trees, 308–309, 374, *see* Trees.
enumeration of, 388–389, 398, 407.
- Ordering: A transitive relation between objects of a set.
lexicographic, 20, 299–300, 306.
linear, 20, 262, 270.
linear, of tree structures, 333, 346.
partial, 261–262, 269–270, 346, 562, 575.
well-, 20, 334.
- Oresme, Nicole, 22.
- Oriented binary trees, 396–397.
- Oriented cycle in a directed graph, 372.
- Oriented forests, 353–355.
- Oriented path in a directed graph, 372.
- Oriented subtrees, enumerated, 378.
- Oriented trees, 308–309, 312, 372–382.
canonically represented, 390–394,
397–398, 590–591.
converted to ordered trees, 347.
defined, 373.
enumerated, 386–387, 389–395, 406.
represented in computer, 347, 353, 377.
with root changed, 377.
- ORIG (origin), 146, 151, 155.
- Orlin, James Berger, 584.
- Orthogonal lists, 298–307.
- Orthogonal vectors of permutations, 184.
- Otoo, Ekow Joseph, 560.
- Otter, Richard Robert, 395, 589.
- OUT (output), 137, 225.
- Out-degree of a vertex, 372.
- Output, 5, 215–228.
buffering, 147, 216–228, 231.
operators of MIX, 136–138.
- Output-restricted deque, 239–243, 269, 274.
- OVERFLOW, 245–251, 256–258, 268–269.
- Overflow toggle of MIX, 126, 134, 142,
208, 214, 228.
- Packed data: Data that has been compressed into a small space, as when two or more elements are placed into the same cell of memory, 128, 158.
- Paging, 452.
- Pallo, Jean Marcel, 577.
- Palm tree, 581.
- Paper tape, 136–137, 231, 229.
- Parallelism, 296.
- Parameters of subroutines, 187, 189, 229.
- Parent, in a tree structure, 311, 317,
334–335.
in a threaded tree, 566.
- Parent links, 347, 353–355, 359–361,
373, 377, 427–433.
- Parentheses, 312–313, 349, 597.
- Parker, Douglass Stott, Jr., 596.
- Parmelee, Richard Paine, 450.

- UNIVAC III computer, 124.
 UNIVAC SS80 computer, 124.
 UNIVAC 1107 computer, 124.
 UNIX operating system, 198.
 Unrooted trees, 363, *see* Free trees.
 Unusual correspondence between permutations, 178–179.
 Updates to memory, synchronous, 298.
 Uspensky, Vladimir Andreevich (Успенский, Владимир Андреевич), 464.
- van Aardenne-Ehrenfest, Tatyana, 375, 379.
 van Ceulen, Ludolph, 596.
 van der Waerden, Bartel Leendert, 385–386, 587.
 van Leeuwen, Jan, 596.
 van Wijngaarden, Adriaan, 461.
 Vandermonde, Alexandre Théophile, 59, 70. matrix, 37–38, 475.
 Vardi, Ilan, 504.
Variable: A quantity that may possess different values as a program is being executed, 3–4, 235.
 link or pointer, 235.
 node, 236.
 Variable-size nodes, 435–456.
 Variance of a probability distribution, 98. deduced from the generating function, 100–103.
 Vauvenargues, Luc de Clapiers, Marquis de, xiv.
 Vectors, *see* Linear lists.
 Velthuis, Frans Jozef, 650.
 Vertex in a graph, 363, 372. isolated, 374.
 Victorius of Aquitania, 518.
 Virtual machine, 201.
 Visiting a node, 320.
 VLSI chips, 563.
 von Ettingshausen, Andreas, 54.
 von Neumann, John (= Margittai Neumann János), 18, 229, 457.
 von Segner, Johann Andreas, 407, 536.
 von Staudt, Karl Georg Christian, 406.
- W-value in MIXAL, 154–155.
 Wadler, Philip Lee, 605.
 Waerden, Bartel Leendert van der, 385–386, 587.
 Wait list, *see* Agenda.
 Waite, William McCastline, 418, 422, 614.
 Wall, Hubert Stanley, 481.
 Wallis, John, 22, 52. product for π , 52, 116.
 Wang, Hao (王浩), 383–384.
 Waring, Edward, 78, 472.
 Warren, Don W., 359.
 Watanabe, Masatoshi (渡邊雅俊), 650.
 Watson, Dan Caldwell, 251.
- Watson, George Neville, 507. lemma, 123.
 Watson, Henry William, 383.
 Weakest precondition, 17.
 Weber, Helmut, 461.
 Webster, Noah, dictionary, 1, 216.
 Wedderburn, Joseph Henry Maclagan, 589.
 Wegbreit, Eliot Ben, 614.
 Wegner, Peter, 306.
 Weierstrass, Karl Theodor Wilhelm, 382.
 Weighted path length, 402–405.
 Weiland, Richard Joel, 608.
 Weizenbaum, Joseph, 414, 459–461.
 Well-ordering, 20, 334.
 Wheeler, David John, 229, 230, 457.
 Whinian, Michael James, 86.
 Whirlwind I computer, 230.
 Whitworth, William Allen, 182.
 Wijngaarden, Adriaan van, 461.
 Wilde, Oscar Fingal O'Flahertie Wills, 422.
 Wiles, Andrew John, 466.
 Wilf, Herbert Saul, 65, 66, 93, 484.
 Wilkes, Maurice Vincent, 229, 230, 457.
 Wilson, John, theorem, 51.
 Wilson, Paul Robinson, 452.
 Windley, Peter F., 523.
 Windsor, House of, 310.
 Winkler, Phyllis Astrid Benson, xi.
 Wirth, Niklaus Emil, 461.
 Wise, David Stephen, 251, 421, 434, 605.
 Wiseman, Neil Ernest, 421.
 Wolman, Eric, 453.
 Wolontis, Vidar Michael, 230.
 Woods, M. L., 230.
 Woodward, Philip Mayne, 460.
Word: An addressable unit of computer memory, 125–127.
Word size, for MIX: The number of different values that might be stored in five bytes.
 Wordsworth, William, 139.
 Worst-fit method of storage allocation, 453, 608.
 Wrench, John William, Jr., 480, 621.
 Wright, Edward Maitland, 492, 520.
 Wright, Jesse Bowdle, 359.
Writing: Doing output, 215.
 Writing large programs, 191–193.
 Wyman, Max, 66.
 Wythoff (= Wijthoff), Willem Abraham, 80.
- X-1 computer, 231.
 X-register of MIX, 125.
 XDS 920 computer, 124.
 XOR (exclusive or), 455.

Yao, Andrew Chi-Chih (姚期智), 543.
 Yngve, Victor Huse, 461.
 Yo-yo list, 240.
 Yoder, Michael Franz, 479.
 Young, David Monaghan, Jr., 586.
z, for complex numbers, 21, 108. .
 Zabell, Sandy Lew, 491.

Zave, Derek Alan, 91, 614.
 Zeilberger, Doron (זילברגֶר), 65.
 Zemanek (= Zemánek), Heinz, 1.
 Zeta function $\zeta(s, z)$, 44, 76.
 Zhang, Linbo (张林波), 650.
 Zimmerman, Seth, 407.
 Zorn, Max, lemma, 547.
 Zuse, Konrad, 457.

*We must not . . . think that computation,
 that is ratiocination,
 has place only in numbers.*

— THOMAS HOBBES, *Elementary Philosophy* (1656)

THIS BOOK was composed on a Sun SPARCstation with Computer Modern typefaces, using the TeX and METAFONT software as described in the author's books *Computers & Typesetting* (Reading, Mass.: Addison-Wesley, 1986), Volumes A-E. The illustrations were produced with John Hobby's METAPOST system. Some names in the index were typeset with additional fonts developed by Yannis Haralambous (Greek, Hebrew, Arabic), Olga G. Lapko (Cyrillic), Frans J. Velthuis (Devanagari), Masatoshi Watanabe (Japanese), and Linbo Zhang (Chinese).

87	J2P T10	Is P $\neq \Lambda$?
88 T9A	DEC6 1	
89	J6P T9	$n \geq k \geq 1$.
90 T11	INC6 1	
91	LDA X,6(QLINK)	
92	JAZ *-2	Find k with QLINK[k] $\neq 0$.
93 T12	ST6 X,6(TOP)	TOP[k] $\leftarrow k$.
94	LD6 X,6(QLINK)	$k \leftarrow \text{QLINK}[k]$.
95	LD1 X,6(TOP)	
96	J1Z T12	Is TOP[k] = 0?
97 T13	ENTA 0,6	
98	CHAR	Convert k to alpha.
99	JBUS *(PRINTER)	
100	STX VALUE	Print.
101	OUT LINE2(PRINTER)	
102	J1Z DONE	Stop when TOP[k] = 0.
103	STZ X,6(TOP)	TOP[k] $\leftarrow 0$.
104	LD6 X,6(QLINK)	$k \leftarrow \text{QLINK}[k]$.
105	LD1 X,6(TOP)	
106	JMP T13	
107 LINE1	ALF LOOP	Title line
108	ALF DETEC	
109	ALF TED I	
110	ALF N INP	
111	ALF UT:	
112 LINE2	ALF	Succeeding lines
113 VALUE	EQU LINE2+3	
114	ORIG LINE2+24	
115 DONE	HLT	End of computation
116 X	END TOPSORT	■

Note: If the relations $9 \prec 1$ and $6 \prec 9$ are added to the data (18), this program will print "9, 6, 8, 5, 9" as the loop.

26. One solution is to proceed in two phases as follows:

Phase 1. (We use the X table as a (sequential) stack as we mark B = 1 or 2 for each subroutine that needs to be used.)

- A0. For $1 \leq J \leq N$ set $B(X[J]) \leftarrow B(X[J]) + 2$, if $B(X[J]) \leq 0$.
- A1. If $N = 0$, go to phase 2; otherwise set $P \leftarrow X[N]$ and decrease N by 1.
- A2. If $|B(P)| = 1$, go to A1, otherwise set $P \leftarrow P + 1$.
- A3. If $B(\text{SUB1}(P)) \leq 0$, set $N \leftarrow N + 1$, $B(\text{SUB1}(P)) \leftarrow B(\text{SUB1}(P)) + 2$, $X[N] \leftarrow \text{SUB1}(P)$. If $\text{SUB2}(P) \neq 0$ and $B(\text{SUB2}(P)) \leq 0$, do a similar set of actions with $\text{SUB2}(P)$. Go to A2. ■

Phase 2. (We go through the table and allocate memory.)

- B1. Set $P \leftarrow \text{FIRST}$.
- B2. If $P = \Lambda$, set $N \leftarrow N + 1$, $\text{BASE}(\text{LOC}(X[N])) \leftarrow \text{MLOC}$, $\text{SUB}(\text{LOC}(X[N])) \leftarrow 0$, and terminate the algorithm.

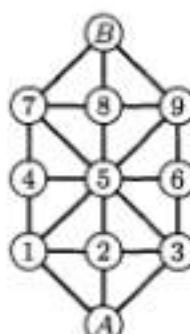
B3. If $B(P) > 0$, set $N \leftarrow N + 1$, $\text{BASE}(\text{LOC}(X[N])) \leftarrow \text{MLOC}$, $\text{SUB}(\text{LOC}(X[N])) \leftarrow P$, $\text{MLOC} \leftarrow \text{MLOC} + \text{SPACE}(P)$.

B4. Set $P \leftarrow \text{LINK}(P)$ and return to B2. ■

27. Comments on the following code are left to the reader.

B EQU 0:1	A1 J1Z B1	INC1 1
SPACE EQU 2:3	LD2 X,1	INCA 2
LINK EQU 4:5	DEC1 1	STA 0,3(B)
SUB1 EQU 2:3	A2 LDA 0,2(1:1)	ST3 X,1
SUB2 EQU 4:5	DECA 1	JMP A2
BASE EQU 0:3	JAZ A1	B1 ENT2 FIRST
SUB EQU 4:5	INC2 1	LDA MLOC
AO LD2 N	A3 LD3 0,2(SUB1)	JMP 1F
	LDA 0,3(B)	B3 LDX 0,2(B)
1H LD3 X,2	JAP 9F	JXNP B4
	INC1 1	INC1 1
LDA 0,3(B)	INCA 2	ST2 X,1(SUB)
JAP **+3	STA 0,3(B)	ADD 0,2(SPACE)
INCA 2	ST3 X,1	1H STA X+1,1(BASE)
STA 0,3(B)	9H LD3 0,2(SUB2)	B4 LD2 0,2(LINK)
DEC2 1	J3Z A2	B2 J2NZ B3
J2P 1B	LDA 0,3(B)	STZ X+1,1(SUB)
LD1 N	JAP A2	■

28. We give here only a few comments related to the military game. Let A be the player with three men whose pieces start on nodes A13; let B be the other player. In this game, A must "trap" B , and if B can cause a position to be repeated for a second time we can consider B the winner. To avoid keeping the entire past history of the game as an integral part of the positions, however, we should modify the algorithm in the following way: Start by marking the positions 157-4, 789-B, 359-6 with B to move as "lost" and apply the suggested algorithm. Now the idea is for player A to move only to B 's lost positions. But A must also take precautions against repeating prior moves. A good computer game-playing program will use a random number generator to select between several winning moves when more than one is present, so an obvious technique would be to make the computer, playing A , just choose randomly among those moves that lead to a lost position for B . But there are interesting situations that make this plausible procedure fail! For example, consider position 258-7 with A to move; this is a won position. From position 258-7, player A might try moving to 158-7 (which is a lost position for B , according to the algorithm). But then B plays to 158-B, and this forces A to play to 258-B, after which B plays back to 258-7; B has won, since the former position has been repeated! This example shows that the algorithm must be re-invoked after every move has been made, starting with each position that has previously occurred marked "lost" (if A is to move) or "won" (if B is to move). The military game makes a very satisfactory computer demonstration program.



Board for the "military game."

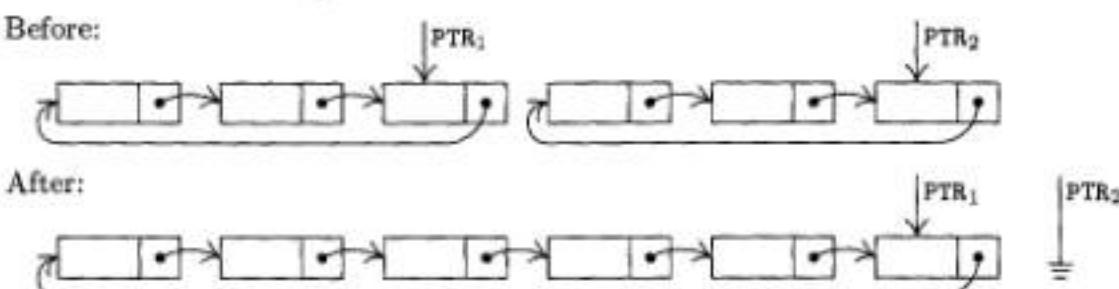
29. (a) If $\text{FIRST} = \Lambda$, do nothing; otherwise set $P \leftarrow \text{FIRST}$, and then repeatedly set $P \leftarrow \text{LINK}(P)$ zero or more times until $\text{LINK}(P) = \Lambda$. Finally set $\text{LINK}(P) \leftarrow \text{AVAIL}$ and $\text{AVAIL} \leftarrow \text{FIRST}$ (and probably also $\text{FIRST} \leftarrow P$). (b) If $F = \Lambda$, do nothing; otherwise set $\text{LINK}(R) \leftarrow \text{AVAIL}$ and $\text{AVAIL} \leftarrow F$ (and probably also $F \leftarrow \Lambda$, $R \leftarrow \text{LOC}(F)$).

30. To insert, set $P \leftarrow \text{AVAIL}$, $\text{INFO}(P) \leftarrow Y$, $\text{LINK}(P) \leftarrow \Lambda$, if $F = \Lambda$ then $F \leftarrow P$ else $\text{LINK}(R) \leftarrow P$, and $R \leftarrow P$. To delete, do (g) with F replacing T . (Although it is convenient to let R be undefined for an empty queue, this lack of discipline might confuse a garbage collection algorithm, as in exercise 6.)

SECTION 2.2.4

1. No, it does not help; it seems to hinder, if anything. (The stated convention is *not* especially consistent with the circular list philosophy, unless we put `NODE(LOC(PTR))` into the list as its list head.)

2. Before:

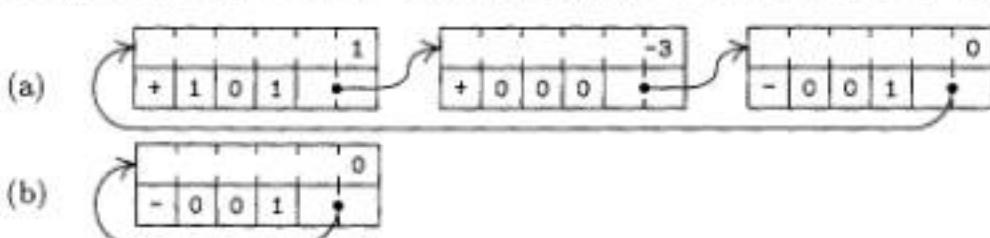


3. If $\text{PTR}_1 = \text{PTR}_2$, the only effect is $\text{PTR}_2 \leftarrow \Lambda$. If $\text{PTR}_1 \neq \text{PTR}_2$, the exchange of links breaks the list into two parts, as if a circle had been broken in two by cutting at two points; the second part of the operation then makes PTR_1 point to a circular list that consists of the nodes that would have been traversed if, in the original list, we followed the links from PTR_1 to PTR_2 .

4. Let HEAD be the address of the list head. To push down Y onto the stack: Set P \leftarrow AVAIL, INFO(P) \leftarrow Y, LINK(P) \leftarrow LINK(HEAD), LINK(HEAD) \leftarrow P. To pop up the stack onto Y: If LINK(HEAD) = HEAD then UNDERFLOW; otherwise set P \leftarrow LINK(HEAD), LINK(HEAD) \leftarrow LINK(P), Y \leftarrow INFO(P), AVAIL \leftarrow P.

5. (Compare with exercise 2.2.3-7.) Set $Q \leftarrow A$, $P \leftarrow \text{PTR}$, and then while $P \neq A$ repeatedly set $R \leftarrow Q$, $Q \leftarrow P$, $P \leftarrow \text{LINK}(Q)$, $\text{LINK}(Q) \leftarrow R$. (Afterwards $Q = \text{PTR}$.)

8.



7. Matching terms in the polynomial are located in one pass over the list, so repeated random searches are avoided. Also, increasing order would be incompatible with the “-1” sentinel.

8. We must know what node points to the current node of interest, if we are going to delete that node or to insert another one ahead of it. There are alternatives, however: We could delete $\text{NODE}(Q)$ by setting $Q_2 \leftarrow \text{LINK}(Q)$ and then setting $\text{NODE}(Q) \leftarrow \text{NODE}(Q_2)$, $\text{AVAIL} \leftarrow Q_2$; we could insert a $\text{NODE}(Q_2)$ in front of $\text{NODE}(Q)$ by first interchanging $\text{NODE}(Q_2) \leftrightarrow \text{NODE}(Q)$, then setting $\text{LINK}(Q) \leftarrow Q_2$, $Q \leftarrow Q_2$. These

clever tricks allow the deletion and insertion *without* knowing which node links to NODE(Q); they were used in early versions of IPL. But they have the disadvantage that the sentinel node at the end of a polynomial will occasionally move, and other link variables may be pointing to this node.

9. Algorithm A with $P = Q$ simply doubles polynomial(Q), as it should—except in the anomalous case that COEF = 0 for some term with ABC ≥ 0 , when it fails badly. Algorithm M with $P = M$ also gives the expected result. Algorithm M with $P = Q$ sets polynomial(P) \leftarrow polynomial(P) times $(1 + t_1)(1 + t_2)\dots(1 + t_k)$ if $M = t_1 + t_2 + \dots + t_k$ (although this is not immediately obvious). When $M = Q$, Algorithm M surprisingly gives the expected result, polynomial(Q) \leftarrow polynomial(Q) + polynomial(Q) \times polynomial(P), except that the computation blows up when the constant term of polynomial(P) is -1.

10. None. (The only possible difference would be in step M2, removing error checks that A, B, or C might individually overflow; these error checks were not specified because we assumed that they were not necessary.) In other words, the algorithms in this section may be regarded as operations on the polynomial $f(x^b, x^b, x)$ instead of on $f(x, y, z)$.

11. COPY STJ 9F	(comments	ST6 1,3(LINK)
ENT3 9F	are	ENT3 0,6
LDA 1,1	left	LD1 1,1(LINK)
1H LD6 AVAIL	to	LDA 1,1
J6Z OVERFLOW	the	JANN 1B
LDX 1,6(LINK)	reader)	LD2 8F(LINK)
STX AVAIL		ST2 1,3(LINK)
STA 1,6		9H JMP *
LDA 0,1		8H CON 0 ■
STA 0,6		

12. Let the polynomial copied have p terms. Program A takes $(29p + 13)u$, and to make it a fair comparison we should add the time to create a zero polynomial, say $18u$ with exercise 14. The program of exercise 11 takes $(21p + 31)u$, about $\frac{3}{4}$ as much.

13. ERASE STJ 9F	
LDX AVAIL	
LDA 1,1(LINK)	
STA AVAIL	
STX 1,1(LINK)	
9H JMP * ■	

14. ZERO STJ 9F	MOVE 1F(2)
LD1 AVAIL	ST2 1,2(LINK)
J1Z OVERFLOW	9H JMP *
LDX 1,1(LINK)	1H CON 0
STX AVAIL	CON -1(ABC) ■
ENT2 0,1	

15. MULT STJ 9F	Entrance to subroutine
LDA 5F	Change settings of switches
STA SW1	
LDA 6F	
STA SW2	
STA SW3	

	JMP	$\star+2$	
2H	JMP	ADD	<u>M2. Multiply cycle.</u>
1H	LD4	1,4(LINK)	<u>M1. Next multiplier.</u> $M \leftarrow \text{LINK}(M)$.
	LDA	1,4	
	JANN	2B	To M2 if $\text{ABC}(M) \geq 0$.
8H	LDA	7F	Restore settings of switches.
	STA	SW1	
	LDA	8F	
	STA	SW2	
	STA	SW3	
9H	JMP	*	Return.
5H	JMP	$\star+1$	New setting of SW1
	LDA	0,1	
	MUL	0,4	$rX \leftarrow \text{COEF}(P) \times \text{COEF}(M)$.
	LDA	1,1(ABC)	$\text{ABC}(P)$
	JAN	$\star+2$	
	ADD	1,4(ABC)	+ $\text{ABC}(M)$, if $\text{ABC}(P) \geq 0$.
	SLA	2	Move into 0:3 field of rA.
	STX	OF	Save rX for use in SW2 and SW3.
	JMP	SW1+1	
6H	LDA	OF	New setting of SW2 and SW3
7H	LDA	1,1	Usual setting of SW1
8H	LDA	0,1	Usual setting of SW2 and SW3
0H	CON	0	Temp storage ■

16. Let r be the number of terms in $\text{polynomial}(M)$. The subroutine requires $21pr + 38r + 29 + 27\sum m' + 18\sum m'' + 27\sum p' + 8\sum q'$ units of time, where the summations refer to the corresponding quantities during the r activations of Program A. The number of terms in $\text{polynomial}(Q)$ goes up by $p' - m'$ each activation of Program A. If we make the not unreasonable assumption that $m' = 0$ and $p' = \alpha p$ where $0 < \alpha < 1$, we get the respective sums equal to 0 , $(1 - \alpha)pr$, αpr , and $rq'_0 + \alpha p(r(r - 1)/2)$, where q'_0 is the value of q' in the first iteration. The grand total is $4\alpha pr^2 + 40pr + 4\alpha pr + 8q'_0r + 38r + 29$. This analysis indicates that the multiplier ought to have fewer terms than the multiplicand, since we have to skip over unmatching terms in $\text{polynomial}(Q)$ more often. (See exercise 5.2.3-29 for a faster algorithm.)

17. There actually is very little advantage; addition and multiplication routines with either type of list would be virtually the same. The efficiency of the ERASE subroutine (see exercise 13) is apparently the only important difference.

18. Let the link field of node x_i contain $\text{LOC}(x_{i+1}) \oplus \text{LOC}(x_{i-1})$, where “ \oplus ” denotes “exclusive or.” Other invertible operations, such as addition or subtraction modulo the pointer field size, could also be used. It is convenient to include two adjacent list heads in the circular list, to help get things started properly. (The origin of this ingenious technique is unknown.)

SECTION 2.2.5

1. Insert Y at the left: $P \leftarrow \text{AVAIL}$; $\text{INFO}(P) \leftarrow Y$; $\text{LLINK}(P) \leftarrow \Lambda$; $\text{RLINK}(P) \leftarrow \text{LEFT}$; if $\text{LEFT} \neq \Lambda$ then $\text{LLINK}(\text{LEFT}) \leftarrow P$ else $\text{RIGHT} \leftarrow P$; $\text{LEFT} \leftarrow P$. Set Y to leftmost and delete: if $\text{LEFT} = \Lambda$ then UNDERFLOW; $P \leftarrow \text{LEFT}$; $\text{LEFT} \leftarrow \text{RLINK}(P)$; if $\text{LEFT} = \Lambda$ then $\text{RIGHT} \leftarrow \Lambda$, else $\text{LLINK}(\text{LEFT}) \leftarrow \Lambda$; $Y \leftarrow \text{INFO}(P)$; $\text{AVAIL} \leftarrow P$.

2. Consider the case of several deletions (at the same end) in succession. After each deletion we must know what to delete next, so the links in the list must point away from that end of the list. Deletion at both ends therefore implies that the links must go both ways. On the other hand, exercise 2.2.4-18 explains how to represent two links in a single link field; in that way general deque operations are possible.

3. To show the independence of CALLUP from CALLDOWN, notice for example that in Table 1 the elevator did not stop at floors 2 or 3 at time 0393-0444 although there were people waiting; these people had pushed CALLDOWN, but if they had pushed CALLUP the elevator would have stopped.

To show the independence of CALLCAR from the others, notice that in Table 1, when the doors start to open at time 1378, the elevator has already decided to be GOINGUP. Its state would have been NEUTRAL at that point if CALLCAR[1] = CALLCAR[2] = CALLCAR[3] = CALLCAR[4] = 0, according to step E2, but in fact CALLCAR[2] and CALLCAR[3] have been set to 1 by users 7 and 9 in the elevator. (If we envision the same situation with all floor numbers increased by 1, the fact that STATE = NEUTRAL or STATE = GOINGUP when the doors open would affect whether the elevator would perhaps continue to go downward or would unconditionally go upward.)

4. If a dozen or more people were getting out at the same floor, STATE might be NEUTRAL all during this time, and when E9 calls the DECISION subroutine this may set a new state before anyone has gotten in on the current floor. It happens very rarely indeed (and it certainly was the most puzzling phenomenon observed by the author during his elevator experiments).

5. The state from the time the doors start to open at time 1063 until user 7 gets in at time 1183 would have been NEUTRAL, since there would have been no calls to floor 0 and nobody on board the elevator. Then user 7 would set CALLCAR[2] \leftarrow 1 and the state would correspondingly change to GOINGUP.

6. Add the condition "if OUT < IN then STATE \neq GOINGUP; if OUT > IN then STATE \neq GOINGDOWN" to the condition "FLOOR = IN" in steps U2 and U4. In step E4, accept users from QUEUE[FLOOR] only if they are headed in the elevator's direction, unless STATE = NEUTRAL (when we accept all comers).

[Stanford's math department has just such an elevator, but its users don't actually pay much attention to the indicator lights; people tend to get on as soon as they can, regardless of direction. Why didn't the elevator designers realize this, and design the logic accordingly by clearing both CALLUP and CALLDOWN? The whole process would be faster, since the elevator wouldn't have to stop as often.]

7. In line 227 this user is assumed to be in the WAIT list. Jumping to U4A makes sure that this assumption is valid. It is assumed that GIVEUPTIME is positive, and indeed that it is probably 100 or more.

8. Comments are left to the reader.

```

277 E8 DEC4 1
278     ENTA 61
279     JMP HOLD
280     LDA CALL,4(3:5)
281     JAP 1F
282     ENT1 -2,4
283     J1Z 2F
284     LDA CALL,4(1:1)

```

285	JAZ	E8		
286	2H	LDA	CALL-1,4	
287		ADD	CALL-2,4	
288		ADD	CALL-3,4	
289		ADD	CALL-4,4	
290		JANZ	E8	
291	1H	ENTA	23	
292		JMP	E2A	
9.	01	DECISION	STJ 9F	Store exit location.
	02		J5NZ 9F	<u>D1. Decision necessary?</u>
	03		LDX ELEV1+2(NEXTINST)	
	04		DECX E1	<u>D2. Should door open?</u>
	05		JXNZ 1F	Jump if elevator not at E1.
	06		LDA CALL+2	
	07		ENT3 E3	Prepare to schedule E3,
	08		JANZ 8F	if there is a call on floor 2.
	09	1H	ENT1 -4	<u>D3. Any calls?</u>
	10		LDA CALL+4,1	Search for a nonzero call variable.
	11		JANZ 2F	
	12	1H	INC1 1	$rI1 \equiv j - 4$
	13		J1NP *-3	
	14		LDA 9F(0:2)	All $\text{CALL}[j]$, $j \neq \text{FLOOR}$, are zero
	15		DECA E6B	Is exit location = line 250?
	16		JANZ 9F	
	17		ENT1 -2	Set $j \leftarrow 2$.
	18	2H	ENT5 4,1	<u>D4. Set STATE.</u>
	19		DEC5 0,4	$\text{STATE} \leftarrow j - \text{FLOOR}$.
	20		J5NZ *+2	
	21		JANZ 1B	$j = \text{FLOOR}$ not allowed in general.
	22		JXNZ 9F	<u>D5. Elevator dormant?</u>
	23		J5Z 9F	Jump if not at E1 or if $j = 2$.
	24		ENT3 E6	Otherwise schedule E6.
	25	8H	ENTA 20	Wait 20 units of time.
	26		ST6 8F(0:2)	Save rI6.
	27		ENT6 ELEV1	
	28		ST3 2,6(NEXTINST)	Set NEXTINST to E3 or E6.
	29		JMP HOLD	Schedule the activity.
30	8H		ENT6 *	Restore rI6.
31	9H		JMP *	Exit from subroutine.

11. Initially let $\text{LINK}[k] = 0$, $1 \leq k \leq n$, and $\text{HEAD} = -1$. During a simulation step that changes $V[k]$, give an error indication if $\text{LINK}[k] \neq 0$; otherwise set $\text{LINK}[k] \leftarrow \text{HEAD}$, $\text{HEAD} \leftarrow k$ and set $\text{NEWV}[k]$ to the new value of $V[k]$. After each simulation step, set $k \leftarrow \text{HEAD}$, $\text{HEAD} \leftarrow -1$, and do the following operation repeatedly zero or more times until $k < 0$: set $V[k] \leftarrow \text{NEWV}[k]$, $t \leftarrow \text{LINK}[k]$, $\text{LINK}[k] \leftarrow 0$, $k \leftarrow t$.

Clearly this method is readily adapted to the case of scattered variables, if we include a NEWV and LINK field in each node associated with a variable field V .

12. The WAIT list has deletions from the left to the right, but insertions are sorted in from the right to the left (since the search is likely to be shorter from that side).

10 ST3 J0 $J_0 \leftarrow \text{COL(PIVOT)}.$
 11 LDA =1.0= Floating point constant 1
 12 FDIV 2,1
 13 STA ALPHA $\text{ALPHA} \leftarrow 1/\text{VAL(PIVOT)}.$
 14 LDA =1.0=
 15 STA 2,1 $\text{VAL(PIVOT)} \leftarrow 1.$
 16 ENT2 BROW,2 $P_0 \leftarrow \text{LOC(BASEROW}[I_0]\text{)}.$
 17 ENT3 BCOL,3 $Q_0 \leftarrow \text{LOC(BASECOL}[J_0]\text{)}.$
 18 JMP S2
 19 2H ENTA BCOL,1
 20 STA BCOL,1(PTR) $\text{PTR}[J] \leftarrow \text{LOC(BASECOL}[J]\text{)}.$
 21 LDA 2,2
 22 FMUL ALPHA
 23 STA 2,2 $\text{VAL}(P_0) \leftarrow \text{ALPHA} \times \text{VAL}(P_0).$
 24 S2 LD2 1,2(LEFT) S2. Process pivot row. $P_0 \leftarrow \text{LEFT}(P_0).$
 25 LD1 1,2(COL) $J \leftarrow \text{COL}(P_0).$
 26 J1NN 2B If $J \geq 0$, process J.
 27 S3 LD3 0,3(UP) S3. Find new row. $Q_0 \leftarrow \text{UP}(Q_0).$
 28 LD4 0,3(ROW) $rI4 \leftarrow \text{ROW}(Q_0).$
 29 9H J4N * If $rI4 < 0$, exit.
 30 CMP4 I0
 31 JE S3 If $rI4 = I_0$, repeat.
 32 ST4 I(ROW) $I \leftarrow rI4.$
 33 ENT4 BROW,4 $P \leftarrow \text{LOC(BASEROW}[I]\text{)}.$
 34 S4A LD5 1,4(LEFT) $P_1 \leftarrow \text{LEFT}(P).$
 35 S4 LD2 1,2(LEFT) S4. Find new column. $P_0 \leftarrow \text{LEFT}(P_0).$
 36 LD1 1,2(COL) $J \leftarrow \text{COL}(P_0).$
 37 CMP1 J0
 38 JE S4 Repeat if $J = J_0.$
 39 ENTA 0,1
 40 SLA 2 $rA(0:3) \leftarrow J.$
 41 J1NN S5
 42 LDAN 2,3 If $J < 0$,
 43 FMUL ALPHA set $\text{VAL}(Q_0) \leftarrow -\text{ALPHA} \times \text{VAL}(Q_0).$
 44 STA 2,3
 45 JMP S3
 46 1H ENT4 0,5 $P \leftarrow P_1.$
 47 LD5 1,4(LEFT) $P_1 \leftarrow \text{LEFT}(P).$
 48 S5 CMPA 1,5(COL) S5. Find I, J element.
 49 JL 1B Loop until $\text{COL}(P_1) \leq J.$
 50 JE S7 If $=$, go right to S7.
 51 S6 LD5 BCOL,1(PTR) S6. Insert I, J element. $rI5 \leftarrow \text{PTR}[J].$
 52 LDA I $rA(0:3) \leftarrow I.$
 53 2H ENT6 0,5 $rI6 \leftarrow rI5.$
 54 LD5 0,6(UP) $rI5 \leftarrow \text{UP}(rI6).$
 55 CMPA 0,5(ROW)
 56 JL 2B Jump if $\text{ROW}(rI5) > I.$
 57 LD5 AVAIL $X \leftarrow \text{AVAIL}.$
 58 J5Z OVERFLOW

```

59      LDA  0,5(UP)
60      STA  AVAIL
61      LDA  0,6(UP)
62      STA  0,5(UP)    UP(X) ← UP(PTR[J])..
63      LDA  1,4(LEFT)
64      STA  1,5(LEFT)   LEFT(X) ← LEFT(P).
65      ST1  1,5(COL)   COL(X) ← J.
66      LDA  I(ROW)
67      STA  0,5(ROW)   ROW(X) ← I.
68      STZ  2,5        VAL(X) ← 0.
69      ST5  1,4(LEFT)   LEFT(P) ← X.
70      ST5  0,6(UP)    UP(PTR[J]) ← X.
71 S7   LDAN 2,3      S7. Pivot. -VAL(Q0)
72      FMUL 2,2       × VAL(P0)
73      FADD 2,5       + VAL(P1).
74      JAZ  S8         If significance lost, to S8.
75      STA  2,5         Otherwise store in VAL(P1).
76      ST5  BCOL,1(PTR) PTR[J] ← P1.
77      ENT4 0,5        P ← P1.
78      JMP  S4A         P1 ← LEFT(P), to S4.
79 S8   LD6  BCOL,1(PTR) S8. Delete I, J element. rI6 ← PTR[J].
80      JMP  **+2
81      LD6  0,6(UP)    rI6 ← UP(rI6).
82      LDA  0,6(UP)
83      DECA 0,5        Is UP(rI6) = P1?
84      JANZ *-3        Loop until equal.
85      LDA  0,5(UP)
86      STA  0,6(UP)    UP(rI6) ← UP(P1).
87      LDA  1,5(LEFT)
88      STA  1,4(LEFT)   LEFT(P) ← LEFT(P1).
89      LDA  AVAIL       AVAIL ← P1.
90      STA  0,5(UP)
91      ST5  AVAIL
92      JMP  S4A         P1 ← LEFT(P), to S4. ■

```

Note: Using the conventions of Chapter 4, lines 71–74 would actually be coded

LDA 2,3; FMUL 2,2; FCMP 2,5; JE S8; STA TEMP; LDA 2,5; FSUB TEMP;

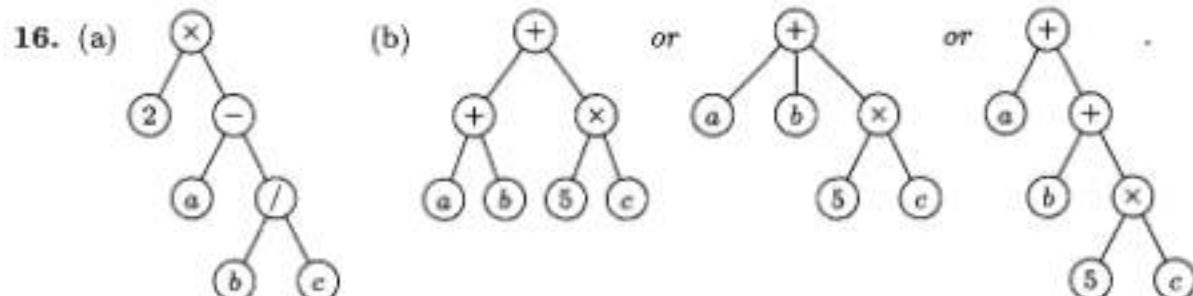
with a suitable parameter EPSILON in location zero.

17. For each row i and each element $A[i,k] \neq 0$, add $A[i,k]$ times row k of B to row i of C . Maintain only the COL links of C while doing this; the ROW links are easily filled in afterwards. [A. Schoor, Inf. Proc. Letters 15 (1982), 87–89.]

18. The three pivot steps, in respective columns 3, 1, 2, yield respectively

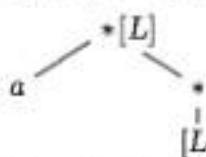
$$\begin{pmatrix} \frac{1}{3} & \frac{2}{3} & \frac{1}{3} \\ -\frac{2}{3} & -\frac{1}{3} & -\frac{2}{3} \\ -\frac{1}{3} & -\frac{2}{3} & -\frac{1}{3} \end{pmatrix}, \quad \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{2} & 1 \\ -\frac{1}{2} & -\frac{1}{2} & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 & 0 \\ -2 & 1 & 1 \\ 1 & -2 & 0 \end{pmatrix};$$

The 0 and 1 convention is interesting for another reason: It gives us an important correspondence between nodes in a binary tree and positive integers expressed in the binary system (namely, memory addresses in a computer).



17. $\text{parent}(F[1]) = A$; $\text{parent}(F[1, 2]) = C$; $\text{parent}(F[1, 2, 2]) = E$.

18. $L[5, 1, 1] = (2)$. $L[3, 1]$ is nonsense, since $L[3]$ is an empty List.

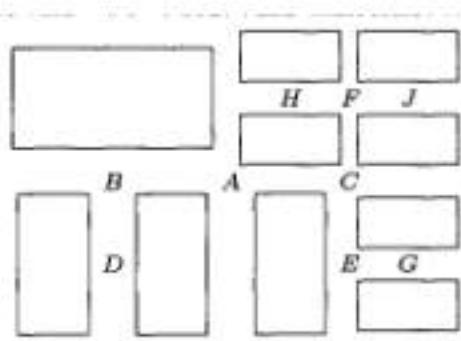
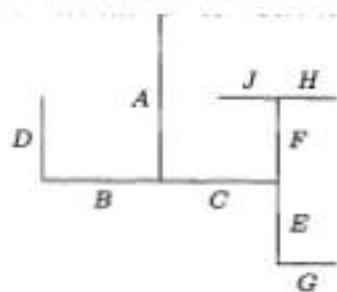
19. 

20. (Intuitively, the correspondence between 0-2-trees and binary trees is obtained by removing all terminal nodes of the 0-2-tree; see the important construction in Section 2.3.4.5.) Let a 0-2-tree with one node correspond to the empty binary tree; and let a 0-2-tree with more than one node, consisting therefore of a root r and 0-2-trees T_1 and T_2 , correspond to the binary tree with root r , left subtree T'_1 , and right subtree T'_2 where T_1 and T_2 correspond respectively to T'_1 and T'_2 .

21. $1 + 0 \cdot n_1 + 1 \cdot n_2 + \dots + (m-1) \cdot n_m$. *Proof:* The number of nodes in the tree is $n_0 + n_1 + n_2 + \dots + n_m$, and this also equals $1 + (\text{number of children in the tree}) = 1 + 0 \cdot n_0 + 1 \cdot n_1 + 2 \cdot n_2 + \dots + m \cdot n_m$.

22. The basic idea is to proceed recursively, with the representation of a nonempty binary tree defined to be the representation of its root plus half-size-and-rotated representations of its left and right subtrees. Thus an arbitrarily large binary tree can be represented on a single sheet of paper, if one has a sufficiently powerful magnifying glass.

Many variations on this theme are possible. For example, one idea is to represent the root by a line from the center of a given landscape-oriented page to the top edge, and to rotate the left-subtree representation by 90° clockwise in the left halfpage, the right-subtree representation by 90° counterclockwise in the right halfpage. Each node is then represented by a line. (When this method is applied to a complete binary tree having $2^k - 1$ nodes on k levels, it yields so-called "H-trees," which are the most efficient layouts of such binary trees on a VLSI chip; see R. P. Brent and H. T. Kung, *Inf. Proc. Letters* 11 (1980), 46–48.)



Another idea is to represent an empty binary tree by some sort of box, and to rotate the subtree representations of nonempty binary trees so that left subsubtrees are alternately to the left of or below the corresponding right subsubtrees, depending on whether the depth of recursion is even or odd. Then the boxes correspond to external nodes in an extended binary tree (see Section 2.3.4.5). This representation, which is strongly related to the 2-D trees and quadtrees discussed in Section 6.5, is especially appropriate when the external nodes carry information but the internal nodes do not.

SECTION 2.3.1

1. $\text{INFO}(T) = A$, $\text{INFO}(\text{RLINK}(T)) = C$, etc.; the answer is H .
2. Preorder: 1245367; symmetric order: 4251637; postorder: 4526731.
3. The statement is true; notice, for example, that nodes 4, 5, 6, 7 always appear in this order in exercise 2. The result is immediately proved by induction on the size of the binary tree.
4. It is the reverse of postorder. (This is easily proved by induction.)
5. In the tree of exercise 2, for example, preorder is 1, 10, 100, 101, 11, 110, 111, using binary notation (which is in this case equivalent to the Dewey system). The strings of digits have been sorted, like words in a dictionary.

In general, the nodes will be listed in preorder if they are sorted lexicographically from left to right, with "blank" $< 0 < 1$. The nodes will be listed in postorder if they are sorted lexicographically with $0 < 1 <$ "blank". For inorder, use $0 <$ "blank" < 1 .

(Moreover, if we imagine the blanks at the left and treat the Dewey labels as ordinary binary numbers, we get *level order*; see 2.3.3-(8).)

6. The fact that $p_1 p_2 \dots p_n$ is obtainable with a stack is readily proved by induction on n , or in fact we may observe that Algorithm T does precisely what is required in its stack actions. (The corresponding sequence of S's and X's, as in exercise 2.2.1-3, is the same as the sequence of 1s and 2s as subscripts in double order; see exercise 18.)

Conversely, if $p_1 p_2 \dots p_n$ is obtainable with a stack and if $p_k = 1$, then $p_1 \dots p_{k-1}$ is a permutation of $\{2, \dots, k\}$ and $p_{k+1} \dots p_n$ is a permutation of $\{k+1, \dots, n\}$; these are the permutations corresponding to the left and right subtrees, and both are obtainable with a stack. The proof now proceeds by induction.

7. From the preorder, the root is known; then from the inorder, we know the left subtree and the right subtree; and in fact we know the preorder and inorder of the nodes in the latter subtrees. Hence the tree is readily constructed (and indeed it is quite amusing to construct a simple algorithm that links the tree together in the normal fashion, starting with the nodes linked together in preorder in LLINK and in inorder in RLINK). Similarly, postorder and inorder together characterize the structure. But preorder and postorder do not; there are two binary trees having AB as preorder and BA as postorder. If all nonterminal nodes of a binary tree have both branches nonempty, its structure is characterized by preorder and postorder.

8. (a) Binary trees with all LLINKs null. (b) Binary trees with zero or one nodes. (c) Binary trees with all RLINKs null.

9. T_1 once, T_2 $2n+1$ times, T_3 n times, T_4 $n+1$ times, T_5 n times. These counts can be derived by induction or by Kirchhoff's law, or by examining Program T.

10. A binary tree with all RLINKs null will cause all n node addresses to be put in the stack before any are removed.

11. Let a_{nk} be the number of binary trees with n nodes for which the stack in Algorithm T never contains more than k items. If $g_k(z) = \sum_n a_{nk} z^n$, we find $g_1(z) = 1/(1-z)$, $g_2(z) = 1/(1-z/(1-z)) = (1-z)/(1-2z)$, ..., $g_k(z) = 1/(1-zg_{k-1}(z)) = q_{k-1}(z)/q_k(z)$ where $q_{-1}(z) = q_0(z) = 1$, $q_{k+1}(z) = q_k(z) - zq_{k-1}(z)$; hence $g_k(z) = (f_1(z)^{k+1} - f_2(z)^{k+1})/(f_1(z)^{k+2} - f_2(z)^{k+2})$ where $f_j(z) = \frac{1}{2}(1 \pm \sqrt{1-4z})$. It can now be shown that $a_{nk} = [u^n](1-u)(1+u)^{2n}(1-u^{k+1})/(1-u^{k+2})$; hence $s_n = \sum_{k \geq 1} k(a_{nk} - a_{n(k-1)})$ is $[u^{n+1}](1-u)^2(1+u)^{2n} \sum_{j \geq 1} u^j/(1-u^j)$, minus a_{nn} . The technique of exercise 5.2.2-52 now yields the asymptotic series

$$s_n/a_{nn} = \sqrt{\pi n} - \frac{3}{2} + \frac{11}{24} \sqrt{\frac{\pi}{n}} + O(n^{-3/2}).$$

[N. G. de Bruijn, D. E. Knuth, and S. O. Rice, in *Graph Theory and Computing*, ed. by R. C. Read (New York: Academic Press, 1972), 15–22.]

When the binary tree represents a forest as described in Section 2.3.2, the quantity analyzed here is the *height* of that forest (the furthest distance between a node and a root, plus one). Generalizations to many other varieties of trees have been obtained by Flajolet and Odlyzko [*J. Computer and System Sci.* 25 (1982), 171–213]; the asymptotic distribution of heights, both near the mean and far away, was subsequently analyzed by Flajolet, Gao, Odlyzko, and Richmond [*Combinatorics, Probability, and Computing* 2 (1993), 145–156].

12. Visit NODE(P) between step T2 and T3, instead of between step T4 and T2. For the proof, demonstrate the validity of the statement “Starting at step T2 with ... original value $A[1] \dots A[m]$,” essentially as in the text.

13. (Solution by S. Araújo, 1976.) Let steps T1 through T4 be unchanged, except that a new variable Q is initialized to Λ in step T1; Q will point to the last node visited, if any. Step T5 becomes two steps:

T5. [Right branch done?] If $RLINK(P) = \Lambda$ or $RLINK(P) = Q$, go on to T6; otherwise set $A \leftarrow P$, $P \leftarrow RLINK(P)$ and return to T2.

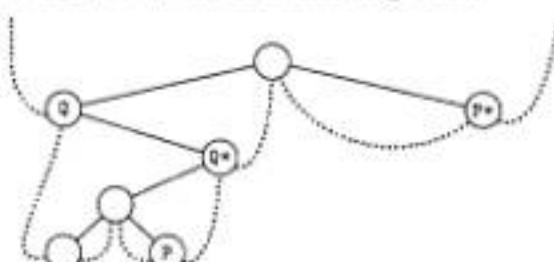
T6. [Visit P.] “Visit” NODE(P), set $Q \leftarrow P$, and return to T4.

A similar proof applies. (Steps T4 and T5 can be streamlined so that nodes are not taken off the stack and immediately reinserted.)

14. By induction, there are always exactly $n+1$ Λ links (counting T when it is null). There are n nonnull links, counting T, so the remark in the text about the majority of null links is justified.

15. There is a thread LLINK or RLINK pointing to a node if and only if it has a nonempty right or left subtree, respectively. (See Fig. 24.)

16. If $LTAG(Q) = 0$, Q^* is $LLINK(Q)$; thus Q^* is one step down and to the left. Otherwise Q^* is obtained by going upwards in the tree (if necessary) repeatedly until the first time it is possible to go down to the right without retracing steps; typical examples are the trips from P to P^* and from Q to Q^* in the following tree:



17. If $\text{LTAG}(P) = 0$, set $Q \leftarrow \text{LLINK}(P)$ and terminate. Otherwise set $Q \leftarrow P$, then set $Q \leftarrow \text{RLINK}(Q)$ zero or more times until finding $\text{RTAG}(Q) = 0$; finally set $Q \leftarrow \text{RLINK}(Q)$ once more.

18. Modify Algorithm T by inserting a step T2.5, "Visit $\text{NODE}(P)$ the first time"; in step T5, we are visiting $\text{NODE}(P)$ the second time.

Given a threaded tree the traversal is extremely simple:

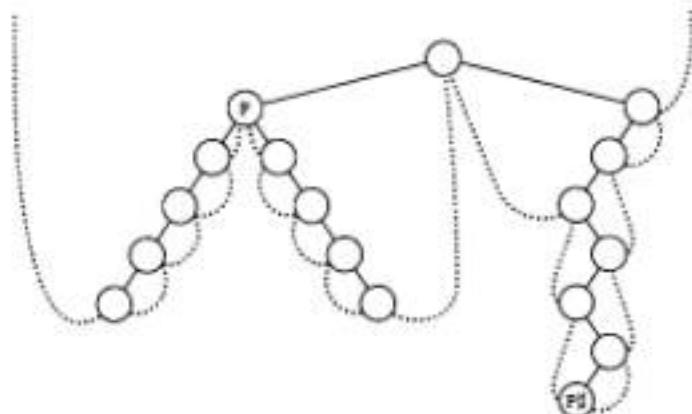
$$(P, 1)^{\Delta} = (\text{LLINK}(P), 1) \text{ if } \text{LTAG}(P) = 0, \text{ otherwise } (P, 2);$$

$$(P, 2)^{\Delta} = (\text{RLINK}(P), 1) \text{ if } \text{RTAG}(P) = 0, \text{ otherwise } (\text{RLINK}(P), 2).$$

In each case, we move at most one step in the tree; in practice, therefore, double order and the values of d and e are embedded in a program and not explicitly mentioned.

Suppressing all the first visits gives us precisely Algorithms T and S; suppressing all the second visits gives us the solutions to exercises 12 and 17.

19. The basic idea is to start by finding the parent Q of P . Then if $P \neq \text{LLINK}(Q)$ we have $P_t = Q$; otherwise we can find P_t by repeatedly setting $Q \leftarrow Q_s$ zero or more times until $\text{RTAG}(Q) = 1$. (See, for example, P and P_t in the tree shown.)



There is no efficient algorithm to find the parent of P in a general right-threaded tree, since a degenerate right-threaded tree in which all left links are null is essentially a circular list in which the links go the wrong way. Therefore we cannot traverse a right-threaded tree in postorder with the same efficiency as the stack method of exercise 13, if we keep no history of how we have reached the current node P .

But if the tree is threaded in both directions, we can find P 's parent efficiently:

F1. Set $Q \leftarrow P$ and $R \leftarrow P$.

F2. If $\text{LTAG}(Q) = \text{RTAG}(R) = 0$, set $Q \leftarrow \text{LLINK}(Q)$ and $R \leftarrow \text{RLINK}(R)$ and repeat this step. Otherwise go to F4 if $\text{RTAG}(R) = 1$.

F3. Set $Q \leftarrow \text{LLINK}(Q)$, and terminate if $P = \text{RLINK}(Q)$. Otherwise set $R \leftarrow \text{RLINK}(R)$ zero or more times until $\text{RTAG}(R) = 1$, then set $Q \leftarrow \text{RLINK}(R)$ and terminate.

F4. Set $R \leftarrow \text{RLINK}(R)$, and terminate with $Q \leftarrow R$ if $P = \text{LLINK}(R)$. Otherwise set $Q \leftarrow \text{LLINK}(Q)$ zero or more times until $\text{LTAG}(Q) = 1$, then set $Q \leftarrow \text{LLINK}(Q)$ and terminate. ■

The average running time of Algorithm F is $O(1)$ when P is a random node of the tree. For if we count only the steps $Q \leftarrow \text{LLINK}(Q)$ when P is a right child, or only the steps $R \leftarrow \text{RLINK}(R)$ when P is a left child, each link is traversed for exactly one node P .

20. Replace lines 06–09 by:

```
T3 ENT4 0,6
LD6 AVAIL
J6Z OVERFLOW
LDX 0,6(LINK)
STX AVAIL
ST5 0,6(INFO)
ST4 0,6(LINK)
```

Replace lines 12–13 by:

```
LD4 0,6(LINK)
LD5 0,6(INFO)
LDX AVAIL
STX 0,6(LINK)
ST6 AVAIL
ENT6 0,4
```

If two more lines of code are added at line 06

```
T3 LD3 0,5(LLINK)
J32 T5
```

To T5 if LLINK(P) = Λ .

with appropriate changes in lines 10 and 11, the running time goes down from $(30n + a + 4)u$ to $(27a + 6n - 22)u$. (This same device would reduce the running time of Program T to $(12a + 6n - 7)u$, which is a slight improvement, if we set $a = (n + 1)/2$.)

21. The following solution by Joseph M. Morris [*Inf. Proc. Letters* 9 (1979), 197–200] traverses also in preorder (see exercise 18).

- U1.** [Initialize.] Set $P \leftarrow T$ and $R \leftarrow \Lambda$.
- U2.** [Done?] If $P = \Lambda$, the algorithm terminates.
- U3.** [Look left.] Set $Q \leftarrow LLINK(P)$. If $Q = \Lambda$, visit $NODE(P)$ in preorder and go to U6.
- U4.** [Search for thread.] Set $Q \leftarrow RLINK(Q)$ zero or more times until either $Q = R$ or $RLINK(Q) = \Lambda$.
- U5.** [Insert or remove thread.] If $Q \neq R$, set $RLINK(Q) \leftarrow P$ and go to U8. Otherwise set $RLINK(Q) \leftarrow \Lambda$ (it had been changed temporarily to P , but we've now traversed P 's left subtree).
- U6.** [Inorder visit.] Visit $NODE(P)$ in inorder.
- U7.** [Go to right or up.] Set $R \leftarrow P$, $P \leftarrow RLINK(P)$, and return to U2.
- U8.** [Preorder visit.] Visit $NODE(P)$ in preorder.
- U9.** [Go to left.] Set $P \leftarrow LLINK(P)$ and return to step U3. ■

Morris also suggested a slightly more complicated way to traverse in postorder.

A completely different solution was found by J. M. Robson [*Inf. Proc. Letters* 2 (1973), 12–14]. Let's say that a node is "full" if its $LLINK$ and $RLINK$ are nonnull, "empty" if its $LLINK$ and $RLINK$ are both empty. Robson found a way to maintain a stack of pointers to the full nodes whose right subtrees are being visited, using the link fields in empty nodes!

Yet another way to avoid an auxiliary stack was discovered independently by G. Lindstrom and B. Dwyer, *Inf. Proc. Letters* 2 (1973), 47–51, 143–145. Their algorithm traverses in *triple order*—it visits every node exactly three times, once in each of preorder, inorder, and postorder—but it does not know which of the three is currently being done.

- W1.** [Initialize.] Set $P \leftarrow T$ and $Q \leftarrow S$, where S is a sentinel value—any number that is known to be different from any link in the tree (e.g., -1).
- W2.** [Bypass null.] If $P = \Lambda$, set $P \leftarrow Q$ and $Q \leftarrow \Lambda$.
- W3.** [Done?] If $P = S$, terminate the algorithm. (We will have $Q = T$ at termination.)

30. Assume that the pointer T in (2) is LLINK(HEAD) in (10).

L1. [Initialize.] Set $Q \leftarrow \text{HEAD}$, $\text{RLINK}(Q) \leftarrow Q$.

L2. [Advance.] Set $P \leftarrow Q$. (See below.)

L3. [Thread.] If $\text{RLINK}(Q) = \Lambda$, set $\text{RLINK}(Q) \leftarrow P$, $\text{RTAG}(Q) \leftarrow 1$; otherwise set $\text{RTAG}(Q) \leftarrow 0$. If $\text{LLINK}(P) = \Lambda$, set $\text{LLINK}(P) \leftarrow Q$, $\text{LTAG}(P) \leftarrow 1$; otherwise set $\text{LTAG}(P) \leftarrow 0$.

L4. [Done?] If $P \neq \text{HEAD}$, set $Q \leftarrow P$ and return to L2. ■

Step L2 of this algorithm implies the activation of an inorder traversal coroutine like Algorithm T, with the additional proviso that Algorithm T visits HEAD after it has fully traversed the tree. This notation is a convenient simplification in the description of tree algorithms, since we need not repeat the stack mechanisms of Algorithm T over and over again. Of course Algorithm S cannot be used during step L2, since the tree hasn't been threaded yet. But the algorithm of exercise 21 can be used in step L2, and this provides us with a very pretty method that threads a tree without using any auxiliary stack.

31. X1. Set $P \leftarrow \text{HEAD}$.

X2. Set $Q \leftarrow P$ (using, say, Algorithm S, modified for a right-threaded tree).

X3. If $P \neq \text{HEAD}$, set $\text{AVAIL} \leftarrow P$.

X4. If $Q \neq \text{HEAD}$, set $P \leftarrow Q$ and go back to X2.

X5. Set $\text{LLINK}(\text{HEAD}) \leftarrow \Lambda$.

Other solutions that decrease the length of the inner loop are clearly possible, although the order of the basic steps is somewhat critical. The stated procedure works because we never return a node to available storage until after Algorithm S has looked at both its LLINK and its RLINK; as observed in the text, each of these links is used precisely once during a complete tree traversal.

32. $\text{RLINK}(Q) \leftarrow \text{RLINK}(P)$, $\text{SUC}(Q) \leftarrow \text{SUC}(P)$, $\text{SUC}(P) \leftarrow \text{RLINK}(P) \leftarrow Q$, $\text{PRED}(Q) \leftarrow P$, $\text{PRED}(\text{SUC}(Q)) \leftarrow Q$.

33. Inserting $\text{NODE}(Q)$ just to the left and below $\text{NODE}(P)$ is quite simple: Set $\text{LLINKT}(Q) \leftarrow \text{LLINKT}(P)$, $\text{LLINK}(P) \leftarrow Q$, $\text{LTAG}(P) \leftarrow 0$, $\text{RLINK}(Q) \leftarrow \Lambda$. Insertion to the right is considerably harder, since it essentially requires finding $*Q$, which is of comparable difficulty to finding Q (see exercise 19); the node-moving technique discussed in exercise 23 could perhaps be used. So general insertions are more difficult with this type of threading. But the insertions required by Algorithm C are not as difficult as insertions are in general, and in fact the copying process is slightly faster for this kind of threading:

C1. Set $P \leftarrow \text{HEAD}$, $Q \leftarrow U$, go to C4. (The assumptions and philosophy of Algorithm C in the text are being used throughout.)

C2. If $\text{RLINK}(P) \neq \Lambda$, set $R \leftarrow \text{AVAIL}$, $\text{LLINK}(R) \leftarrow \text{LLINK}(Q)$, $\text{LTAG}(R) \leftarrow 1$, $\text{RLINK}(R) \leftarrow \Lambda$, $\text{RLINK}(Q) \leftarrow \text{LLINK}(Q) \leftarrow R$.

C3. Set $\text{INFO}(Q) \leftarrow \text{INFO}(P)$.

C4. If $\text{LTAG}(P) = 0$, set $R \leftarrow \text{AVAIL}$, $\text{LLINK}(R) \leftarrow \text{LLINK}(Q)$, $\text{LTAG}(R) \leftarrow 1$, $\text{RLINK}(R) \leftarrow \Lambda$, $\text{LLINK}(Q) \leftarrow R$, $\text{LTAG}(Q) \leftarrow 0$.

C5. Set $P \leftarrow \text{LLINK}(P)$, $Q \leftarrow \text{LLINK}(Q)$.

C6. If $P \neq \text{HEAD}$, go to C2. ■

The algorithm now seems almost too simple to be correct!

Algorithm C for threaded or right-threaded binary trees takes slightly longer due to the extra time to calculate P^* , Q^* in step C5.

It would be possible to thread RLINKs in the usual way or to put $\&P$ in RLINK(P), in conjunction with this copying method, by appropriately setting the values of RLINK(R) and RLINKT(Q) in steps C2 and C4.

34. **A1.** Set $Q \leftarrow P$, and then repeatedly set $Q \leftarrow \text{RLINK}(Q)$ zero or more times until $\text{RTAG}(Q) = 1$.

A2. Set $R \leftarrow \text{RLINK}(Q)$. If LLINK(R) = P, set LLINK(R) $\leftarrow \Lambda$. Otherwise set $R \leftarrow \text{LLINK}(R)$, then repeatedly set $R \leftarrow \text{RLINK}(R)$ zero or more times until $\text{RLINK}(R) = P$; then finally set RLINKT(R) $\leftarrow \text{RLINKT}(Q)$. (This step has removed NODE(P) and its subtrees from the original tree.)

A3. Set RLINK(Q) $\leftarrow \text{HEAD}$, LLINK(HEAD) $\leftarrow P$. ■

(The key to inventing and/or understanding this algorithm is the construction of good "before and after" diagrams.)

36. No; see the answer to exercise 1.2.1-15(e).

37. If LLINK(P) = RLINK(P) = Λ in the representation (2), let LINK(P) = Λ ; otherwise let LINK(P) = Q where NODE(Q) corresponds to NODE(LLINK(P)) and NODE(Q+1) to NODE(RLINK(P)). The condition LLINK(P) or RLINK(P) = Λ is represented by a sentinel in NODE(Q) or NODE(Q+1) respectively. This representation uses between n and $2n - 1$ memory positions; under the stated assumptions, (2) would require 18 words of memory, compared to 11 in the present scheme. Insertion and deletion operations are approximately of equal efficiency in either representation. But this representation is not quite as versatile in combination with other structures.

SECTION 2.3.2

1. If B is empty, $F(B)$ is an empty forest. Otherwise, $F(B)$ consists of a tree T plus the forest $F(\text{right}(B))$, where $\text{root}(T) = \text{root}(B)$ and $\text{subtrees}(T) = F(\text{left}(B))$.

2. The number of zeros in the binary notation is the number of decimal points in the decimal notation; the exact formula for the correspondence is

$$a_1.a_2.\dots.a_k \leftrightarrow 1^{a_1}01^{a_2-1}0\dots01^{a_k-1},$$

where 1^n denotes n ones in a row.

3. Sort the Dewey decimal notations for the nodes lexicographically (from left to right, as in a dictionary), placing a shorter sequence $a_1.\dots.a_k$ in front of its extensions $a_1.\dots.a_k.\dots.a_r$ for preorder, and behind its extensions for postorder. Thus, if we were sorting words instead of sequences of numbers, we would place the words *cat*, *cataract* in the usual dictionary order, to get preorder; we would reverse the order of initial subwords (*cataract*, *cat*), to get postorder. These rules are readily proved by induction on the size of the tree.

4. True, by induction on the number of nodes.

5. (a) Inorder. (b) Postorder. It is interesting to formulate rigorous induction proofs of the equivalence of these traversal algorithms.

6. We have $\text{preorder}(T) = \text{preorder}(T')$, and $\text{postorder}(T) = \text{inorder}(T')$, even if T has nodes with only one child. The remaining two orders are not in any simple relation; for example, the root of T comes at the end in one case and about in the middle in the other.

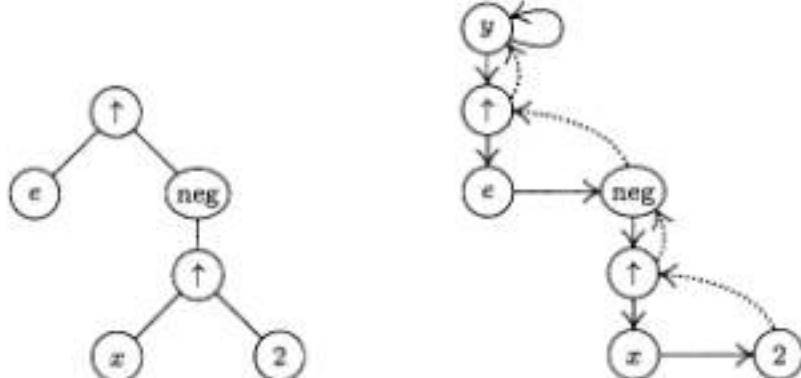
7. (a) Yes; (b) no; (c) no; (d) yes. Note that reverse preorder of a forest equals postorder of the left-right reversed forest (in the sense of mirror reflection).

8. $T \preceq T'$ means that either $\text{info}(\text{root}(T)) \prec \text{info}(\text{root}(T'))$, or these info's are equal and the following condition holds: Suppose the subtrees of $\text{root}(T)$ are T_1, \dots, T_n and the subtrees of $\text{root}(T')$ are $T'_1, \dots, T'_{n'}$, and let $k \geq 0$ be as large as possible such that T_j is equivalent to T'_j for $1 \leq j \leq k$. Then either $k = n$ or $k < n$ and $T_{k+1} \preceq T'_{k+1}$.

9. The number of nonterminal nodes is one less than the number of right links that are Λ , in a nonempty forest, because the null right links correspond to the rightmost child of each nonterminal node, and also to the root of the rightmost tree in the forest. (This fact gives another proof of exercise 2.3.1-14, since the number of null left links is obviously equal to the number of terminal nodes.)

10. The forests are similar if and only if $n = n'$ and $d(u_j) = d(u'_j)$, for $1 \leq j \leq n$; they are equivalent if and only if in addition $\text{info}(u_j) = \text{info}(u'_j)$, $1 \leq j \leq n$. The proof is similar to the previous proof, by generalizing Lemma 2.3.1P; let $f(u) = d(u) - 1$.

11.



12. If $\text{INFO}(Q1) \neq 0$: Set $R \leftarrow \text{COPY}(P1)$; then if $\text{TYPE}(P2) = 0$ and $\text{INFO}(P2) \neq 2$, set $R \leftarrow \text{TREE}("↑", \text{TREE}(\text{INFO}(P2) - 1))$; if $\text{TYPE}(P2) \neq 0$, set $R \leftarrow \text{TREE}("↑", R, \text{TREE}("¬", \text{COPY}(P2), \text{TREE}(1)))$; then set $Q1 \leftarrow \text{MULT}(Q1, \text{MULT}(\text{COPY}(P2), R))$.

If $\text{INFO}(Q) \neq 0$: Set $Q \leftarrow \text{TREE}("×", \text{MULT}(\text{TREE}("in", \text{COPY}(P1)), Q), \text{TREE}("↑", \text{COPY}(P1), \text{COPY}(P2)))$.

Finally go to DIFF[4].

13. The following program implements Algorithm 2.3.1C with $rI1 \equiv P$, $rI2 \equiv Q$, $rI3 \equiv R$, and with appropriate changes to the initialization and termination conditions:

```

064 ST3 6F(0:2)      Save contents of rI3, rI2.
065 ST2 7F(0:2)      C1. Initialize.
066 ENT2 8F           Start by creating NODE(U) with
067 JMP 1F             RLINK(U) = Λ.
068 8H CON 0          Zero constant for initialization
069 4H LD1 0,1(LLINK) Set P ← LLINK(P) = P*.
070 1H LD3 AVAIL      R ← AVAIL.
071 J3Z OVERFLOW
072 LDA 0,3(LLINK)
073 STA AVAIL
074 ST3 0,2(LLINK)    LLINK(Q) ← R.
075 ENNA 0,2
076 STA 0,3(RLINKT)  RLINK(R) ← Q, RTAG(R) ← 1.

```

077	INCA 8B	rA \leftarrow LOC(init node) - Q.
078	ENT2 0,3	Set Q \leftarrow R = Q*.
079	JAZ C3	To C3, the first time.
080	C2 LDA 0,1	<u>C2. Anything to right?</u>
081	JAN C3	Jump if RTAG(P) = 1.
082	LD3 AVAIL	R \leftarrow AVAIL.
083	J3Z OVERFLOW	
084	LDA 0,3(LLINK)	
085	STA AVAIL	
086	LDA 0,2(RLINKT)	
087	STA 0,3(RLINKT)	Set RLINKT(R) \leftarrow RLINKT(Q).
088	ST3 0,2(RLINKT)	RLINK(Q) \leftarrow R, RTAG(Q) \leftarrow 0.
089	C3 LDA 1,1	<u>C3. Copy INFO.</u>
090	STA 1,2	INFO field copied.
091	LDA 0,1(TYPE)	
092	STA 0,2(TYPE)	TYPE field copied.
093	C4 LDA 0,1(LLINK)	<u>C4. Anything to left?</u>
094	JANZ 4B	Jump if LLINK(P) \neq A.
095	STZ 0,2(LLINK)	LLINK(Q) \leftarrow A.
096	C5 LD2N 0,2(RLINKT)	<u>C5. Advance.</u> Q \leftarrow -RLINKT(Q).
097	LD1 0,1(RLINK)	P \leftarrow RLINK(P).
098	J2P C5	Jump if RTAG(Q) was 1.
099	ENN2 0,2	Q \leftarrow -Q.
100	C6 J2NZ C2	<u>C6. Test if complete.</u>
101	LD1 8B(LLINK)	rI1 \leftarrow location of first node created.
102	6H ENT3 *	Restore index registers.
103	7H ENT2 *	■

14. Let a be the number of nonterminal (operator) nodes copied. The number of executions of the various lines in the previous program is as follows: 064-067, 1; 069, a ; 070-079, $a + 1$; 080-081, $n - 1$; 082-088, $n - 1 - a$; 089-094, n ; 095, $n - a$; 096-098, $n + 1$; 099-100, $n - a$; 101-103, 1. The total time is $(36n + 22)a$; we use about 20% of the time to get available nodes, 40% to traverse, and 40% to copy the INFO and LINK information.

15. Comments are left to the reader.

218	DIV LDA 1,6	231	ENTA UPARROW
219	JAZ 1F	232	1H ENTX *
220	JMP COPYP2	233	JMP TREE2
221	ENTA SLASH	234	ST1 1F(0:2)
222	ENTX 0,6	235	JMP COPYP1
223	JMP TREE2	236	ENTA 0,1
224	ENT6 0,1	237	ENT1 0,5
225	1H LDA 1,5	238	JMP MULT
226	JAZ SUB	239	ENTX 0,1
227	JMP COPYP2	240	1H ENT1 *
228	ST1 1F(0:2)	241	ENTA SLASH
229	ENTA CON2	242	JMP TREE2
230	JMP TREEO	243	ENTS 0,1
		244	JMP SUB ■

16. Comments are left to the reader.

245	PWR	LDA	1,6	263	JMP	TREEO	281	ENTA LOG
246		JAZ	4F	264	1H	ENTX *	282	JMP TREE1
247		JMP	COPYP1	265	ENTA	MINUS	283	ENTA 0,1
248		ST1	R(0:2)	266	JMP	TREE2	284	ENT1 0,5
249		LDA	0,3(TYPE).	267	5H	LDX	285	JMP MULT
250		JANZ	2F	268	ENTA	UPARROW	286	ST1 1F(0:2)
251		LDA	1,3	269	JMP	TREE2	287	JMP COPYP1
252		DECA	2	270	ST1	R(0:2)	288	ST1 2F(0:2)
253		JAZ	3F	271	3H	JMP	289	JMP COPYP2
254		INCA	1	272	ENTA	0,1	290	2H ENTX *
255		STA	CON0+1	273	R	ENT1 *	291	ENTA UPARROW
256		ENTA	CON0	274	JMP	MULT	292	JMP TREE2
257		JMP	TREEO	275	ENTA	0,6	293	1H ENTX *
258		STZ	CON0+1	276	JMP	MULT	294	ENTA TIMES
259		JMP	5F	277	ENT6	0,1	295	JMP TREE2
260	2H	JMP	COPYP2	278	4H	LDA	296	ENT5 0,1
261		ST1	1F(0:2)	279	JAZ	ADD	297	JMP ADD ■
262		ENTA	CON1	280	JMP	COPYP1		

17. References to early work on such problems can be found in a survey article by J. Sammet, CACM 9 (1966), 555-569.

18. First set $\text{LLINK}[j] \leftarrow \text{RLINK}[j] \leftarrow j$ for all j , so that each node is in a circular list of length 1. Then for $j = n, n-1, \dots, 1$ (in this order), if $\text{PARENT}[j] = 0$ set $r \leftarrow j$, otherwise insert the circular list starting with j into the circular list starting with $\text{PARENT}[j]$ as follows: $k \leftarrow \text{PARENT}[j], l \leftarrow \text{RLINK}[k], i \leftarrow \text{LLINK}[j], \text{LLINK}[j] \leftarrow k, \text{RLINK}[k] \leftarrow j, \text{LLINK}[l] \leftarrow i, \text{RLINK}[i] \leftarrow l$. This works because (a) each nonroot node is always preceded by its parent or by a descendant of its parent; (b) nodes of each family appear in their parent's list, in order of location; (c) preorder is the unique order satisfying (a) and (b).

20. If u is an ancestor of v , it is immediate by induction that u precedes v in preorder and follows v in postorder. Conversely, suppose u precedes v in preorder and follows v in postorder; we must show that u is an ancestor of v . This is clear if u is the root of the first tree. If u is another node of the first tree, v must be also, since u follows v in postorder; so induction applies. Similarly if u is not in the first tree, v must not be either, since u precedes v in preorder. (This exercise also follows easily from the result of exercise 3. It gives us a quick test for ancestorhood, if we know each node's position in preorder and postorder.)

21. If $\text{NODE}(P)$ is a binary operator, pointers to its two operands are $P_1 = \text{LLINK}(P)$ and $P_2 = \text{RLINK}(P_1) = sP$. Algorithm D makes use of the fact that $P_2s = P$, so that $\text{RLINK}(P_1)$ may be changed to Q_1 , a pointer to the derivative of $\text{NODE}(P_1)$; then $\text{RLINK}(P_1)$ is reset later in step D3. For ternary operations, we would have, say, $P_1 = \text{LLINK}(P), P_2 = \text{RLINK}(P_1), P_3 = \text{RLINK}(P_2) = sP$, so it is difficult to generalize the binary trick. After computing the derivative Q_1 , we could set $\text{RLINK}(P_1) \leftarrow Q_1$ temporarily, and then after computing the next derivative Q_2 we could set $\text{RLINK}(Q_2) \leftarrow Q_1$ and $\text{RLINK}(P_2) \leftarrow Q_2$ and reset $\text{RLINK}(P_1) \leftarrow P_2$. But this is certainly inelegant, and it becomes progressively more so as the degree of the operator becomes higher. Therefore the device of temporarily changing $\text{RLINK}(P_1)$ in Algorithm D is definitely a trick,

not a *technique*. A more aesthetic way to control a differentiation process, because it generalizes to operators of higher degree and does not rely on isolated tricks, can be based on Algorithm 2.3.3F; see exercise 2.3.3-3.

22. From the definition it follows immediately that the relation is transitive; that is, if $T \subseteq T'$ and $T' \subseteq T''$ then $T \subseteq T''$. (In fact the relation is easily seen to be a partial ordering.) If we let f be the function taking nodes into themselves, clearly $l(T) \subseteq T$ and $r(T) \subseteq T$. Therefore if $T \subseteq l(T')$ or $T \subseteq r(T')$ we must have $T \subseteq T'$.

Suppose f_l and f_r are functions that respectively show $l(T) \subseteq l(T')$ and $r(T) \subseteq r(T')$. Let $f(u) = f_l(u)$ if u is in $l(T)$, $f(u) = \text{root}(T')$ if u is $\text{root}(T)$, otherwise $f(u) = f_r(u)$. Now it follows easily that f shows $T \subseteq T'$; for example, if we let $r'(T)$ denote $r(T) \setminus \text{root}(T)$ we have $\text{preorder}(T) = \text{root}(T) \text{ preorder}(l(T)) \text{ preorder}(r'(T))$; $\text{preorder}(T') = f(\text{root}(T)) \text{ preorder}(l(T')) \text{ preorder}(r'(T'))$.

The converse does not hold: Consider the subtrees with roots b and b' in Fig. 25.

SECTION 2.3.3

1. Yes, we can reconstruct them just as (3) is deduced from (4), but interchanging LTAG and RTAG, LLINK and RLINK, and using a queue instead of a stack.

2. Make the following changes in Algorithm F: Step F1, change to "last node of the forest in preorder." Step F2, change " $f(x_d), \dots, f(x_1)$ " to " $f(x_1), \dots, f(x_d)$ " in two places. Step F4, "If P is the first node in preorder, terminate the algorithm. (Then the stack contains $f(\text{root}(T_1)), \dots, f(\text{root}(T_m))$, from top to bottom, where T_1, \dots, T_m are the trees of the given forest, from left to right.) Otherwise set P to its predecessor in preorder ($P \leftarrow P - 1$ in the given representation), and return to F2."

3. In step D1, also set $S \leftarrow \Lambda$. (S is a link variable that links to the top of the stack.) Step D2 becomes, for example, "If $\text{NODE}(P)$ denotes a unary operator, set $Q \leftarrow S$, $S \leftarrow \text{RLINK}(Q)$, $P_1 \leftarrow \text{LLINK}(P)$; if it denotes a binary operator, set $Q \leftarrow S$, $Q_1 \leftarrow \text{RLINK}(Q)$, $S \leftarrow \text{RLINK}(Q_1)$, $P_1 \leftarrow \text{LLINK}(P)$, $P_2 \leftarrow \text{RLINK}(P_1)$. Then perform $\text{DIFF}[\text{TYPE}(P)]$." Step D3 becomes "Set $\text{RLINK}(Q) \leftarrow S$, $S \leftarrow Q$." Step D4 becomes "Set $P \leftarrow P_1$." The operation $\text{LLINK}(DY) \leftarrow Q$ may be avoided in step D5 if we assume that $S \equiv \text{LLINK}(DY)$. This technique clearly generalizes to ternary and higher-order operators.

4. A representation like (10) takes $n - m$ LLINKs and $n + (n - m)$ RLINKs. The difference in total number of links is $n - 2m$ between the two forms of representation. Arrangement (10) is superior when the LLINK and INFO fields require about the same amount of space in a node and when m is rather large, namely when the nonterminal nodes have rather large degrees.

5. It would certainly be silly to include threaded RLINKs, since an RLINK thread just points to PARENT anyway. Threaded LLINKs as in 2.3.2-(4) would be useful if it is necessary to move leftward in the tree, for example if we wanted to traverse a tree in reverse postorder, or in family order; but these operations are not significantly harder without threaded LLINKs unless the nodes tend to have very high degrees.

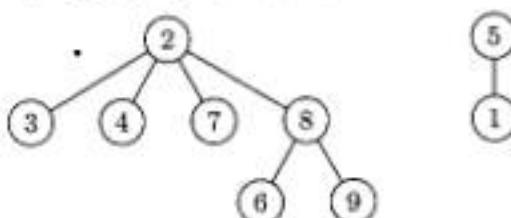
6. L1. Set $P \leftarrow \text{FIRST}$, $\text{FIRST} \leftarrow \Lambda$.

L2. If $P = \Lambda$, terminate. Otherwise set $Q \leftarrow \text{RLINK}(P)$.

L3. If $\text{PARENT}(P) = \Lambda$, set $\text{RLINK}(P) \leftarrow \text{FIRST}$, $\text{FIRST} \leftarrow P$; otherwise set $R \leftarrow \text{PARENT}(P)$, $\text{RLINK}(P) \leftarrow \text{LCHILD}(R)$, $\text{LCHILD}(R) \leftarrow P$.

L4. Set $P \leftarrow Q$ and return to L2. ■

7. $\{1, 5\} \{2, 3, 4, 7\} \{6, 8, 9\}$.
 8. Perform step E3 of Algorithm E, then test if $j = k$.
 9. PARENT[k]: 5 0 2 2 0 8 2 2 8
 k : 1 2 3 4 5 6 7 8 9



10. One idea is to set PARENT of each root node to the negative of the number of nodes in its tree (these values being easily kept up to date); then if $|\text{PARENT}[j]| > |\text{PARENT}[k]|$ in step E4, the roles of j and k are interchanged. This technique (due to M. D. McIlroy) ensures that each operation takes $O(\log n)$ steps.

For still more speed, we can use the following suggestion due to Alan Tritter: In step E4, set $\text{PARENT}[x] \leftarrow k$ for all values $x \neq k$ that were encountered in step E3. This makes an extra pass up the trees, but it collapses them so that future searches are faster. (See Section 7.4.1.)

11. It suffices to define the transformation that is done for each input (P, j, Q, k) :

- T1. If $\text{PARENT}(P) \neq \Lambda$, set $j \leftarrow j + \text{DELTA}(P)$, $P \leftarrow \text{PARENT}(P)$, and repeat this step.
 T2. If $\text{PARENT}(Q) \neq \Lambda$, set $k \leftarrow k + \text{DELTA}(Q)$, $Q \leftarrow \text{PARENT}(Q)$, and repeat this step.
 T3. If $P = Q$, check that $j = k$ (otherwise the input erroneously contains contradictory equivalences). If $P \neq Q$, set $\text{DELTA}(Q) \leftarrow j - k$, $\text{PARENT}(Q) \leftarrow P$, $\text{LBD}(P) \leftarrow \min(\text{LBD}(P), \text{LBD}(Q) + \text{DELTA}(Q))$, and $\text{UBD}(P) \leftarrow \max(\text{UBD}(P), \text{UBD}(Q) + \text{DELTA}(Q))$. ■

Note: It is possible to allow the "ARRAY X[i:u]" declarations to occur intermixed with equivalences, or to allow assignment of certain addresses of variables before others are equivalenced to them, etc., under suitable conditions that are not difficult to understand. For further development of this algorithm, see CACM 7 (1964), 301–303, 506.

12. (a) Yes. (If this condition is not required, it would be possible to avoid the loops on S that appear in steps A2 and A9.) (b) Yes.

13. The crucial fact is that the UP chain leading upward from P always mentions the same variables and the same exponents for these variables as the UP chain leading upward from Q, except that the latter chain may include additional steps for variables with exponent zero. (This condition holds throughout most of the algorithm, except during the execution of steps A9 and A10.) Now we get to step A8 either from A3 or from A10, and in each case it was verified that $\text{EXP}(Q) \neq 0$. Therefore $\text{EXP}(P) \neq 0$, and in particular it follows that $P \neq \Lambda$, $Q \neq \Lambda$, $\text{UP}(P) \neq \Lambda$, $\text{UP}(Q) \neq \Lambda$; the result stated in the exercise now follows. Thus the proof depends on showing that the UP chain condition stated above is preserved by the actions of the algorithm.

16. We prove (by induction on the number of nodes in a single tree T) that if P is a pointer to T, and if the stack is initially empty, steps F2 through F4 will end with the single value $f(\text{root}(T))$ on the stack. This is true for $n = 1$. If $n > 1$, there are

$0 < d = \text{DEGREE}(\text{root}(T))$ subtrees T_1, \dots, T_d ; by induction and the nature of a stack, and since postorder consists of T_1, \dots, T_d followed by $\text{root}(T)$, the algorithm computes $f(T_1), \dots, f(T_d)$, and then $f(\text{root}(T))$, as desired. The validity of Algorithm F for forests follows.

17. G1. Set the stack empty, and let P point to the root of the tree (the last node in postorder). Evaluate $f(\text{NODE}(P))$.
- G2. Push $\text{DEGREE}(P)$ copies of $f(\text{NODE}(P))$ onto the stack.
- G3. If P is the first node in postorder, terminate the algorithm. Otherwise set P to its predecessor in postorder (this would be simply $P \leftarrow P - 1$ in (g)).
- G4. Evaluate $f(\text{NODE}(P))$ using the value at the top of the stack, which is equal to $f(\text{NODE}(\text{PARENT}(P)))$. Pop this value off the stack, and return to G2. ■

Note: An algorithm analogous to this one can be based on preorder instead of postorder as in exercise 2. In fact, family order or level order could be used; in the latter case we would use a queue instead of a stack.

18. The INFO1 and RLINK tables, together with the suggestion for computing LTAG in the text, give us the equivalent of a binary tree represented in the usual manner. The idea is to traverse this tree in postorder, counting degrees as we go:

- P1. Let R, D, and I be stacks that are initially empty; then set $R \leftarrow n + 1$, $D \leftarrow 0$, $j \leftarrow 0$, $k \leftarrow 0$.
- P2. If $\text{top}(R) > j + 1$, go to P5. (If an LTAG field were present, we could have tested $\text{LTAG}[j] = 0$ instead of $\text{top}(R) > j + 1$.)
- P3. If I is empty, terminate the algorithm; otherwise set $i \leftarrow I$, $k \leftarrow k + 1$, $\text{INFO2}[k] \leftarrow \text{INFO}[i]$, $\text{DEGREE}[k] \leftarrow D$.
- P4. If $\text{RLINK}[i] = 0$ go to P3; otherwise delete the top of R (which will equal $\text{RLINK}[i]$).
- P5. Set $\text{top}(D) \leftarrow \text{top}(D) + 1$, $j \leftarrow j + 1$, $I \leftarrow i$, $D \leftarrow 0$, and if $\text{RLINK}[j] \neq 0$ set $R \leftarrow \text{RLINK}[j]$. Go to P2. ■

19. (a) This is equivalent to saying that SCOPE links do not cross each other. (b) The first tree of the forest contains $d_1 + 1$ elements, and we can proceed by induction. (c) The condition of (a) is preserved when we take minima.

Notes: By exercise 2.3.2-20, it follows that $d_1 d_2 \dots d_n$ can also be interpreted in terms of inversions: If the k th node in postorder is the p_k th node in preorder, then d_k is the number of elements $> k$ that appear to the left of k in $p_1 p_2 \dots p_n$.

A similar scheme, in which we list the number of descendants of each node in postorder of the forest, leads to sequences of numbers $c_1 c_2 \dots c_n$ characterized by the properties (i) $0 \leq c_k < k$ and (ii) $k \geq j \geq k - c_k$ implies $j - c_j \geq k - c_k$. Algorithms based on such sequences have been investigated by J. M. Pallo, Comp. J. 29 (1986), 171-175. Notice that c_k is the size of the left subtree of the k th node in symmetric order of the corresponding binary tree. We can also interpret d_k as the size of the right subtree of the k th node in symmetric order of a suitable binary tree, namely the binary tree that corresponds to the given forest by the dual method of exercise 2.3.2-5.

The relation $d_k \leq d'_k$ for $1 \leq k \leq n$ defines an interesting lattice ordering of forests and binary trees, first introduced in another way by D. Tamari [Thèse (Paris: 1951)]; see exercise 6.2.3-32.

SECTION 2.3.4.1

1. $(B, A, C, D, B), (B, A, C, D, E, B), (B, D, C, A, B), (B, D, E, B), (B, E, D, B), (B, E, D, C, A, B)$.

2. Let (V_0, V_1, \dots, V_n) be a path of smallest possible length from V to V' . If now $V_j = V_k$ for some $j < k$, then $(V_0, \dots, V_j, V_{k+1}, \dots, V_n)$ would be a shorter path.

3. (The fundamental path traverses e_3 and e_4 once, but cycle C_2 traverses them -1 times, giving a net total of zero.) Traverse the following edges: $e_1, e_2, e_6, e_7, e_9, e_{10}, e_{11}, e_{12}, e_{14}$.

4. If not, let G'' be the subgraph of G' obtained by deleting each edge e_j for which $E_j = 0$. Then G'' is a finite graph that has no cycles and at least one edge, so by the proof of Theorem A there is at least one vertex, V , that is adjacent to exactly one other vertex, V' . Let e_j be the edge joining V to V' ; then Kirchhoff's equation (1) at vertex V is $E_j = 0$, contradicting the definition of G'' .

5. $A = 1 + E_8, B = 1 + E_8 - E_2, C = 1 + E_8, D = 1 + E_8 - E_5, E = 1 + E_{17} - E_{21}, F = 1 + E_{13}'' + E_{17} - E_{21}, G = 1 + E_{13}'', H = E_{17} - E_{21}, J = E_{17}, K = E_{19}'' + E_{20}, L = E_{17} + E_{19}'' + E_{20} - E_{21}, P = E_{17} + E_{20} - E_{21}, Q = E_{20}, R = E_{17} - E_{21}, S = E_{25}$. Note: In this case it is also possible to solve for E_2, E_5, \dots, E_{25} in terms of A, B, \dots, S ; hence there are nine independent solutions, explaining why we eliminated six variables in Eq. 1.3.3-(8).

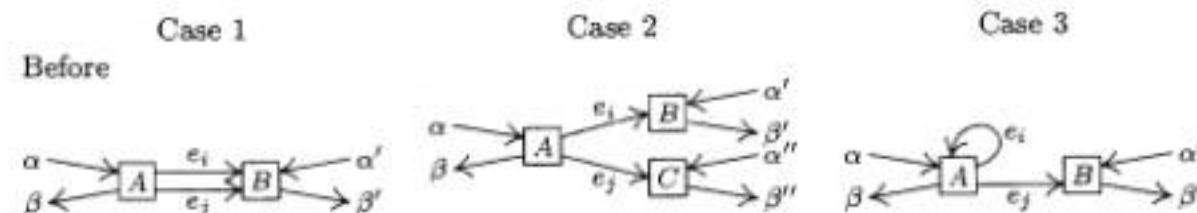
6. (The following solution is based on the idea that we may print out each edge that does not make a cycle with the preceding edges.) Use Algorithm 2.3.3E, with each pair (a_i, b_i) representing $a_i \equiv b_i$ in the notation of that algorithm. The only change is to print (a_i, b_i) if $j \neq k$ in step E4.

To show that this algorithm is valid, we must prove that (a) the algorithm prints out no edges that form a cycle, and (b) if G contains at least one free subtree, the algorithm prints out $n - 1$ edges. Define $j \equiv k$ if there exists a path from V_j to V_k or if $j = k$. This is clearly an equivalence relation, and moreover $j \equiv k$ if and only if this relation can be deduced from the equivalences $a_1 \equiv b_1, \dots, a_m \equiv b_m$. Now (a) holds because the algorithm prints out no edges that form a cycle with previously printed edges; (b) is true because $\text{PARENT}[k] = 0$ for precisely one k if all vertices are equivalent.

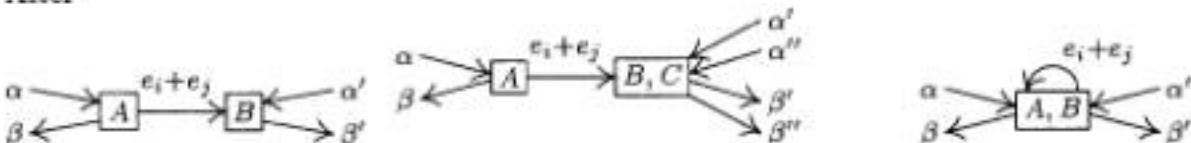
A more efficient algorithm can, however, be based on depth-first search; see Algorithm 2.3.5A and Section 7.4.1.

7. Fundamental cycles: $C_0 = e_0 + e_1 + e_4 + e_9$ (fundamental path is $e_1 + e_4 + e_9$); $C_5 = e_5 + e_3 + e_2$; $C_6 = e_6 - e_2 + e_4$; $C_7 = e_7 - e_4 - e_3$; $C_8 = e_8 - e_9 - e_4 - e_3$. Therefore we find $E_1 = 1, E_2 = E_5 - E_6, E_3 = E_5 - E_7 - E_8, E_4 = 1 + E_6 - E_7 - E_8, E_9 = 1 - E_8$.

8. Each step in the reduction process combines two arrows e_i and e_j that start at the same box, and it suffices to prove that such steps can be reversed. Thus we are given the value of $e_i + e_j$ after combination, and we must assign consistent values to e_i and e_j before the combination. There are three essentially different situations:



After



Here A , B , and C stand for vertices or supervertices, and the α 's and β 's stand for the other given flows besides $e_i + e_j$; these flows may each be distributed among several edges, although only one is shown. In Case 1 (e_i and e_j lead to the same box), we may choose e_i arbitrarily, then $e_j \leftarrow (e_i + e_j) - e_i$. In Case 2 (e_i and e_j lead to different boxes), we must set $e_i \leftarrow \beta' - \alpha'$, $e_j \leftarrow \beta'' - \alpha''$. In Case 3 (e_i is a loop but e_j is not), we must set $e_j \leftarrow \beta' - \alpha'$, $e_i \leftarrow (e_i + e_j) - e_j$. In each case we have reversed the combination step as desired.

The result of this exercise essentially proves that the number of fundamental cycles in the reduced flow chart is the minimum number of vertex flows that must be measured to determine all the others. In the given example, the reduced flow chart reveals that only three vertex flows (e.g., a , c , d) need to be measured, while the original chart of exercise 7 has four independent edge flows. We save one measurement every time Case 1 occurs during the reduction.

A similar reduction procedure could be based on combining the arrows flowing into a given box, instead of those flowing out. It can be shown that this would yield the same reduced flow chart, except that the supervertices would contain different names.

The construction in this exercise is based on ideas due to Armen Nahapetian and F. Stevenson. For further comments, see D. E. Knuth and F. Stevenson, *BIT* 13 (1973), 313–322.

9. Each edge from a vertex to itself becomes a "fundamental cycle" all by itself. If there are $k + 1$ edges $e, e', \dots, e^{(k)}$ between vertices V and V' , make k fundamental cycles $e' \pm e, \dots, e^{(k)} \pm e$ (choosing + or – according as the edges go in the opposite or the same direction), and then proceed as if only edge e were present.

Actually this situation would be much simpler conceptually if we had defined a graph in such a way that multiple edges are allowed between vertices, and edges are allowed from a vertex to itself; paths and cycles would be defined in terms of edges instead of vertices. Such a definition is, in fact, made for directed graphs in Section 2.3.4.2.

10. If the terminals have all been connected together, the corresponding graph must be connected in the technical sense. A minimum number of wires will involve no cycles, so we must have a free tree. By Theorem A, a free tree contains $n - 1$ wires, and a graph with n vertices and $n - 1$ edges is a free tree if and only if it is connected.

11. It is sufficient to prove that when $n > 1$ and $c(n - 1, n)$ is the minimum of the $c(i, n)$, there exists at least one minimum cost tree in which T_{n-1} is wired to T_n . (For, any minimum cost tree with $n > 1$ terminals and with T_{n-1} wired to T_n must also be a minimum cost tree with $n - 1$ terminals if we regard T_{n-1} and T_n as "common", using the convention stated in the algorithm.)

To prove the statement above, suppose we have a minimum cost tree in which T_{n-1} is not wired to T_n . If we add the wire $T_{n-1} — T_n$ we obtain a cycle, and any of the other wires in that cycle may be removed; removing the other wire touching T_n gives us another tree, whose total cost is not greater than the original, and $T_{n-1} — T_n$ appears in that tree.

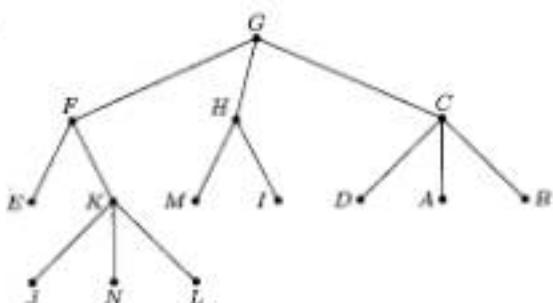
12. Keep two auxiliary tables, $a(i)$ and $b(i)$, for $1 \leq i < n$, representing the fact that the cheapest connection from T_i to a chosen terminal is to $T_{b(i)}$, and its cost is $a(i)$; initially $a(i) = c(i, n)$ and $b(i) = n$. Then do the following operation $n - 1$ times: Find i such that $a(i) = \min_{1 \leq j < n} a(j)$; connect T_i to $T_{b(i)}$; for $1 \leq j < n$ if $c(i, j) < a(j)$ set $a(j) \leftarrow c(i, j)$ and $b(j) \leftarrow i$; and set $a(i) \leftarrow \infty$. Here $c(i, j)$ means $c(j, i)$ when $j < i$.

(It is somewhat more efficient to avoid the use of ∞ , keeping instead a one-way linked list of those j that have not yet been chosen. With or without this straightforward improvement, the algorithm takes $O(n^2)$ operations.) See also E. W. Dijkstra, Proc. Nederl. Akad. Wetensch. A63 (1960), 196–199; D. E. Knuth, The Stanford GraphBase (New York: ACM Press, 1993), 460–497. Significantly better algorithms to find a minimum-cost spanning tree are discussed in Section 7.5.4.

13. If there is no path from V_i to V_j , for some $i \neq j$, then no product of the transpositions will move i to j . So if all permutations are generated, the graph must be connected. Conversely if it is connected, remove edges if necessary until we have a tree. Then renumber the vertices so that V_n is adjacent to only one other vertex, namely V_{n-1} . (See the proof of Theorem A.) Now the transpositions other than $(n-1\ n)$ form a tree with $n - 1$ vertices; so by induction if π is any permutation of $\{1, 2, \dots, n\}$ that leaves n fixed, π can be written as a product of those transpositions. If π moves n to j then $\pi(j\ n-1)(n-1\ n) = \rho$ fixes n ; hence $\pi = \rho(n-1\ n)(j\ n-1)$ can be written as a product of the given transpositions.

SECTION 2.3.4.2

- Let (e_1, \dots, e_n) be an oriented path of smallest possible length from V to V' . If now $\text{init}(e_j) = \text{init}(e_k)$ for $j < k$, $(e_1, \dots, e_{j-1}, e_k, \dots, e_n)$ would be a shorter path; a similar argument applies if $\text{fin}(e_j) = \text{fin}(e_k)$ for $j < k$. Hence (e_1, \dots, e_n) is simple.
- Those cycles in which all signs are the same: $C_0, C_8, C''_{13}, C_{17}, C''_{19}, C_{20}$.
- For example, use three vertices A, B, C , with arcs from A to B and A to C .
- If there are no oriented cycles, Algorithm 2.2.3T topologically sorts G . If there is an oriented cycle, topological sorting is clearly impossible. (Depending on how this exercise is interpreted, oriented cycles of length 1 could be excluded from consideration.)
- Let k be the smallest integer such that $\text{fin}(e_k) = \text{init}(e_j)$ for some $j \leq k$. Then (e_j, \dots, e_k) is an oriented cycle.
- False (on a technicality), just because there may be several different arcs from one vertex to another.
- True for finite directed graphs: If we start at any vertex V and follow the only possible oriented path, we never encounter any vertex twice, so we must eventually reach the vertex R (the only vertex with no successor). For infinite directed graphs the result is obviously false since we might have vertices R, V_1, V_2, V_3, \dots and arcs from V_j to V_{j+1} for $j \geq 1$.
- All arcs point upward.



- 10.** **G1.** Set $k \leftarrow P[j]$, $P[j] \leftarrow 0$.
G2. If $k = 0$, stop; otherwise set $m \leftarrow P[k]$, $P[k] \leftarrow j$, $j \leftarrow k$, $k \leftarrow m$, and repeat step G2. ■
- 11.** This algorithm combines Algorithm 2.3.3E with the method of the preceding exercise, so that all oriented trees have arcs that correspond to actual arcs in the directed graph; $S[j]$ is an auxiliary table that tells whether an arc goes from j to $P[j]$ ($S[j] = +1$) or from $P[j]$ to j ($S[j] = -1$). Initially $P[1] = \dots = P[n] = 0$. The following steps may be used to process each arc (a, b) :
- C1.** Set $j \leftarrow a$, $k \leftarrow P[j]$, $P[j] \leftarrow 0$, $s \leftarrow S[j]$.
 - C2.** If $k = 0$, go to C3; otherwise set $m \leftarrow P[k]$, $t \leftarrow S[k]$, $P[k] \leftarrow j$, $S[k] \leftarrow -s$, $s \leftarrow t$, $j \leftarrow k$, $k \leftarrow m$, and repeat step C2.
 - C3.** (Now a appears as the root of its tree.) Set $j \leftarrow b$, and then if $P[j] \neq 0$ repeatedly set $j \leftarrow P[j]$ until $P[j] = 0$.
 - C4.** If $j = a$, go to C5; otherwise set $P[a] \leftarrow b$, $S[a] \leftarrow +1$, print (a, b) as an arc belonging to the free subtree, and terminate.
 - C5.** Print "CYCLE" followed by " (a, b) ".
 - C6.** If $P[b] = 0$ terminate. Otherwise if $S[b] = +1$, print " $+(b, P[b])$ ", else print " $-(P[b], b)$ "; set $b \leftarrow P[b]$ and repeat step C6. ■
- Note:* This algorithm will take at most $O(m \log n)$ steps if we incorporate the suggestion of McIlroy in answer 2.3.3-10. But there is a much better solution that needs only $O(m)$ steps: Use depth-first search to construct a "palm tree," with one fundamental cycle for each "frond" [R. E. Tarjan, *SICOMP* 1 (1972), 146–150].
- 12.** It equals the in-degree; the out-degree of each vertex can be only 0 or 1.
- 13.** Define a sequence of oriented subtrees of G as follows: G_0 is the vertex R alone. G_{k+1} is G_k , plus any vertex V of G that is not in G_k but for which there is an arc from V to V' where V' is in G_k , plus one such arc $e[V]$ for each such vertex. It is immediate by induction that G_k is an oriented tree for all $k \geq 0$, and that if there is an oriented path of length k from V to R in G then V is in G_k . Therefore G_∞ , the set of all V and $e[V]$ in any of the G_k , is the desired oriented subtree of G .
- 14.** $(e_{12}, e_{20}, e_{00}, e'_{01}, e_{10}, e_{01}, e'_{12}, e_{22}, e_{21}), (e_{12}, e_{20}, e_{00}, e'_{01}, e'_{12}, e_{22}, e_{21}, e_{10}, e_{01}),$
 $(e_{12}, e_{20}, e'_{01}, e_{10}, e_{00}, e_{01}, e'_{12}, e_{22}, e_{21}), (e_{12}, e_{20}, e'_{01}, e'_{12}, e_{22}, e_{21}, e_{10}, e_{00}, e_{01}),$
 $(e_{12}, e_{22}, e_{20}, e_{00}, e'_{01}, e_{10}, e_{01}, e'_{12}, e_{21}), (e_{12}, e_{22}, e_{20}, e_{00}, e'_{01}, e'_{12}, e_{21}, e_{10}, e_{01}),$
 $(e_{12}, e_{22}, e_{20}, e'_{01}, e_{10}, e_{00}, e_{01}, e'_{12}, e_{21}), (e_{12}, e_{22}, e_{20}, e'_{01}, e'_{12}, e_{21}, e_{10}, e_{00}, e_{01}),$
- in lexicographic order; the eight possibilities come from the independent choices of which of e_{00} or e'_{01} , e_{10} or e'_{12} , e_{20} or e_{22} , should precede the other.
- 15.** True: If it is connected and balanced and has more than one vertex, it has an Eulerian circuit that touches all the vertices.
- 16.** Consider the directed graph G with vertices V_1, \dots, V_{13} and with an arc from V_j to V_k for each k in pile j . Winning the game is equivalent to tracing out an Eulerian circuit in this directed graph (for if the game is won the final card turned up must come from the center; this graph is balanced). Now if the game is won, the stated digraph is an oriented subtree by Lemma E. Conversely if the stated digraph is an oriented tree, the game is won by Theorem D.
- 17.** $\frac{1}{13}$. This answer can be obtained, as the author first obtained it, by laborious enumeration of oriented trees of special types and the application of generating functions, etc., based on the methods of Section 2.3.4.4; it also follows easily from the following

simple, direct proof: Define an order for turning up *all* cards of the deck, as follows: Obey the rules of the game until getting stuck, then "cheat" by turning up the first available card (find the first pile that is not empty, going clockwise from pile 1) and continue as before, until eventually all cards have been turned up. The cards in the order of turning up are in completely random order (since the value of a card need not be specified until after it is turned up). So the problem is just to calculate the probability that in a randomly shuffled deck the last card is a king. More generally the probability that k cards are still face down when the game is over is the probability that the last king in a random shuffle is followed by k cards, namely $4! \binom{51-k}{3} \frac{48!}{52!}$. Hence a person playing this game without cheating will turn up an average of exactly 42.4 cards per game. Note: Similarly, it can be shown that the probability that the player will have to "cheat" k times in the process described above is exactly given by the Stirling number $\left[\begin{smallmatrix} 13 \\ k+1 \end{smallmatrix} \right] / 13!$. (See Section Eq. 1.2.10-(9) and exercise 1.2.10-7; the case of a more general card deck is considered in exercise 1.2.10-18.)

18. (a) If there is a cycle (V_0, V_1, \dots, V_k) , where necessarily $3 \leq k \leq n$, the sum of the k rows of A corresponding to the k edges of this cycle, with appropriate signs, is a row of zeros; so if G is not a free tree the determinant of A_0 is zero.

But if G is a free tree we may regard it as an ordered tree with root V_0 , and we can rearrange the rows and columns of A_0 so that columns are in preorder and so that the k th row corresponds to the edge from the k th vertex (column) to its parent. Then the matrix is triangular with ± 1 's on the diagonal, so the determinant is ± 1 .

- (b) By the Binet-Cauchy formula (exercise 1.2.3-46) we have

$$\det A_0^T A_0 = \sum_{1 \leq i_1 < \dots < i_n \leq m} (\det A_{i_1 \dots i_n})^2$$

where $A_{i_1 \dots i_n}$ represents a matrix consisting of rows i_1, \dots, i_n of A_0 (thus corresponding to a choice of n edges of G). The result now follows from (a).

[See S. Okada and R. Onodera, Bull. Yamagata Univ. 2 (1952), 89-117.]

19. (a) The conditions $a_{00} = 0$ and $a_{jj} = 1$ are just conditions (a), (b) of the definition of oriented tree. If G is not an oriented tree there is an oriented cycle (by exercise 7), and the rows of A_0 corresponding to the vertices in this oriented cycle will sum to a row of zeros; hence $\det A_0 = 0$. If G is an oriented tree, assign an arbitrary order to the children of each family and regard G as an ordered tree. Now permute rows and columns of A_0 until they correspond to preorder of the vertices. Since the same permutation has been applied to the rows as to the columns, the determinant is unchanged; and the resulting matrix is triangular with +1 in every diagonal position.

- (b) We may assume that $a_{0j} = 0$ for all j , since no arc emanating from V_0 can participate in an oriented subtree. We may also assume that $a_{jj} > 0$ for all $j \geq 1$ since otherwise the whole j th row is zero and there obviously are no oriented subtrees. Now use induction on the number of arcs: If $a_{jj} > 1$ let e be some arc leading from V_j ; let B_0 be a matrix like A_0 but with arc e deleted, and let C_0 be the matrix like A_0 but with all arcs except e that lead from V_j deleted. Example: If $A_0 = \begin{pmatrix} 3 & -2 \\ -1 & 2 \end{pmatrix}$, $j = 1$, and e is an arc from V_1 to V_0 , then $B_0 = \begin{pmatrix} 2 & -2 \\ -1 & 2 \end{pmatrix}$, $C_0 = \begin{pmatrix} 1 & 0 \\ -1 & 2 \end{pmatrix}$. In general we have $\det A_0 = \det B_0 + \det C_0$, since the matrices agree in all rows except row j , and A_0 is the sum of B_0 and C_0 in that row. Moreover, the number of oriented subtrees of G is the number of subtrees that do not use e (namely, $\det B_0$, by induction) plus the number that do use e (namely, $\det C_0$).

Notes: The matrix A is often called the *Laplacian* of the graph, by analogy with a similar concept in the theory of partial differential equations. If we delete any set S of

rows from the matrix A , and the same set of columns, the determinant of the resulting matrix is the number of oriented forests whose roots are the vertices $\{V_k \mid k \in S\}$ and whose arcs belong to the given digraph. The matrix tree theorem for oriented trees was stated without proof by J. J. Sylvester in 1857 (see exercise 28), then forgotten for many years until it was independently rediscovered by W. T. Tutte [Proc. Cambridge Phil. Soc. 44 (1948), 463–482, §3]. The first published proof in the special case of undirected graphs, when the matrix A is symmetric, was given by C. W. Borchardt [Crelle 57 (1860), 111–121]. Several authors have ascribed the theorem to Kirchhoff, but Kirchhoff proved a quite different (though related) result.

20. Using exercise 18 we find $B = A_0^T A_0$. Or, using exercise 19, B is the matrix A_0 for the directed graph G' with two arcs (one in each direction) in place of each edge of G ; each free subtree of G corresponds uniquely to an oriented subtree of G' with root V_0 , since the directions of the arcs are determined by the choice of root.

21. Construct the matrices A and A^* as in exercise 19. For the example graphs G and G^* in Figs. 36 and 37,

$$A = \begin{pmatrix} 2 & -2 & 0 \\ -1 & 3 & -2 \\ -1 & -1 & 2 \end{pmatrix}, \quad A^* = \begin{pmatrix} [00] & [10] & [20] & [01] & [01] & [21] & [12] & [12] & [22] \\ [00] & 2 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ [10] & -1 & 3 & 0 & -1 & -1 & 0 & 0 & 0 \\ [20] & -1 & 0 & 3 & -1 & -1 & 0 & 0 & 0 \\ \hline [01] & 0 & -1 & 0 & 3 & 0 & 0 & -1 & -1 & 0 \\ [01] & 0 & -1 & 0 & 0 & 3 & 0 & -1 & -1 & 0 \\ [21] & 0 & -1 & 0 & 0 & 0 & 3 & -1 & -1 & 0 \\ \hline [12] & 0 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & -1 \\ [12] & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 3 & -1 \\ [22] & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 2 \end{pmatrix}.$$

Add the indeterminate λ to the upper left corner element of A and A^* (in the example this gives $2 + \lambda$ in place of 2). If $t(G)$ and $t(G^*)$ are the numbers of oriented subtrees of G and G^* , we have $t(G) = \lambda^{-1}(n+1)\det A$, $t(G^*) = \lambda^{-1}m(n+1)\det A^*$. (The number of oriented subtrees of a balanced graph is the same for any given root, by exercise 22.)

If we group vertices V_{jk} for equal k the matrix A^* can be partitioned as shown above. Let $B_{kk'}$ be the submatrix of A^* consisting of the rows for V_{jk} and the columns for $V_{j'k'}$, for all j and j' such that V_{jk} and $V_{j'k'}$ are in G^* . By adding the 2nd, ..., m th columns of each submatrix to the first column and then subtracting the first row of each submatrix from the 2nd, ..., m th rows, the matrix A^* is transformed so that

$$B_{kk'} = \begin{pmatrix} a_{kk'} & * & \dots & * \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \quad \text{for } k \neq k', \quad B_{kk} = \begin{pmatrix} a_{kk} + \lambda \delta_{k0} & * & \dots & * \\ -\lambda \delta_{k0} & m & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\lambda \delta_{k0} & 0 & \dots & m \end{pmatrix}.$$

It follows that $\det A^*$ is $m^{m(n-1)}$ times the determinant of

$$\begin{pmatrix} \lambda + a_{00} & * & * & \dots & * & a_{01} & \dots & a_{0n} \\ -\lambda & m & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & & & & & & \\ -\lambda & 0 & 0 & \dots & m & 0 & \dots & 0 \\ a_{10} & * & * & \dots & * & a_{11} & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots & & & & \\ a_{n0} & * & * & \dots & * & a_{n1} & \dots & a_{nn} \end{pmatrix}.$$

In this derivation, "*" indicates values that are more or less irrelevant; the remaining asterisks are all zero except for precisely one -1 in each column. Add the last n rows to the top row, and expand the determinant by the first row, to get $m^{n(m-1)+m-1} \det A - (m-1)m^{n(m-1)+m-2} \det A$.

This derivation can be generalized to determine the number of oriented subtrees of G^* when G is an arbitrary directed graph; see R. Dawson and I. J. Good, *Ann. Math. Stat.* 28 (1957), 946–956; D. E. Knuth, *Journal of Combinatorial Theory* 3 (1967), 309–314.) An alternative, purely combinatorial proof has been given by J. B. Orlin, *Journal of Combinatorial Theory* B25 (1978), 187–198.

22. The total number is $(\sigma_1 + \dots + \sigma_n)$ times the number of Eulerian circuits starting with a given edge e_1 , where $\text{init}(e_1) = V_1$. Each such circuit determines an oriented subtree with root V_1 by Lemma E, and for each of the T oriented subtrees there are $\prod_{j=1}^n (\sigma_j - 1)!$ paths satisfying the three conditions of Theorem D, corresponding to the different order in which the arcs $\{e \mid \text{init}(e) = V_j, e \neq e[V_j], e \neq e_1\}$ are entered into P . (Exercise 14 provides a simple example.)

23. Construct the directed graph G_k with m^{k-1} vertices as in the hint, and denote by $[x_1, \dots, x_k]$ the arc mentioned there. For each function that has maximum period length, we can define a unique corresponding Eulerian circuit, by letting $f(x_1, \dots, x_k) = x_{k+1}$ if arc $[x_1, \dots, x_k]$ is followed by $[x_2, \dots, x_{k+1}]$. (We regard Eulerian circuits as being the same if one is just a cyclic permutation of the other.) Now $G_k = G_{k-1}^*$ in the sense of exercise 21, so G_k has $m^{m^{k-1}-m^{k-2}}$ times as many oriented subtrees as G_{k-1} ; by induction G_k has $m^{m^{k-1}-1}$ oriented subtrees, and $m^{m^{k-1}-k}$ with a given root. Therefore by exercise 22 the number of functions with maximum period, namely the number of Eulerian circuits of G_k starting with a given arc, is $m^{-k}(m!)^{m^{k-1}}$. [For $m = 2$ this result is due to C. Flye Sainte-Marie, *L'Intermédiaire des Mathématiciens* 1 (1894), 107–110.]

24. Define a new directed graph having E_j copies of e_j , for $0 \leq j \leq m$. This graph is balanced, hence it contains an Eulerian circuit (e_0, \dots) by Theorem G. The desired oriented path comes by deleting the edge e_0 from this Eulerian circuit.

25. Assign an arbitrary order to all arcs in the sets $I_j = \{e \mid \text{init}(e) = V_j\}$ and $F_j = \{e \mid \text{fin}(e) = V_j\}$. For each arc e in I_j , let $\text{ATAG}(e) = 0$ and $\text{ALINK}(e) = e'$ if e' follows e in the ordering of I_j ; also let $\text{ATAG}(e) = 1$ and $\text{ALINK}(e) = e'$ if e is last in I_j and e' is first in F_j . Let $\text{ALINK}(e) = \Lambda$ in the latter case if F_j is empty. Define BLINK and BTAG by the same rules, reversing the roles of init and fin .

Examples (using alphabetic order in each set of arcs):

arc	ALINK	ATAG	BLINK	BTAG	arc	ALINK	ATAG	BLINK	BTAG
a	d	1	b	0	a	c	0	b	1
b	Λ	1	c	0	b	a	1	d	0
c	f	1	r	1	c	e	0	a	1
d	Λ	1	e	0	d	h	0	f	0
e	Λ	1	a	1	e	g	0	f	1
f	Λ	1	c	1	f	j	0	d	1
r	a	1	Λ	1	g	c	1	h	0
					h	b	1	j	0
					j	e	1	Λ	1

Note: If in the oriented tree representation we add another arc from H to itself, we get an interesting situation: Either we get the standard conventions 2.3.1–(8) with LLINK, LTAG, RLINK, RTAG interchanged in the list head, or (if the new arc is placed last in the ordering) we get the standard conventions except RTAG = 0 in the node associated with the root of the tree.

This exercise is based on an idea communicated to the author by W. C. Lynch. Can tree traversal algorithms like Algorithm 2.3.1S be generalized to classes of digraphs that are not oriented trees, using such a representation?

27. Let a_{ij} be the sum of $p(e)$ over all arcs e from V_i to V_j . We are to prove that $t_j = \sum_i a_{ij} t_i$ for all j . Since $\sum_i a_{ii} = 1$, we must prove that $\sum_i a_{ji} t_j = \sum_i a_{ij} t_i$. But this is not difficult, because both sides of the equation represent the sum of all products $p(e_1) \dots p(e_n)$ taken over subgraphs $\{e_1, \dots, e_n\}$ of G such that $\text{init}(e_i) = V_i$ and such that there is a unique oriented cycle contained in $\{e_1, \dots, e_n\}$, where this cycle includes V_j . Removing any arc of the cycle yields an oriented tree; the left-hand side of the equation is obtained by factoring out the arcs that leave V_j , while the right-hand side corresponds to those that enter V_j .

In a sense, this exercise is combination of exercises 19 and 26.

28. Every term in the expansion is $a_{1p_1} \dots a_{mp_m} b_{1q_1} \dots b_{nq_n}$, where $0 \leq p_i \leq n$ for $1 \leq i \leq m$ and $0 \leq q_j \leq m$ for $1 \leq j \leq n$, times some integer coefficient. Represent this product as a directed graph on the vertices $\{0, u_1, \dots, u_m, v_1, \dots, v_n\}$, with arcs from u_i to v_{p_i} and from v_j to u_{q_j} , where $u_0 = v_0 = 0$.

If the digraph contains a cycle, the integer coefficient is zero. For each cycle corresponds to a factor of the form

$$a_{i_0j_0} b_{j_0i_1} a_{i_1j_1} \dots a_{i_{k-1}j_{k-1}} b_{j_{k-1}i_0} \quad (*)$$

where the indices $(i_0, i_1, \dots, i_{k-1})$ are distinct and so are the indices $(j_0, j_1, \dots, j_{k-1})$. The sum of all terms containing $(*)$ as a factor is the determinant obtained by setting $a_{i,j} \leftarrow [j = i_1]$ for $0 \leq j \leq n$ and $b_{j,i} \leftarrow [i = i_{(l+1) \bmod k}]$ for $0 \leq i \leq m$, for $0 \leq l < k$, leaving the variables in the other $m+n-2k$ rows unchanged. This determinant is identically zero, because the sum of rows i_0, i_1, \dots, i_{k-1} in the top section equals the sum of rows j_0, j_1, \dots, j_{k-1} in the bottom section.

On the other hand, if the directed graph contains no cycles, the integer coefficient is +1. This follows because each factor a_{i,p_i} and b_{j,q_j} must have come from the diagonal of the determinant: If any off-diagonal element $a_{i_0j_0}$ is chosen in row i_0 of the top section, we must choose some off-diagonal $b_{j_0i_1}$ from column j_0 of the left section, hence we must choose some off-diagonal $a_{i_1j_1}$ from row i_1 of the top section, etc., forcing a cycle.

Thus the coefficient is +1 if and only if the corresponding digraph is an oriented tree with root 0. The number of such terms (hence the number of such oriented trees) is obtained by setting each a_{ij} and b_{ji} to 1; for example,

$$\begin{aligned} \det \begin{pmatrix} 4 & 0 & 1 & 1 & 1 \\ 0 & 4 & 1 & 1 & 1 \\ 1 & 1 & 3 & 0 & 0 \\ 1 & 1 & 0 & 3 & 0 \\ 1 & 1 & 0 & 0 & 3 \end{pmatrix} &= \det \begin{pmatrix} 4 & 0 & 1 & 1 & 1 \\ -4 & 4 & 0 & 0 & 0 \\ 1 & 1 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 & 0 \\ 0 & 0 & -3 & 0 & 3 \end{pmatrix} = \det \begin{pmatrix} 4 & 0 & 3 & 1 & 1 \\ 0 & 4 & 0 & 0 & 0 \\ 2 & 1 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{pmatrix} \\ &= \det \begin{pmatrix} 4 & 3 \\ 2 & 3 \end{pmatrix} \cdot 4 \cdot 3 \cdot 3. \end{aligned}$$

In general we obtain $\det \begin{pmatrix} n+1 & n \\ m & m+1 \end{pmatrix} \cdot (n+1)^{m-1} \cdot (m+1)^{n-1}$.

Notes: J. J. Sylvester considered the special case $m = n$ and $a_{10} = a_{20} = \dots = a_{mn} = 0$ in *Quarterly J. of Pure and Applied Math.* 1 (1857), 42–56, where he conjectured (correctly) that the total number of terms is then $n^n(n+1)^{n-1}$. He also stated without proof that the $(n+1)^{n-1}$ nonzero terms present when $a_{ij} = \delta_{ij}$ correspond to all connected cycle-free graphs on $\{0, 1, \dots, n\}$. In that special case, he reduced the determinant to the form in the matrix tree theorem of exercise 19, e.g.,

$$\det \begin{pmatrix} b_{10} + b_{12} + b_{13} & -b_{12} & -b_{13} \\ -b_{21} & b_{20} + b_{21} + b_{23} & -b_{23} \\ -b_{31} & -b_{32} & b_{30} + b_{31} + b_{32} \end{pmatrix}.$$

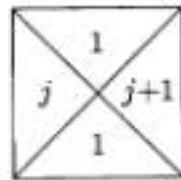
Cayley quoted this result in *Crelle* 52 (1856), 279, ascribing it to Sylvester; thus it is ironic that the theorem about the number of such graphs is often attributed to Cayley.

By negating the first m rows of the given determinant, then negating the first m columns, we can reduce this exercise to the matrix tree theorem.

[Matrices having the general form considered in this exercise are important in iterative methods for the solution of partial differential equations, and they are said to have "Property A." See, for example, Louis A. Hageman and David M. Young, *Applied Iterative Methods* (Academic Press, 1981), Chapter 9.]

SECTION 2.3.4.3

1. The root is the empty sequence; arcs go from (x_1, \dots, x_n) to (x_1, \dots, x_{n-1}) .
2. Take one tetrad type and rotate it 180° to get another tetrad type; these two types give an obvious way to tile the plane (without further rotations) by replication of a 2×2 pattern.



3. Consider the set of tetrad types for all positive integers j . The right half plane can be tiled in uncountably many ways; but whatever square is placed in the center of the plane puts a finite limit on the distance it can be continued to the left.

4. Systematically enumerate all possible ways to tile an $n \times n$ block, for $n = 1, 2, \dots$, looking for toroidal solutions within these blocks. If there is no way to tile the plane, the infinity lemma tells us there is an n with no $n \times n$ solutions. If there is a way to tile the plane, the assumption tells us that there is an n with an $n \times n$ solution containing a rectangle that yields a toroidal solution. Hence in either case the algorithm will terminate. (But the stated assumption is false, as shown in the next exercise, and in fact there is no algorithm that will determine in a finite number of steps whether or not there exists a way to tile the plane with a given set of types.)

5. Start by noticing that we need classes $\begin{smallmatrix} \alpha & \beta \\ \gamma & \delta \end{smallmatrix}$ replicated in 2×2 groups in any solution. Then, step 1: Considering just the α squares, show that the pattern $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ must be replicated in 2×2 groups of α squares. Step $n > 1$: Determine a pattern that must appear in a cross-shaped region of height and width $2^n - 1$. The middle of the crosses has the pattern $\begin{smallmatrix} Na & Nb \\ Nc & Nd \end{smallmatrix}$ replicated throughout the plane.

For example, after step 3 we will know the contents of 7×7 blocks throughout the plane, separated by unit length strips, every eight units. The 7×7 blocks that are of

class Na in the center have the form

αa	βKQ	αb	βQP	αa	βBK	αb
γPJ	δNa	γRB	δQK	γLJ	δNb	γPB
αc	βDS	αd	βQTY	αc	βBS	αd
γPQ	δPJ	γPXB	δNa	γRQ	δRB	γRB
αa	βUK	αb	βDP	αa	βBK	αb
γTJ	δNe	γSB	δDS	γST	δNd	γTB
αc	βQS	αd	βDT	αc	βBS	αd

The middle column and the middle row is the “cross” just filled in during step 3; the other four 3×3 squares were filled in after step 2; the squares just to the right and below this 7×7 square are part of a 15×15 cross to be filled in at step 4.

For a similar construction that leads to a set of only 35 tetrad types having nothing but nontoroidal solutions, see R. M. Robinson, *Inventiones Math.* **12** (1971), 177–209. Robinson also exhibits a set of six squarish shapes that tile the plane only nontoroidally, even when rotations and reflections are allowed. In 1974, Roger Penrose discovered a set of only two polygons, based on the golden ratio instead of a square grid, that tile the plane only aperiodically; this led to a set of only 16 tetrad types with only nontoroidal solutions [see B. Grünbaum and G. C. Shephard, *Tilings and Patterns* (Freeman, 1987), Chapters 10–11; Martin Gardner, *Penrose Tiles to Trapdoor Ciphers* (Freeman, 1989), Chapters 1–2].

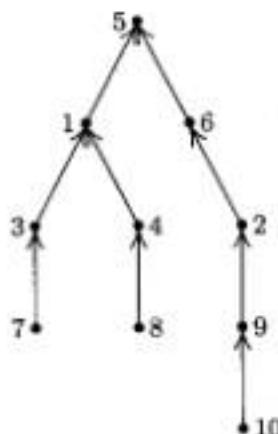
6. Let k and m be fixed. Consider an oriented tree whose vertices each represent, for some n , one of the partitions of $\{1, \dots, n\}$ into k parts, containing no arithmetic progression of length m . A node that partitions $\{1, \dots, n+1\}$ is a child of one for $\{1, \dots, n\}$ if the two partitions agree on $\{1, \dots, n\}$. If there were an infinite path to the root we would have a way to divide all integers into k sets with no arithmetic progression of length m . Hence, by the infinity lemma and van der Waerden’s theorem, this tree is finite. (If $k = 2, m = 3$, the tree can be rapidly calculated by hand, and the least value of N is 9. See *Studies in Pure Mathematics*, ed. by L. Mirsky (Academic Press, 1971), 251–260, for van der Waerden’s interesting account of how the proof of his theorem was discovered.)

7. The positive integers can be partitioned into two sets S_0 and S_1 such that neither set contains any infinite computable sequence (see exercise 3.5–32). So in particular there is no infinite arithmetic progression. Theorem K does not apply because there is no way to put partial solutions into a tree with finite degrees at each vertex.

8. Let a “counterexample sequence” be an infinite sequence of trees that violates Kruskal’s theorem, if such sequences exist. Assume that the theorem is false; then let T_1 be a tree with the smallest possible number of nodes such that T_1 can be the first tree in a counterexample sequence; if T_1, \dots, T_j have been chosen, let T_{j+1} be a tree with the smallest possible number of nodes such that T_1, \dots, T_j, T_{j+1} is the beginning of a counterexample sequence. This process defines a counterexample sequence $\langle T_n \rangle$. None of these T ’s is just a root. Now, we look at this sequence very carefully:

(a) Suppose there is a subsequence T_{n_1}, T_{n_2}, \dots for which $l(T_{n_1}), l(T_{n_2}), \dots$ is a counterexample sequence. This is impossible; otherwise $T_1, \dots, T_{n_1-1}, l(T_{n_1}), l(T_{n_2}), \dots$ would be a counterexample sequence, contradicting the definition of T_{n_1} .

13.



14. True, since the root will not become a leaf until all other branches have been removed.

15. In the canonical representation, $V_1, V_2, \dots, V_{n-1}, f(V_{n-1})$ is a topological sort of the oriented tree considered as a directed graph, but this order would not in general be output by Algorithm 2.2.3T. Algorithm 2.2.3T can be changed so that it determines the values of V_1, V_2, \dots, V_{n-1} if the "insert into queue" operation of step T6 is replaced by a procedure that adjusts links so that the entries of the list appear in ascending order from front to rear; then the queue becomes a priority queue.

(However, a general priority queue isn't needed to find the canonical representation; we only need to sweep through the vertices from 1 to n , looking for leaves, while pruning off paths from new leaves less than the sweep pointer; see the following exercise.)

16. D1. Set $C[1] \leftarrow \dots \leftarrow C[n] \leftarrow 0$, then set $C[f(V_j)] \leftarrow C[f(V_j)] + 1$ for $1 \leq j \leq n$. (Thus vertex k is a leaf if and only if $C[k] = 0$.) Set $k \leftarrow 0$ and $j \leftarrow 1$.

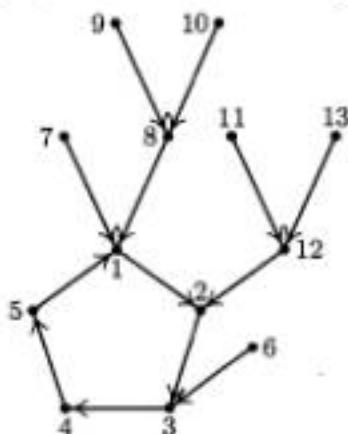
D2. Increase k one or more times until $C[k] = 0$, then set $l \leftarrow k$.

D3. Set $PARENT[l] \leftarrow f(V_j)$, $l \leftarrow f(V_j)$, $C[l] \leftarrow C[l] - 1$, and $j \leftarrow j + 1$.

D4. If $j = n$, set $PARENT[l] \leftarrow 0$ and terminate the algorithm.

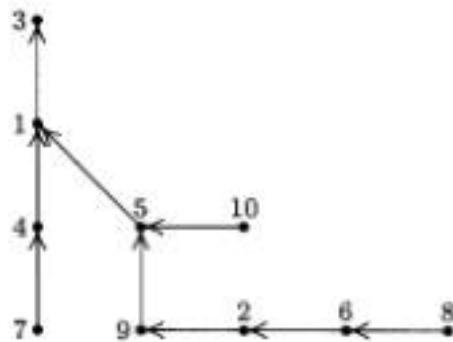
D5. If $C[l] = 0$ and $l < k$, go to D3; otherwise go back to D2. ■

17. There must be exactly one cycle x_1, x_2, \dots, x_k where $f(x_j) = x_{j+1}$ and $f(x_k) = x_1$. We will enumerate all f having a cycle of length k such that the iterates of each x ultimately come into this cycle. Define the canonical representation $f(V_1), f(V_2), \dots, f(V_{m-k})$ as in the text; now $f(V_{m-k})$ is in the cycle, so we continue to get a "canonical representation" by writing down the rest of the cycle $f(f(V_{m-k})), f(f(f(V_{m-k}))),$ etc. For example, the function with $m = 13$ whose graph is shown here leads to the representation 3, 1, 8, 8, 1, 12, 12, 2, 3, 4, 5, 1. We obtain a sequence of $m - 1$ numbers in which the last k are distinct. Conversely, from any such sequence we can reverse the construction (assuming that k is known); hence there are precisely $m^k m^{m-k-1}$ such functions having a k -cycle. (For related results, see exercise 3.1-14. The formula $m^{m-1} Q(m)$ was first obtained by L. Katz, Annals of Math. Statistics 26 (1955), 512-517.)



18. To reconstruct the tree from a sequence s_1, s_2, \dots, s_{n-1} , begin with s_1 as the root and successively attach arcs to the tree that point to s_1, s_2, \dots ; if vertex s_k has appeared earlier, leave the initial vertex of the arc leading to s_{k-1} nameless, otherwise give this vertex the name s_k . After all $n-1$ arcs have been placed, give names to all vertices that remain nameless by using the numbers that have not yet appeared, assigning names in increasing order to nameless vertices in the order of their creation.

For example, from 3, 1, 4, 1, 5, 9, 2, 6, 5 we would construct the tree shown on the right. There is no simple connection between this method and the one in the text. Several more representations are possible; see the article by E. H. Neville, Proc. Cambridge Phil. Soc. 49 (1953), 381–385.



19. The canonical representation will have precisely $n-k$ different values, so we enumerate the sequences of $n-1$ numbers with this property. The answer is $\frac{n-k}{n-k} \binom{n-1}{n-k}$.

20. Consider the canonical representation of such trees. We are asking how many terms of $(x_1 + \dots + x_n)^{n-1}$ have k_0 exponents zero, k_1 exponents one, etc. This is plainly the coefficient of such a term times the number of such terms, namely

$$\frac{(n-1)!}{(0!)^{k_0}(1!)^{k_1}\dots(n!)^{k_n}} \times \frac{n!}{k_0! k_1! \dots k_n!}.$$

21. There are none with $2m$ vertices; if there are $n = 2m+1$ vertices, the answer is obtained from exercise 20 with $k_0 = m+1, k_2 = m$, namely $\binom{2m+1}{m}(2m)!/2^m$.

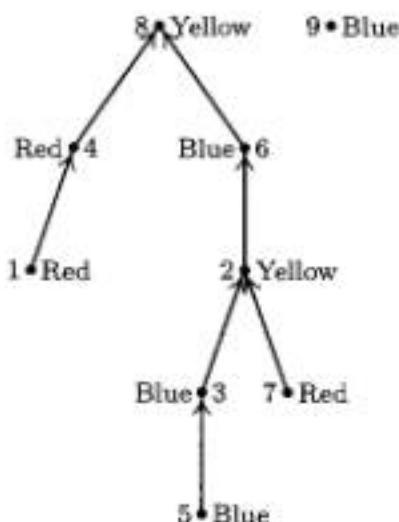
22. Exactly n^{n-2} ; for if X is a particular vertex, the free trees are in one-to-one correspondence with oriented trees having root X .

23. It is possible to put the labels on every unlabeled, ordered tree in $n!$ ways, and each of these labeled, ordered trees is distinct. So the total number is $n! b_{n-1} = (2n-2)!/(n-1)!$.

24. There are as many with one given root as with another, so the answer in general is $1/n$ times the answer in exercise 23; and in this particular case the answer is 30.

25. For $0 \leq q < n$, $r(n, q) = (n-q)n^{q-1}$. (The special case $s=1$ in Eq. (24).)

26. ($k=7$)



The methods above can be generalized to show that the number of (ordered, unlabeled) forests having f trees and n_j nodes of degree j is $(n-1)! f/n_0! n_1! \dots n_m!$, provided that the condition $n_0 = f + n_2 + 2n_3 + \dots$ is satisfied.

33. Consider the number of trees with n_1 nodes labeled 1, n_2 nodes labeled 2, ..., and such that each node labeled j has degree e_j . Let this number be $c(n_1, n_2, \dots)$, with the specified degrees e_1, e_2, \dots regarded as fixed. The generating function $G(z_1, z_2, \dots) = \sum c(n_1, n_2, \dots) z_1^{n_1} z_2^{n_2} \dots$ satisfies the identity $G = z_1 G^{e_1} + \dots + z_r G^{e_r}$, since $z_j G^{e_j}$ enumerates the trees whose root is labeled j . And by the result of the previous exercise,

$$c(n_1, n_2, \dots) = \begin{cases} \frac{(n_1 + n_2 + \dots - 1)!}{n_1! n_2! \dots}, & \text{if } (1 - e_1)n_1 + (1 - e_2)n_2 + \dots = 1; \\ 0, & \text{otherwise.} \end{cases}$$

More generally, since G^f enumerates the number of ordered forests having such labels, we have for integer $f > 0$

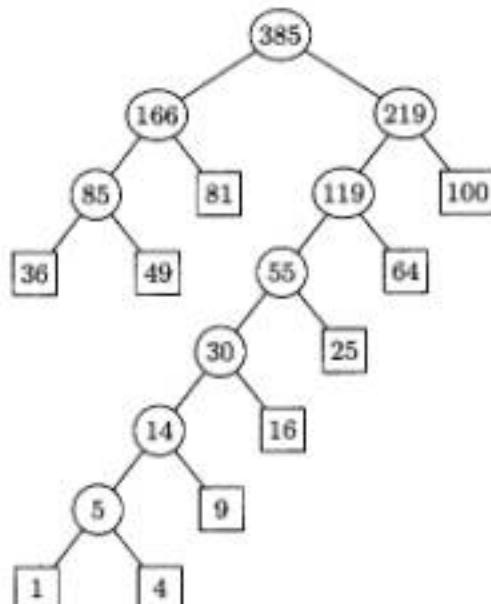
$$w^f = \sum_{f=(1-e_1)n_1+(1-e_2)n_2+\dots} \frac{(n_1 + n_2 + \dots - 1)! f}{n_1! n_2!} z_1^{n_1} z_2^{n_2} \dots$$

These formulas are meaningful when $r = \infty$, and they are essentially equivalent to Lagrange's inversion formula.

SECTION 2.3.4.5

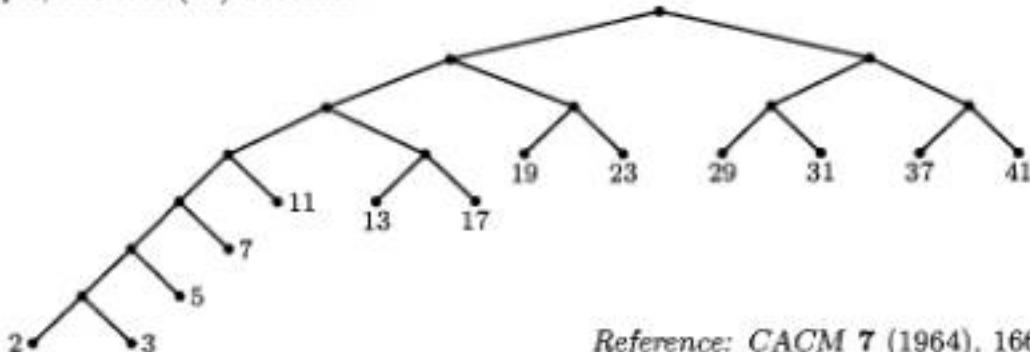
1. There are $\binom{8}{5}$ in all, since the nodes numbered 8, 9, 10, 11, 12 may be attached in any of eight positions below 4, 5, 6, and 7.

2.



3. By induction on m , the condition is necessary. Conversely if $\sum_{j=1}^m 2^{-l_j} = 1$, we want to construct an extended binary tree with path lengths l_1, \dots, l_m . When $m = 1$, we have $l_1 = 0$ and the construction is trivial. Otherwise we may assume that the l 's are ordered so that $l_1 = l_2 = \dots = l_q > l_{q+1} \geq l_{q+2} \geq \dots \geq l_m > 0$ for some q with $1 \leq q \leq m$. Now $2^{l_1-1} = \sum_{j=1}^m 2^{l_1-l_j-1} = \frac{1}{2}q + \text{integer}$, hence q is even. By induction on m there is a tree with path lengths $l_1 - 1, l_3, l_4, \dots, l_m$; take such a tree and replace one of the external nodes at level $l_1 - 1$ by an internal node whose children are at level $l_1 = l_2$.

4. First, find a tree by Huffman's method. If $w_j < w_{j+1}$, then $l_j \geq l_{j+1}$, since the tree is optimal. The construction in the answer to exercise 3 now gives us another tree with these same path lengths and with the weights in the proper sequence. For example, the tree (11) becomes



Reference: CACM 7 (1964), 166–169.

5. (a) $b_{np} = \sum_{\substack{k+l=n-1 \\ r+s+n-1=p}} b_{kr}b_{ls}$. Hence $zB(w, wz)^2 = B(w, z) - 1$.

- (b) Take the partial derivative with respect to w :

$$2zB(w, wz)(B_w(w, wz) + zB_z(w, wz)) = B_w(w, z).$$

Therefore if $H(z) = B_w(1, z) = \sum_n h_n z^n$, we find $H(z) = 2zB(z)(H(z) + zB'(z))$; and the known formula for $B(z)$ implies

$$H(z) = \frac{1}{1-4z} - \frac{1}{z} \left(\frac{1-z}{\sqrt{1-4z}} - 1 \right), \quad \text{so} \quad h_n = 4^n - \frac{3n+1}{n+1} \binom{2n}{n}.$$

The average value is h_n/b_n . (c) Asymptotically, this comes to $n\sqrt{\pi n} - 3n + O(\sqrt{n})$.

For the solution to similar problems, see John Riordan, *IBM J. Res. and Devel.* 4 (1960), 473–478; A. Rényi and G. Szekeres, *J. Australian Math. Soc.* 7 (1967), 497–507; John Riordan and N. J. A. Sloane, *J. Australian Math. Soc.* 10 (1969), 278–282; and exercise 2.3.1–11.

6. $n + s - 1 = tn$.

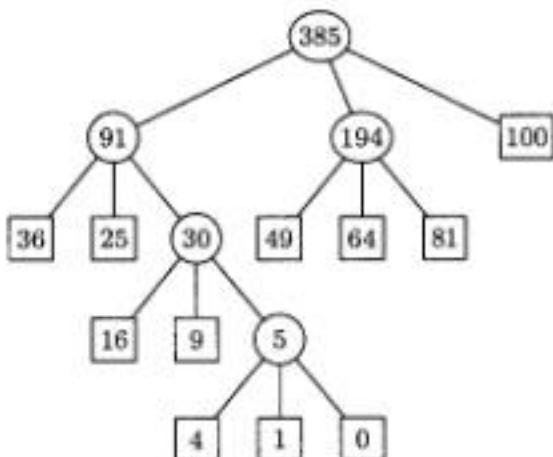
7. $E = (t-1)I + tn$.

8. Summation by parts gives $\sum_{k=1}^n \lfloor \log_t((t-1)k) \rfloor = nq - \sum k$, where the sum on the right is over values of k such that $0 \leq k \leq n$ and $(t-1)k + 1 = t^j$ for some j . The latter sum may be rewritten $\sum_{j=1}^q (t^j - 1)/(t-1)$.

9. Induction on the size of the tree.

10. By adding extra zero weights, if necessary, we may assume that $m \bmod (t-1) = 1$. To obtain a t -ary tree with minimum weighted path length, combine the smallest t values at each step and replace them by their sum. The proof is essentially the same as the binary case. The desired ternary tree is shown.

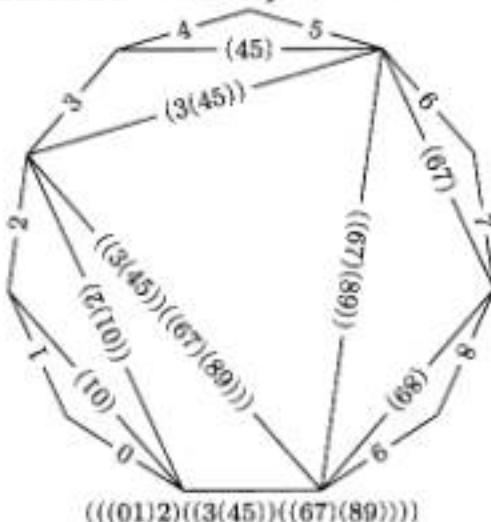
F. K. Hwang has observed [*SIAM J. Appl. Math.* 37 (1979), 124–127] that a similar procedure is valid for minimum weighted path length trees having any prescribed multiset of degrees: Combine the smallest t weights at each step, where t is as small as possible.



SECTION 2.3.4.6

1. Choose one edge of the polygon and call it the base. Given a triangulation, let the triangle on the base correspond to the root of a binary tree, and let the other two sides of that triangle define bases of left and right subpolygons, which correspond to left and right subtrees in the same way. We proceed recursively until reaching "2-sided" polygons, which correspond to empty binary trees.

Stating this correspondence another way, we can label the non-base edges of a triangulated polygon with the integers $0, \dots, n$; and when two adjacent sides of a triangle are labeled α and β in clockwise order, we can label the third side $(\alpha\beta)$. The label of the base then characterizes the binary tree and the triangulation. For example,



corresponds to the binary tree shown in 2.3.1-(1).

2. (a) Take a base edge as in exercise 1, and give it d descendants if that edge is part of a $(d+1)$ -gon in the dissected r -gon. The other d edges are then bases for subtrees. This defines a correspondence between Kirkman's problem and all ordered trees with $r-1$ leaves and $k+1$ nonleaves, having no nodes of degree 1. (When $k=r-3$ we have the situation of exercise 1.)

(b) There are $\binom{r+k}{k+1} \binom{r-3}{k}$ sequences $d_1 d_2 \dots d_{r+k}$ of nonnegative integers such that $r-1$ of the d 's are 0, none of them are 1, and the sum is $r+k-1$. Exactly one of the cyclic permutations $d_1 d_2 \dots d_{r+k}, d_2 \dots d_{r+k} d_1, \dots, d_{r+k} d_1 \dots d_{r+k-1}$ satisfies the additional property that $\sum_{j=1}^q (1-d_j) > 0$ for $1 \leq q \leq r+k$.

[Kirkman gave evidence for his conjecture in *Philos. Trans.* 147 (1857), 217–272, §22. Cayley proved it in *Proc. London Math. Soc.* 22 (1891), 237–262, without noticing the connection to trees.]

3. (a) Let the vertices be $\{1, 2, \dots, n\}$. Draw an RLINK from j to k if they are consecutive elements of the same part and $j < k$; draw an LLINK from j to $j+1$ if $j+1$ is the smallest of its part. Then there are $k-1$ nonnull LLINKs, $n-k$ nonnull RLINKs, and we have a binary tree whose nodes are $12\dots n$ in preorder. Using the natural correspondence of Section 2.3.2, this rule defines a one-to-one correspondence between "partitions of an n -gon's vertices into k noncrossing parts" and "forests with n vertices and $n-k+1$ leaves." Interchanging LLINK with RLINK also gives "forests with n vertices and k leaves."

(b) A forest with n vertices and k leaves also corresponds to a sequence of nested parentheses, containing n left parentheses, n right parentheses, and k occurrences of " $()$ ". We can enumerate such sequences as follows:

Say that a string of 0s and 1s is an (m, n, k) string if there are m 0s, n 1s, and k occurrences of "01". Then 0010101001110 is a $(7, 6, 4)$ string. The number of (m, n, k) strings is $\binom{m}{k} \binom{n}{k}$, because we are free to choose which 0s and 1s will form the 01 pairs.

Let $S(\alpha)$ be the number of 0s in α minus the number of 1s. We say that a string σ is *good* if $S(\alpha) \geq 0$ whenever α is a prefix of σ (in other words, if $\sigma = \alpha\beta$ implies that $S(\alpha) \geq 0$); otherwise σ is *bad*. The following alternative to the “reflection principle” of exercise 2.2.1-4 establishes a one-to-one correspondence between bad (n, n, k) strings and arbitrary $(n - 1, n + 1, k)$ strings:

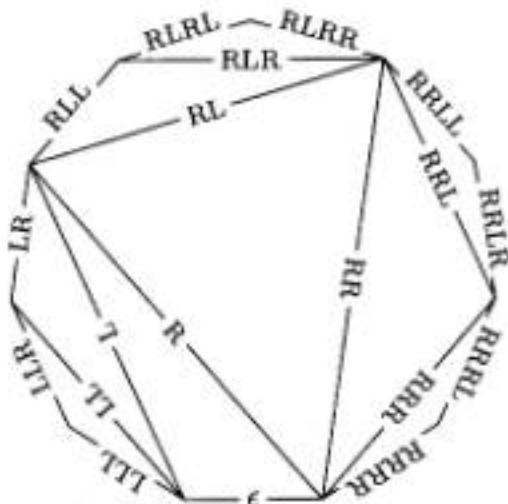
Any bad (n, n, k) string σ can be written uniquely in the form $\sigma = \alpha 0\beta$, where $\bar{\alpha}^R$ and β are good. (Here $\bar{\alpha}^R$ is the string obtained from α by reversing it and complementing all the bits.) Then $\sigma' = \alpha 1\beta$ is an $(n - 1, n + 1, k)$ string. Conversely, every $(n - 1, n + 1, 1)$ string can be written uniquely in the form $\alpha 1\beta$ where $\bar{\alpha}^R$ and β are good, and $\alpha 0\beta$ is then a bad (n, n, k) string.

Consequently the number of forests with n vertices and k leaves is $\binom{n}{k} \binom{n}{k} - \binom{n-1}{k} \binom{n+1}{k} = n! (n-1)! / (n-k+1)! (n-k)! k! (k-1)!$.

Notes: G. Kreweras, *Discrete Math.* **1** (1972), 333–350, enumerated noncrossing partitions in a different way. The partial ordering of partitions by refinement leads to an interesting partial ordering of forests, different from the discussed in exercise 2.3.3–19; see Y. Poupart, *Cahiers du Bureau Univ. de Recherche Opérationnelle* **16** (1970), Chapter 8; *Discrete Math.* **2** (1972), 279–288; P. Edelman, *Discrete Math.* **31** (1980), 171–180, **40** (1982), 171–179.

A third way to define a natural lattice ordering of forests was introduced by R. Stanley in *Fibonacci Quarterly* 13 (1975), 215–232: Suppose we represent a forest by a string σ of 0s and 1s representing left and right parentheses as above; then $\sigma \leq \sigma'$ if and only if $S(\sigma_k) \leq S(\sigma'_k)$ for all k , where σ_k denotes the first k bits of σ . Stanley's lattice is *distributive*, unlike the other two.

4. Let $m = n + 2$; by exercise 1, we want a correspondence between triangulated m -gons and $(m - 1)$ -rowed friezes. First let's look more closely at the previous correspondence, by giving a "top-down" labeling to the edges of a triangulation instead of the "bottom-up" one considered earlier: Assign the empty label ϵ to the base, then recursively give the labels αL and αR to the opposite edges of a triangle whose base is labeled α . For example, the previous diagram becomes



under these the new conventions. If the base edge in this example is called 10, while the other edges are 0 to 9 as before, we can write $0 = 10LLL$, $1 = 10LLR$, $2 = 10LR$,

$3 = 10RLL$, etc. Any of the other edges can also be chosen as the base; thus, if 0 is chosen we have $1 = 0L$, $2 = 0RL$, $3 = 0RRLLL$, etc. It is not difficult to verify that if $u = u\alpha$ we have $v = u\alpha^T$, where α^T is obtained by reading α from right to left and interchanging L with R . For example, $10 = 0RRR = 1LRR = 2LR = 3RRL$, etc. If u , v , and w are edges of the polygon with $w = u\alpha L\gamma$ and $w = v\beta R\gamma$, then $u = v\beta L\alpha^T$ and $v = u\alpha R\beta^T$.

Given a triangulation of a polygon whose edges are numbered $0, 1, \dots, m-1$, we define (u, v) for any pair of distinct edges u and v as follows: Let $u = v\alpha$, and interpret α as a 2×2 matrix by letting $L = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ and $R = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$. Then (u, v) is defined to be the element in the upper left corner of α . Notice that α^T is the transpose of the matrix α , since $R = L^T$; hence we have $(v, u) = (u, v)$. Notice also that $(u, v) = 1$ if and only if u_- and v_- are joined by an edge of the triangulation, where u_- denotes the vertex between edges u and $u-1$.

Let $(u, u) = 0$ for all polygon edges u . We can now prove that $v = u\alpha$ implies

$$\alpha = \begin{pmatrix} (u, v) & (u, v+1) \\ (u+1, v) & (u+1, v+1) \end{pmatrix} \quad \text{for all } u \neq v, \quad (*)$$

where $u+1$ and $v+1$ are the clockwise successors of u and v . The proof is by induction on m : Eq. (*) is trivial when $m = 2$, since the two parallel edges u and v are then related by $u = ve$, and $\alpha = e$ is the identity matrix. If any triangulation is augmented by extending some edge v with a triangle $v v' v''$, then $v = ua$ implies $v' = uaL$ and $v'' = uaR$; hence (u, v') and (u, v'') in the extended polygon are respectively equal to (u, v) and $(u, v) + (u, v+1)$ in the original one. It follows that

$$\alpha L = \begin{pmatrix} (u, v') & (u, v'') \\ (u+1, v') & (u+1, v'') \end{pmatrix} \quad \text{and} \quad \alpha R = \begin{pmatrix} (u, v'') & (u, v''+1) \\ (u+1, v'') & (u+1, v''+1) \end{pmatrix},$$

and (*) remains true in the extended polygon.

The frieze pattern corresponding to the given triangulation is now defined to be the periodic sequence

(0, 1)	(1, 2)	(2, 3)	...	(m-1, 0)	(0, 1)	(1, 2)	...
(0, 2)	(1, 3)	(2, 4)	...	(m-1, 1)	(0, 2)	(1, 3)	...
(m-1, 2)	(0, 3)	(1, 4)	...	(m-2, 1)	(m-1, 2)	(0, 3)	...
(m-1, 3)	(0, 4)	(1, 5)	...	(m-2, 2)	(m-1, 3)	(0, 4)	...

and so on until $m - 1$ rows have been defined; the final row begins with $([m/2] + 1, [m/2])$ when $m > 3$. Condition (*) proves that this pattern is a frieze, namely that

$$(u, v)(u+1, v+1) = (u, v+1)(u+1, v) = 1, \quad (**)$$

because $\det L = \det R = 1$ implies $\det \alpha = 1$. Our example triangulation yields

05	E1	LD1 P0	1	<u>E1. Initialize.</u> $P \leftarrow P_0$.
06		ENT2 0	1	$T \leftarrow \Lambda$.
07		ENTX -1	1	$rX \leftarrow -1$.
08	E2	STX 0,1(MARK)	1	<u>E2. Mark.</u> $MARK(P) \leftarrow 1$.
09	E3	LDA 0,1(ATOM)	1	<u>E3. Atom?</u>
10		JAZ E4	1	Jump if $ATOM(P) = 0$.
11	E6	J2Z DONE	n	<u>E6. Up.</u>
12		ENT3 0,2	$n-1$	$Q \leftarrow T$.
13		LDA 0,3(ATOM)	$n-1$	
14		JANZ 1F	$n-1$	Jump if $ATOM(T) = 1$.
15		LD2 0,3(BLINK)	t_2	$T \leftarrow BLINK(Q)$.
16		ST1 0,3(BLINK)	t_2	$BLINK(Q) \leftarrow P$.
17		ENT1 0,3	t_2	$P \leftarrow Q$.
18		JMP E6	t_2	
19	1H	STZ 0,2(ATOM)	t_1	$ATOM(T) \leftarrow 0$.
20		LD2 0,3(ALINK)	t_1	$T \leftarrow ALINK(Q)$.
21		ST1 0,3(ALINK)	t_1	$ALINK(Q) \leftarrow P$.
22		ENT1 0,3	t_1	$P \leftarrow Q$.
23	E5	LD3 0,1(BLINK)	n	<u>E5. Down BLINK.</u> $Q \leftarrow BLINK(P)$.
24		J3Z E6	n	Jump if $Q = \Lambda$.
25		LDA 0,3	$n-b_2$	
26		STX 0,3(MARK)	$n-b_2$	$MARK(Q) \leftarrow 1$.
27		JANP E6	$n-b_2$	Jump if $NODE(Q)$ was already marked.
28		LDA 0,3(ATOM)	t_2+a_2	
29		JANZ E6	t_2+a_2	Jump if $ATOM(Q) = 1$.
30		ST2 0,1(BLINK)	t_2	$BLINK(P) \leftarrow T$.
31	E4A	ENT2 0,1	$n-1$	$T \leftarrow P$.
32		ENT1 0,3	$n-1$	$P \leftarrow Q$.
33	E4	LD3 0,1(ALINK)	n	<u>E4. Down ALINK.</u> $Q \leftarrow ALINK(P)$.
34		J3Z E5	n	Jump if $Q = \Lambda$.
35		LDA 0,3	$n-b_1$	
36		STX 0,3(MARK)	$n-b_1$	$MARK(Q) \leftarrow 1$.
37		JANP E5	$n-b_1$	Jump if $NODE(Q)$ was already marked.
38		LDA 0,3(ATOM)	t_1+a_1	
39		JANZ E5	t_1+a_1	Jump if $ATOM(Q) = 1$.
40		STX 0,1(ATOM)	t_1	$ATOM(P) \leftarrow 1$.
41		ST2 0,1(ALINK)	t_1	$ALINK(P) \leftarrow T$.
42		JMP E4A	t_1	$T \leftarrow P$, $P \leftarrow Q$, to E4. ■

By Kirchhoff's law, $t_1 + t_2 + 1 = n$. The total time is $(34n + 4t_1 + 3a - 5b - 8)u$, where n is the number of nonatomic nodes marked, a is the number of atoms marked, b is the number of Λ links encountered in marked nonatomic nodes, and t_1 is the number of times we went down an ALINK ($0 \leq t_1 < n$).

5. (The following is the fastest known marking algorithm for a one-level memory.)
 - S1. Set $MARK(P_0) \leftarrow 1$. If $ATOM(P_0) = 1$, the algorithm terminates; otherwise set $S \leftarrow 0$, $R \leftarrow P_0$, $T \leftarrow \Lambda$.
 - S2. Set $P \leftarrow BLINK(R)$. If $P = \Lambda$ or $MARK(P) = 1$, go to S3. Otherwise set $MARK(P) \leftarrow 1$. Now if $ATOM(P) = 1$, go to S3; otherwise if $S < N$ set $S \leftarrow S + 1$, $STACK[S] \leftarrow P$, and go to S3; otherwise go to S5.

12. When the **AVAIL** list contains N nodes, where N is a specified constant to be chosen as discussed below, initiate another coroutine that shares computer time with the main routine and does the following: (a) Marks all N nodes on the **AVAIL** list; (b) marks all other nodes that are accessible to the program; (c) links all unmarked nodes together to prepare a new **AVAIL** list for use when the current **AVAIL** list is empty, and (d) resets the mark bits in all nodes. One must choose N and the ratio of time sharing so that operations (a), (b), (c), and (d) are guaranteed to be complete before N nodes are taken from the **AVAIL** list, yet the main routine is running sufficiently fast. It is necessary to use some care in step (b) to make sure that all nodes "accessible to the program" are included, as the program continues to run; details are omitted here. If the list formed in (c) has fewer than N nodes, it may be necessary to stop eventually because memory space might become exhausted. [For further information, see Guy L. Steele Jr., *CACM* 18 (1975), 495–508; P. Wadler, *CACM* 19 (1976), 491–500; E. W. Dijkstra, L. Lamport, A. J. Martin, C. S. Scholten, and E. F. M. Steffens, *CACM* 21 (1978), 966–975; H. G. Baker, Jr., *CACM* 21 (1978), 280–294.]

SECTION 2.4

1. Preorder.
 2. It is essentially proportional to the number of Data Table entries created.
 3. Change step A5 to:
- A5'. [Remove top level.] Remove the top stack entry; and if the new level number at the top of the stack is $\geq L$, let (L_1, P_1) be the new entry at the top of the stack and repeat this step. Otherwise set $SIB(P_1) \leftarrow Q$ and then let (L_1, P_1) be the new entry at the top of the stack.
4. (Solution by David S. Wise.) Rule (c) is violated if and only if there is a data item whose *complete qualification* $A_0 \text{ OF } \dots \text{ OF } A_n$ is also a COBOL reference to some other data item. Since the parent $A_1 \text{ OF } \dots \text{ OF } A_n$ must also satisfy rule (c), we may assume that this other data item is a descendant of the same parent. Therefore Algorithm A would be extended to check, as each new data item is added to the Data Table, whether its parent is an ancestor of any other item of the same name, or if the parent of any other item of the same name is in the stack. (When the parent is Λ , it is everybody's ancestor and always on the stack.)

On the other hand, if we leave Algorithm A as it stands, the COBOL programmer will get an error message from Algorithm B when trying to use an illegal item. Only **MOVE CORRESPONDING** can make use of such items without error.

5. Make these changes:	Step	replace	by
	B1.	$P \leftarrow \text{LINK}(P_0)$	$P \leftarrow \text{LINK}(\text{INFO}(T))$
	B2.	$k \leftarrow 0$	$K \leftarrow T$
	B3.	$k < n$	$\text{RLINK}(K) \neq \Lambda$
	B4.	$k \leftarrow k + 1$	$K \leftarrow \text{RLINK}(K)$
	B6.	$\text{NAME}(S) = P_k$	$\text{NAME}(S) = \text{INFO}(K)$

6. A simple modification of Algorithm B makes it search only for complete references (if $k = n$ and $\text{PARENT}(S) \neq \Lambda$ in step B3, or if $\text{NAME}(S) \neq P_k$ in step B6, set $P \leftarrow \text{PREV}(P)$ and go to B2). The idea is to run through this modified Algorithm B first; then, if Q is still Λ , to perform the unmodified algorithm.

7. **MOVE MONTH OF DATE OF SALES TO MONTH OF DATE OF PURCHASES.** **MOVE DAY OF DATE OF SALES TO DAY OF DATE OF PURCHASES.** **MOVE YEAR OF DATE OF SALES TO YEAR OF**

A4. Set $\text{ROVER} \leftarrow \text{LINK}(P)$, $K \leftarrow \text{SIZE}(P) - N$. If $K < c$ (where c is a constant ≥ 2), set $\text{LINK}(\text{LINK}(P + 1)) \leftarrow \text{ROVER}$, $\text{LINK}(\text{ROVER} + 1) \leftarrow \text{LINK}(P + 1)$, $L \leftarrow P$; otherwise set $L \leftarrow P + K$, $\text{SIZE}(P) \leftarrow \text{SIZE}(L - 1) \leftarrow K$, $\text{TAG}(L - 1) \leftarrow "-"$, $\text{SIZE}(L) \leftarrow N$. Finally set $\text{TAG}(L) \leftarrow \text{TAG}(L + \text{SIZE}(L) - 1) \leftarrow "+"$. ■

13. $rI1 \equiv P$, $rX \equiv F$, $rI2 \equiv L$.

LINK	EQU	4:5	
SIZE	EQU	1:2	
TSIZE	EQU	0:2	
TAG	EQU	0:0	
A1	LDA	N	$rA \leftarrow N$.
	SLA	3	Shift into SIZE field.
	ENTX	0	$F \leftarrow 0$.
	LD1	ROVER	$P \leftarrow \text{ROVER}$.
	JMP	A2	
A3	CMPA	0,1(SIZE)	
	JLE	A4	Jump if $N \leq \text{SIZE}(P)$.
	LD1	0,1(LINK)	$P \leftarrow \text{LINK}(P)$.
A2	ENT2	-AVAIL,1	$rI2 \leftarrow P - \text{LOC}(\text{AVAIL})$.
	J2NZ	A3	
	JXNZ	OVERFLOW	Is $F \neq 0$?
	ENTX	1	Set $F \leftarrow 1$.
	LD1	AVAIL(LINK)	$P \leftarrow \text{AVAIL}$.
	JMP	A2	
A4	LD2	0,1(LINK)	
	ST2	ROVER	$\text{ROVER} \leftarrow \text{LINK}(P)$.
	LDA	0,1(SIZE)	$rA \equiv K \leftarrow \text{SIZE}(P) - N$.
	SUB	N	
	CMPA	=c=	
	JGE	1F	Jump if $K \geq c$.
	LD3	1,1(LINK)	$rI3 \leftarrow \text{LINK}(P + 1)$.
	ST2	0,3(LINK)	$\text{LINK}(rI3) \leftarrow \text{ROVER}$.
	ST3	1,2(LINK)	$\text{LINK}(\text{ROVER} + 1) \leftarrow rI3$.
	ENT2	0,1	$L \leftarrow P$.
	LD3	0,1(SIZE)	$rI3 \leftarrow \text{SIZE}(P)$.
	JMP	2F	
1H	STA	0,1(SIZE)	$\text{SIZE}(P) \leftarrow K$.
	LD2	0,1(SIZE)	
	INC2	0,1	$L \leftarrow P + K$.
	LDAN	0,1(SIZE)	$rA \leftarrow -K$.
	STA	-1,2(TSIZE)	$\text{SIZE}(L - 1) \leftarrow K$, $\text{TAG}(L - 1) \leftarrow "-"$.
	LD3	N	$rI3 \leftarrow N$.
2H	ST3	0,2(TSIZE)	$\text{TAG}(L) \leftarrow "+"$, also set $\text{SIZE}(L) \leftarrow rI3$.
	INC3	0,2	
	STZ	-1,3(TAG)	$\text{TAG}(L + \text{SIZE}(L) - 1) \leftarrow "+"$. ■

14. (a) This field is needed to locate the beginning of the block, in step C2. It could be replaced (perhaps to advantage) by a link to the first word of the block. See also exercise 19. (b) This field is needed because we sometimes need to reserve more than N words (for example if $K = 1$), and the amount reserved must be known when the block is subsequently freed.

15, 16. $r11 \equiv P_0$, $r12 \equiv P_1$, $r13 \equiv F$, $r14 \equiv B$, $r16 \equiv -N$.

D1	LD1 PO	<u>D1.</u>
	LD2 0,1(SIZE)	
	ENN6 0,2	$N \leftarrow \text{SIZE}(P_0)$.
	INC2 0,1	$P_1 \leftarrow P_0 + N$.
	LD5 0,2(TSIZE)	.
	J5N D4	To D4 if $\text{TAG}(P_1) = "-"$.
D2	LD5 -1,1(TSIZE)	<u>D2.</u>
	J5N D7	To D7 if $\text{TAG}(P_0 - 1) = "-"$.
D3	LD3 AVAIL(LINK)	<u>D3.</u> Set $F \leftarrow \text{AVAIL}$.
	ENT4 AVAIL	$B \leftarrow \text{LOC}(\text{AVAIL})$.
	JMP D5	To D5.
D4	INC6 0,5	<u>D4.</u> $N \leftarrow N + \text{SIZE}(P_1)$.
	LD3 0,2(LINK)	$F \leftarrow \text{LINK}(P_1)$.
	LD4 1,2(LINK)	$B \leftarrow \text{LINK}(P_1 + 1)$.
	CMP2 ROVER	(New code, because of the ROVER feature of exercise 12: If $P_1 = \text{ROVER}$, set $\text{ROVER} \leftarrow \text{LOC}(\text{AVAIL})$.)
	JNE *+3	
	ENTX AVAIL	
	STX ROVER	
	DEC2 0,5	$P_1 \leftarrow P_1 + \text{SIZE}(P_1)$.
	LD5 -1,1(TSIZE)	
	J5N D6	To D6 if $\text{TAG}(P_0 - 1) = "-"$.
D5	ST3 0,1(LINK)	<u>D5.</u> $\text{LINK}(P_0) \leftarrow F$.
	ST4 1,1(LINK)	$\text{LINK}(P_0 + 1) \leftarrow B$.
	ST1 1,3(LINK)	$\text{LINK}(F + 1) \leftarrow P_0$.
	ST1 0,4(LINK)	$\text{LINK}(B) \leftarrow P_0$.
	JMP D8	To D8.
D6	ST3 0,4(LINK)	<u>D6.</u> $\text{LINK}(B) \leftarrow F$.
	ST4 1,3(LINK)	$\text{LINK}(F + 1) \leftarrow B$.
D7	INC6 0,5	<u>D7.</u> $N \leftarrow N + \text{SIZE}(P_0 - 1)$.
	INC1 0,5	$P_0 \leftarrow P_0 - \text{SIZE}(P_0 - 1)$.
D8	ST6 0,1(TSIZE)	<u>D8.</u> $\text{SIZE}(P_0) \leftarrow N$, $\text{TAG}(P_0) \leftarrow "-"$.
	ST6 -1,2(TSIZE)	$\text{SIZE}(P_1 - 1) \leftarrow N$, $\text{TAG}(P_1 - 1) \leftarrow "-"$. ■

17. Both LINK fields equal to $\text{LOC}(\text{AVAIL})$.

18. Algorithm A reserves the upper end of a large block. When storage is completely available, the first-fit method actually begins by reserving the high-order locations, but once these become available again they are not re-reserved since a fit is usually found already in the lower locations; thus the initial large block at the lower end of memory quickly disappears with first-fit. A large block rarely is the best fit, however, so the best-fit method leaves a large block at the beginning of memory.

19. Use the algorithm of exercise 12, except delete the references to $\text{SIZE}(L - 1)$, $\text{TAG}(L - 1)$, and $\text{TAG}(L + \text{SIZE}(L) - 1)$ from step A4; also insert the following new step between steps A2 and A3:

A2½. Set $P_1 \leftarrow P + \text{SIZE}(P)$. If $\text{TAG}(P_1) = "+"$, proceed to step A3. Otherwise set $P_2 \leftarrow \text{LINK}(P_1)$, $\text{LINK}(P_2 + 1) \leftarrow \text{LINK}(P_1 + 1)$, $\text{LINK}(\text{LINK}(P_1 + 1)) \leftarrow P_2$, $\text{SIZE}(P) \leftarrow \text{SIZE}(P) + \text{SIZE}(P_1)$. If $\text{ROVER} = P_1$, set $\text{ROVER} \leftarrow P_2$. Repeat step A2½.

09	1H	ENT5 AVAIL,2	1 + R
10		DEC5 0,4	1 + R
11		J5NZ R2	1 + R Jump if AVAILF[j] ≠ LOC(AVAIL[j]).
12		INC2 1	R Increase j.
13		INC3 1	R
14		LD4N AVAIL,2(TLNKF)	R
15		J4NN 1B	R Is $j \leq m$?
16		JMP OVERFLOW	
17	R2	LD5 0,4(LINKF)	1 <u>R2. Remove from list.</u>
18		ST5 AVAIL,2(LINKF)	1 AVAILF[j] ← LINKF(L).
19		ENTA AVAIL,2	1
20		STA 0,5(LINKB)	1 LINKB(L) ← LOC(AVAIL[j]).
21		STZ 0,4(TAG)	1 TAG(L) ← 0.
22	R3	J3Z DONE	1 <u>R3. Split required?</u>
23	R4	DEC3 1	R <u>R4. Split.</u>
24		DEC2 1	R Decrease j.
25		LD5 TWO,2	R rI5 ≡ P.
26		INC5 0,4	R P ← L + 2^j .
27		ENNA AVAIL,2	R
28		STA 0,5(TLNKF)	R TAG(P) ← 1, LINKF(P) ← LOC(AVAIL[j]).
29		STA 0,5(LINKB)	R LINKB(P) ← LOC(AVAIL[j]).
30		ST5 AVAIL,2(LINKF)	R AVAILF[j] ← P.
31		ST5 AVAIL,2(LINKB)	R AVAILB[j] ← P.
32		ST2 0,5(KVAL)	R KVAL(P) ← j.
33		J3P R4	R Go to R3.
34	DONE	...	■

28. rI1 ≡ k, rI5 ≡ P, rI4 ≡ L; assume TAG(2^m) = "+".

01	S1	LD4 L	1 <u>S1. Is buddy available?</u>
02		LD1 K	1
03	1H	ENTA 0,4	1 + S
04		XOR TWO,1	1 + S rA ← buddy _k L.
05		STA TEMP	1 + S
06		LD5 TEMP	1 + S P ← rA.
07		LDA 0,5	1 + S
08		JANN S3	1 + S Jump if TAG(P) = 0.
09		CMP1 0,5(KVAL)	B + S
10		JNE S3	B + S Jump if KVAL(P) ≠ k.
11	S2	LD2 0,5(LINKF)	S <u>S2. Combine with buddy.</u>
12		LD3 0,5(LINKB)	S
13		ST3 0,2(LINKF)	S LINKF(LINKB(P)) ← LINKF(P).
14		ST2 0,3(LINKB)	S LINKB(LINKF(P)) ← LINKB(P).
15		INC1 1	S Increase k.
16		CMP4 TEMP	S
17		JL 1B	S
18		ENT4 0,5	A If L > P, set L ← P.
19		JMP 1B	A
20	S3	LD2 AVAIL,1(LINKF)	1 <u>S3. Put on list.</u>
21		ENNA AVAIL,1	1
22		STA 0,4(0:4)	1 TAG(L) ← 1, LINKB(L) ← LOC(AVAIL[k]).

16 (1976), 426–441; R. B. K. Dewar and A. P. McCann, *Software Practice & Exp.* 7 (1977), 95–113; F. Lockwood Morris, *CACM* 21 (1978), 662–665, 22 (1979), 571; H. B. M. Jonkers, *Inf. Proc. Letters* 9 (1979), 26–30; J. J. Martin, *CACM* 25 (1982), 571–581; F. Lockwood Morris, *Inf. Proc. Letters* 15 (1982), 139–142, 16 (1983), 215. Other methods have been published by B. K. Haddon and W. M. Waite, *Comp. J.* 10 (1967), 162–165; B. Wegbreit, *Comp. J.* 15 (1972), 204–208; D. A. Zave, *Inf. Proc. Letters* 3 (1975), 167–169. Cohen and Nicolau have analyzed four of these approaches in *ACM Trans. Prog. Languages and Systems* 5 (1983), 532–553.]

34. Let $\text{TOP} \equiv rI1$, $\text{Q} \equiv rI2$, $\text{P} \equiv rI3$, $k \equiv rI4$, $\text{SIZE(P)} \equiv rI5$. Assume further that $\Lambda = 0$, and $\text{LINK}(0) \neq 0$ to simplify step G4. Step G1 is omitted.

01	LINK EQU	4:5		
02	INFO EQU	0:3		
03	SIZE EQU	1:2		
04	T EQU	3:3		
05	G2 LD1 USE	1	<u>G2. Initialize marking phase.</u>	$\text{TOP} \leftarrow \text{USE}$.
06	LD2 AVAIL	1		
07	ST2 0,1(LINK)	1		$\text{LINK}(\text{TOP}) \leftarrow \text{AVAIL}$.
08	STZ 0,2(LINK)	1		$\text{LINK}(\text{AVAIL}) \leftarrow \Lambda$.
09	G3 ENT3 0,1	a + 1	<u>G3. Pop up stack.</u>	$\text{P} \leftarrow \text{TOP}$.
10	LD1 0,1(LINK)	a + 1		$\text{TOP} \leftarrow \text{LINK}(\text{TOP})$.
11	J1Z G5	a + 1		To G5 if $\text{TOP} = \Lambda$.
12	G4 LD4 0,3(T)	a	<u>G4. Put new links on stack.</u>	$k \leftarrow \text{T}(\text{P})$.
13	1H J4Z G3	a + b		$k = 0?$
14	INC3 1	b		$\text{P} \leftarrow \text{P} + 1$.
15	DEC4 1	b		$k \leftarrow k - 1$.
16	LD2 0,3(LINK)	b		$\text{Q} \leftarrow \text{LINK}(\text{P})$.
17	LDA 0,2(LINK)	b		
18	JANZ 1B	b		Jump if $\text{LINK}(\text{Q}) \neq \Lambda$.
19	ST1 0,2(LINK)	a - 1		Otherwise set $\text{LINK}(\text{Q}) \leftarrow \text{TOP}$,
20	ENT1 0,2	a - 1		$\text{TOP} \leftarrow \text{Q}$.
21	JMP 1B	a - 1		
22	G5 ENT2 1	1	<u>G5. Initialize next phase.</u>	$\text{Q} \leftarrow 1$.
23	ST2 0,3	1		$\text{LINK}(\text{AVAIL}) \leftarrow 1$, $\text{SIZE}(\text{AVAIL}) \leftarrow 0$.
24	ENT3 1	1		$\text{P} \leftarrow 1$.
25	JMP G6	1		
26	1H ST2 0,3(LINK)	a		$\text{LINK}(\text{P}) \leftarrow \text{Q}$.
27	INC2 0,5	a		$\text{Q} \leftarrow \text{Q} + \text{SIZE}(\text{P})$.
28	INC3 0,5	a		$\text{P} \leftarrow \text{P} + \text{SIZE}(\text{P})$.
29	G6 LDA 0,3(LINK)	a + 1	<u>G6. Assign new addresses.</u>	
30	G6A LD5 0,3(SIZE)	a + c + 1		
31	JAZ G7	a + c + 1		Jump if $\text{LINK}(\text{P}) = \Lambda$.
32	J5NZ 1B	a + 1		Jump if $\text{SIZE}(\text{P}) \neq 0$.
33	G8 LD1 USE	1	<u>G8. Translate all links.</u>	
34	LDA 0,1(LINK)	1		
35	STA USE	1		$\text{USE} \leftarrow \text{LINK}(\text{USE})$.
36	ST2 AVAIL	1		$\text{AVAIL} \leftarrow \text{Q}$.
37	ENT3 1	1		$\text{P} \leftarrow 1$.
38	JMP G8P	1		

39	1H	LD6 0,6(SIZE)	<i>d</i>	
40		INC5 0,6	<i>d</i>	$rI5 \leftarrow rI5 + \text{SIZE}(P + \text{SIZE}(P))$.
41	G7	ENT6 0,3	<i>c+d</i>	<u>G7. Collapse available areas.</u>
42		INC6 0,5	<i>c+d</i>	$rI6 \leftarrow P + \text{SIZE}(P)$.
43		LDA 0,6(LINK)	<i>c+d</i>	
44		JAZ 1B	<i>c+d</i>	Jump if $\text{LINK}(rI6) \equiv \Lambda$.
45		ST5 0,3(SIZE)	<i>c</i>	$\text{SIZE}(P) \leftarrow rI5$.
46		INC3 0,5	<i>c</i>	$P \leftarrow P + \text{SIZE}(P)$.
47		JMP G6A	<i>c</i>	
48	2H	DEC4 1	<i>b</i>	$k \leftarrow k - 1$.
49		INC2 1	<i>b</i>	$Q \leftarrow Q + 1$.
50		LD6 0,2(LINK)	<i>b</i>	
51		LDA 0,6(LINK)	<i>b</i>	
52		STA 0,2(LINK)	<i>b</i>	$\text{LINK}(Q) \leftarrow \text{LINK}(\text{LINK}(Q))$.
53	1H	J4NZ 2B	<i>a+b</i>	Jump if $k \neq 0$.
54	3H	INC3 0,5	<i>a+c</i>	$P \leftarrow P + \text{SIZE}(P)$.
55	G8P	LDA 0,3(LINK)	$1+a+c$	
56		LD5 0,3(SIZE)	$1+a+c$	
57		JAZ 3B	$1+a+c$	Is $\text{LINK}(P) = \Lambda$?
58		LD4 0,3(T)	$1+a$	$k \leftarrow T(P)$.
59		ENT2 0,3	$1+a$	$Q \leftarrow P$.
60		J5NZ 1B	$1+a$	Jump unless $\text{SIZE}(P) = 0$.
61	G9	ENT3 1	1	<u>G9. Move.</u> $P \leftarrow 1$.
62		ENT1 1	1	Set $rI1$ for MOVE instructions.
63		JMP G9P	1	
64	1H	STZ 0,3(LINK)	a	$\text{LINK}(P) \leftarrow \Lambda$.
65		ST5 **+1(4:4)	a	
66		MOVE 0,3(*)	a	$\text{NODE}(rI1) \leftarrow \text{NODE}(P)$, $rI1 \leftarrow rI1 + \text{SIZE}(P)$.
67	3H	INC3 0,5	$a+c$	$P \leftarrow P + \text{SIZE}(P)$.
68	G9P	LDA 0,3(LINK)	$1+a+c$	
69		LD5 0,3(SIZE)	$1+a+c$	
70		JAZ 3B	$1+a+c$	Jump if $\text{LINK}(P) = \Lambda$.
71		J5NZ 1B	$1+a$	Jump unless $\text{SIZE}(P) = 0$. ■

In line 66 we are assuming that the size of each node is sufficiently small that it can be moved with a single MOVE instruction; this seems a fair assumption for most cases when this kind of garbage collection is applicable.

The total running time for this program is $(44a + 17b + 2w + 25c + 8d + 47)u$, where a is the number of accessible nodes, b is the number of link fields therein, c is the number of inaccessible nodes that are *not* preceded by an inaccessible node, d is the number of inaccessible nodes that are preceded by an inaccessible node, and w is the total number of words in the accessible nodes. If the memory contains n nodes, with ρn of them inaccessible, then we may estimate $a = (1 - \rho)n$, $c = (1 - \rho)\rho n$, $d = \rho^2 n$. Example: five-word nodes (on the average), with two link fields per node (on the average), and a memory of 1000 nodes. Then when $\rho = \frac{1}{5}$, it takes 374u per available node recovered; when $\rho = \frac{1}{2}$, it takes 104u; and when $\rho = \frac{4}{5}$, it takes only 33u.

36. A single customer will be able to sit in one of the sixteen seats 1, 3, 4, 6, ..., 23. If a pair enters, there must be room for them; otherwise there are at least two people in seats (1, 2, 3), at least two in (4, 5, 6), ..., at least two in (19, 20, 21), and at least one in 22 or 23, so at least fifteen people are already seated.

Conversely for $r \leq 5$ it can be shown that a buddy system sometimes requires as many as $a_r n$ cells, if the mechanism of steps R1 and R2 is modified to choose the worst possible available 2^j -block to split instead of the first such block.

Robson's proof that $N(2^r) \leq 1 + r$ (see exercise 40) is easily modified to show that such a "leftmost" strategy will never need more than $(1 + \frac{1}{2}r)n$ cells to allocate space for blocks of sizes 1, 2, 4, ..., 2^r , since blocks of size 2^k will never be placed in locations $\geq (1 + \frac{1}{2}k)n$. Although his algorithm seems very much like the buddy system, it turns out that no buddy system will be this good, even if we modify steps R1 and R2 to choose the best possible available 2^j -block to split. For example, consider the following sequence of "snapshots" of the memory, for $n = 16$ and $r = 3$:

11111111	11111111	00000000	00000000
10101010	10101010	2-2-2-2-	00000000
11110000	11110000	2-110000	00000000
11111111	11110000	11110000	00000000
10101010	10102-2-	10102-2-	00000000
10001000	10002-00	10002-00	4---4---
10000000	10000000	10000000	4---0000

Here 0 denotes an available location and k denotes the beginning of a k -block. In a similar way there is a sequence of operations, whenever n is a multiple of 16, that forces $\frac{3}{16}n$ blocks of size 8 to be $\frac{1}{8}$ full, and another $\frac{1}{16}n$ to be $\frac{1}{2}$ full. If n is a multiple of 128, a subsequent request for $\frac{9}{128}n$ blocks of size 8 will require more than $2.5n$ memory cells. (The buddy system allows unwanted 1s to creep into $\frac{3}{16}n$ of the 8-blocks, since there are no other available 2s to be split at a crucial time; the "leftmost" algorithm keeps all 1s confined.)

42. We can assume that $m \geq 6$. The main idea is to establish the occupancy pattern $R_{m-2}(F_{m-3}R_1)^k$ at the beginning of the memory, for $k = 0, 1, \dots$, where R_j and F_j denote reserved and free blocks of size j . The transition from k to $k+1$ begins with

$$\begin{aligned} R_{m-2}(F_{m-3}R_1)^k &\rightarrow R_{m-2}(F_{m-3}R_1)^k R_{m-2} R_{m-2} \\ &\rightarrow R_{m-2}(F_{m-3}R_1)^{k-1} F_{2m-4} R_{m-2} \\ &\rightarrow R_{m-2}(F_{m-3}R_1)^{k-1} R_m R_{m-5} R_1 R_{m-2} \\ &\rightarrow R_{m-2}(F_{m-3}R_1)^{k-1} F_m R_{m-5} R_1; \end{aligned}$$

then the commutation sequence $F_{m-3}R_1 F_m R_{m-5} R_1 \rightarrow F_{m-3}R_1 R_{m-2} R_2 R_{m-5} R_1 \rightarrow F_{2m-4} R_2 R_{m-5} R_1 \rightarrow R_m R_{m-5} R_1 R_2 R_{m-5} R_1 \rightarrow F_m R_{m-5} R_1 F_{m-3} R_1$ is used k times until we get $F_m R_{m-5} R_1 (F_{m-3}R_1)^k \rightarrow F_{2m-5} R_1 (F_{m-3}R_1)^k \rightarrow R_{m-2}(F_{m-3}R_1)^{k+1}$. Finally, when k gets large enough, there is an endgame that forces overflow unless the memory size is at least $(n - 4m + 11)(m - 2)$; details appear in *Comp. J.* 20 (1977), 242–244. [Notice that the worst conceivable worst case, which begins with the pattern $F_{m-1}R_1 F_{m-1}R_1 F_{m-1}R_1 \dots$, is only slightly worse than this; the next-fit strategy of exercise 6 can produce this pessimal pattern.]

43. We will show that if D_1, D_2, \dots is any sequence of numbers such that $D_1/m + D_2/(m+1) + \dots + D_m/(2m-1) \geq 1$ for all $m \geq 1$, and if $C_m = D_1/1 + D_2/2 + \dots + D_m/m$, then $N_{PF}(n, m) \leq nC_m$. In particular, since

$$\frac{1}{m} + \frac{1}{m+1} + \dots + \frac{1}{2m-1} = 1 - \frac{1}{2} + \dots + \frac{1}{2m-3} - \frac{1}{2m-2} + \frac{1}{2m-1} > \ln 2,$$

the constant sequence $D_m = 1/\ln 2$ satisfies the necessary conditions. The proof is by induction on m . Let $N_j = nC_j$ for $j \geq 1$, and suppose that some request for a block of size m cannot be allocated in the leftmost N_m cells of memory. Then $m > 1$. For $0 \leq j < m$, we let N'_j denote the rightmost position allocated to blocks of sizes $\leq j$, or 0 if all reserved blocks are larger than j ; by induction we have $N'_j \leq N_j$. Furthermore we let N'_m be the rightmost occupied position $\leq N_m$, so that $N'_m \geq N_m - m + 1$. Then the interval $(N'_{j-1} \dots N'_j]$ contains at least $[j(N'_j - N'_{j-1})/(m+j-1)]$ occupied cells, since its free blocks are of size $< m$ and its reserved blocks are of size $\geq j$. It follows that $n - m \geq$ number of occupied cells $\geq \sum_{j=1}^m j(N'_j - N'_{j-1})/(m+j-1) = mN'_m/(2m-1) - (m-1)\sum_{j=1}^{m-1} N'_j/(m+j)(m+j-1) > mN_m/(2m-1) - m - (m-1)\sum_{j=1}^{m-1} N_j(1/(m+j-1) - 1/(m+j)) = \sum_{j=1}^m nD_j/(m+j-1) - m \geq n - m$, a contradiction.

[This proof establishes slightly more than was asked. If we define the D 's by $D_1/m + \dots + D_m/(2m-1) = 1$, then the sequence C_1, C_2, \dots is $1, \frac{7}{4}, \frac{161}{72}, \frac{7483}{2880}, \dots$; and the result can be improved further, even in the case $m = 2$, as in exercise 38.]

44. $[F^{-1}(1/N)], [F^{-1}(2/N)], \dots, [F^{-1}(N/N)]$.

APPENDIX A

TABLES OF NUMERICAL QUANTITIES

Table 1

QUANTITIES THAT ARE FREQUENTLY USED IN STANDARD SUBROUTINES
AND IN ANALYSIS OF COMPUTER PROGRAMS (40 DECIMAL PLACES)

$\sqrt{2}$ = 1.41421 35623 73095 04880 16887 24209 69807 85697-
$\sqrt{3}$ = 1.73205 08075 68877 29352 74463 41505 87236 69428+
$\sqrt{5}$ = 2.23606 79774 99789 69640 91736 68731 27623 54406+
$\sqrt{10}$ = 3.16227 76601 68379 33199 88935 44432 71853 37196-
$\sqrt[3]{2}$ = 1.25992 10498 94873 16476 72106 07278 22835 05703-
$\sqrt[3]{3}$ = 1.44224 95703 07408 38232 16383 10780 10958 83919-
$\sqrt[4]{2}$ = 1.18920 71150 02721 06671 74999 70560 47591 52930-
$\ln 2$ = 0.69314 71805 59945 30941 72321 21458 17656 80755+
$\ln 3$ = 1.09861 22886 68109 69139 52452 36922 52570 46475-
$\ln 10$ = 2.30258 50929 94045 68401 79914 54684 36420 76011+
$1/\ln 2$ = 1.44269 50408 88963 40735 99246 81001 89213 74266+
$1/\ln 10$ = 0.43429 44819 03251 82765 11289 18916 60508 22944-
π = 3.14159 26535 89793 23846 26433 83279 50288 41972-
1° = $\pi/180$ = 0.01745 32925 19943 29576 92369 07684 88612 71344+
$1/\pi$ = 0.31830 98861 83790 67153 77675 26745 02872 40689+
π^2 = 9.86960 44010 89358 61883 44909 99876 15113 53137-
$\sqrt{\pi} = \Gamma(1/2)$ = 1.77245 38509 05516 02729 81674 83341 14518 27975+
$\Gamma(1/3)$ = 2.67893 85347 07747 63365 56929 40974 67764 41287-
$\Gamma(2/3)$ = 1.35411 79394 26400 41694 52880 28154 51378 55193+
e = 2.71828 18284 59045 23536 02874 71352 66249 77572+
$1/e$ = 0.36787 94411 71442 32159 55237 70161 46086 74458+
e^2 = 7.38905 60989 30650 22723 04274 60575 00781 31803+
γ = 0.57721 56649 01532 86060 65120 90082 40243 10422-
$\ln \pi$ = 1.14472 98858 49400 17414 34273 51353 05871 16473-
ϕ = 1.61803 39887 49894 84820 45868 34365 63811 77203+
e^γ = 1.78107 24179 90197 98523 65041 03107 17954 91696+
$e^{\pi/4}$ = 2.19328 00507 38015 45655 97696 59278 73822 34616+
$\sin 1$ = 0.84147 09848 07896 50665 25023 21630 29899 96226-
$\cos 1$ = 0.54030 23058 68139 71740 09366 07442 97660 37323+
$-\zeta'(2)$ = 0.93754 82543 15843 75370 25740 94567 86497 78979-
$\zeta(3)$ = 1.20205 69031 59594 28539 97381 61511 44999 07650-
$\ln \phi$ = 0.48121 18250 59603 44749 77589 13424 36842 31352-
$1/\ln \phi$ = 2.07808 69212 35027 53760 13226 06117 79576 77422-
$-\ln \ln 2$ = 0.36651 29205 81664 32701 24391 58232 66946 94543-

Formal symbolism	Meaning	Where defined
$(B \Rightarrow E; E')$	conditional expression: denotes E if B is true, E' if B is false	
$[B]$	characteristic function of condition B : $(B \Rightarrow 1; 0)$	1.2.3
δ_{kj}	Kronecker delta: $[j = k]$	1.2.6
$[z^n] g(z)$	coefficient of z^n in power series $g(z)$	1.2.9
$\sum_{R(k)} f(k)$	sum of all $f(k)$ such that the variable k is an integer and relation $R(k)$ is true	1.2.3
$\prod_{R(k)} f(k)$	product of all $f(k)$ such that the variable k is an integer and relation $R(k)$ is true	1.2.3
$\min_{R(k)} f(k)$	minimum value of all $f(k)$ such that the variable k is an integer and relation $R(k)$ is true	1.2.3
$\max_{R(k)} f(k)$	maximum value of all $f(k)$ such that the variable k is an integer and relation $R(k)$ is true	1.2.3
$j \mid k$	j divides k : $k \bmod j = 0$ and $j > 0$	1.2.4
$S \setminus T$	set difference: $\{a \mid a \text{ in } S \text{ and } a \text{ not in } T\}$	
$\gcd(j, k)$	greatest common divisor of j and k : $(j = k = 0 \Rightarrow 0; \max_{d \mid j, d \mid k} d)$	1.1
$j \perp k$	j is relatively prime to k : $\gcd(j, k) = 1$	1.2.4
A^T	transpose of rectangular array A : $A^T[j, k] = A[k, j]$	1.2.3
α^R	left-right reversal of α	
x^y	x to the y power (when x is positive)	1.2.2
x^k	x to the k th power: $(k \geq 0 \Rightarrow \prod_{0 \leq j < k} x; 1/x^{-k})$	1.2.2
$x^{\bar{k}}$	x to the k rising: $\Gamma(x + k)/\Gamma(x) =$ $(k \geq 0 \Rightarrow \prod_{0 \leq j < k} (x + j); 1/(x + k)^{-k})$	1.2.5
$x^{\underline{k}}$	x to the k falling: $x!/(x - k)! =$ $(k \geq 0 \Rightarrow \prod_{0 \leq j < k} (x - j); 1/(x - k)^{-k})$	1.2.5

Formal symbolism	Meaning	Where defined
$n!$	n factorial: $\Gamma(n+1) = n^n$	1.2.5
$\binom{x}{k}$	binomial coefficient: ($k < 0 \Rightarrow 0$; $x^k/k!$)	1.2.6
$\binom{n}{n_1, n_2, \dots, n_m}$	multinomial coefficient (defined only when $n = n_1 + n_2 + \dots + n_m$)	1.2.6
$\left[\begin{smallmatrix} n \\ m \end{smallmatrix} \right]$	Stirling number of the first kind: $\sum_{0 < k_1 < k_2 < \dots < k_{n-m} < n} k_1 k_2 \dots k_{n-m}$	1.2.6
$\left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\}$	Stirling number of the second kind: $\sum_{1 \leq k_1 \leq k_2 \leq \dots \leq k_{n-m} \leq m} k_1 k_2 \dots k_{n-m}$	1.2.6
$\{a \mid R(a)\}$	set of all a such that the relation $R(a)$ is true	
$\{a_1, \dots, a_n\}$	the set or multiset $\{a_k \mid 1 \leq k \leq n\}$	
$\{x\}$	fractional part (used in contexts where a real value, not a set, is implied): $x - [x]$	1.2.11.2
$[a..b]$	closed interval: $\{x \mid a \leq x \leq b\}$	1.2.2
$(a..b)$	open interval: $\{x \mid a < x < b\}$	1.2.2
$[a..b)$	half-open interval: $\{x \mid a \leq x < b\}$	1.2.2
$(a..b]$	half-closed interval: $\{x \mid a < x \leq b\}$	1.2.2
$ S $	cardinality: the number of elements in set S	
$ x $	absolute value of x : ($x \geq 0 \Rightarrow x$; $-x$)	
$ \alpha $	length of α	
$[x]$	floor of x , greatest integer function: $\max_{k \leq x} k$	1.2.4
$[x]$	ceiling of x , least integer function: $\min_{k \geq x} k$	1.2.4
$x \bmod y$	mod function: ($y = 0 \Rightarrow x$; $x - y[x/y]$)	1.2.4
$x \equiv x' \pmod{y}$	relation of congruence: $x \bmod y = x' \bmod y$	1.2.4
$O(f(n))$	big-oh of $f(n)$, as the variable $n \rightarrow \infty$	1.2.11.1
$O(f(z))$	big-oh of $f(z)$, as the variable $z \rightarrow 0$	1.2.11.1
$\Omega(f(n))$	big-omega of $f(n)$, as the variable $n \rightarrow \infty$	1.2.11.1
$\Theta(f(n))$	big-theta of $f(n)$, as the variable $n \rightarrow \infty$	1.2.11.1

Formal symbolism	Meaning	Where defined
$\log_b x$	logarithm, base b , of x (when $x > 0$, $b > 0$, and $b \neq 1$): the y such that $x = b^y$	1.2.2
$\ln x$	natural logarithm: $\log_e x$	1.2.2
$\lg x$	binary logarithm: $\log_2 x$	1.2.2
$\exp x$	exponential of x : e^x	1.2.2
(X_n)	the infinite sequence X_0, X_1, X_2, \dots (here the letter n is part of the symbolism)	1.2.9
$f'(x)$	derivative of f at x	1.2.9
$f''(x)$	second derivative of f at x	1.2.10
$f^{(n)}(x)$	n th derivative: $(n = 0 \Rightarrow f(x); g'(x))$, where $g(x) = f^{(n-1)}(x)$	1.2.11.2
$H_n^{(x)}$	harmonic number of order x : $\sum_{1 \leq k \leq n} 1/k^x$	1.2.7
H_n	harmonic number: $H_n^{(1)}$	1.2.7
F_n	Fibonacci number: $(n \leq 1 \Rightarrow n; F_{n-1} + F_{n-2})$	1.2.8
B_n	Bernoulli number: $n! [z^n] z/(e^z - 1)$	1.2.11.2
$\det(A)$	determinant of square matrix A	1.2.3
$\text{sign}(x)$	sign of x : $[x > 0] - [x < 0]$	
$\zeta(x)$	zeta function: $\lim_{n \rightarrow \infty} H_n^{(x)}$ (when $x > 1$)	1.2.7
$\Gamma(x)$	gamma function: $(x - 1)! = \gamma(x, \infty)$	1.2.5
$\gamma(x, y)$	incomplete gamma function: $\int_0^y e^{-t} t^{x-1} dt$	1.2.11.3
γ	Euler's constant: $\lim_{n \rightarrow \infty} (H_n - \ln n)$	1.2.7
e	base of natural logarithms: $\sum_{n \geq 0} 1/n!$	1.2.2
π	circle ratio: $4 \sum_{n \geq 0} (-1)^n / (2n + 1)$	
∞	infinity: larger than any number	
Λ	null link (pointer to no address)	2.1
ϵ	empty string (string of length zero)	
\emptyset	empty set (set with no elements)	
ϕ	golden ratio: $\frac{1}{2}(1 + \sqrt{5})$	1.2.8
$\varphi(n)$	Euler's totient function: $\sum_{0 \leq k < n} [k \perp n]$	1.2.4
$x \approx y$	x is approximately equal to y	1.2.5, 4.2.2

INDEX AND GLOSSARY

*Some Men pretend to understand a Book
by scouting thro' the Index:*

*as if a Traveller should go about to describe a Palace
when he had seen nothing but the Privy.*

— JONATHAN SWIFT, *Mechanical Operation of the Spirit* (1704)

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.

- (), 164, *see* Identity permutation.
0-2-trees, 317.
 oriented, 398.
0-origin indexing, 254, 282, 299–301,
 305–306.
2-D trees, 564.
 π (circle ratio), 21, 619–620.
 Wallis's product for, 52, 116.
 ϕ (golden ratio), 13, 18, 21, 80, 83–86,
 619–620.
- A-register of MIX, 125.
Aardenne-Ehrenfest, Tatyana van, 375, 379.
Aarons, Roger M., 528.
Abel, Niels Henrik, 58, 498.
 binomial theorem, 71–73, 398.
 limit theorem, 95.
Absolute error, 116.
Absolute value, 21.
Absolutely convergent series, 29.
ACE computer, 193, 229.
 Pilot, 230.
Adams, Charles William, 230.
ADD, 131–132, 208.
Add to list: *see* Insertion.
Addition of polynomials, 275–280, 357–359.
Address: A number used to identify a
 position in memory.
 field of MIXAL line, 145, 151–153.
 of node, 233.
 portion of MIX instruction, 127.
Address transfer operators of MIX,
 133–134, 210.
Adjacent vertices of a graph, 363.
Adobe Systems, 202.
Agenda, 288, 291, 296, *see* Priority queue.
Aho, Alfred Vaino, 560.
Ahrens, Wilhelm Ernst Martin Georg, 162.
al-Khwārizmī, Abū 'Abd Allah
 Muhammad ibn Mūsā
 (أبو عبد الله محمد بن موسى الخوارزمي),
 1, 79.
ALF (alphabetic data), 151, 152, 155.
- Algebraic formulas, 313.
differentiation, 90, 338–347, 459.
manipulation of, 459–462.
representation as trees, 337, 459.
simplification of, 339, 347.
ALGOL language, 202, 229.
Algorithm, origin of word, 1–2.
Algorithms, 1–9.
 analysis of, vi, 7, 96–107, 170–172, 179,
 250, 253, 268, 278–280, 324–326, 331,
 380–382, 444–445, 451.
 communication of, 16.
 effective, 6, 8, 9.
 equivalence between, 467.
 form of in this book, 2–4.
 hardware-oriented, 26, 252, 611.
 how to read, 4, 16.
 proof of, 5–6, 13–17, 321, 361, 422, 434.
 properties of, 4–6, 9.
 random paths in, 380–381.
 set-theoretic definition, 7–9.
 theory of, 7, 9.
Allocation of tables, *see* Dynamic storage
 allocation, Linked allocation,
 Representation, Sequential allocation.
Alpern, Steven Robert, 526.
Alphabetic character: A letter, digit, or
 special character symbol.
codes for MIX, 136, 138, 140–141.
AMM: *American Mathematical Monthly*,
 published by the Mathematical
 Association of America since 1894.
Amortized running time, 254.
Analysis of algorithms, vi, 7, 96–107,
 170–172, 179, 250, 253, 268, 278–280,
 324–326, 331, 380–382, 444–445, 451.
Analytical Engine, 1, 229.
Ancestor, in a tree structure, 311, 348.
André, Antoine Désiré, 536.
Anticipated input, 216, *see* Buffering.
Antisymmetric relation, 261.
Apostol, Tom Mike, 28.
Araújo, Saulo, 565.
Arbogast, Louis François Antoine, 52, 105.
Arborescences, 363, *see* Oriented trees.

- Arc digraph, 379.
 Arc in a directed graph, 372.
 Area of memory, 435.
 Arguments of subroutines, 187, 189.
 Arithmetic: Addition, subtraction,
 multiplication, and division, vi, ix.
 fixed point, 158.
 floating point, 131, 306.
 operators of MIX, 131–133, 208.
 polynomial, 275–280, 357–359, 361.
 scaled decimal, 160–161.
 Arithmetic expressions, *see* Algebraic formulas.
 Arithmetic progression, sum of, 11, 31–32, 56.
 Array: A table that usually has a k -dimensional rectangular structure, 4, 232, 298–307.
 linked, 301–307.
 one-dimensional, *see* Linear list.
 represented as a tree, 315.
 sequential allocation, 159, 299–301, 305–307.
 tetrahedral, 300–301, 306.
 two-dimensional, *see* Matrix.
 uninitialized, 307.
 Arrows, used to represent links in diagrams, 234.
 Assembly language: A language that is intended to facilitate the construction of programs in machine language by making use of symbolic and mnemonic conventions to denote machine language instructions.
 contrasted with high-level language, 236, 535.
 for MIX, 144–157.
 Assembly program, 145, 153.
 Assigning a buffer, 219–223, 226–227.
 Assignment operation (\leftarrow), 3.
 Asterisk (“*”) in assembly language, 146, 149, 153.
 Asymmetric relations, 261.
 Asymptotic values: Functions that express the limiting behavior approached by numerical quantities.
 derivation of, 107–123, 243, 396–397, 525, 565.
 Atom (in a List), 315, 408–412, 418.
 purpose of, 410.
 Automata theory, 230, 240, 463–464.
 Automaton: An abstract machine that is defined formally, often intended to be a model of some aspects of actual computers (plural: Automata), 463–464.
 AVAIL stack: Available space list, 256.
 Available space list, 256–261, 266, 269, 278, 291, 413–414, 435–444.
 history, 458.
 variable-size blocks, 435–456.
 Average value of a random variable, 97–98, 103.
 from a generating function, 100–103.
 Babbage, Charles, 1, 229.
 Bachmann, Paul Gustav Heinrich, 107.
 Backus, John Warner, 230.
 Bailey, Michael John, 462.
 Baker, Henry Givens, Jr., 605.
 Balanced directed graph, 374–375.
 Ball, Thomas Jaudon, 369.
 Ball, Walter William Rouse, 162.
 Ballot problem, 536–537.
 Barnes, Ernest William, 507.
 Barnett, Michael Peter, 462.
 Barrington, David Arno, 526.
 Barry, David McAlister, subtle reference to, xi, 274.
 Barton, David Elliott, 66, 536.
 Base address, 244.
 Bead, 233, *see* Node.
 Before and after diagrams, 260–261, 278, 281, 571.
 Beigel, Richard, 467.
 Bell, Eric Temple, 87.
 Bell Interpretive System, 230.
 Bellman, Richard Ernest, xv.
 Bendix G20, 124.
 Bennett, John Makepeace, 230.
 Berger, Robert, 385.
 Bergeron, François, 395.
 Bergman, George Mark, 495, 593.
 Berlekamp, Elwyn Ralph, 273.
 Berman, Martin Fredric, 523.
 Bernoulli, Jacques (= Jakob = James), 112, 115.
 numbers B_n , 76, 91, 112–115.
 numbers, table, 621.
 polynomials, 44, 113–115, 503.
 Bernoulli, Jean (= Johann = John), III, 495.
 Bertrand, Joseph Louis François, postulate, 512.
 Beratiss, Alfs Teodors, 462.
 Best-fit method of storage allocation, 436–437, 447, 453–456.
 Beta function $B(x, y)$, 72.
 Bhāskara Āchārya (भास्कर आचार्य), 53–54.
 Bienaymé, Irénée Jules, 98.
 Bienstock, Daniel, 514.
 Big-Oh notation, 107–111, 118.
 Big-Omega notation, 110–111.
 Big-Theta notation, 110.
 Bigelow, Richard Henry, 562.
 Binary computer: A computer that manipulates numbers primarily in the binary (radix 2) number system.
 Binary logarithms, 23, 26.
 Binary number system, 24–26.

- Binary trees, 311–312, 317, 318–337, 363, 459.
 complete, 401, 405, 563.
 copying of, 329–330, 333, 347.
 correspondence to trees and forests, 334–335, 346.
 definition of, 312.
 “Dewey” notation for, 317, 331, 346, 405.
 enumeration of, 388–389, 405.
 equivalent, 328–329.
 erasing of, 333.
 extended, 399–406.
 linked, 318.
 oriented, 396–397.
 path length of, 399–406.
 representation of, 318, 322, 327, 333–334.
 right-threaded, 327, 332–334.
 sequential, 401.
 similar, 327–329.
 threaded, 322, 331–332, 460.
 traversal of, 319–320, 323, 331–332, 459–460.
 with shared subtrees, 326, 603.
- Binet, Jacques Philippe Marie, 36, 407, 475, 582.
- Binomial coefficients, 52–74, 89.
 asymptotic values, 72.
 bounds on, 74.
 combinatorial interpretation, 52–53, 73.
 defined, 53.
 generalized, 65, 72, 85.
 generating functions, 90, 94.
 history, 53–54.
 sums involving, 56–74, 76–78, 85, 96.
 table, 54.
- Binomial distribution, 101–102.
 tail, 106.
- Binomial theorem, 57–58, 90.
 Abel’s generalization, 71–73, 398.
 generalizations of, 64, 70–73, 90, 398–399, 488.
 Hurwitz’s generalization, 399, 488.
- Binomial tree, xx.
- Bipartite trees, 398.
- Bit: “Binary digit”, either zero or unity.
- BIT: *Nordisk Tidskrift for Informations- Behandling*, an international journal published in Scandinavia since 1961.
- Bitwise operations, 442, 455, 510, 553.
- Blaauw, Gerrit Anne, 458.
- Blikle, Andrzej Jacek, 329.
- Block of external data, 136–137.
- Block of memory, 435.
- Blocking of records, 218, 225.
- Bobrow, Daniel Gureasko, 421, 460, 461.
- Boles, David Alan, 452.
- Bolzano, Bernhard, 382.
- Boncompagni, Prince Baldassarre, 79.
- Boothroyd, John, 177.
- Bootstrapping, 143.
- Borchardt, Carl Wilhelm, 406, 583.
- Bottom of stack, 241.
- Bottom-up process, 309, 351.
- Boundary tag method of storage allocation, 440–442, 453–454, 461.
- Bourne, Charles Percy, 516.
- Bracket notation for coefficients, 92.
- Bracket notation for logical statements, 32–33, *see* Iverson’s convention.
- Branch instruction: A conditional “jump” instruction.
- Branch node of tree, 308.
- Breadth-first search, 351.
- Brenner, Norman Mitchell, 523.
- Brent, Richard Peirce, 563.
- Briggs, Henry, 26.
- Broline, Duane Marvin, 601.
- Brother, in a tree structure, 311.
- Brouwer, Luitzen Egbertus Jan, 406.
- Bruijn, Nicolaas Govert de, 121, 122, 375, 379, 380, 478, 504, 543, 565.
- Buddy system for storage allocation, 442–444, 447–448, 454–456, 461.
- Buffering of input-output, 158, 216–228.
 history, 231.
 swapping, 147, 159, 217–218, 225.
- Bugs: Errors or defects; *see* Debugging.
- Burke, John, 310.
- Burks, Arthur Walter, 359.
- Burleson, Peter Barrus, 462.
- Burroughs B220, 124.
- Burroughs B5000, 461.
- Busche, Conrad Heinrich Edmund Friedrich, 43.
- Busy waiting, 216.
- Byte: Basic unit of data, usually associated with alphabetic characters, 125.
 in MIX, 124–125, 139.
- Byte size in MIX: The number of distinct values that might be stored in a byte.
- C language, 556.
- Cache, 528.
- CACM: *Communications of the ACM*, a publication of the Association for Computing Machinery since 1958.
- Cajori, Florian, 24.
- Calendar, 160.
- California Institute of Technology, x, 282.
- Call: To activate another routine in a program.
- Calling sequence, 187–190, 193, 196–197.
- Campbell, John Arthur, 450.
- Canonical cycle notation for permutations, 178–179.
- Canonical representation of oriented trees, 390–394, 397–398, 590–591.
- Capelli, Alfredo, 50, 71.

- Car: LISP terminology for the first component of a List; analogous to INFO and DLINK on page 411, or to ALINK on page 415.
- Cards, playing, 51, 69, 233–237, 238, 377–378.
- Cards, punched, 136–137, 152, 229.
- Carlitz, Leonard, 499, 506, 96.
- Carlyle, Thomas, xiv.
- Carpenter, Brian Edward, 229.
- Carr, John Weber, III, 458.
- Cassini, Jean Dominique, 81.
- Catalan, Eugène Charles, 407. numbers, 407.
- Cate, Esko George, 523.
- Cauchy, Augustin Louis, 92, 475, 490, 506, 520, 582. inequality: $(\sum a_k b_k)^2 \leq (\sum a_k^2)(\sum b_k^2)$. matrix, 37–38, 475.
- Cayley, Arthur, 396, 406–407, 586, 597.
- CDC 1604 computer, 124, 529.
- Cdr: LISP terminology for the remainder of a List with its first component deleted; analogous to RLINK on page 411, or to BLINK on page 415.
- Ceiling function $[x]$, 39, 41.
- Cell: A word of the computer memory, 127.
- Cellar, 240.
- Central moment of a probability distribution, 105.
- Centroid of a free tree, 387–388, 397, 589.
- Ceulen, Ludolph van, 596.
- Chain: A word used by some authors to denote a linked linear list, by others to denote a linearly ordered set.
- Chain rule for differentiation, 52.
- Chaining: A word used by some authors in place of "linking".
- Chakravarti, Gurugovinda (ଗୁରୁଗୋବିନ୍ଦ ଚକ୍ରବାର୍ତ୍ତୀ), 53.
- Change of summation variable, 28, 32–33.
- Channel: A data-transmission device connected to a computer, 224.
- CHAR (convert to characters), 138.
- Character codes of MIX, 136, 138, 140–141.
- Characteristic function of a probability distribution, 103.
- Characteristic polynomial of a matrix, 499.
- Charles Philip Arthur George, Prince of Wales, 310.
- Cheam, Tat Ong, 450.
- Cheating, 582.
- Chebyshev, Pafnutii Lvovich (Чебышев, Павел Львович), inequality, 98, 104.
- Checkerboard, 435–436.
- Checkerboarding, *see* Fragmentation.
- Chen, Tien Chi (陳天機), 471.
- Cheney, Christopher John, 421.
- Cheney, Ednah Dow Littlehale, 377.
- Chernoff, Herman, 502.
- Chess, 6, 194, 272.
- Child link, 427–433.
- Children in tree structures, 311, 317, 334–335, 352.
- Chowla, Paromita (পারমিতা চৌধুরী), 307.
- Christian IX, King of Denmark, 310, 311, 562.
- Christie Mallowan, Agatha Mary Clarissa (Miller), xvii.
- Chu Shih-Chieh (= Zhu Shijié 漢庭, Zhu Sōngting; 朱世傑漢卿, 朱松庭), 53, 59, 70.
- Chung, Fan Rong King (鍾金芳華), 514.
- Chung, Kai Lai (鍾開萊), 105.
- CI: The comparison indicator of MIX, 126, 134, 142, 211, 213, 228.
- Circle of buffers, 218–227, 231.
- Circuit, Eulerian, in a directed graph, 374–376, 379–380, 584.
- Circuit, Hamiltonian, in a directed graph, 374, 379.
- Circular definition, 263, 308, *see* Definition, circular.
- Circular linking, 273–280, 302, 355, 411, 459.
- Circular store, 240.
- Circulating shift, 135.
- CITRUS, 457.
- Clark, Douglas Wells, 604.
- Clavius, Christopher, 159.
- Clock, for real time, 228.
- Clock, simulated, 283, 288.
- Clock, solitaire game, 377–378.
- Closed subroutine, *see* Subroutine.
- CMath: Concrete Mathematics, a book by R. L. Graham, D. E. Knuth, and O. Patashnik, 11.
- CMP1 (compare r11), 134, 210–211.
- CMPA (compare rA), 134, 210–211.
- CNPX (compare rX), 134, 210–211.
- COBOL: "Common Business-Oriented Language", 424–43, 457, 458.
- Codes for difficulty of exercises, xvii.
- Coding: Synonym for "programming", but with even less prestige.
- Coefficient extraction, 92.
- Cofactor of element in square matrix: Determinant of the matrix obtained by replacing this element by unity and replacing all other elements in the same row or column by zero, 37, 381.
- Coffman, Edward Grady, Jr., 450–451.
- Cohen, Jacques, 421, 461, 614.
- Coin tossing, 101–102.
- tail of distribution, 106.
- Collins, George Edwin, 461.

- Combinations of n objects taken k at a time, 52–53, 69.
 with repetitions permitted, 73, 95, 386, 388.
 with restricted repetitions, 95.
- Combinatorial matrix, 37–38, 589.
- Combinatorial number system, 73, 560.
- Comfort, Webb T., 461.
- COMIT, 461.
- Command: Synonym for "instruction".
- Comments, 2–3.
 in assembly language, 145, 149.
- Commutative law, 165.
- Comp. J.*: The Computer Journal, a publication of the British Computer Society since 1958.
- Compacting memory, 423, 439, 449, 452, 455.
- Comparability, 270.
- Comparison indicator of MIX, 126, 134, 142, 211, 213, 228.
- Comparison operators of MIX, 134, 210–211.
- Compiler: A program that translates computer languages.
 algorithms especially for use in, 360, 424–434, 556.
- Complete binary tree, 401, 405, 563.
- Complete t -ary tree, 401–402.
- Complex conjugate, 21.
- Complex number, 21.
- Compound interest, 23–24.
- Compression of messages, 407.
- Computational error, 24–26.
- Computational method, 5, 7–8.
- Compute: To process data.
- Computer: A data processor.
- Computer language, *see* Assembly language, Machine language, Programming language.
- CON (constant), 149–150, 155.
- Concatenation of strings, 274.
- Concave function, 406.
- Conditional expression, 460, 624.
- Congruence, 40–42.
- Connected directed graph, 363.
 strongly, 372, 377.
- Connected graph, 363.
- Conservative law, 170, *see* Kirchhoff's law.
- Constants in assembly language, 149–150, 155.
- Construction of trees, 340–341, 343, 428–429.
- CONTENTS, 127, 235–236.
- Context-free grammar, 539.
- Continuants, 600–601.
- Continued fractions, 498.
- Continuous simulation, 282, 298.
- Convergence: An infinite sequence (X_n) converges if it approaches a limit as n approaches infinity; an infinite sum or product is said to "converge" or to "exist" if it has a value according to the conventions of mathematical calculus; see Eq. 1.2.3–(3).
- absolute, 29.
- of power series, 87, 396.
- Conversion operations of MIX, 138.
- Convolution of probability distributions: The distribution obtained by adding two independent variables, 103.
- Conway, Melvin Edward, 19, 80, 151, 229, 273, 408, 600.
- Copy a data structure: To duplicate a structured object by producing another distinct object that has the same data values and structural relationships.
- binary tree, 329–330, 333, 347.
- linear list, 279.
- List, 423.
 two-dimensional linked list, 306.
- Copying and compacting, 421.
- Corless, Robert Malcolm, 395.
- Couroutines, 193–200, 222–223, 283–296, 320.
 history, 229.
 linkage, 194, 200, 223, 291.
- Correspondence between binary trees and forests, 334–335, 346.
- Cousins, 317.
- Coxeter, Harold Scott Macdonald, 80, 162, 407, 408.
- Crelle: Journal für die reine und angewandte Mathematik*, an international journal founded by A. L. Crelle in 1826.
- Crelle, August Leopold, 58, 632.
- Critical path time, 217.
- Crossword puzzle, 163.
- Crowe, Donald Warren, 601.
- Cumulants of a probability distribution, 103–106.
- Cycle: Path from a vertex to itself.
 detection of, 271.
 fundamental, 366–370, 377.
 in directed graph, 363.
 in graph, 363.
 in permutation, 164–167, 176–178, 182–184.
 in random permutation, 179–184.
 notation for permutations, 164–167, 172–175, 182.
 oriented, in directed graph, 372.
 singleton, in permutation, 164, 171, 180–181.
- Dahl, Ole-Johan, 230, 461, 462.
- Dahm, David Michael, 433, 434.
- Data (originally the plural of "datum", but now used collectively as singular or plural, like "information"): Representation in a precise, formalized

- language of some facts or concepts, often numeric or alphabetic values, to facilitate manipulation by a computational method, 215.
- packed, 128, 158.
- Data organization:** A way to represent information in a data structure, together with algorithms that access and/or modify the structure.
- Data structure:** A table of data including structural relationships, 232–465.
- linear list structures, 238–298.
 - List structures, 408–423.
 - multilinked structures, 424–434.
 - orthogonal lists, 298–307, 424–434.
 - tree structures, 308–408.
- Daughter, in a tree structure, 311.
- David, Florence Nightingale, 66.
- Davies, David Julian Meredith, 445.
- Davis, Philip Jacob, 50.
- Dawson, Reed, 584.
- de Bruijn, Nicolaas Govert, 121, 122, 375, 379, 380, 478, 504, 543, 565.
- De Moivre, Abraham, 74, 83, 87, 106, 182, 536.
- De Morgan, Augustus, 17.
- Deallocation, *see* Liberation.
- Debugging: Detecting and removing bugs (errors), 192–193, 201, 257, 297, 413, 556.
- DEC1 (decrease rI1), 134, 210.
- DECA (decrease rA), 134, 210.
- Decimal computer: A computer that manipulates numbers primarily in the decimal (radix ten) number system.
- Decimal number system, 21, 619.
- DECX (decrease rX), 134, 210.
- Defined symbol, an assembly language, 153.
- Definition, circular, *see* Circular definition.
- Degree, of node in tree, 308, 317, 377.
- of vertex in directed graph, 372.
- Deletion of a node: Removing it from a data structure and possibly returning it to available storage.
- from available space list, *see* Reservation.
 - from deque, 251, 297.
 - from doubly linked list, 281, 290–291, 297.
 - from doubly linked ring structure, 358.
 - from linear list, 239.
 - from linked list, 236, 255, 276, 305.
 - from queue, 242, 244–245, 254, 261, 265, 273–274.
 - from stack, 241, 242, 244–245, 247, 254, 259, 269, 273–274, 278, 458.
 - from tree, 358.
 - from two-dimensional list, 305.
- Demuth, Howard B., 120.
- Depth-first search, 578, 581.
- Deque: Double-ended queue, 239–243, 269.
- deletion from, 251, 297.
 - input-restricted, 239–243, 416.
 - insertion into, 251, 297.
 - linked allocation, 280, 297.
 - output-restricted, 239–243, 269, 274.
 - sequential allocation, 251.
- Derangements, 180, 183.
- Derivative, 90, 338.
- Dershowitz, Nachum, 518, 588.
- Descendant, in a tree structure, 311.
- Determinant of a square matrix, 37–39, 81, 378–379, 382.
- Deuel, Phillip DeVere, Jr., 556.
- Deutsch, Laurence Peter, 418, 421, 422.
- Dewar, Robert Berriedale Keith, 614.
- Dewey, Melvil, notation for binary trees (due to Galton), 317, 331, 346, 405.
- notation for trees, 313, 317, 382–383, 460.
- Diaconis, Persi Warren, 491.
- Diagonals of polygons, 408.
- Diagrams of structural information, 234, 279.
- before-and-after, 260–261, 278, 281, 571.
 - binary trees, 312, 318, 563.
 - List structures, 315–317, 408–409.
 - tree structures, 309–315, 337, 346, 349, 460.
- Dickman, Karl Daniel, 525.
- Dickson, Leonard Eugene, 81.
- Dictionaries of English, 1–2, 215–216.
- Differences of polynomials, 64.
- Differentiation, 90, 338–347, 459.
- chain rule for, 52.
- Digamma function $\psi(z)$, 44, 75, 493.
- Digit: One of the symbols used in radix notation; usually a decimal digit, one of the symbols 0, 1, ..., or 9.
- Digraph, 372, *see* Directed graph.
- Dijkstra, Edsger Wijbe, 17, 230, 231, 240, 459, 462, 545, 580, 605.
- d'Imperio, Mary E., 462.
- Directed graphs, 372–374, 422.
- as flow charts, 364–365, 377.
 - balanced, 374–375.
 - connected, 363.
 - regular, 379.
 - strongly connected, 372, 377.
- Discrete system simulation, 203, 282–298.
- synchronous, 282, 298.
- Disjoint sets: Sets with no common elements.
- Disk files, 136–137, 435, 463.
- Disk input, buffered, 228.
- Disposal, *see* Garbage collection, Liberation.
- Dissection of a polygon, 408.
- Distributed-fit method of storage allocation, 450–451, 456.

- Distribution: A specification of probabilities that govern the value of a random variable.
- binomial, 101–102.
 - negative binomial, 107.
 - normal, 104–106, 122.
 - Poisson, 106, 502, 524.
 - tails of, 104, 106–107.
 - uniform, 102, 253, 446.
- Distributive law, 28, 37, 42, 598.
- DIV (divide), 131–133, 139, 208.
- Divergent series, 28, 75.
- Division converted to multiplication, 516–518.
- Divisor: x is a divisor of y if $y \bmod x = 0$ and $x > 0$; it is a *proper divisor* if in addition $1 < x < y$.
- Dixon, Alfred Cardew, 490.
- Dixon, Robert Dan, 509.
- DLINK: Link downward, 409, 411.
- Doran, Robert William, 229.
- Double generating function: A generating function of two variables, 94, 396, 405, 537–539.
- Double order for traversing trees, 332, 333, 564.
- Doubly linked lists, 280–281, 288–291, 297–298, 357, 411, 441, 443, 452, 459. compared to singly linked, 281, 298.
- Dougall, John, 490.
- Doyle, Arthur Conan, 465.
- Drum memory, 136–137, 457.
- Dull, Brutus Cyclops, 111.
- Dummy variable, 27.
- Dunlap, James Robert, 457.
- Dutka, Jacques, 50, 65.
- Dvoretzky, Aryeh, 593.
- Dwyer, Barry, 567.
- Dynamic storage allocation, 246–254, 256–259, 413–414, 435–456. history, 457–458, 461–462. running time estimates, 449.
- Dynastic order, 336, *see Preorder*.
- DYSEAC computer, 231.
- Earley, Jackson Clark, 462.
- Easter date, 159–160.
- Edelman, Paul Henry, 598.
- Edge in graph, 363.
- Edwards, Daniel James, 423.
- Effective algorithm, 6, 8, 9.
- Egorychev, Georgii Petrovich (Егорычев, Георгий Петрович), 499.
- Eisele, Peter, 480.
- Eisenstein, Ferdinand Gotthold Max, 479.
- Elementary symmetric functions, 38, 94, 497.
- Elevator (lift) system, 282–298.
- Embedding of partial order into linear order, 262, *see Topological sorting*.
- Embedding of tree in another tree, 348, 386.
- Emulation, 202.
- END, 151, 156, 296.
- End of file, 216, 227.
- Endorder, *see Postorder*.
- Engles, Robert William, 462.
- English letter frequencies, 159.
- ENN1 (enter negative into rII), 133, 210.
- ENNA (enter negative into rA), 133, 210.
- ENNX (enter negative into rX), 133, 210.
- ENT1 (enter into rII), 133, 210.
- ENTA (enter into rA), 133, 210.
- Entity, 233, *see Node*.
- Entrances to subroutines, 186–191. multiple, 189.
- ENTX (enter into rX), 133, 210.
- Enumeration of subtrees, 378–379.
- Enumeration of tree structures, 386–399. history, 406–407.
- Epictetus of Hierapolis (Ἐπίκτητος ὁ Ἰεραπόλεως), 1.
- EQU (equivalent to), 146, 149, 155.
- Equivalence algorithm, 360–361, 578, 581.
- Equivalence classes, 354.
- Equivalence declarations, 360.
- Equivalence relations, 353–355, 487.
- Equivalent algorithms, 467.
- Equivalent binary trees, 328–329.
- Equivalent forests, 346.
- Equivalent Lists, 423.
- Equivalent of a MIXAL symbol, 156.
- Equivalent trees, 346.
- Erase a data structure: To return all of its nodes to available storage.
- linear list, 273, 274, 279.
 - List, 413–414.
 - right-threaded binary tree, 333.
- Erdélyi, Arthur, 399.
- Erdwinn, Joel Dyne, 229.
- Errors, avoiding, 260–261, 556. computational, 305. detection of, 192–193, 201, 257, 297, 413.
- Etherington, Ivor Malcolm Haddon, 399.
- Ettingshausen, Andreas von, 54.
- Etymology, 1–2.
- Euclid (Εὐκλεῖδης), 2, 5. algorithm for gcd, 2–9, 80. extended, 13–14, 42.
- Euclidean domains, 468.
- Euler, Leonhard (Эйлер, Леонард), 49, 50, 52, 57, 75, 76, 87, 111, 374, 407, 472, 496, 536, 600. constant γ , 75, 114, 619–620. summation formula, 111–116, 120, 123. theorem, 42. totient function $\varphi(n)$, 42, 184.
- Eulerian circuit in directed graph, 374–376, 379, 584. enumeration of, 380.

- Eulerian numbers, second-order, 506.
 Evaluation of powers, 509.
 Evaluation of tree functions, 351, 361.
 Evans, Arthur, Jr., 202.
 Exchange operation (\leftrightarrow), 3, 182, 274.
 Exclusive or, 442, 455, 553.
 Execution time, methods for studying, 96, 170–172.
 for MIX instructions, 138–141.
 Exercises, notes on, xv–xvii, 284.
 Exit: The place where control leaves a routine.
 Exits from subroutines, 186–191.
 Expected value of a random variable: The average or "mean" value, 98, 103.
 from a generating function, 100–103.
 Exponential generating function for (a_n) :
 $\sum a_n z^n/n!$, 89.
 Exponential integral $E_1(z)$, 498.
 Exponents, laws of, 22, 25, 52.
 Extended binary trees, 399–406.
 Extended Euclidean algorithm, 13–14, 42.
 Extendible matrix, 307.
 External nodes, 400–405.
 External path length, 400, 405.
 Extreme and mean ratio, 80.
- Faà di Bruno, Francesco, 483.
 Factorials, 46–52, 55.
 related to gamma function, 49.
 Factorial powers, 50, 52, 67, 71, 109–110.
 FADD (floating add), 306.
 Fail-safe program, 270.
 Fallacious reasoning, 18, 111, 465.
 Falling powers, 50, 67, 69, 624.
 Family order, 350, 577.
 sequential representation of trees, 350.
 Family trees, 310–311, 317, 406.
 Farber, David Jack, 461.
 Farey, John, 520.
 series, 161.
 Father, in a tree structure, 311.
 FCMP (floating compare), 507, 559.
 FDIV (floating divide), 306.
 Ferguson, David Elton, 231, 334.
 Fermat, Pierre de, 17, 466.
 theorem, 41.
 Ferranti Mark I computer, 18.
 Feynman, Richard Phillips, 26.
 Fibonacci, Leonardo, of Pisa, 79–80, 84.
 buddy system, 455.
 generating function, 82–83.
 number system, 86, 495.
 numbers F_n : Elements of the Fibonacci sequence, 13, 18, 79–86.
 numbers, table of, 621.
 sequence, 13, 18, 79–86, 621.
 strings, 86.
 trees, 496.
- Fibonomial coefficients, 85, 499.
 Fich, Faith Ellen, 523.
 Field: A designated portion of a set of data, usually consisting of contiguous (adjacent) symbols.
 partial, of MIX word, 126–128, 139, 143, 207.
 within a node, 233–237.
 within a node, notations for, 235–237, 458.
 FIFO, 240, 459, *see Queue*.
 Fifty-percent rule, 444–445, 447, 448.
 Filters, 198.
 Final vertex of an arc, 372.
 Fine, Nathan Jacob, 484.
 First-fit method of storage allocation, 436–438, 453–456, 616.
 First in, first out, 240, 351, 459, 607, *see Queue*.
 Fischer, Michael John, 353.
 Fixed element of permutation, 164, 180–181.
 Fixed point arithmetic, 158.
 Flag, *see Sentinel*.
 Flajolet, Philippe Patrick, 501, 506, 543, 565.
 Floating point arithmetic, 131, 306.
 operators of MIX, 131, 557–559.
 Floor function $[x]$, 39–41.
 Flow charts, viii, 2–3, 15–18, 364–365, 377.
 Floyd, Robert W, x, 17, 19, 20, 422, 467, 509.
 PLPL, 460–461.
 Flye Sainte-Marie, Camille, 584.
 FMUL (floating multiply), 306.
 FOCS: Proceedings of the IEEE Symposia on Foundations of Computer Science (1975–), formerly called the Symposia on Switching Circuit Theory and Logic Design (1960–1965), Symposia on Switching and Automata Theory (1966–1974).
 Ford, Donald Floyd, 516.
 Forecasting, 224.
 Forest: Zero or more trees, 309, 408, *see Trees*.
 correspondence to binary trees, 334–335, 346.
 enumeration, 389, 594.
 index notation for, 313, 315, 317.
 Formulas, algebraic, *see Algebraic formulas*.
 Förstemann, Wilhelm, 490.
 FORTRAN language, 231, 233, 296, 360, 458, 460.
 Foster, Frederic Gordon, 100.
 Fourier, Jean Baptiste Joseph, 27.
 Fractional part, 40.
 Fraenkel, Aviezri S, 251.
 Fragmentation, 439, 449, 450, 456.
 Friedman, Michael Lawrence, 514.
 Free lattice, 347.

- Free storage, *see* Available space.
 Free subtrees, 365–370.
 enumeration of, 378–379.
 minimum cost, 371.
 Free trees, 363–371.
 definition of, 363.
 enumeration of, 387–388, 398, 407.
 Friedman, Daniel Paul, 421.
 Frieze patterns, 407–408.
 Front of queue, 241.
 FSUB (floating subtract), 306.
 Fuchs, David Raymond, 202.
 Fukuoka, Hirobumi (福岡博文), 508.
 Full-word logical (bitwise) operations, 442, 455, 510, 553.
 Fundamental cycles in a graph, 366–370, 377.
 Fundamental path, 368.
 Fundamental theorem of arithmetic, 42.
 Furch, Robert, 121.
 Future reference in MIXAL, 153, 156.
 restrictions on, 156.
- Galler, Bernard Aaron, 353.
 Galton, Francis, 562, 633.
 Games, solution of, 86, 272–273.
 Gamma function $\Gamma(z)$, 49–52, 72, 79, 116–119.
 incomplete, 117–122.
 Gao, Zhicheng (高志成), 565.
 Garbage collection, 257, 413–423, 438–439, 449, 455, 461, 546, 551.
 efficiency of, 420–421.
 Gardner, Martin, 19, 80, 587.
 Garwick, Jan Vaumund, 248, 457.
 Gaskell, Robert Eugene, 86.
 Gasper, George, Jr., 490.
 Gates, William Henry, III, xi.
 Gauß (= Gauss), Johann Friedrich Carl
 (= Carl Friedrich), 49, 58, 95.
 gcd: Greatest common divisor.
 Gelernter, Herbert Leo, 460.
 Generating functions, 82–84, 87–96, 243, 386–389, 391–392, 394–399, 539.
 double, 94, 396, 405, 537–539.
 for discrete probability distributions, 99–107, 181.
 Genuys, François, 231.
 Geometric progression, sum of, 31, 88.
 Gerberich, Carl Luther, 460.
 Gill, Stanley, 229, 230, 457.
 Girard, Albert, 497.
 Glaisher, James Whitbread Lee, 504.
 Glassey, Charles Roger, 406.
 Gnedenko, Boris Vladimirovich (Гнеденко, Борис Владимирович), 105.
 GO button of MIX, 126, 139, 143, 211.
 Goldbach, Christian, 49, 472.
 Goldberg, Joel, 528.
- Golden ratio, 13, 18, 21, 80, 83–86, 619–620.
 Goldman, Alan Joseph, 589.
 Goldstine, Herman Heine, 18, 229.
 Golomb, Solomon Wolf, 184.
 Columbic, Martin Charles
 (מרטין כהן נולומביק), 596.
 Goncharov, Vasilii Leonidovich (Гончаров, Василий Леонидович), 501.
 Gonnet Haas, Gaston Henry, 395.
 Good, Irving John, 374, 395, 483, 584.
 Gopala (గోపాల), 80.
 Gorn, Saul, 460.
 Gosper, Ralph William, Jr., 65.
 Gould, Henry Wadsworth, 58, 63, 121, 485, 492.
 Gourdon, Xavier Richard, 525.
 Gower, John Clifford, 459.
 Grabner, Peter Johannes, 506.
 Graham, Ronald Lewis, 11, 631.
 Graphs, 363–372, 464.
 directed, *see* Directed graphs.
 Greatest common divisor, 2–9, 13–14, 40, 81–82.
 Greatest integer function, *see* Floor function.
 Griswold, Ralph Edward, 461.
 Grounded wire symbol, 234.
 Grünbaum, Branko, 384, 587.
 Guy, Richard Kenneth, 19, 80, 273, 600.
- H-trees, 563.
 Haddon, Bruce Kenneth, 614.
 Hadeler, Karl-Peter Fritz, 480.
 Hageman, Louis Alfred, 586.
 Halayudha (හළයුදා), 53.
 Hamel, Georg, 480.
 Hamilton, William Rowan, circuit, 374, 379.
 Hamlet, Prince of Denmark, 232.
 Hamming, Richard Wesley, 26.
 Hankel, Hermann, 49.
 Hansen, James Rone, 460.
 Hansen, Wilfred James, 421.
 Haralambous, Yannis (Χαραλάμπους, Ιωάννης), 650.
 Harary, Frank, 407.
 Hardware-oriented algorithms, 26, 252, 611.
 Hardy, Godfrey Harold, 12, 406, 492, 520.
 Hare, David Edwin George, 395.
 Hare and hounds, *see* Military game.
 Harmonic numbers H_n , 75–79, 114.
 generating function, 90.
 table, 621–623.
 Harmonic series, 75, 160–161.
 Haros, C., 520.
 Hartmanis, Juris, 464.
 Hautus, Matheus Lodewijk Johannes, 489.
 hcf, *see* gcd.
 Head of list, *see* List head.
 Heap, 435, *see* Pool.
 Height of tree or forest, 565.
 Heine, Heinrich Eduard, 490.

- Hellerman, Herbert, 459.
 Hemachandra, Āchārya (आचार्य हेमचन्द्र), 80.
 Henkin, Léon Albert, 17.
 Henrici, Peter Karl Eugen, 88.
 Herbert, George, xiv.
 Hermite, Charles, 49, 478.
 Hesse-Kassel, Louise Wilhelmine Friederike Karoline Auguste Julia von, 310, 311.
 Heyting, Arend, 406.
 Hilbert, David, matrix, 38.
 Hiles, John Owen, 421.
 Hill, Robert, 518.
 Hipparchus of Nicaea ("Ιππαρχος Νικαιας"), 593.
 HLT (halt), 136, 143.
 Hoare, Charles Antony Richard, 17, 230, 461, 462.
 Hobbes, Thomas, 650.
 Hobby, John Douglas, 650.
 Holmes, Thomas Sherlock Scott, 465.
 Holt Hopfenberg, Anatol Wolf, 460.
 Honeywell H800, 124.
 Hopcroft, John Edward, 560.
 Hopper, Grace Brewster Murray, 229.
 Hu, Te Chiang (胡德強), 405, 596.
 Huang Bing-Chao (黃秉超), 176.
 Huffman, David Albert, 402, 407.
 algorithm, 402–406.
 Hurwitz, Adolf, 44.
 binomial theorem, 399, 488.
 Hwang, Frank Kwangming (黃光明), 66, 405, 595.
 Hypergeometric functions, 65.
 basic, 490.
- I/O: Input or output, 215.
 II-register of MIX, 125, 142.
 IBM 650 computer, i, 124, 230, 529.
 IBM 701 computer, 230.
 IBM 705 computer, 230.
 IBM 709 computer, 124, 529.
 IBM 7070 computer, 124.
 Identity permutation, 164, 175.
 Iliffe, John Kenneth, 462.
 Illiac I computer, 230.
 Imaginary part of complex number, 21.
 d'Imperio, Mary E., 462.
 IN (input), 137, 215–216.
 In-degree of vertex, 372.
 INC1 (increase r11), 133, 210.
 INCA (increase rA), 133, 210.
 Incidence matrix, 270.
 Inclusion and exclusion principle, 181, 184.
 Incomplete gamma function $\gamma(a, x)$, 117–122.
 INCX (increase rX), 133, 210.
 Indentation, 312.
 Index: A number that indicates a particular element of an array (often called a "subscript"), 4, 299, 313, 315.
- Index register, 125, 127, 158, 266.
 modification of MIX instructions, 127, 251–252.
 Index variable, 27.
 Indirect addressing, 246, 251–252, 306.
 Induction, mathematical, 11–21, 32, 316, 475.
 generalized, 20.
 Inductive closure, 475.
 Infinite series: A sum over infinitely many values, 27–29, 58, 87–96.
 Infinite trees, 317, 382.
 Infinity lemma, 382–386.
 Information: The meaning associated with data — the facts or concepts represented by data; often used also in a narrower sense as a synonym for "data", or in a broader sense to include any concepts that can be deduced from data.
 Information structure, see Data structure.
 Ingalls, Daniel Henry Holmes, 522.
 Initial vertex of an arc, 372.
 inorder for a binary tree, 319–323, 330–332, 346.
 Input, 5, 215–228.
 anticipated, 159, 216, 224.
 buffering, 159, 216–228, 231.
 operators of MIX, 136–138, 215–216.
 Input-restricted deque, 239–243, 416.
 Insertion of a node: Entering it into a data structure.
 into available space list, see Liberation.
 into deque, 251, 297.
 into doubly linked list, 281, 290, 297.
 into doubly linked ring structure, 358.
 into linear list, 239.
 into linked list, 235, 255, 276, 305.
 into quadruply linked binary tree, 333.
 into queue, 242, 244–245, 254, 260, 265, 273–274.
 into threaded binary tree, 327, 332.
 into two-dimensional list, 305.
 onto a stack, 241, 242, 244–245, 247, 254, 258, 269, 273–274, 278, 458.
 Instruction, machine language: A code that, when interpreted by the circuitry of a computer, causes the computer to perform some action.
 in MIX, 127–144.
 symbolic form, 128, 144.
 INT (interrupt), 228.
 Integers, 21.
 Integration, 90.
 by parts, 77, 112–113.
 related to summation, 111–116.
 Interchange operation (\leftrightarrow), 3, 182, 274.
 Interchanging the order of summation, 29, 33, 35, 43.
 Interest, compound, 23–24.

- Krogdahl, Stein, 616.
- Kronecker, Leopold, delta notation, 33, 61, 624.
- Kruskal Joseph Bernard, 386, 588.
- Kummer, Ernst Eduard, 70.
- Kung, Hsiang Tsung (孔祥重), 563.
- Labeled trees, enumeration of, 389, 407.
- Labelle, Gilbert, 395.
- Lagrange (= de la Grange), Joseph Louis, Comte, inversion formula, 392, 594.
- Lamé, Gabriel, 407.
- Lamport, Leslie B., 605.
- Language: A set of strings of symbols, usually accompanied by conventions that assign a "meaning" to each string in the set, 5, 241, 460–461.
- machine, viii–x, 124.
- Laplace (= de la Place), Pierre Simon, Marquis de, 87.
- transform, 94.
- Laplacian of graph, 582.
- Lapko, Olga Georgievna (Лапко, Ольга Георгиевна), 650.
- Large programs, writing, 191–193.
- Larus, James Richard, 369.
- Last in, first out, 240, 452, 459, *see Stack*. almost, 446, 449, 455.
- Latency, 228, 457.
- Lattice: An algebraic system that generalizes operations like \cup and \cap . defined on forests, 577, 598. free, 347.
- Lawson, Harold Wilbur, Jr., 433, 461.
- LCHILD, 352–353, 359–360.
- LD1 (load r1), 129, 208.
- LD1N (load r1 negative), 129, 208.
- LDA (load rA), 129, 208.
- LDAN (load rA negative), 129, 208.
- LDX (load rX), 129, 208.
- LDXN (load rX negative), 129, 208.
- Leaf of tree, 308, *see Terminal node*.
- Least-recently-used replacement, 452.
- Leeuwen, Jan van, 596.
- Left subtree in binary tree, 312, 318.
- Left-to-right maximum or minimum, 97–101, 104–106, 179.
- Legendre (= Le Gendre), Adrien Marie, 49, 51.
- symbol, 45.
- Léger, Émile, 80.
- Lehmer, Derrick Henry, 465.
- Leibniz, Gottfried Wilhelm, Freiherr von, 2, 51.
- Leighton, Frank Thomson, 450–451.
- Leiner, Alan Lewine, 231.
- Leonardo of Pisa, 79–80, 84.
- Leroux, Pierre, 395.
- Letter frequencies in English, 159.
- Level of node in a tree, 308, 316, 317.
- Level order, 351, 564, 577.
- sequential representation, 351, 359.
- LeVeque, William Judson, 466.
- Lévy, Paul, 105.
- Levy, Silvio Vieira Ferreira, xi.
- Lexicographic order, 20, 299–300, 306.
- Liberation of reserved storage, 256, 259, 291, 413–414, 420–421, 438–442, 443–444, 452–456.
- LIFO, 240, 459, *see Stack*.
- Lilius, Aloysius, 159.
- Lindstrom, Gary Edward, 567–568.
- Line printer, 136–137.
- Lineal chart, 310–311.
- Linear extensions, *see Topological sorting*.
- Linear lists, 232, 238–307.
- Linear ordering, 20, 262, 270.
- embedding a partial ordering into, 262, *see Topological sorting*.
- of binary trees, 333.
- of trees, 346.
- Linear probing, 451.
- Linear recurrences, 83, 88.
- Link, 233–237.
- diagram, 234.
- field, purpose, 432–433, 462.
- manipulation, avoiding errors in, 260–261.
- null, 234–235.
- Link variable, 234.
- Linked allocation of tables, 234, 254–256.
- arrays, 301–307.
- contrasted to sequential, 254–256, 296, 433.
- history, 457–461.
- linear lists, 234–237, 264–266, 269.
- tree structures, 334, 352–357.
- Linked memory philosophy, 255, 435.
- Linking automaton, 463–464.
- Linsky, Vladislav Sergeevich (Линский, Владислав Сергеевич), 471.
- LISP, 233, 460–461.
- LISP 2 garbage collector, 613.
- List: Ordered sequence of zero or more elements.
- circular, 273–280, 302, 355, 411, 459.
- doubly linked, 280–281, 288–291, 297–298, 357, 411, 441, 443, 452, 459.
- linear, 232, 238–307.
- of available space, *see Available space list*.
- List (capital-List) structures, 315–316, 408–423, 460–462.
- copying, 423.
- diagrams of, 315–317, 408–409.
- distinguished from lists, 233, 411.
- equivalence between, 423.
- notations for, 315–317, 408–409.
- representation of, 409–412, 421.
- sequential allocation, 421.

- List head, in circular lists, 275, 302–303.
 in doubly linked lists, 280–281,
 288–289, 441, 443.
 in Lists, 410, 414, 417.
 in threaded binary tree, 324, 334.
 in threaded trees, 337.
- List processing systems, 233, 412, 460–462.
- Listing, Johann Benedict, 406.
- Literal constants in MIXAL, 150, 156.
- Literate programming, 193.
- Littlewood, John Edensor, 406.
- LLINK: Link to the left, 280–281, 288–291.
 in binary trees, 318, 322, 327, 333, 459.
 in Lists, 411.
 in trees, 338, 348, 359, 380.
- LLINKT, 324.
- Lloyd, Stuart Phinney, 183, 184.
- Loading operators of MIX, 129, 139, 208.
- Loading routine, 144, 271–272.
- LOC, 235–236.
- Local symbols of MIXAL, 150–151, 157.
- Locally defined function in tree, 351, 361.
- Location counter in MIXAL, 155.
- Location field of MIXAL line, 145, 152.
- Logan, Benjamin Franklin (= Tex), Jr., 74.
- Logarithms, 22–26.
 binary, 23, 26.
 common, 23, 26.
 natural, 23, 26.
 power series, 91.
- Loop in a directed graph: Arc from a vertex to itself, 372.
- Loopstra, Bram Jan, 231.
- Louise Wilhelmine Friederike Karoline Auguste Julia von Hesse-Kassel, 310, 311.
- Lovász, László, 491.
- Lovelace, Augusta Ada Byron King, Countess of, 1.
- LTAG, 322, 333, 349–350, 352, 359, 380.
- Lucas, François Édouard Anatole, 69, 80, 81, 85, 273, 494.
 numbers L_n , 495.
- Luhn, Hans Peter, 457.
- Lukasiewicz, Jan, 338.
- Lunch counter problem, 456.
- Luo, Jianjin (罗见今), 407.
- Lynch, William Charles, 585.
- Machine language: A language that directly governs a computer's actions, as it is interpreted by the computer's circuitry, viii–x, 124.
 symbolic, 144, *see* Assembly language.
- MacMahon, Percy Alexander, 490, 589.
- Macro instruction: Specification of a pattern of instructions and/or pseudo-operators that may be repeated frequently within a program.
- Madnick, Stuart Elliot, 461.
- Magic square, 162.
- Magnetic tape, 136–137, 463.
- Mailoux, Barry James, 461.
- malloc, *see* Dynamic storage allocation.
- Mallows, Colin Lingwood, 536.
- Margolin, Barry Herbert, 450.
- Mark I calculator (Harvard), 229.
- Mark I computer (Ferranti), 18.
- Mark bits, 413–414.
- Marking algorithms: Algorithms that "mark" all of the nodes accessible from some given node, 271–272, 415–423.
- Markov, Andrei Andreevich (Марков, Андрей Андреевич), the elder, 495.
- chain: Path taken by a Markov process, 253, 380–382.
- Markov, Andrei Andreevich (Марков, Андрей Андреевич), the younger, 8.
- Markowitz, Harry Max, 461.
- Markowsky, George, 80, 404.
- Martin, Alain Jean, 605.
- Martin, Johannes Jakob, 614.
- Math. Comp.: Mathematics of Computation (1960–), a publication of the American Mathematical Society since 1965; founded by the National Research Council of the National Academy of Sciences under the original title Mathematical Tables and Other Aids to Computation (1943–1959).
- Mathematical induction, 11–21, 32, 316, 475.
 generalized, 20.
- Matiyasevich, Yuri Vladimirovich (Матијасевић, Јуриј Владимировић), 86.
- Matrix: Two-dimensional array, 298–299, 315.
 Cauchy, 37–38, 475.
 characteristic polynomial of, 499.
 combinatorial, 37–38, 589.
 determinant of, 37–39, 81, 378–379, 382.
 extendible, 307.
 Hilbert, 38.
 incidence, 270.
 inverse of, 37–38, 73, 307.
 multiplication, 306.
 permanent of, 51.
 representation of, 158–159, 298–307.
 singular, 307.
 sparse, 302–306.
 transpose of, 182.
 triangular, 300, 305.
 tridiagonal, 307.
 unimodular, 601.
 Vandermonde, 37–38, 475.
- Matrix (Bush), Irving Joshua, 35, 36.
- Matrix tree theorem, 378–379, 586.
- Mauchly, John William, 230.

Maurolico, Francesco, 17.
 Maximum, algorithm to find, 96–101,
 145, 186.
 Maximum norm, 106.
 McCall's, v.
 McCann, Anthony Paul, 614.
 McCarthy, John, 460–461.
 McEliece, Robert James, 477.
 McIlroy, Malcolm Douglas, 576, 581.
 Mealy, George, 462.
 Mean value, *see* Expected value.
 Meek, H. V., 230.
 Meggitt, John Edward, 471.
 Melville, Robert Christian, 540.
 Memory: Part of a computer system
 used to store data, 126.
 cell of, 127.
 hierarchy, 199, 421, 435, 463.
 map, 435–436.
 types of, 238.
 Merner, Jack Newton Forsythe, 229.
 Merrett, Timothy Howard, 560.
 Merrington, Maxine, 66.
 METAFONT, xi, 611, 650.
 METAPOST, xi, 650.
 Military game, 273, 550.
 Miller, Kenneth William, 123.
 Ming, An-T'u (明安圖), 407.
 Minimum path length, 399–406.
 Minimum spanning tree, 371.
 Minimum wire length, 371.
 Minsky, Marvin Lee, 423.
 Mirsky, Leon, 587.
 Mitchell, William Charles, 525.
 MIX computer, viii–x, 124–144.
 assembly language for, 144–157.
 extensions to, 143, 228, 251–252, 455.
 instructions, summary, 140–141.
 simulator of, 203–212.
 MIXAL: MIX Assembly Language, 144–157,
 235–236.
 Mixed-radix number system, 300.
 Mixture of probability distributions, 106.
 MMIX computer, 124, 187, 215, 325.
 Mock, Owen Russell, 231.
 mod, 39–40.
 modulo, 40.
 Mohammed, John Llewelyn, 527.
 Moments of probability distributions, 105.
 Monitor routine, 212, *see* Trace routine.
 Monte Carlo method: Experiments with
 random data, 445–447.
 Moon, John Wesley, 407.
 Mordell, Louis Joel, 479.
 Morris, Francis Lockwood, 18, 614.
 Morris, Joseph Martin, 567.
 Morrison, Emily Kramer, 229.
 Morrison, Philip, 229.

Moschopoulos, Manuel (Μωσχόπουλος,
 Μανώλης), 162.
 Moser, Leo, 66.
 Mother, in a tree structure, 311.
 Motzkin, Theodor Samuel
 (תֹּהֵד מַזְקִין), 85, 593.
 MOVE, 135, 142, 193, 211.
 MOVE CORRESPONDING, 426, 430–431, 434.
 MUG: MIX User's Group, 641.
 MUL (multiply), 131–132, 208.
 Multilinked structures, 232, 288–289,
 357, 424–434, 458.
 Multinomial coefficients, 65, 394.
 Multinomial theorem, 65.
 Multipass algorithms, 198–200, 201–202.
 Multiple: x is a multiple of y if $x = ky$
 for some integer k .
 Multiple entrances to subroutines, 189.
 Multiple exits from subroutines, 190, 269.
 Multiple precision arithmetic, 202.
 Multiple precision constants, 619–621.
 Multiple summation, 33–36.
 Multiplication of permutations, 165–167,
 172–173, 371.
 Multiplication of polynomials, 277, 280, 361.
 Multiplication of sparse matrices, 306.
 Multiplicative function, 42–43.
 Multiset: Analogous to a set, but elements
 may appear more than once.
 Multiway decisions, 158.
 Munro, James Ian, 523.
 Nagorny, Nikolai Makarovich (Нагорный,
 Николай Макарович), 9.
 Nahapetian, Armen, 579.
 Napier, John, Laird of Merchiston, 23.
 Nash, Paul, 556.
 National Science Foundation, x.
 Natural correspondence between binary
 trees and forests, 334–335, 346.
 Natural logarithms, 23, 26.
 Naur, Peter, 17.
 Needham, Joseph, 59.
 Neely, Michael, 452.
 Negative: Less than zero (*not* zero).
 Negative binomial distribution, 107.
 Nested parentheses, 312–313, 349, 597.
 Nested sets, 312, 317.
 Nesting store, 240.
 Network: A graph together with additional
 data, such as weights on the edges
 or vertices.
 Neumann, John von (= Margittai Neumann
 János), 18, 229, 457.
 Neville, Eric Harold, 591.
 Newell, Allen, 230, 457–458, 460.
 Newton, Isaac, 22, 57, 497.
 identities, 497.
 Next-fit method of storage allocation,
 448, 453, 617.

- Partial field designations in MIX, 126–128, 139, 143, 207.
- Partial fractions, 62–63, 72, 83.
- Partial ordering, 261–262, 269–270, 346, 562, 575.
- Partitions of a set, 74, 482.
- Partitions of an integer, 12, 34, 93. generating function, 87, 94.
- Pascal, Blaise, 17, 53. triangle, 53–54, 69, 71, 73, 85, 499, *see also* Binomial coefficients.
- Pass, in a program, 198–200.
- Patashnik, Oren, 11, 631.
- Path, in a graph or directed graph, 363. oriented, 372. random, 380–381. simple, 363, 369, 372, 376.
- Path compression, 576.
- Path length of a tree structure, 399–406. average, 405.
- Patience (solitaire), 377–378.
- Patt, Yale Nance, 509.
- Pawlak, Zdzislaw, 460.
- PDP-4 computer, 124.
- Peck, John Edward L., 461.
- Pedigree, 310–311.
- Peirce, Charles Santiago Sanders, 593.
- Penrose, Roger, 587.
- Peripheral device: An I/O component of a computer system, 136.
- Perlis, Alan Jay, 322, 459–460.
- Permanent of a square matrix, 51.
- Permutations, 45–46, 51, 97–98, 164–185, 242–243. in place, 9, 165, 184–185, 523. inverse of, 106, 175–178, 182. multiplication of, 165–167, 172–173, 371. notations for, 164. orthogonal vectors of, 184.
- PERT network, 261–262.
- Petkovsek, Marko, 65.
- Petolino, Joseph Anthony, Jr., 516.
- Pfaff, Johann Friedrich, 486.
- Pflug, Georg Christian, 445.
- Phi (ϕ), 81, *see also* Golden ratio. number system, 86.
- Phidias, son of Charmides ($\Phi\delta\delta\alphaς$ Φεδίας), 81.
- Philco S2000 computer, 124.
- Phyllotaxis, 80.
- Pi (π), 21, 619–620. Wallis's product for, 52, 116.
- Pingala, Āchārya (आचार्य पिङ्गल), 53.
- Pile, 240.
- Pilot ACE computer, 230.
- Pipe, 198.
- Pipeline, 528.
- Pisano, Leonardo, 79–80, 84.
- Pivot step, 302–305, 307.
- PL/I language, 433–434.
- PL/MIX language, 156.
- Plane trees, 308, *see also* Ordered trees.
- Plex, 458.
- Poblete Olivares, Patricio Vicente, 523.
- Poincaré, Jules Henri, 491.
- Pointer, *see also* Link.
- Pointer machines, 464.
- Poirot, Hercule, xvii.
- Poisson, Siméon Denis, distribution, 106, 524. tail of, 502.
- Polish notation, *see also* Prefix notation, Postfix notation.
- Polonsky, Ivan Paul, 461.
- Pólya, George (= György), 17, 93, 395, 396, 406, 407, 496.
- Polynomials, 55, 57, 64, 67, 68, 70, 108. addition of, 275–280, 357–359. Bernoulli, 44, 113–115, 503. differences of, 64. multiplication of, 277, 280, 361. representation of, 275–276, 280, 356–357.
- Pool of available nodes, 257, *see also* Available space list.
- Pooled buffers, 224, 227.
- Pop up a stack: Delete its top element, 241, 242, 244–245, 247, 254, 259, 269, 273–274, 278, 458.
- Positive: Greater than zero (*not* zero).
- Postfix notation, 338, 352, 593.
- Posting a new item, *see also* Insertion.
- Postorder for a binary tree, 319, 321, 330–332, 346.
- Postorder for a tree, 336–340, 346, 348, 460.
- Postorder with degrees, representation of trees, 351, 361–362.
- PostScript, 202.
- Poupard, Yves, 598.
- Power of a number, 22. evaluation, 509.
- Power series: A sum of the form $\sum_{k \geq 0} a_k z^k$, *see also* Generating function. convergence of, 87, 396. manipulation of, 118.
- Pratt, Vaughan Ronald, 45, 539, 592.
- Prefix notation, 338.
- Preorder for a binary tree, 319, 321, 330–332, 346.
- Preorder for a tree, 336–338, 346, 348, 460.
- Preorder sequential representation of trees, 349, 362. with degrees, 359, 460.
- Prepostorder, 568.
- Prim, Robert Clay, 371.
- Prime numbers, 19, 41, 45, 47–48, 51, 69–70, 84–85. algorithm to compute, 147–149. factorization into, 42.

- Prinz, Dietrich G., 230.
 Priority queue, 435, 556, 590.
 Probability distribution: A specification of probabilities that govern the value of a random variable, 98–107.
 average (“expected”) value of, 98–103.
 variance of, 98–103.
 Probability generating function, 103.
 Procedure, *see* Subroutine.
 Procedure for reading this set of books, xii–xiv, 9.
 Prodinger, Helmut, 506.
 Profile of a program: The number of times each instruction is performed, 145, 170, 214, 296, 528.
 Program: Representation of a computational method in some precise, formalized language, 5.
 Programming language: A precise, formalized language in which programs are written.
 Programs, hints for construction of, 191–193, 296.
 Progression, arithmetic, sum of, 11, 31–32, 56.
 Progression, geometric, sum of, 31, 88.
 Proof of algorithms, 5–6, 13–17, 321, 361, 422, 434.
 Proof of termination, 16–17, 19–21, 386.
 Proper divisor, *see* Divisor.
 Property *A*, 586.
 Prosody, 53, 80.
 Prüfer, Ernst Paul Heinz, 407.
 Pseudo-operator: A construction in a programming language that is used to control the translation of that language into machine language, 146.
 Psi function $\psi(z)$, 44, 75, 493.
 Purdom, Paul Walton, Jr., 448, 450.
 Push down list, 240, *see* Stack.
 Push down a stack: Insert a new top element, 241, 242, 244–245, 247, 254, 258, 269, 273–274, 278, 458.
 q-nomial coefficients, 65, 73, 491.
 q-nomial theorem, 73, 494.
 Quadratic Euclidean domains, 468.
 Quadratic reciprocity law, 45.
 Quadruply linked binary tree, 333.
 Quadruply linked trees, 357.
 Quadtrees, 564.
 Qualification of names, 424–434.
 Quasi-parallel processing, 296.
 Queue, 239–243, 264–266, 459, 577, 607.
 deletion from the front, 242, 244–245, 254, 261, 265, 273–274.
 insertion at the rear, 242, 244–245, 254, 260, 265, 273–274.
 linked allocation, 259–261, 269, 273–274, 288.
 sequential allocation, 244–245, 251, 252, 254.
 Quick, Jonathan Horatio, 502.
 Quotient, 40.
 Rahman, Mizanur (মিজানুর রহমান), 490.
 Railway network, 240.
 Ramanan, Prakash Viriyur (பிரகாஷ் விரைவுர்), 538.
 Ramanujan Iyengar, Srinivasa (ஸ்ரீநிவாஸ் ராமானுஜன் தழைக்காரி), 12, 121, 122.
 Ramshaw, Lyle Harold, 487.
 Ramus, Christian, 71.
 Randell, Brian, 202, 450.
 Random path, 380–381.
 Raney, George Neal, 392, 394, 593.
 Raphael, Bertram, 460.
 Rational number, 21, 25, 161.
 Raz, Yoav (יָאוֹב רָזִי), 534.
 RCA 601 computer, 124.
 Reactive process, 5.
 Read, Ronald Cedric, 565.
 Reading: Doing input, 215.
 Real number, 21.
 Real part of complex number, 21.
 Real-time garbage collection, 423.
 Reallocation of memory, 247–251, 452, 457,
 see also Compacting memory.
 Rear of queue, 241.
 Recently used bit, 452.
 Recipe, 6.
 Reciprocity formulas, 44–45.
 Recomp II computer, 124.
 Record: A set of contiguous data; *see also* Node, 233.
 Records, blocking of, 218, 225.
 Rectangular arrays, 298–307.
 Recurrence relation: A rule that defines each element of a sequence in terms of the preceding elements, 87–89.
 Recursive definition, 308, 312, 315, 318, 319, 335, 346, 357.
 Recursive Lists, 316.
 Recursive use of subroutines, 191.
 Reeves, Colin Morrison, 445.
 Reference, 233, *see* Link.
 Reference counters, 413–414, 421, 461.
 Reflection principle, 536, 593, 598.
 Reflective laws, 55, 489.
 Reflexive relation, 261, 353.
 Registers: Portions of a computer’s internal circuitry in which data is most accessible.
 of MIX, 125.
 saving and restoring contents of, 188, 198, 213, 228.
 Regular directed graph, 379.
 Reingold, Edward Martin, 23, 518.

- Relation: A property that holds for certain sets (usually ordered pairs) of elements; for example, " $<$ " is a relation defined on ordered pairs (x, y) of integers, and the property " $x < y$ " holds if and only if x is less than y .
- antisymmetric, 261.
 - asymmetric, 261.
 - equivalence, 353–355, 487.
 - irreflexive, 261.
 - reflexive, 261, 353.
 - symmetric, 353.
 - transitive, 108, 261, 353, *see Ordering*.
- Relative error, 116.
- Relatively prime integers, 40.
- Releasing a buffer, 219–223, 226–227.
- Remainder, 40.
- Remove from a structure, *see Deletion*.
- Rényi, Alfréd, 595.
- Replacement operation (\leftarrow), 3.
- Replicative function, 43–44.
- Representation (inside a computer),
 - methods for choosing, 238, 424–433.
 - of algebraic formulas, 337, 459.
 - of binary trees, 318, 322, 327, 333–334.
 - of deques, 251, 280, 297.
 - of forests, 334.
 - of Lists, 409–412, 421.
 - of oriented trees, 347, 353, 377.
 - of polynomials, 275–276, 280, 356–357.
 - of queues, 244–245, 251–254, 259–261, 269, 273–274, 288.
 - of stacks, 244–254, 258, 269–270, 273–274, 332, 417.
 - of trees, 348–357, 359–362, 459–460.
- Reprogramming, 203.
- Reservation of free storage, 256–258, 291, 436–438, 443, 452–456.
- Reversing a list, 269, 279.
- Reversion storage, 240.
- Ribenboim, Paolo, 466.
- Rice, Stephan Oswald, 565.
- Richmond, Lawrence Bruce, 565.
- Riemann, Georg Friedrich Bernhard,
 - zeta function* $\zeta(s)$, 76.
- Right subtree of a binary tree, 312, 318.
- Right-threaded binary trees, 327, 332–334.
- Right-threaded trees, 338, 380.
- Right-to-left maximum or minimum, 97–101, 104–106, 179.
- Ring structure, 355–357.
- Riordan, John, 397, 494, 595.
- RISC: Reduced Instruction Set Computer, 124.
- Rising factorial powers, 50, 71, 624.
- Ritchie, Dennis MacAlistair, 461.
- RLINK: Link to the right, 280–281, 288–291.
 - in binary trees, 318, 322, 327, 333, 459.
 - in Lists, 409, 411.
- in trees, 338, 348–353, 359, 380.
- RLINKT, 324.
- Robinson, Raphael Mitchel, 587.
- Robson, John Michael, 448, 452, 456, 567, 604, 616.
- Rodrigues, Benjamin Olinde, 407.
- Roes, Piet Bernard Marie, 100.
- Rogers, Leonard James, 490.
- Rokicki, Tomas Gerhard, 202.
- Roll, 240.
- Root of a directed graph, 372.
- Root of a number, 22, 25.
- Root of a tree, 110, 308, 309, 317.
 - change of, 377.
- Rooting a free tree, 373.
- Roots of unity, 89.
- Rosenberg, Arnold Leonard, 560.
- Rosenstiel, Pierre, 243.
- Ross, Douglas Taylor, 450, 458, 462.
- Rotating memory devices, 228, 457.
- Rothe, Heinrich August, 63.
- Rounding, 41, 83, 160, 183.
- Rousseau, Cecil Clyde, 507.
- Roving pointer, 607.
- Row major order, 159, 182, 299.
- RTAG, 322, 332–334, 338, 349–351, 359, 380.
- Running time, *see Execution time*.
- Russell, David Lewis, 613.
- Russell, Lawford John, 202.
- Saddle point, 159.
- Salton, Gerard Anton, 351, 459.
- Sammet, Jean Elaine, 462, 574.
- Satterthwaite, Edwin Hallowell, Jr., 230.
- Saving and restoring registers, 188, 198, 213, 228.
- Schäffer, Alejandro Alberto, 514.
- Schatzoff, Martin, 450.
- Scherk, Heinrich Ferdinand, 490.
- Schlatter, Charles Fordemwalt, 459.
- Schlatter, William Joseph, 459.
- Schleswig-Holstein-Sonderburg-Glücksberg, Christian von, *see Christian IX*.
- Scholten, Carel Steven, 231, 605.
- Schoor, Amir (שׁוֹר אַמִּיר), 559.
- Schorr, Herbert, 418, 422.
- Schreiber, Peter, 81.
- Schreier, Otto, 385.
- Schröder, Friedrich Wilhelm Karl Ernst, 592.
- Schwartz, Eugene Sidney, 404.
- Schwarz, Karl Hermann Amandus, inequality, 36.
- Schwenk, Allen John, 495.
- Schweppé, Earl Justin, 459.
- SCOPE link, 350, 362, 434.
- Scroll, 240.
- Segner, Johann Andreas von, 407, 536.
- Seki, Takakazu (關孝和), 112, 115.
- Self-modifying code, 187, 193.
- Selfridge, John Lewis, 78.

- Semaphores, 231.
 Semi-invariants of a probability distribution, 103–106.
 Sentinel: A special value placed in a table, designed to be easily recognizable by the accompanying program, 217–218, 276, 567.
 Sequential (consecutive) allocation of tables, 244.
 arrays, 158–159, 299–301, 305–307.
 contrasted to linked, 254–256, 296, 433.
 history, 457.
 linear lists, 244–254, 264–266, 325.
 tree structures, 348–352, 359–362, 433.
 Series, infinite: An infinite sum.
 Series-parallel networks, 589.
 Sets, partition of, 74, 482.
 Sha, Jichang (沙基昌), 547.
 Shakespeare (= Shakspere), William, 232, 466.
 Shaw, John Clifford, 230, 457–458.
 Shelf, 240.
 Shephard, Geoffrey Colin, 384, 587.
 Shepp, Lawrence Alan, 183, 184.
 Shift operators of MIX, 135, 211.
 Shor, Peter Williston, 514.
 Shore, John Edward, 445, 450.
 Shylock, 466.
 Sibling, in a tree structure, 311.
 Sibling link, 427–433.
 SICOMP: SIAM Journal on Computing, published by the Society for Industrial and Applied Mathematics since 1972.
 Sideways addition, 131.
 Sign function ($\text{sign } x$), 475.
 Similar binary trees, 327–329.
 Similar forests, 346.
 Simon, Herbert Alexander, 230, 457–458.
 Simonovits, Miklós, 505.
 Simple oriented path, 372, 376.
 Simple path, 363, 369.
 SIMSCRIPT language, 461.
 SIMULA I language, 229.
 Simulated time, 283, 288.
 Simulation: Imitation of some process, 445.
 continuous, 282, 298.
 discrete, 203, 282–298.
 of one computer on another, 9, 202–203.
 of one computer on itself, 212–214.
 Singh, Parmanand (परमानन्द सिंह), 80.
 Singleton cycle of a permutation, 164, 171, 180–181.
 Singular matrix, 307.
 Singularity of a function, 396.
 Sister, in a tree structure, 311.
 SLA (shift left rA), 135, 530.
 SLAX (shift left rAX), 135, 530.
 SLC (shift left rAX circularly), 135, 530.
 SLIP, 460–461.
 Sloane, Neil James Alexander, 595.
 Smallest in, first out, 556.
 SNOBOL, 461.
 SODA: Proceedings of the ACM-SIAM Symposia on Discrete Algorithms, inaugurated in 1990.
 Son, in a tree structure, 311.
 Software: General-purpose programs that extend the capabilities of computer hardware.
 Soria, Michèle, 501.
 Spanning subtrees, 365–370.
 minimum cost, 371.
 Sparse array trick, 307.
 Sparse matrices, 302–306.
 Sparse-update memory, 298.
 Speedcoding, 230.
 Spieß, Jürgen, 91.
 Spine of a binary tree, 568.
 SRA (shift right rA), 135, 530.
 SRAK (shift right rAX), 135, 530.
 SRC (shift right rAX circularly), 135, 530.
 STA (store r11), 130, 209.
 STA (store rA), 130, 209.
 Stack, 239–243, 320–321, 323, 325–326, 351, 361, 415–416, 422, 428–429, 458–459.
 deletion ("popping"), 241, 242, 244–245, 247, 254, 259, 269, 273–274, 278, 458.
 insertion ("pushing"), 241, 242, 244–245, 247, 254, 258, 269, 273–274, 278, 458.
 linked allocation, 258, 269, 270, 273–274, 332, 417.
 pointer to, 244, 258.
 sequential allocation, 244–254, 325.
 Stack permutations, 242–243, 331.
 Standard deviation of a probability distribution: The square root of the variance, an indication of how much a random quantity tends to deviate from its expected value, 98.
 Stanford University, ii, x, 296, 554.
 Stanley, Richard Peter, 407, 593, 598.
 Staudt, Karl Georg Christian von, 406.
 Steady state, 381–382.
 Stearns, Richard Edwin, 464.
 Steele, Guy Lewis, Jr., 605.
 Steffens, Elisabeth Francisca Maria, 605.
 Steffensen, Johan Frederik, 503.
 Stevenson, Francis Robert, 579.
 Stickelberger, Ludwig, 51.
 Stigler, Stephen Mack, 448, 450.
 Stirling, James, 47–49, 67, 69, 73, 87, 115, 181.
 approximation, 51, 115–116.
 Stirling numbers, 66–69, 71–74, 78, 99–100, 506, 582.
 asymptotic behavior, 66.
 combinatorial interpretations, 66, 74, 179.
 duality law, 68.

- generating functions, 91.
modulo p , 492.
table, 66.
- STJ** (store rJ), 130, 187, 209.
- STOC: Proceedings of the ACM Symposia on Theory of Computing**, inaugurated in 1969.
- Stolarsky**, Kenneth Barry, 495.
- Storage allocation**: Choosing memory cells in which to store data, *see* Available space list, Dynamic storage allocation, Linked allocation, Sequential allocation.
- Storage mapping function**: The function whose value is the location of an array node, given the indices of that node, 299–301, 305–307.
- Store**: British word for “memory”.
- Storing operators of MIX**, 130, 209.
- Straight linkage**, 254, 258–259, 411.
- String**: A finite sequence of zero or more symbols from a given alphabet, 8, 86, 185, 274, 495, *see* Linear lists.
- binary, 598–599.
concatenation of, 274.
manipulation of, 461, 462.
- Strong**, Hovey Raymond, Jr., 560.
- Strongly connected directed graph**, 372, 377.
- Structure**, how to represent, 238, 424–433, 462.
- Struik**, Dirk Jan, 57, 481, 497.
- Stuart**, Alan, 100.
- STX** (store rX), 130, 209.
- STZ** (store zero), 130, 209.
- SUB** (subtract), 131–132, 208.
- Subadditive law**, 616.
- Subi**, Carlos Samuel, 527.
- Subroutines**, 158, 186–197, 202, 206–207, 211, 269, 279–280, 290–291, 343.
allocation of, 271–272.
history, 229–230.
linkage of, 186, 229.
- Subscript**, 3–4, *see* Index.
- Substitution operation** (\leftarrow), 3.
- Subtrees**, 308.
average size of, 405.
free, enumeration of, 378–379.
- Summation**, 27–39.
by parts, 44, 76–78.
Euler's formula, 111–116, 120, 123.
interchange of order, 29, 33, 35, 43.
multiple, 33–36.
of arithmetic progression, 11, 31–32, 56.
of binomial coefficients 56–74,
76–78, 85, 96.
of geometric progression, 31, 88.
of powers, 115.
related to integration, 111–116.
- Sun SPARCstation**, 650.
- Supremum**: Least upper bound, 37.
- Suri, Subhash** (सुराज सुरी), 514.
- Sutherland, Ivan Edward**, 459.
- Swainson, William**, 330.
- Swapping buffers**, 147, 159, 217–218, 225.
- Swift, Charles James**, 231.
- Swift, Jonathan**, 628.
- Switching table**, 158, 205–206, 209, 210, 530.
- Sylvester, James Joseph**, 407, 473, 583, 586.
- Symbol manipulation**: A general term for data processing, usually applied to nonnumeric processes such as the manipulation of strings or algebraic formulas.
- Symbol table algorithms**, 175, 426.
- Symbolic machine language**, *see* Assembly language.
- Symmetric functions**, 92–94, 497.
elementary, 38, 94, 497.
- Symmetric order for a binary tree**, 319–323, 330–332, 346.
- Symmetric relation**, 353.
- Synchronous discrete simulation**, 282, 298.
- System**: A set of objects or processes that are connected to or interacting with each other.
- System/360 computers**, 124, 189.
- Szekeres, George**, 595.
- Szilrajn, Edward**, 268.
- t-ary trees**, 334, 405.
enumeration of, 397, 593.
sequential allocation, 401.
- Table-driven program**, *see* Interpreter, Switching table.
- Tables**, arrangement of, inside a computer, *see* Representation.
- Tables of numerical quantities**, 54, 66, 619–621.
- Tag field in tree node**, 322, *see* LTAG, RTAG.
- Tail inequalities**, 104, 106–107.
- Tamaki, Jeanne Keiko** (玉置恵子), 596.
- Tamari, Dov**, 577.
lattice, 577, 598.
- Tape**, magnetic, 136–137, 463.
paper, 136–137, 231, 229.
- Tarjan, Robert Endre**, 243, 581.
- Taylor, Brook**, formula with remainder, 117.
- Temme, Nicolaas Maria**, 66, 121.
- Temporary storage**: Part of memory used to hold a value for a comparatively short time while other values occupy the registers, 191.
- Terminal function**, 48, 51.
- Terminal node of a tree**, 308, 318, 352, 397, 597.
- Terminology**, viii, 45, 240, 311, 362, 435.
- Ternary trees**, 334, 401.
- Tetrad tiling**, 383–385.
- Tetrahedral arrays**, 300–301, 306.
- Tex**, xi, 202, 611, 650.

- Theory of algorithms, 7, 9.
 Theory of automata, 230, 240, 463–464.
 Thiele, Thorvald Nicolai, 103.
 Thorelli, Lars-Erik, 603, 613.
 Thornton, Charles, 322, 459–460.
 Thread an unthreaded tree, 333.
 Thread links, 422.
 Threaded binary trees, 322, 331–332, 460.
 compared to unthreaded, 326.
 insertion into, 327, 332.
 list head in, 324, 334.
 Threaded trees, 335–336, 459.
 Three-address code, 337, 459.
 Tiling the plane, 383.
 Time taken by a program, *see* Execution time.
 Todd, John, 475.
 Tonge, Frederic McLanahan, Jr., 460.
 Top-down process, 309, 361.
 Top of stack, 241–242.
 Topological sorting, 261–271, 346, 376, 397.
 Torelli, Gabriele, 71, 488.
 Toroidal tiling, 384.
 Total ordering, 270, *see* Linear ordering.
 Totient function $\varphi(n)$, 42, 184.
 Trace routine, 192, 212–214, 230, 296.
 Traffic signals, 161–162.
 Transfer instruction: A “jump” instruction.
 Transitive relation, 108, 261, 353, *see* Ordering.
 Transposing a rectangular matrix, 182.
 Transposing blocks of data, 184–185.
 Transpositions: Permutations that interchange two elements, 3, 182, 274, 371.
 Traversal of binary trees, 319, 459–460.
 inorder, 320, 323.
 postorder, 331–332.
 preorder, 331–332.
 Traversal of trees, 336, 459–460.
 prepostorder, 568.
 Tree function $T(z)$, 395.
 Tree mappings, 390.
 Trees, 232, 308–423.
 binary, *see* Binary trees.
 comparison of different types, 308–309, 374.
 complete, 401–402, 405, 563.
 construction of, 340–341, 343, 428–429.
 copying of, 329–330, 333, 347.
 definition of, 308, 317, 363, 373.
 deletion from, 358.
 Dewey notation for, 313, 317, 382–383, 460.
 diagrams of, 309–315, 337, 346, 349, 460.
 disjoint, *see* Forest.
 embedding of, 348, 386.
 enumeration of, 386–399, 408.
 equivalent, 346.
 erasing of, 333.
 free, *see* Free trees.
 history, 406–407, 459–460.
 infinite, 317, 382.
 insertion into, 327, 332.
 labeled, enumeration of, 389, 407.
 linear ordering of, 346.
 linked allocation for, 334, 352–357.
 mathematical theory of, 362–408.
 ordered, 308, 374, 388–389, *see* Trees.
 oriented, *see* Oriented trees.
 quadruply linked, 357.
 representation of, 348–357, 359–362, 459–460.
 right-threaded, 338, 380.
 sequential allocation for, 348–352, 359–362, 433.
 similar, 346.
 t-ary, 334, 397, 401–402, 405, 593.
 ternary, 334, 401.
 threaded, 335–336, 459.
 traversal of, 336, 459–460.
 triply linked, 353, 359–360, 427–433.
 unordered, *see* Oriented trees.
 unrooted, 363, *see* Free trees.
 Triangular matrix, 300, 305.
 Triangulations of polygons, 407, 598–601.
 Tricomi, Francesco Giacomo Filippo, 121, 122.
 Tridiagonal matrix, 307.
 Trigonometric functions, 44, 229, 471.
 Trilling, Laurent, 461.
 Triple order for a binary tree, 567–568.
 Triply linked tree, 353, 359–360, 427–433.
 Trit, 139.
 Tritter, Alan Levi, 576.
 Tucker, Alan Curtiss, 405.
 Turing, Alan Mathison, 18, 193, 229, 230, 459.
 machines, 8, 230, 464.
 Tutte, William Thomas, 583.
 Twain, Mark (= Clemens, Samuel Langhorne), 54.
 Twigg, David William, 523.
 Two-line notation for permutations, 164, 182.
 Two stacks, 246, 251, 253–254.
 Two-way linkage, 280–281, 411.
 Typewriter, 136–137, 231.
 Uhler, Horace Scudder, 480.
 Ullman, Jeffrey David, 560.
 UNDERFLOW, 245, 247, 259, 268–269, 274.
 Uniform distribution: A probability distribution in which every value is equally likely, 102, 253, 446.
 Unimodular matrices, 601.
 Uninitialized arrays, 307.
 Union-find algorithm, 354, 360.
 UNIVAC I computer, 151, 229, 231, 480.

5. Inserting at the front is essentially like the basic insertion operation (8), with an additional test for empty queue: $P \Leftarrow \text{AVAIL}$, $\text{INFO}(P) \leftarrow Y$, $\text{LINK}(P) \leftarrow F$; if $F = \Lambda$ then $R \leftarrow P$; $F \leftarrow P$.

To delete from the rear, we would have to find which node links to $\text{NODE}(R)$, and that is necessarily inefficient since we have to search all the way from F . This could be done, for example, as follows:

- If $F = \Lambda$ then UNDERFLOW, otherwise set $P \leftarrow \text{LOC}(F)$.
- If $\text{LINK}(P) \neq R$ then set $P \leftarrow \text{LINK}(P)$ and repeat this step until $\text{LINK}(P) = R$.
- Set $Y \leftarrow \text{INFO}(R)$, $\text{AVAIL} \leftarrow R$, $R \leftarrow P$, $\text{LINK}(P) \leftarrow \Lambda$.

6. We could remove the operation $\text{LINK}(P) \leftarrow \Lambda$ from (14), if we delete the commands " $F \leftarrow \text{LINK}(P)$ " and "if $F = \Lambda$ then set $R \leftarrow \text{LOC}(F)$ " from (17); the latter are to be replaced by "if $F = R$ then $F \leftarrow \Lambda$ and $R \leftarrow \text{LOC}(F)$, otherwise set $F \leftarrow \text{LINK}(P)$ ".

The effect of these changes is that the LINK field of the rear node in the queue will contain spurious information that is never interrogated by the program. A trick like this saves execution time and it is quite useful in practice, although it violates one of the basic assumptions of garbage collection (see Section 2.3.5) so it cannot be used in conjunction with such algorithms.

7. (Make sure that your solution works for empty lists.)

- I1. Set $P \leftarrow \text{FIRST}$, $Q \leftarrow \Lambda$.
- I2. If $P \neq \Lambda$, set $R \leftarrow Q$, $Q \leftarrow P$, $P \leftarrow \text{LINK}(Q)$, $\text{LINK}(Q) \leftarrow R$, and repeat this step.
- I3. Set $\text{FIRST} \leftarrow Q$. ■

In essence we are popping nodes off one stack and pushing them onto another.

8.	LD1 FIRST	1	<u>I1.</u> $P \equiv rI1 \leftarrow \text{FIRST}$.
	ENT2 0	1	$Q \equiv rI2 \leftarrow \Lambda$.
	J1Z 2F	1	<u>I2.</u> If the list is empty, jump.
1H	ENTA 0,2	n	$R \equiv rA \leftarrow Q$.
	ENT2 0,1	n	$Q \leftarrow P$.
	LD1 0,2(LINK)	n	$P \leftarrow \text{LINK}(Q)$.
	STA 0,2(LINK)	n	$\text{LINK}(Q) \leftarrow R$.
	J1NZ 1B	n	Is $P \neq \Lambda$?
2H	ST2 FIRST	1	<u>I3.</u> $\text{FIRST} \leftarrow Q$. ■

The time is $(7n + 6)u$. Better speed $(5n + \text{constant})u$ is attainable; see exercise 1.1-3.

9. (a) Yes. (b) Yes, if biological parenthood is considered; no, if legal parenthood is considered (a man's daughter might marry his father, as in the song "I'm My Own Grampa"). (c) No ($-1 \prec 1$ and $1 \prec -1$). (d) Let us hope so, or else there is a circular argument. (e) $1 \prec 3$ and $3 \prec 1$. (f) The statement is ambiguous. If we take the position that the subroutines called by y are dependent upon which subroutine calls y , we would have to conclude that the transitive law does not necessarily hold. (For example, a general input-output subroutine might call on different processing routines for each I/O device present, but these processing subroutines are usually not all needed in a single program. This is a problem that plagues many automatic programming systems.)

10. For (i) there are three cases: $x = y$; $x \subset y$ and $y = z$; $x \subset y$ and $y \subset z$. For (ii) there are two cases: $x = y$; $x \neq y$. Each case is handled trivially, as is (iii).

11. "Multiply out" the following to get all 52 solutions: $13749(25 + 52)86 + (1379 + 1397 + 1937 + 9137)(4258 + 4528 + 2458 + 5428 + 2548 + 5248 + 2584 + 5284)6 + (1392 + 1932 + 1923 + 9123 + 9132 + 9213)7(458 + 548 + 584)6$.

12. For example: (a) List all sets with k elements (in any order) before all sets with $k+1$ elements, $0 \leq k < n$. (b) Represent a subset by a sequence of 0s and 1s showing which elements are in the set. This gives a correspondence between all subsets and the integers 0 through $2^n - 1$, via the binary number system. The order of correspondence is a topological sequence.
13. Sha and Kleitman, *Discrete Math.* 63 (1987), 271–278, have proved that the number is at most $\prod_{k=0}^n \binom{n}{k}$. This exceeds the obvious lower bound $\prod_{k=0}^n \binom{n}{k}! = 2^{2^n(n+O(1))}$ by a factor of $2^{2^n+O(n)}$; they conjecture that the lower bound is closer to the truth.

14. If $a_1 a_2 \dots a_n$ and $b_1 b_2 \dots b_n$ are two possible topological sorts, let j be minimal such that $a_j \neq b_j$; then $a_k = b_j$ and $a_j = b_m$ for some $k, m > j$. Now $b_j \not\leq a_j$ since $k > j$, and $a_j \not\leq b_j$ since $m > j$, hence (iv) fails. Conversely if there is only one topological sort $a_1 a_2 \dots a_n$, we must have $a_j \leq a_{j+1}$ for $1 \leq j < n$, since otherwise a_j and a_{j+1} could be interchanged. This and transitivity imply (iv).

Note: The following alternative proofs work also for infinite sets. (a) Every partial ordering can be embedded in a linear ordering. For if we have two elements with $x_0 \not\leq y_0$ and $y_0 \not\leq x_0$ we can generate another partial ordering by the rule " $x \leq y$ or $x \leq z_0$ and $y_0 \leq y$ ". The latter ordering "includes" the former and has $x_0 \leq y_0$. Now apply Zorn's lemma or transfinite induction in the usual way to complete the proof. (b) Obviously a linear ordering cannot be embedded in any different linear ordering. (c) A partial ordering that has incomparable elements x_0 and y_0 as in (a) can be extended to two linear orderings in which $x_0 \leq y_0$ and $y_0 \leq x_0$, respectively, so at least two linear orderings exist.

15. If S is finite, we can list all relations $a \prec b$ that are true in the given partial ordering. By successively removing, one at a time, any relations that are implied by others, we arrive at an irredundant set. The problem is to show there is just one such set, no matter in what order we go about removing redundant relations. If there were two irredundant sets U and V , in which " $a \prec b$ " appears in U but not in V , there are $k+1$ relations $a \prec c_1 \prec \dots \prec c_k \prec b$ in V for some $k \geq 1$. But it is possible to deduce $a \prec c_1$ and $c_1 \prec b$ from U , without using the relation $a \prec b$ (since $b \not\leq c_1$ and $c_1 \not\leq a$), hence the relation $a \prec b$ is redundant in U .

The result is false for infinite sets S , when there is *at most* one irredundant set of relations. For example if S denotes the integers plus the element ∞ and we define $n \prec n+1$ and $n \prec \infty$ for all n , there is no irredundant set of relations that characterizes this partial ordering.

16. Let $x_{p_1} x_{p_2} \dots x_{p_n}$ be a topological sorting of S ; apply the permutation $p_1 p_2 \dots p_n$ to both rows and columns.
17. If k increases from 1 to n in step T4, the output is 1932745860. If k decreases from n to 1 in step T4, as it does in Program T, the output is 9123745860.
18. They link together the items in sorted order: QLINK[0] first, QLINK[QLINK[0]] second, and so on; QLINK[last] = 0.
19. This would fail in certain cases; when the queue contains only one element in step T5, the modified method would set $F = 0$ (thereby emptying the queue), but other entries could be placed in the queue in step T6. The suggested modification would therefore require an additional test of $F = 0$ in step T6.
20. Indeed, a stack *could* be used, in the following way. (Step T7 disappears.)

- T4.** Set $T \leftarrow 0$. For $1 \leq k \leq n$ if $COUNT[k]$ is zero do the following: Set $SLINK[k] \leftarrow T$, $T \leftarrow k$. ($SLINK[k] \equiv QLINK[k]$.)
- T5.** Output the value of T . If $T=0$, go to T8; otherwise, set $N \leftarrow N - 1$, $P \leftarrow TOP[T]$, $T \leftarrow SLINK[T]$.
- T6.** Same as before, except go to T5 instead of T7; and when $COUNT[SUC(P)]$ goes down to zero, set $SLINK[SUC(P)] \leftarrow T$ and $T \leftarrow SUC(P)$.
- 21.** Repeated relations only make the algorithm a little slower and take up more space in the storage pool. A relation " $j \prec j$ " would be treated like a loop (an arrow from a box to itself in the corresponding diagram), which violates partial order.
- 22.** To make the program "fail-safe" we should (a) check that $0 < n <$ some appropriate maximum; (b) check each relation $j \prec k$ for the conditions $0 < j, k \leq n$; (c) make sure that the number of relations doesn't overflow the storage pool area.
- 23.** At the end of step T5, add " $TOP[F] \leftarrow \Lambda$ ". (Then at all times $TOP[1], \dots, TOP[n]$ point to all the relations not yet canceled.) In step T8, if $N > 0$, print "LOOP DETECTED IN INPUT:", and set $QLINK[k] \leftarrow 0$ for $1 \leq k \leq n$. Now add the following steps:
- T9.** For $1 \leq k \leq n$ set $P \leftarrow TOP[k]$, $TOP[k] \leftarrow 0$, and perform step T10. (This will set $QLINK[j]$ to one of the predecessors of object j , for each j not yet output.) Then go to T11.
- T10.** If $P \neq \Lambda$, set $QLINK[SUC(P)] \leftarrow k$, $P \leftarrow NEXT(P)$, and repeat this step.
- T11.** Find a k with $QLINK[k] \neq 0$.
- T12.** Set $TOP[k] \leftarrow 1$ and $k \leftarrow QLINK[k]$. Now if $TOP[k] = 0$, repeat this step.
- T13.** (We have found the start of a loop.) Print the value of k , set $TOP[k] \leftarrow 0$, $k \leftarrow QLINK[k]$, and if $TOP[k] = 1$ repeat this step.
- T14.** Print the value of k (the beginning and end of the loop) and stop. (Note: The loop has been printed backwards; if it is desired to print the loop in forward order, an algorithm like that in exercise 7 should be used between steps T12 and T13.) ■
- 24.** Insert three lines in the program of the text:

```
08a PRINTER EQU 18
14a      ST6 NO
59a      STZ X,1(TOP)      TOP[F] ← Λ.
```

Replace lines 74–75 by the following:

```
74      J6Z DONE
75      OUT LINE1(PRINTER)  Print indication of loop.
76      LD6 NO
77      STZ X,6(QLINK)      QLINK[k] ← 0.
78      DEC6 1
79      J6P *-2            n ≥ k ≥ 1.
80      LD6 NO
81  T9      LD2 X,6(TOP)    P ← TOP[k].
82      STZ X,6(TOP)      TOP[k] ← 0.
83      J2Z T9A            Is P = Λ?
84  T10     LD1 0,2(SUC)    r11 ← SUC(P).
85      ST6 X,1(QLINK)    QLINK[r11] ← k.
86      LD2 0,2(NEXT)      P ← NEXT(P).
```

Also we delete nodes from all three lists in several places when we do not know the predecessor or successor of the node being deleted. Only the ELEVATOR list could be converted to a one-way list, without much loss of efficiency.

Note: It may be preferable to use a nonlinear list as the WAIT list in a discrete simulator, to reduce the time for sorting in. Section 5.2.3 discusses the general problem of maintaining priority queues, or "smallest in, first out" lists, such as this. Several ways are known in which only $O(\log n)$ operations are needed to insert or delete when there are n elements in the list, although there is of course no need for such a fancy method when n is known to be small.

SECTION 2.2.6

1. (Here the indices run from 1 to n , not from 0 to n as in Eq. (6).) $\text{LOC}(A[J,K]) = \text{LOC}(A[0,0]) + 2nJ + 2K$, where $A[0,0]$ is an assumed node that is actually nonexistent. If we set $J = K = 1$, we get $\text{LOC}(A[1,1]) = \text{LOC}(A[0,0]) + 2n + 2$, so the answer can be expressed in several ways. The fact that $\text{LOC}(A[0,0])$ might be negative has led to many bugs in compilers and loading routines.

2. $\text{LOC}(A[I_1, \dots, I_k]) = \text{LOC}(A[0, \dots, 0]) + \sum_{1 \leq r \leq k} a_r I_r = \text{LOC}(A[l_1, \dots, l_k]) + \sum_{1 \leq r \leq k} a_r I_r - \sum_{1 \leq r \leq k} a_r l_r$, where $a_r = c \prod_{r < s \leq k} (l_s - l_r + 1)$.

Note: For a generalization to the structures occurring in programming languages such as C, and a simple algorithm to compute the relevant constants, see P. Deuel, CACM 9 (1966), 344–347.

3. $1 \leq k \leq j \leq n$ if and only if $0 \leq k-1 \leq j-1 \leq n-1$; so replace k, j, n respectively by $k-1, j-1, n-1$ in all formulas derived for lower bound zero.

$$4. \text{LOC}(A[J,K]) = \text{LOC}(A[0,0]) + nJ - J(J-1)/2 + K.$$

5. Let $A0 = \text{LOC}(A[0,0])$. There are at least two solutions, assuming that J is in rI1 and K is in rI2. (i) "LDA TA2,1:7", where location TA2+j is "NOP j+1*j/2+A0,2"; (ii) "LDA C1,7:2", where location C1 contains "NOP TA,1:7" and location TA+j says "NOP j+1*j/2+A0". The latter takes one more cycle but doesn't tie the table down to index register 2.

$$6. (a) \text{LOC}(A[I,J,K]) = \text{LOC}(A[0,0,0]) + \binom{I+2}{3} + \binom{J+1}{2} + \binom{K}{1}.$$

$$(b) \text{LOC}(B[I,J,K]) = \text{LOC}(B[0,0,0])$$

$$+ \binom{n+3}{3} - \binom{n+3-I}{3} + \binom{n+2-I}{2} - \binom{n+2-J}{2} + K - J,$$

hence the stated form is possible in this case also.

$$7. \text{LOC}(A[I_1, \dots, I_k]) = \text{LOC}(A[0, \dots, 0]) + \sum_{1 \leq r \leq k} \binom{I_r+k-r}{1+k-r}. \text{ See exercise 1.2.6-56.}$$

8. (Solution by P. Nash.) Let $X[I,J,K]$ be defined for $0 \leq I \leq n, 0 \leq J \leq n+1, 0 \leq K \leq n+2$. We can let $A[I,J,K] = X[I,J,K]; B[I,J,K] = X[J,I+1,K]; C[I,J,K] = X[I,K,J+1]; D[I,J,K] = X[J,K,I+2]; E[I,J,K] = X[K,I+1,J+1]; F[I,J,K] = X[K,J+1,I+2]$. This scheme is the best possible, since it packs the $(n+1)(n+2)(n+3)$ elements of the six tetrahedral arrays into consecutive locations with no overlap. *Proof:* A and B exhaust all cells $X[i,j,k]$ with $k = \min(i,j,k)$; C and D exhaust all cells with $j = \min(i,j,k) \neq k$; E and F exhaust all cells with $i = \min(i,j,k) \neq j, k$.

(The construction generalizes to m dimensions, if anybody ever wants to pack the elements of $m!$ generalized tetrahedral arrays into $(n+1)(n+2)\dots(n+m)$ consecutive

locations. Associate a permutation $a_1 a_2 \dots a_m$ with each array, and store its elements in $X[I_{a_1} + b_1, I_{a_2} + b_2, \dots, I_{a_m} + b_m]$, where $b_1 b_2 \dots b_m$ is the inversion table for $a_1 a_2 \dots a_m$ as defined in Section 5.2.1.)

9. G1. Set pointer variables P1, P2, P3, P4, P5, P6 to the first locations of the lists FEMALE, A21, A22, A23, BLOND, BLUE, respectively. Assume in what follows that the end of each list is given by link Λ , and Λ is smaller than any other link. If $P6 = \Lambda$, stop (the list, unfortunately, is empty).
- G2. (Many possible orderings of the following actions could be done; we have chosen to examine EYES first, then HAIR, then AGE, then SEX.) Set $P5 \leftarrow \text{HAIR}(P5)$ zero or more times until $P5 \leq P6$. If now $P5 < P6$, go to step G5.
- G3. Set $P4 \leftarrow \text{AGE}(P4)$ repeatedly if necessary until $P4 \leq P6$. Similarly do the same to P3 and P2 until $P3 \leq P6$ and $P2 \leq P6$. If now P4, P3, P2 are all smaller than P6, go to G5.
- G4. Set $P1 \leftarrow \text{SEX}(P1)$ until $P1 \leq P6$. If $P1 = P6$, we have found one of the young ladies desired, so output her address, P6. (Her age can be determined from the settings of P2, P3, and P4.)
- G5. Set $P6 \leftarrow \text{EYES}(P6)$. Now stop if $P6 = \Lambda$; otherwise return to G2. ■

This algorithm is interesting but not the best way to organize a list for such a search.

10. See Section 6.5.

11. At most $200 + 200 + 3 \cdot 4 \cdot 200 = 2800$ words.

12. $\text{VAL}(Q0) = c$, $\text{VAL}(P0) = b/a$, $\text{VAL}(P1) = d$.

13. It is convenient to have at the end of each list a sentinel that "compares low" in some field on which the list is ordered. A straight one-way list could have been used, for example by retaining just the LEFT links in $\text{BASEROW}[i]$ and the UP links in $\text{BASECOL}[j]$, by modifying Algorithm S thus: In S2, test if $P0 = \Lambda$ before setting $J \leftarrow \text{COL}(P0)$; if so, set $P0 \leftarrow \text{LOC}(\text{BASEROW}[IO])$ and go to S3. In S3, test if $Q0 = \Lambda$; if so, terminate. Step S4 should change by analogy with step S2. In S5, test if $P1 = \Lambda$; if so, act as if $\text{COL}(P1) < 0$. In S6, test if $\text{UP}(\text{PTR}[J]) = \Lambda$; if so, act as if its ROW field were negative.

These modifications make the algorithm more complicated and save no storage space except a ROW or COL field in the list heads (which in the case of MIX is no saving at all).

14. One could first link together those columns that have a nonzero element in the pivot row, so that all other columns could be skipped as we pivot on each row. Rows in which the pivot column is zero are skipped over immediately.

15. Let $rI1 \equiv \text{PIVOT}, J, rI2 \equiv P0, rI3 \equiv Q0, rI4 \equiv P, rI5 \equiv P1, X; \text{LOC}(\text{BASEROW}[i]) \equiv \text{BROW} + i; \text{LOC}(\text{BASECOL}[j]) \equiv \text{BCOL} + j; \text{PTR}[j] \equiv \text{BCOL} + j(1:3)$.

```

01 ROW      EQU 0:3
02 UP       EQU 4:5
03 COL      EQU 0:3
04 LEFT     EQU 4:5
05 PTR      EQU 1:3
06 PIVOTSTEP STJ 9F          Subroutine entrance, rI1 = PIVOT
07 S1       LD2 0,1(ROW)    S1. Initialize.
08           ST2 IO          IO ← ROW(PIVOT).
09           LD3 1,1(COL)

```

after the final permutations, we have the answer

$$\begin{pmatrix} 1 & -2 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}.$$

20. $a_0 = \text{LOC}(A[1,1]) - 3$, $a_1 = 1$ or 2 , $a_2 = 3 - a_1$.

21. For example, $M \leftarrow \max(I, J)$, $\text{LOC}(A[I, J]) = \text{LOC}(A[1, 1]) + M(M-1) + I - J$. (Such formulas have been proposed independently by many people. A. L. Rosenberg and H. R. Strong have suggested the following k -dimensional generalization: $\text{LOC}(A[I_1, \dots, I_k]) = L_k$ where $L_1 = \text{LOC}(A[1, \dots, 1]) + I_1 - 1$, $L_r = L_{r-1} + (M_r - I_r) \times (M_r^{r-1} - (M_r - 1)^{r-1})$, and $M_r = \max(I_1, \dots, I_r)$ [IBM Tech. Disclosure Bull. 14 (1972), 3026–3028]. See Current Trends in Programming Methodology 4 (Prentice-Hall, 1978), 263–311, for further results of this kind.)

22. According to the combinatorial number system (exercise 1.2.6–56), we can let

$$p(i_1, \dots, i_k) = \binom{i_1}{1} + \binom{i_1 + i_2 + 1}{2} + \dots + \binom{i_1 + i_2 + \dots + i_k + k - 1}{k}.$$

[Det Kongelige Norske Videnskabers Selskabs Forhandlinger 34 (1961), 8–9.]

23. Let $c[J] = \text{LOC}(A[0, J]) = \text{LOC}(A[0, 0]) + mJ$, if there were m rows when the matrix grew from J to $J+1$ columns; similarly, let $r[I] = \text{LOC}(A[I, 0]) = \text{LOC}(A[0, 0]) + nI$, if there were n columns when we created row I . Then we can use the allocation function

$$\text{LOC}(A[I, J]) = \begin{cases} I + c[J], & \text{if } c[J] \geq r[I]; \\ J + r[I], & \text{otherwise.} \end{cases}$$

It is not hard to prove that $c[J] \geq r[I]$ implies $c[J] \geq r[I] + J$, and $c[J] \leq r[I]$ implies $c[J] + I \leq r[I]$; therefore the relation

$$\text{LOC}(A[I, J]) = \max(I + \text{LOC}(A[0, J]), J + \text{LOC}(A[I, 0]))$$

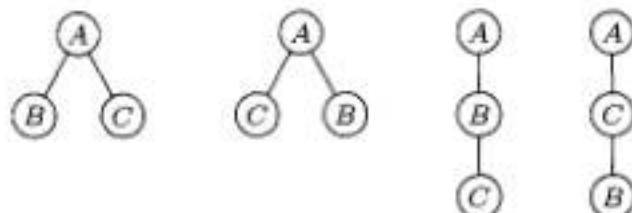
also holds. We need not restrict allocation to mn consecutive locations; the only constraint is that, when the matrix grows, we allocate m or n consecutive new cells in locations greater than those previously used. This construction is due to E. J. Otoo and T. H. Merrett [Computing 31 (1983), 1–9], who also generalized it to k dimensions.

24. [Aho, Hopcroft, and Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, 1974), exercise 2.12.] Besides the array A , maintain also a verification array V of the same size, and a list L of the locations used. Let n be the number of items in L ; initially $n = 0$ and the contents of L , A , and V are arbitrary. Whenever you want to access $A[k]$ for a value of k that you might not have used before, first check whether $0 \leq V[k] < n$ and $L[V[k]] = k$. If not, set $V[k] \leftarrow n$, $L[n] \leftarrow k$, $A[k] \leftarrow 0$, and $n \leftarrow n + 1$. Otherwise you can be sure that $A[k]$ already contains legitimate data. (By a slight extension of this method, it is possible to save and eventually restore the contents of all entries of A and V that change during the computation.)

SECTION 2.3

- There are three ways to choose the root. Once the root has been chosen, say A , there are three ways to partition the other nodes into subtrees: $\{B\}, \{C\}; \{C\}, \{B\}; \{B, C\}$. In the latter case there are two ways to make $\{B, C\}$ into a tree, depending

on which is the root. Hence we get the four trees shown when A is the root, and 12 in all. This problem is solved for any number n of nodes in exercise 2.3.4.4–23.



2. The first two trees in the answer to exercise 1 are the same, as oriented trees, so we get only 9 different possibilities in this case. For the general solution, see Section 2.3.4.4, where the formula n^{n-1} is proved.

3. Part 1: To show there is at least one such sequence. Let the tree have n nodes. The result is clear when $n = 1$, since X must be the root. If $n > 1$, the definition implies there is a root X_1 and subtrees T_1, T_2, \dots, T_m ; either $X = X_1$ or X is a member of a unique T_j . In the latter case, there is by induction a path X_2, \dots, X where X_2 is the root of T_j , and since X_1 is the parent of X_2 we have a path X_1, X_2, \dots, X .

Part 2: To show there is at most one such sequence. We will prove by induction that if X is not the root of the tree, X has a unique parent (so that X_k determines X_{k-1} determines X_{k-2} , etc.) If the tree has one node, there is nothing to prove; otherwise X is in a unique T_j . Either X is the root of T_j , in which case X has a unique parent by definition; or X is not the root of T_j , in which case X has a unique parent in T_j by induction, and no node outside of T_j can be X 's parent.

4. True (unfortunately).

5. 4.

6. Let $\text{parent}^{[0]}(X)$ denote X , and let $\text{parent}^{[k+1]}(X) = \text{parent}(\text{parent}^{[k]}(X))$, so that $\text{parent}^{[1]}(X)$ is X 's parent, and $\text{parent}^{[2]}(X)$ is X 's grandparent; when $k \geq 2$, $\text{parent}^{[k]}(X)$ is X 's “(great-) $k-2$ grandparent.” The requested cousinship condition is that $\text{parent}^{[m+1]}(X) = \text{parent}^{[m+n+1]}(Y)$ but $\text{parent}^{[m]}(X) \neq \text{parent}^{[m+n]}(Y)$. When $n > 0$, this relation is not symmetrical with respect to X and Y , although people usually treat it as symmetrical in everyday conversation.

7. Use the (unsymmetric) condition defined in exercise 6, with the convention that $\text{parent}^{[j]}(X) \neq \text{parent}^{[k]}(Y)$ if either j or k (or both) is -1 . To show that this relation is always valid for some unique m and n , consider the Dewey decimal notation for X and Y , namely $1.a_1 \dots a_p.b_1 \dots b_q$ and $1.a_1 \dots a_p.c_1 \dots c_r$, where $p \geq 0$, $q \geq 0$, $r \geq 0$ and (if $qr \neq 0$) $b_1 \neq c_1$. The Dewey numbers of any pair of nodes can be written in this form, and clearly we must take $m = q - 1$ and $m + n = r - 1$.

8. No binary tree is really a tree; the concepts are quite separate, even though the diagram of a nonempty binary tree may look treelike.

9. A is the root, since we conventionally put the root at the top.

10. Any finite collection of nested sets corresponds to a forest as defined in the text, as follows: Let A_1, \dots, A_n be the sets of the collection that are contained in no other. For fixed j , the sub-collection of all sets contained in A_j is nested; hence we may assume that this sub-collection corresponds to a tree (unordered) with A_j as the root.

11. In a nested collection C let $X \equiv Y$ if there is some $Z \in C$ such that $X \cup Y \subseteq Z$. This relation is obviously reflexive and symmetric, and it is in fact an equivalence relation since $W \equiv X$ and $X \equiv Y$ implies that there are Z_1 and Z_2 in C with $W \subseteq Z_1$, $X \subseteq Z_1 \cap Z_2$, and $Y \subseteq Z_2$. Since $Z_1 \cap Z_2 \neq \emptyset$, either $Z_1 \subseteq Z_2$ or $Z_2 \subseteq Z_1$; hence $W \cup Y \subseteq Z_1 \cup Z_2 \in C$. Now if C is a nested collection, define an oriented forest

W4. [Visit.] Visit $\text{NODE}(P)$.

W5. [Rotate.] Set $R \leftarrow \text{LLINK}(P)$, $\text{LLINK}(P) \leftarrow \text{RLINK}(P)$, $\text{RLINK}(P) \leftarrow Q$, $Q \leftarrow P$, $P \leftarrow R$, and return to W2. ■

Correctness follows from the fact that if we start at W2 with P pointing to the root of a binary tree T and Q pointing to X, where X is not a link in that tree, the algorithm will triple traverse the tree and reach step W3 with $P = X$ and $Q = T$.

If $\alpha(T) = x_1x_2\dots x_{3n}$ is the resulting sequence of nodes in triple order, we have $\alpha(T) = T \alpha(\text{LLINK}(T)) T \alpha(\text{RLINK}(T)) T$. Therefore, as Lindstrom observed, the three subsequences $x_1x_4\dots x_{3n-2}$, $x_2x_5\dots x_{3n-1}$, $x_3x_6\dots x_{3n}$ each include every tree node just once. (Since x_{j+1} is either the parent or child of x_j , these subsequences visit the nodes in such a way that each is at most three links away from its predecessor. Section 7.2.1 describes a general traversal scheme called *prepostorder* that has this property not only for binary trees but for trees in general.)

22. This program uses the conventions of Programs T and S, with Q in rI6 and/or rI4. The old-fashioned MIX computer is not good at comparing index registers for equality, so variable R is omitted and the test " $Q = R$ " is changed to " $\text{RLINK}(Q) = P$ ".

01	U1	LD5 HEAD(LLINK)	1	<u>U1. Initialize.</u> $P \leftarrow T$.
02	U2A	J5Z DONE	1	Stop if $P = \Lambda$.
03	U3	LD6 0,5(LLINK)	$n+a-1$	<u>U3. Look left.</u> $Q \leftarrow \text{LLINK}(P)$.
04		J6Z U6	$n+a-1$	To U6 if $Q = \Lambda$.
05	U4	CMP5 1,6(RLINK)	$2n-2b$	<u>U4. Search for thread.</u>
06		JE 5F	$2n-2b$	Jump if $\text{RLINK}(Q) = P$.
07		ENT4 0,6	$2n-2b-a+1$	$rI4 \leftarrow Q$.
08		LD6 1,6(RLINK)	$2n-2b-a+1$	
09		J6NZ U4	$2n-2b-a+1$	To U4 with $Q \leftarrow \text{RLINK}(Q)$ if it's $\neq 0$.
10	U5	ST5 1,4(RLINK)	$a-1$	<u>U5a. Insert thread.</u> $\text{RLINK}(Q) \leftarrow P$.
11	U9	LD5 0,5(LLINK)	$a-1$	<u>U9. Go to left.</u> $P \leftarrow \text{LLINK}(P)$.
12		JMP U3	$a-1$	To U3.
13	5H	STZ 1,6(RLINK)	$a-1$	<u>U5b. Remove thread.</u> $\text{RLINK}(Q) \leftarrow \Lambda$.
14	U6	JMP VISIT	n	<u>U6. Inorder visit.</u>
15	U7	LD5 1,5(RLINK)	n	<u>U7. Go to right or up.</u> $P \leftarrow \text{RLINK}(P)$.
16	U2	J5NZ U3	n	<u>U2. Done?</u> To U3 if $P \neq \Lambda$.
17	DONE	...		

The total running time is $21n + 6a - 3 - 14b$, where n is the number of nodes, a is the number of null RLINKs (hence $a-1$ is the number of nonnull LLINKs), and b is the number of nodes on the tree's "right spine" T, RLINK(T), RLINK(RLINK(T)), etc.

23. Insertion to the right: $\text{RLINK}_T(Q) \leftarrow \text{RLINK}_T(P)$, $\text{RLINK}(P) \leftarrow Q$, $\text{RTAG}(P) \leftarrow 0$, $\text{LLINK}(Q) \leftarrow \Lambda$. Insertion to the left, assuming $\text{LLINK}(P) = \Lambda$: Set $\text{LLINK}(P) \leftarrow Q$, $\text{LLINK}(Q) \leftarrow \Lambda$, $\text{RLINK}(Q) \leftarrow P$, $\text{RTAG}(Q) \leftarrow 1$. Insertion to the left, between P and $\text{LLINK}(P) \neq \Lambda$: Set $R \leftarrow \text{LLINK}(P)$, $\text{LLINK}(Q) \leftarrow R$, and then set $R \leftarrow \text{RLINK}(R)$ zero or more times until $\text{RTAG}(R) = 1$; finally set $\text{RLINK}(R) \leftarrow Q$, $\text{LLINK}(P) \leftarrow Q$, $\text{RLINK}(Q) \leftarrow P$, $\text{RTAG}(Q) \leftarrow 1$.

(A more efficient algorithm for the last case can be used if we know a node F such that $P = \text{LLINK}(F)$ or $P = \text{RLINK}(F)$; assuming the latter, for example, we could set $\text{INFO}(P) \leftrightarrow \text{INFO}(Q)$, $\text{RLINK}(F) \leftarrow Q$, $\text{LLINK}(Q) \leftarrow P$, $\text{RLINK}(P) \leftarrow Q$, $\text{RTAG}(P) \leftarrow 1$; this takes a fixed amount of time, but it is generally not recommended because it switches nodes around in memory.)

24. No:



25. We first prove (b), by induction on the number of nodes in T , and similarly (c). Now (a) breaks into several cases; write $T \preceq_1 T'$ if (i) holds, $T \preceq_2 T'$ if (ii) holds, etc. Then $T \preceq_1 T'$ and $T' \preceq T''$ implies $T \preceq_1 T''$; $T \preceq_2 T'$ and $T' \preceq T''$ implies $T \preceq_2 T''$; and the remaining two cases are treated by proving (a) by induction on the number of nodes in T .

26. If the double order of T is $(u_1, d_1), (u_2, d_2), \dots, (u_{2n}, d_{2n})$ where the u 's are nodes and the d 's are 1 or 2, form the "trace" of the tree $(v_1, s_1), (v_2, s_2), \dots, (v_{2n}, s_{2n})$, where $v_j = \text{info}(u_j)$, and $s_j = l(u_j)$ or $r(u_j)$ according as $d_j = 1$ or 2. Now $T \preceq T'$ if and only if the trace of T (as defined here) lexicographically precedes or equals the trace of T' . Formally, this means that we have either $n \leq n'$ and $(v_j, s_j) = (v'_j, s'_j)$ for $1 \leq j \leq n$, or else there is a k for which $(v_j, s_j) = (v'_j, s'_j)$ for $1 \leq j < k$ and either $v_k < v'_k$ or $v_k = v'_k$ and $s_k < s'_k$.

27. R1. [Initialize.] Set $P \leftarrow \text{HEAD}$, $P' \leftarrow \text{HEAD}'$; these are the respective list heads of the given right-threaded binary trees. Go to R3.

R2. [Check INFO.] If $\text{INFO}(P) \prec \text{INFO}(P')$, terminate ($T \prec T'$); if $\text{INFO}(P) \succ \text{INFO}(P')$, terminate ($T \succ T'$).

R3. [Go to left.] If $\text{LLINK}(P) = \Lambda = \text{LLINK}(P')$, go to R4; if $\text{LLINK}(P) = \Lambda \neq \text{LLINK}(P')$, terminate ($T \prec T'$); if $\text{LLINK}(P) \neq \Lambda = \text{LLINK}(P')$, terminate ($T \succ T'$); otherwise set $P \leftarrow \text{LLINK}(P)$, $P' \leftarrow \text{LLINK}(P')$ and go to R2.

R4. [End of tree?] If $P = \text{HEAD}$ (or, equivalently, if $P' = \text{HEAD}'$), terminate (T is equivalent to T').

R5. [Go to right.] If $\text{RTAG}(P) = 1 = \text{RTAG}(P')$, set $P \leftarrow \text{RLINK}(P)$, $P' \leftarrow \text{RLINK}(P')$, and go to R4. If $\text{RTAG}(P) = 1 \neq \text{RTAG}(P')$, terminate ($T \prec T'$). If $\text{RTAG}(P) \neq 1 = \text{RTAG}(P')$, terminate ($T \succ T'$). Otherwise, set $P \leftarrow \text{RLINK}(P)$, $P' \leftarrow \text{RLINK}(P')$, and go to R2. ■

To prove the validity of this algorithm (and therefore to understand how it works), one may show by induction on the size of the tree T_0 that the following statement is valid: Starting at step R2 with P and P' pointing to the roots of two nonempty right-threaded binary trees T_0 and T'_0 , the algorithm will terminate if T_0 and T'_0 are not equivalent, indicating whether $T_0 \prec T'_0$ or $T_0 \succ T'_0$; the algorithm will reach step R4 if T_0 and T'_0 are equivalent, with P and P' then pointing respectively to the successor nodes of T_0 and T'_0 in symmetric order.

28. Equivalent and similar.

29. Prove by induction on the size of T that the following statement is valid: Starting at step C2 with P pointing to the root of a nonempty binary tree T and with Q pointing to a node that has empty left and right subtrees, the procedure will ultimately arrive at step C6 after setting $\text{INFO}(Q) \leftarrow \text{INFO}(P)$ and attaching copies of the left and right subtrees of $\text{NODE}(P)$ to $\text{NODE}(Q)$, and with P and Q pointing respectively to the preorder successor nodes of the trees T and $\text{NODE}(Q)$.

(b) Because of (a), there are only finitely many j for which $l(T_j)$ cannot be embedded in $l(T_k)$ for any $k > j$. Therefore by taking n_1 larger than any such j we can find a subsequence for which $l(T_{n_1}) \subseteq l(T_{n_2}) \subseteq l(T_{n_3}) \subseteq \dots$.

(c) Now by the result of exercise 2.3.2-22, $r(T_{n_j})$ cannot be embedded in $r(T_{n_k})$ for any $k > j$, else $T_{n_j} \subseteq T_{n_k}$. Therefore $T_1, \dots, T_{n_1-1}, r(T_{n_1}), r(T_{n_2}), \dots$ is a counterexample sequence. But this contradicts the definition of T_{n_1} .

Notes: Kruskal, in *Trans. Amer. Math. Soc.* 95 (1960), 210–225, actually proved a stronger result, using a weaker notion of embedding. His theorem does not follow directly from the infinity lemma, although the results are vaguely similar. Indeed, König himself proved a special case of Kruskal's theorem, showing that there is no infinite sequence of pairwise incomparable n -tuples of nonnegative integers, where comparability means that all components of one n -tuple are \leq the corresponding components of the other [Matematikai és Fizikai Lapok 39 (1932), 27–29]. For further developments, see *J. Combinatorial Theory A* 13 (1972), 297–305. See also N. Dershowitz, *Inf. Proc. Letters* 9 (1979), 212–215, for applications to termination of algorithms.

SECTION 2.3.4.4

$$1. \ln A(z) = \ln z + \sum_{k \geq 1} a_k \ln \left(\frac{1}{1-z^k} \right) = \ln z + \sum_{k,t \geq 1} \frac{a_k z^{kt}}{t} = \ln z + \sum_{t \geq 1} \frac{A(z^t)}{t}.$$

2. By differentiation, and equating the coefficients of z^n , we obtain the identity

$$na_{n+1} = \sum_{k \geq 1} \sum_{d \mid k} da_d a_{n+1-k}.$$

Now interchange the order of summation.

4. (a) $A(z)$ certainly converges at least for $|z| < \frac{1}{4}$, since a_n is less than the number of ordered trees b_{n-1} . Since $A(1)$ is infinite and all coefficients are positive, there is a positive number $\alpha \leq 1$ such that $A(z)$ converges for $|z| < \alpha$, and there is a singularity at $z = \alpha$. Let $\psi(z) = A(z)/z$; since $\psi(z) > e^{z\psi(z)}$, we see that $\psi(z) = m$ implies $z < \ln m/m$, so $\psi(z)$ is bounded and $\lim_{z \rightarrow \alpha^-} \psi(z)$ exists. Thus $\alpha < 1$, and by Abel's limit theorem $a = \alpha \cdot \exp(a + \frac{1}{2}A(\alpha^2) + \frac{1}{3}A(\alpha^3) + \dots)$.

(b) $A(z^2), A(z^3), \dots$ are analytic for $|z| < \sqrt{\alpha}$, and $\frac{1}{2}A(z^2) + \frac{1}{3}A(z^3) + \dots$ converges uniformly in a slightly smaller disk.

(c) If $\partial F/\partial w = a - 1 \neq 0$, the implicit function theorem implies that there is an analytic function $f(z)$ in a neighborhood of $(\alpha, a/\alpha)$ such that $F(z, f(z)) = 0$. But this implies $f(z) = A(z)/z$, contradicting the fact that $A(z)$ is singular at α .

(d) Obvious.

(e) $\partial F/\partial w = A(z) - 1$ and $|A(z)| < A(\alpha) = 1$, since the coefficients of $A(z)$ are all positive. Hence, as in (c), $A(z)$ is regular at all such points.

(f) Near $(\alpha, 1/\alpha)$ we have the identity $0 = \beta(z - \alpha) + (\alpha/2)(w - 1/\alpha)^2 + \text{higher order terms}$, where $w = A(z)/z$; so w is an analytic function of $\sqrt{z - \alpha}$ here by the implicit function theorem. Consequently there is a region $|z| < \alpha_1$ minus a cut $[\alpha, \alpha_1]$ in which $A(z)$ has the stated form. (The minus sign is chosen since a plus sign would make the coefficients ultimately negative.)

(g) Any function of the stated form has coefficient asymptotically $\frac{\sqrt{2\beta}}{\alpha^n} \binom{1/2}{n}$. Note that

$$\binom{3/2}{n} = O\left(\frac{1}{n} \binom{1/2}{n}\right).$$

For further details, and asymptotic values of the number of free trees, see R. Otter, *Ann. Math.* (2) **49** (1948), 583–599.

$$5. \quad c_n = \sum_{j_1+2j_2+\dots=n} \binom{c_1+j_1-1}{j_1} \dots \binom{c_n+j_n-1}{j_n} - c_n, \quad n > 1.$$

Therefore

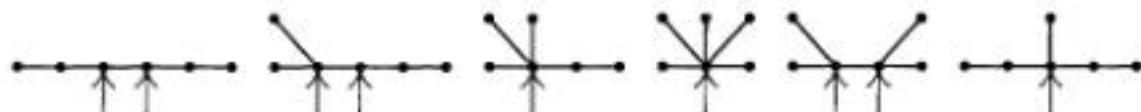
$$2C(z) + 1 - z = (1-z)^{-c_1}(1-z^2)^{-c_2}(1-z^3)^{-c_3}\dots = \exp(C(z) + \frac{1}{2}C(z^2) + \dots).$$

We find $C(z) = z + z^2 + 2z^3 + 5z^4 + 12z^5 + 33z^6 + 90z^7 + 261z^8 + 766z^9 + \dots$. When $n > 1$, the number of series-parallel networks with n edges is $2c_n$ [see P. A. MacMahon, *Proc. London Math. Soc.* **22** (1891), 330–339].

6. $zG(z)^2 = 2G(z) - 2 - zG(z^2)$; $G(z) = 1 + z + z^2 + 2z^3 + 3z^4 + 6z^5 + 11z^6 + 23z^7 + 46z^8 + 98z^9 + \dots$. The function $F(z) = 1 - zG(z)$ satisfies the simpler relation $F(z^2) = 2z + F(z)^2$. [J. H. M. Wedderburn, *Annals of Math.* **24** (1922), 121–140.]

7. $g_n = ca^n n^{-3/2}(1 + O(1/n))$, where $c \approx 0.7916031835775$, $a \approx 2.483253536173$.

8.



9. If there are two centroids, by considering a path from one to the other we find that there can't be intermediate points, so any two centroids are adjacent. A tree cannot contain three mutually adjacent vertices, so there are at most two.

10. If X and Y are adjacent, let $s(X, Y)$ be the number of vertices in the Y -subtree of X . Then $s(X, Y) + s(Y, X) = n$. The argument in the text shows that if Y is a centroid, $\text{weight}(X) = s(X, Y)$. Therefore if both X and Y are centroids, $\text{weight}(X) = \text{weight}(Y) = n/2$.

In terms of this notation, the argument in the text goes on to show that if $s(X, Y) \geq s(Y, X)$, there is a centroid in the Y subtree of X . So if two free trees with m vertices are joined by an edge between X and Y , we obtain a free tree in which $s(X, Y) = m = s(Y, X)$, and there must be two centroids (namely X and Y).

[It is a nice programming exercise to compute $s(X, Y)$ for all adjacent X and Y in $O(n)$ steps; from this information we can quickly find the centroid(s). An efficient algorithm for centroid location was first given by A. J. Goldman, *Transportation Sci.* **5** (1971), 212–221.]

11. $zT(z)^t = T(z) - 1$; thus $z + T(z)^{-t} = T(z)^{1-t}$. By Eq. 1.2.9–(21), $T(z) = \sum_n A_n(1, -t)z^n$, so the number of t -ary trees is

$$\binom{1+tn}{n} \frac{1}{1+tn} = \binom{tn}{n} \frac{1}{(t-1)n+1}.$$

12. Consider the directed graph that has one arc from V_i to V_j for all $i \neq j$. The matrix A_0 of exercise 2.3.4.2–19 is a combinatorial $(n-1) \times (n-1)$ matrix with $n-1$ on the diagonal and -1 off the diagonal. So its determinant is

$$(n + (n-1)(-1))n^{n-2} = n^{n-2},$$

the number of oriented trees with a given root. (Exercise 2.3.4.2–20 could also be used.)

27. Given a function g from $\{1, 2, \dots, r\}$ to $\{1, 2, \dots, q\}$ such that adding arcs from V_k to $U_{g(k)}$ introduces no oriented cycles, construct a sequence a_1, \dots, a_r as follows: Call vertex V_k "free" if there is no oriented path from V_j to V_k for any $j \neq k$. Since there are no oriented cycles, there must be at least one free vertex. Let b_1 be the smallest integer for which V_{b_1} is free; and assuming that b_1, \dots, b_t have been chosen, let b_{t+1} be the smallest integer different from b_1, \dots, b_t for which $V_{b_{t+1}}$ is free in the graph obtained by deleting the arcs from V_{b_k} to $U_{g(b_k)}$ for $1 \leq k \leq t$. This rule defines a permutation $b_1 b_2 \dots b_r$ of the integers $\{1, 2, \dots, r\}$. Let $a_k = g(b_k)$ for $1 \leq k \leq r$; this defines a sequence such that $1 \leq a_k \leq q$ for $1 \leq k < r$, and $1 \leq a_r \leq p$.

Conversely if such a sequence a_1, \dots, a_r is given, call a vertex V_k "free" if there is no j for which $a_j > p$ and $f(a_j) = k$. Since $a_r \leq p$ there are at most $r - 1$ non-free vertices. Let b_1 be the smallest integer for which V_{b_1} is free; and assuming that b_1, \dots, b_t have been chosen, let b_{t+1} be the smallest integer different from b_1, \dots, b_t for which $V_{b_{t+1}}$ is free with respect to the sequence a_{t+1}, \dots, a_r . This rule defines a permutation $b_1 b_2 \dots b_r$ of the integers $\{1, 2, \dots, r\}$. Let $g(b_k) = a_k$ for $1 \leq k \leq r$; this defines a function such that adding arcs from V_k to $U_{g(k)}$ introduces no oriented cycles.

28. Let f be any of the n^{m-1} functions from $\{2, \dots, m\}$ to $\{1, 2, \dots, n\}$, and consider the directed graph with vertices $U_1, \dots, U_m, V_1, \dots, V_n$ and arcs from U_k to $V_{f(k)}$ for $1 < k \leq m$. Apply exercise 27 with $p = 1, q = m, r = n$, to show that there are m^{n-1} ways to add further arcs from the V 's to the U 's to obtain an oriented tree with root U_1 . Since there is a one-to-one correspondence between the desired set of free trees and the set of oriented trees with root U_1 , the answer is $n^{m-1} m^{n-1}$. [This construction can be extensively generalized; see D. E. Knuth, *Canadian J. Math.* **20** (1968), 1077–1086.]

29. If $y = x^t$, then $(tz)y = \ln y$, and we see that it is sufficient to prove the identity for $t = 1$. Now if $zx = \ln x$ we know by exercise 25 that $x^m = \sum_k E_k(m, 1)z^k$ for nonnegative integers m . Hence

$$\begin{aligned} x^r = e^{zx^r} &= \sum_k \frac{(zxr)^k}{k!} = \sum_{j,k} \frac{r^k z^{k+j} E_j(k, 1)}{k!} = \sum_{j,k} \frac{z^k}{k!} \sum_j \binom{k}{j} j! E_j(k-j, 1) r^{k-j} \\ &= \sum_k \frac{z^k}{k!} \binom{k-1}{j} k^j r^{k-j} = \sum_k z^k E_k(r, 1). \end{aligned}$$

[Exercise 4.7-22 derives considerably more general results.]

30. Each graph described defines a set $C_x \subseteq \{1, \dots, n\}$, where j is in C_x if and only if there is a path from t_j to r_i for some $i \leq x$. For a given C_x each graph described is composed of two independent parts: one of the $x(x + \epsilon_1 z_1 + \dots + \epsilon_n z_n)^{\epsilon_1 + \dots + \epsilon_n - 1}$ graphs on the vertices r_i, s_{jk}, t_j for $i \leq x$ and $j \in C_x$, where $\epsilon_j = [j \in C_x]$, plus one of the $y(y + (1 - \epsilon_1)z_1 + \dots + (1 - \epsilon_n)z_n)^{(1-\epsilon_1)+\dots+(1-\epsilon_n)-1}$ graphs on the remaining vertices.

31. $G(z) = z + G(z)^2 + G(z)^3 + G(z)^4 + \dots = z + G(z)^2/(1 - G(z))$. Hence $G(z) = \frac{1}{4}(1 + z - \sqrt{1 - 6z + z^2}) = z + z^2 + 3z^3 + 11z^4 + 45z^5 + \dots$. [Notes: Another problem equivalent to this one was posed and solved by E. Schröder, *Zeitschrift für Mathematik und Physik* **15** (1870), 361–376, who determined the number of ways to insert nonoverlapping diagonals in a convex $(n + 1)$ -gon. These numbers for $n > 1$ are just half the values obtained in exercise 2.2.1–11, since Pratt's grammar allows the root node of the associated parse tree to have degree one. The asymptotic value is calculated in exercise 2.2.1–12. Curiously, the value $[z^{10}]G(z) = 103049$ seems

to have been calculated already by Hipparchus in the second century B.C., as the number of "affirmative compound propositions that can be made from only ten simple propositions"; see R. P. Stanley, *AMM* 104 (1997), 344–350.]

32. Zero if $n_0 \neq 1 + n_2 + 2n_3 + 3n_4 + \dots$ (see exercise 2.3–21), otherwise

$$(n_0 + n_1 + \dots + n_m - 1)! / n_0! n_1! \dots n_m!.$$

To prove this result we recall that an unlabeled tree with $n = n_0 + n_1 + \dots + n_m$ nodes is characterized by the sequence $d_1 d_2 \dots d_n$ of the degrees of the nodes in postorder (Section 2.3.3). Furthermore such a sequence of degrees corresponds to a tree if and only if $\sum_{j=1}^k (1 - d_j) > 0$ for $0 < k \leq n$. (This important property of Polish postfix notation is readily proved by induction; see Algorithm 2.3.3F with f a function that creates a tree, like the TREE function of Section 2.3.2.) In particular, d_1 must be 0. The answer to our problem is therefore the number of sequences $d_2 \dots d_n$ with n_j occurrences of j for $j > 0$, namely the multinomial coefficient

$$\binom{n-1}{n_0-1, n_1, \dots, n_m},$$

minus the number of such sequences $d_2 \dots d_n$ for which $\sum_{j=2}^k (1 - d_j) < 0$ for some $k \geq 2$.

We may enumerate the latter sequences as follows: Let t be minimal such that $\sum_{j=2}^t (1 - d_j) < 0$; then $\sum_{j=2}^t (1 - d_j) = -s$ where $1 \leq s < d_t$, and we may form the subsequence $d'_2 \dots d'_n = d_{t-1} \dots d_{t-1} d_{t+1} \dots d_n$, which has n_j occurrences of $j \neq d_t$, $n_j - 1$ occurrences of j for $j = d_t$. Now $\sum_{j=2}^k (1 - d'_j)$ is equal to d_t when $k = n$, and equal to $d_t - s$ when $k = t$; when $k < t$, it is

$$\sum_{2 \leq j < t} (1 - d_j) - \sum_{2 \leq j \leq t-k} (1 - d_j) \leq \sum_{2 \leq j < t} (1 - d_j) = d_t - s - 1.$$

It follows that, given s and any sequence $d'_2 \dots d'_n$, the construction can be reversed; hence the number of sequences $d_2 \dots d_n$ that have a given value of d_t and s is the multinomial coefficient

$$\binom{n-1}{n_0, \dots, n_{d_t}-1, \dots, n_m}.$$

The number of sequences $d_2 \dots d_n$ that correspond to trees is therefore obtained by summing over the possible values of d_t and s :

$$\sum_{j=0}^m (1-j) \binom{n-1}{n_0, \dots, n_j-1, \dots, n_m} = \frac{(n-1)!}{n_0! n_1! \dots n_m!} \sum_{j=0}^m (1-j) n_j$$

and the latter sum is 1.

An even simpler proof of this result has been given by G. N. Raney (*Transactions of the American Math. Society* 94 (1960), 441–451). If $d_1 d_2 \dots d_n$ is any sequence with n_j appearances of j , there is precisely one cyclic rearrangement $d_k \dots d_n d_1 \dots d_{k-1}$ that corresponds to a tree, namely the rearrangement where k is maximal such that $\sum_{j=1}^k (1 - d_j)$ is minimal. [This argument in the case of binary trees was apparently first discovered by C. S. Peirce in an unpublished manuscript; see his *New Elements of Mathematics* 4 (The Hague: Mouton, 1976), 303–304. It was discovered in the case of t -ary trees by Dvoretzky and Motzkin, *Duke Math. J.* 14 (1947), 305–313.]

Still another proof, by G. Bergman, inductively replaces $d_k d_{k+1}$ by $(d_k + d_{k+1} - 1)$ if $d_k > 0$ [*Algebra Universalis* 8 (1978), 129–130].

11. The "Dewey" notation is the binary representation of the node number.
 12. By exercise 9, it is the internal path length divided by n , plus 1. (This result holds for general trees as well as binary trees.)
 13. [See J. van Leeuwen, Proc. 3rd International Colloq. Automata, Languages and Programming (Edinburgh University Press, 1976), 382–410.]

H1. [Initialize.] Set $A[m-1+i] \leftarrow w_i$ for $1 \leq i \leq m$. Then set $A[2m] \leftarrow \infty$, $x \leftarrow m$, $i \leftarrow m+1$, $j \leftarrow m-1$, $k \leftarrow m$. (During this algorithm $A[i] \leq \dots \leq A[2m-1]$ is the queue of unused external weights; $A[k] \geq \dots \geq A[j]$ is the queue of unused internal weights, empty if $j < k$; the current left and right pointers are x and y .)

H2. [Find right pointer.] If $j < k$ or $A[i] \leq A[j]$, set $y \leftarrow i$ and $i \leftarrow i+1$; otherwise set $y \leftarrow j$ and $j \leftarrow j-1$.

H3. [Create internal node.] Set $k \leftarrow k-1$, $L[k] \leftarrow x$, $R[k] \leftarrow y$, $A[k] \leftarrow A[x]+A[y]$.

H4. [Done?] Terminate the algorithm if $k = 1$.

H5. [Find left pointer.] (At this point $j \geq k$ and the queues contain a total of k unused weights. If $A[y] < 0$ we have $j = k$, $i = y+1$, and $A[i] > A[j]$.) If $A[i] \leq A[j]$, set $x \leftarrow i$ and $i \leftarrow i+1$; otherwise set $x \leftarrow j$ and $j \leftarrow j-1$. Return to step H2. ■

14. The proof for $k = m-1$ applies with little change. [See SIAM J. Appl. Math. **21** (1971), 518.]

15. Use the combined-weight functions (a) $1 + \max(w_1, w_2)$ and (b) $xw_1 + xw_2$, respectively, instead of $w_1 + w_2$ in (g). [Part (a) is due to M. C. Golumbic, IEEE Trans. C-25 (1976), 1164–1167; part (b) to T. C. Hu, D. Kleitman, and J. K. Tamaki, SIAM J. Appl. Math. **37** (1979), 246–256. Huffman's problem is the limiting case of (b) as $x \rightarrow 1$, since $\sum(1+\epsilon)^{l_j} w_j = \sum w_j + \epsilon \sum w_j l_j + O(\epsilon^2)$.]

D. Stott Parker, Jr., has pointed out that a Huffman-like algorithm will also find the minimum of $w_1x^{l_1} + \dots + w_mx^{l_m}$ when $0 < x < 1$, if the two maximum weights are combined at each step. In particular, the minimum of $w_12^{-l_1} + \dots + w_m2^{-l_m}$, when $w_1 \leq \dots \leq w_m$, is $w_1/2 + \dots + w_{m-1}/2^{m-1} + w_m/2^{m-1}$. See D. E. Knuth, J. Comb. Theory **A32** (1982), 216–224, for further generalizations.

16. Let $l_{m+1} = l'_{m+1} = 0$. Then

$$\sum_{j=1}^m w_j l_j \leq \sum_{j=1}^m w_j l'_j = \sum_{k=1}^m (l'_j - l'_{j+1}) \sum_{j=1}^k w_j \leq \sum_{k=1}^m (l'_j - l'_{j+1}) \sum_{j=1}^k w'_j = \sum_{j=1}^m w'_j l'_j,$$

since $l'_j \geq l'_{j+1}$ as in exercise 4. The same proof holds for many other kinds of optimum trees, including those of exercise 10.

17. (a) This is exercise 14. (b) We can extend $f(n)$ to a concave function $f(x)$, so the stated inequality holds. Now $F(m)$ is the minimum of $\sum_{j=1}^{m-1} f(s_j)$, where the s_j are internal node weights of an extended binary tree on the weights 1, 1, ..., 1. Huffman's algorithm, which constructs the complete binary tree with $m-1$ internal nodes in this case, yields the optimum tree. The choice $k = 2^{\lceil \lg(n/3) \rceil}$ defines a binary tree with the same internal weights, so it yields the minimum in the recurrence, for each n . [SIAM J. Appl. Math. **31** (1976), 368–378.] We can evaluate $F(n)$ in $O(\log n)$ steps; see exercises 5.2.3–20 and 21. If $f(n)$ is convex instead of concave, so that $\Delta^2 f(n) \geq 0$, the solution to the recurrence is obtained when $k = \lfloor n/2 \rfloor$.

The relation $(u, v) = 1$ defines the edges of the triangulation, hence different triangulations yield different friezes. To complete the proof of one-to-one correspondence, we must show that every $(m - 1)$ -rowed frieze pattern of positive integers is obtained in this way from some triangulation.

Given any frieze of $m - 1$ rows, extend it by putting a new row 0 at the top and a new row m at the bottom, both consisting entirely of zeros. Now let the elements of row 0 be called $(0, 0), (1, 1), (2, 2)$, etc., and for all nonnegative integers $u < v \leq u + m$ let (u, v) be the element in the diagonal southeast of (u, u) and in the diagonal southwest of (v, v) . By assumption, condition $(**)$ holds for all $u < v < u + m$. We can in fact extend $(**)$ to the considerably more general relation

$$(t, u)(v, w) + (t, w)(u, v) = (t, v)(u, w) \quad \text{for } t \leq u \leq v \leq w \leq t + m. \quad (***)$$

For if $(***)$ is false, let (t, u, v, w) be a counterexample with the smallest value of $(w - t)m + u - t + w - v$. *Case 1:* $t + 1 < u$. Then $(***)$ holds for $(t, t + 1, v, w), (t, t + 1, u, v)$, and $(t + 1, u, v, w)$, so we find $((t, u)(v, w) + (t, w)(v, u))(t + 1, v) = (t, v)(u, w)(t + 1, v)$; this implies $(t + 1, v) = 0$, a contradiction. *Case 2:* $v + 1 < w$. Then $(***)$ holds for $(t, u, w - 1, w), (u, v, w - 1, w)$, and $(t, u, v, w - 1)$; we obtain a similar contradiction $(u, w - 1) = 0$. *Case 3:* $u = t + 1$ and $w = v + 1$. In this case $(***)$ reduces to $(**)$.

Now we set $u = t + 1$ and $w = t + m$ in $(***)$, obtaining $(t, v) = (v, t + m)$ for $t \leq v \leq t + m$, because $(t + 1, t + m) = 1$ and $(t, t + m) = 0$. We conclude that the entries of any $(m - 1)$ -rowed frieze are periodic: $(u, v) = (v, u + m) = (u + m, v + m) = (v + m, u + 2m) = \dots$.

Every frieze pattern of positive integers contains a 1 in row 2. For if we set $t = 0, v = u + 1$, and $w = u + 2$ in $(***)$ we get $(0, u + 1)(u, u + 2) = (0, u) + (0, u + 2)$, hence $(0, u + 2) - (0, u + 1) \geq (0, u + 1) - (0, u)$ if and only if $(u, u + 2) \geq 2$. This cannot hold for all u in the range $0 \leq u \leq m - 2$, because $(0, 1) - (0, 0) = 1$ and $(0, m) - (0, m - 1) = -1$.

Finally, if $m > 3$ we cannot have two consecutive 1s in row 2, because $(u, u + 2) = (u + 1, u + 3) = 1$ implies $(u, u + 3) = 0$. Therefore we can reduce the frieze to another one with m reduced by 1, as illustrated here for 7 rows reduced to 6:

$$\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ a & b & c & d+1 & 1 & e+1 & y & z & \dots \\ p & q & c+r & d & e & u+y & v & w & \dots \\ u & q+v & r & s & u & q+v & r & s & \dots \\ u+y & v & w & p & q & c+r & d & e & \dots \\ y & z & a & b & c & d+1 & 1 & e+1 & \dots \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \end{array} \quad \begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ a & b & c & d & e & y & z & \dots \\ p & q & r & s & u & v & w & \dots \\ u & v & w & p & q & r & s & \dots \\ y & z & a & b & c & d & e & \dots \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \end{array}$$

The reduced frieze corresponds to a triangulation, by induction, and the unreduced frieze corresponds to attaching one more triangle. [Math. Gazette 57 (1974), 87–94, 175–183; Conway and Guy, *The Book of Numbers* (New York: Copernicus, 1996), 74–76, 96–97, 101–102.]

Notes: This proof demonstrates that the function (u, v) , which we defined on any triangulation via 2×2 matrices, satisfies $(***)$ whenever (t, u, v, w) are edges of the polygon in clockwise order. We can express each (u, v) as a polynomial in the numbers $a_j = (j - 1, j + 1)$; these polynomials are essentially identical to the “continuants” discussed in Section 4.5.3, except for the signs of individual terms. In fact, $(j, k) = i^{1-k+j} K_{k-j-1}(ia_{j+1}, ia_{j+2}, \dots, ia_{k-1})$. Thus $(***)$ is equivalent to Euler’s identity

for continuants in the answer to exercise 4.5.3–32. The matrices L and R have the interesting property that any 2×2 matrix of nonnegative integers with determinant 1 can be expressed uniquely as a product of L 's and R 's.

Many other interesting relationships are present; for example, the numbers in row 2 of an integer frieze count the number of triangles touching each vertex of the corresponding triangulated polygon. The total number of occurrences of $(u, v) = 1$ in the basic region $0 \leq u < v - 1 < m - 1$ and $(u, v) \neq (0, m - 1)$ is the number of diagonals (chords) of the triangulation, namely $m - 3 = n - 1$. The total number of 2s is also $n - 1$, because $(u, v) = 2$ if and only if u_- and v_- are opposing vertices of the two triangles adjacent to a chord.

Another interpretation of (u, v) was found by D. Broline, D. W. Crowe, and I. M. Isaacs [Geometriae Dedicata 3 (1974), 171–176]: It is the number of ways to match the $v - u - 1$ vertices between edges u and $v - 1$ with distinct triangles adjacent to those vertices.

SECTION 2.3.5

1. A List structure is a directed graph in which the arcs leaving each vertex are ordered, and where some of the vertices that have out-degree 0 are designated "atoms." Furthermore there is a vertex S such that there is an oriented path from S to V for all vertices $V \neq S$. (With directions of arcs reversed, S would be a "root.")

2. Not in the same way, since thread links in the usual representation lead back to "PARENT," which is not unique for sub-Lists. The representation discussed in exercise 2.3.4.2–25, or some similar method, could perhaps be used (but this idea has not yet been exploited at the time of writing).

3. As mentioned in the text, we prove also that $P = P_0$ upon termination. If only P_0 is to be marked, the algorithm certainly operates correctly. If $n > 1$ nodes are to be marked, we must have $\text{ATOM}(P_0) = 0$. Step E4 then sets $\text{ALINK}(P_0) \leftarrow \Lambda$ and executes the algorithm with P_0 replaced by $\text{ALINK}(P_0)$ and T replaced by P_0 . By induction (note that since $\text{MARK}(P_0) = 1$, all links to P_0 are equivalent to Λ by steps E4 and E5), we see that ultimately we will mark all nodes on paths that start with $\text{ALINK}(P_0)$ and do not pass through P_0 ; and we will then get to step E6 with $T = P_0$ and $P = \text{ALINK}(P_0)$. Now since $\text{ATOM}(T) = 1$, step E6 restores $\text{ALINK}(P_0)$ and $\text{ATOM}(P_0)$ and we reach step E5. Step E5 sets $\text{BLINK}(P_0) \leftarrow \Lambda$, etc., and a similar argument shows that we will ultimately mark all nodes on paths that start with $\text{BLINK}(P_0)$ and do not pass through P_0 or nodes reachable from $\text{ALINK}(P_0)$. Then we will get to E6 with $T = P_0$, $P = \text{BLINK}(P_0)$, and finally we get to E6 with $T = \Lambda$, $P = P_0$.

4. The program that follows incorporates the suggested improvements in the speed of processing atoms that appear in the text after the statement of Algorithm E.

In steps E4 and E5 of the algorithm, we want to test if $\text{MARK}(Q) = 0$. If $\text{NODE}(Q) = +0$, this is an unusual case that can be handled properly by setting it to -0 and treating it as if it were originally -0 , since it has ALINK and BLINK both Λ . This simplification is not reflected in the timing calculations below.

$r11 \equiv P$, $r12 \equiv T$, $r13 \equiv Q$, and $rX \equiv -1$ (for setting MARKs).

```

01 MARK EQU 0:0
02 ATOM EQU 1:1
03 ALINK EQU 2:3
04 BLINK EQU 4:5

```

- S3.** Set $P \leftarrow \text{ALINK}(R)$. If $P = \Lambda$ or $\text{MARK}(P) = 1$, go to S4. Otherwise set $\text{MARK}(P) \leftarrow 1$. Now if $\text{ATOM}(P) = 1$, go to S4; otherwise set $R \leftarrow P$ and return to S2.
- S4.** If $S = 0$, terminate the algorithm; otherwise set $R \leftarrow \text{STACK}[S]$, $S \leftarrow S - 1$, and go to S2.
- S5.** Set $Q \leftarrow \text{ALINK}(P)$. If $Q = \Lambda$ or $\text{MARK}(Q) = 1$, go to S6. Otherwise set $\text{MARK}(Q) \leftarrow 1$. Now if $\text{ATOM}(Q) = 1$, go to S6; otherwise set $\text{ATOM}(P) \leftarrow 1$, $\text{ALINK}(P) \leftarrow T$, $T \leftarrow P$, $P \leftarrow Q$, go to S5.
- S6.** Set $Q \leftarrow \text{BLINK}(P)$. If $Q = \Lambda$ or $\text{MARK}(Q) = 1$, go to S7; otherwise set $\text{MARK}(Q) \leftarrow 1$. Now if $\text{ATOM}(Q) = 1$, go to S7; otherwise set $\text{BLINK}(P) \leftarrow T$, $T \leftarrow P$, $P \leftarrow Q$, go to S5.
- S7.** If $T = \Lambda$, go to S3. Otherwise set $Q \leftarrow T$. If $\text{ATOM}(Q) = 1$, set $\text{ATOM}(Q) \leftarrow 0$, $T \leftarrow \text{ALINK}(Q)$, $\text{ALINK}(Q) \leftarrow P$, $P \leftarrow Q$, and return to S6. If $\text{ATOM}(Q) = 0$, set $T \leftarrow \text{BLINK}(Q)$, $\text{BLINK}(Q) \leftarrow P$, $P \leftarrow Q$, and return to S7. ■

Reference: CACM 10 (1967), 501–506.

6. From the second phase of garbage collection (or perhaps also the initial phase, if all mark bits are set to zero at that time).
7. Delete steps E2 and E3, and delete “ $\text{ATOM}(P) \leftarrow 1$ ” in E4. Set $\text{MARK}(P) \leftarrow 1$ in step E5 and use “ $\text{MARK}(Q) = 0$ ”, “ $\text{MARK}(Q) = 1$ ” in step E6 in place of the present “ $\text{ATOM}(Q) = 1$ ”, “ $\text{ATOM}(Q) = 0$ ” respectively. The idea is to set the MARK bit only after the left subtree has been marked. This algorithm works even if the tree has overlapping (shared) subtrees, but it does not work for all recursive List structures such as those with $\text{NODE}(\text{ALINK}(Q))$ an ancestor of $\text{NODE}(Q)$. (Note that ALINK of a marked node is never changed.)

8. Solution 1: Analogous to Algorithm E, but simpler.

- F1.** Set $T \leftarrow \Lambda$, $P \leftarrow P_0$.
- F2.** Set $\text{MARK}(P) \leftarrow 1$, and set $P \leftarrow P + \text{SIZE}(P)$.
- F3.** If $\text{MARK}(P) = 1$, go to F5.
- F4.** Set $Q \leftarrow \text{LINK}(P)$. If $Q \neq \Lambda$ and $\text{MARK}(Q) = 0$, set $\text{LINK}(P) \leftarrow T$, $T \leftarrow P$, $P \leftarrow Q$ and go to F2. Otherwise set $P \leftarrow P - 1$ and return to F3.
- F5.** If $T = \Lambda$, stop. Otherwise set $Q \leftarrow T$, $T \leftarrow \text{LINK}(Q)$, $\text{LINK}(Q) \leftarrow P$, $P \leftarrow Q - 1$, and return to F3. ■

A similar algorithm, which sometimes decreases the storage overhead and which avoids all pointers into the middle of nodes, has been suggested by Lars-Erik Thorelli, BIT 12 (1972), 555–568.

Solution 2: Analogous to Algorithm D. For this solution, we assume that the SIZE field is large enough to contain a link address. Such an assumption is probably not justified by the statement of the problem, but it lets us use a slightly faster method than the first solution when it is applicable.

- G1.** Set $T \leftarrow \Lambda$, $\text{MARK}(P_0) \leftarrow 1$, $P \leftarrow P_0 + \text{SIZE}(P_0)$.
- G2.** If $\text{MARK}(P) = 1$, go to G5.
- G3.** Set $Q \leftarrow \text{LINK}(P)$, $P \leftarrow P - 1$.
- G4.** If $Q \neq \Lambda$ and $\text{MARK}(Q) = 0$, set $\text{MARK}(Q) \leftarrow 1$, $S \leftarrow \text{SIZE}(Q)$, $\text{SIZE}(Q) \leftarrow T$, $T \leftarrow Q + S$. Go back to G2.

G5. If $T = \Lambda$, stop. Otherwise set $P \leftarrow T$ and find the first value of $Q = P, P - 1, P - 2, \dots$ for which $\text{MARK}(Q) = 1$; set $T \leftarrow \text{SIZE}(Q)$ and $\text{SIZE}(Q) \leftarrow P - Q$. Go back to G2. ■

- 9.** **H1.** Set $L \leftarrow 0, K \leftarrow M + 1, \text{MARK}(0) \leftarrow 1, \text{MARK}(M + 1) \leftarrow 0$.
- H2.** Increase L by one, and if $\text{MARK}(L) = 1$ repeat this step.
- H3.** Decrease K by one, and if $\text{MARK}(K) = 0$ repeat this step.
- H4.** If $L > K$, go to step H5; otherwise set $\text{NODE}(L) \leftarrow \text{NODE}(K), \text{ALINK}(K) \leftarrow L, \text{MARK}(K) \leftarrow 0$, and return to H2.
- H5.** For $L = 1, 2, \dots, K$ do the following: Set $\text{MARK}(L) \leftarrow 0$. If $\text{ATOM}(L) = 0$ and $\text{ALINK}(L) > K$, set $\text{ALINK}(L) \leftarrow \text{ALINK}(\text{ALINK}(L))$. If $\text{ATOM}(L) = 0$ and $\text{BLINK}(L) > K$, set $\text{BLINK}(L) \leftarrow \text{ALINK}(\text{BLINK}(L))$. ■

See also exercise 2.5-33.

- 10.** **Z1.** [Initialize.] Set $F \leftarrow P_0, R \leftarrow \text{AVAIL}, \text{NODE}(R) \leftarrow \text{NODE}(F), \text{REF}(F) \leftarrow R$. (Here F and R are pointers for a queue set up in the REF fields of all header nodes encountered.)
- Z2.** [Begin new List.] Set $P \leftarrow F, Q \leftarrow \text{REF}(P)$.
- Z3.** [Advance to right.] Set $P \leftarrow \text{RLINK}(P)$. If $P = \Lambda$, go to Z6.
- Z4.** [Copy one node.] Set $Q_1 \leftarrow \text{AVAIL}, \text{RLINK}(Q) \leftarrow Q_1, Q \leftarrow Q_1, \text{NODE}(Q) \leftarrow \text{NODE}(P)$.
- Z5.** [Translate sub-List link.] If $T(P) = 1$, set $P_1 \leftarrow \text{REF}(P)$, and if $\text{REF}(P_1) = \Lambda$ set $\text{REF}(R) \leftarrow P_1, R \leftarrow \text{AVAIL}, \text{REF}(P_1) \leftarrow R, \text{NODE}(R) \leftarrow \text{NODE}(P_1), \text{REF}(Q) \leftarrow R$. If $T(P) = 1$ and $\text{REF}(P_1) \neq \Lambda$, set $\text{REF}(Q) \leftarrow \text{REF}(P_1)$. Go to Z3.
- Z6.** [Move to next List.] Set $\text{RLINK}(Q) \leftarrow \Lambda$. If $\text{REF}(F) \neq R$, set $F \leftarrow \text{REF}(\text{REF}(F))$ and return to Z2. Otherwise set $\text{REF}(R) \leftarrow \Lambda, P \leftarrow P_0$.
- Z7.** [Final cleanup.] Set $Q \leftarrow \text{REF}(P)$. If $Q \neq \Lambda$, set $\text{REF}(P) \leftarrow \Lambda$ and $P \leftarrow Q$ and repeat step Z7. ■

Of course, this use of the REF fields makes it impossible to do garbage collection with Algorithm D; moreover, Algorithm D is ruled out by the fact that the Lists aren't well-formed during the copying.

Several elegant List-moving and List-copying algorithms that make substantially weaker assumptions about List representation have been devised. See D. W. Clark, *CACM* 19 (1976), 352-354; J. M. Robson, *CACM* 20 (1977), 431-433.

- 11.** Here is a pencil-and-paper method that can be written out more formally to answer the problem: First attach a unique name (e.g., a capital letter) to each List in the given set; in the example we might have $A = (a: C, b, a: F), F = (b: D), B = (a: F, b, a: E), C = (b: G), G = (a: C), D = (a: F), E = (b: G)$. Now make a list of pairs of List names that must be proved equal. Successively add pairs to this list until either a contradiction is found because we have a pair that disagrees on the first level (then the originally given Lists are unequal), or until the list of pairs does not imply any further pairs (then the originally given Lists are equal). In the example, this list of pairs would originally contain only the given pair, AB ; then it gets the further pairs CF, EF (by matching A and B), DG (from CF); and then we have a self-consistent set.

To prove the validity of this method, observe that (i) if it returns the answer "unequal", the given Lists are unequal; (ii) if the given Lists are unequal, it returns the answer "unequal"; (iii) it always terminates.

DATE OF PURCHASES. MOVE ITEM OF TRANSACTION OF SALES TO ITEM OF TRANSACTION OF PURCHASES. MOVE QUANTITY OF TRANSACTION OF SALES TO QUANTITY OF TRANSACTION OF PURCHASES. MOVE PRICE OF TRANSACTION OF SALES TO PRICE OF TRANSACTION OF PURCHASES. MOVE TAX OF TRANSACTION OF SALES TO TAX OF TRANSACTION OF PURCHASES.

8. If and only if α or β is an elementary item. (It may be of interest to note that the author failed to handle this case properly in his first draft of Algorithm C, and it actually made the algorithm more complicated.)

9. "MOVE CORRESPONDING α TO β ", if neither α nor β is elementary, is equivalent to the set of statements "MOVE CORRESPONDING A OF α TO A OF β " taken over all names A common to groups α and β . (This is a more elegant way to state the definition than the more traditional and more cumbersome definition of "MOVE CORRESPONDING" given in the text.) We may verify that Algorithm C satisfies this definition, using an inductive proof that steps C2 through C5 will ultimately terminate with $P = P_0$ and $Q = Q_0$. Further details of the proof are filled in as we have done many times before in a "tree induction" (see, for example, the proof of Algorithm 2.3.1T).

10. (a) Set $S_1 \leftarrow \text{LINK}(P_k)$. Then repeatedly set $S_1 \leftarrow \text{PREV}(S_1)$ zero or more times until either $S_1 = \Lambda$ ($\text{NAME}(S) \neq P_k$) or $S_1 = S$ ($\text{NAME}(S) = P_k$). (b) Set $P_1 \leftarrow P$ and then set $P_1 \leftarrow \text{PREV}(P_1)$ zero or more times until $\text{PREV}(P_1) = \Lambda$; do a similar operation with variables Q_1 and Q ; then test if $P_1 = Q_1$. Alternatively, if the Data Table entries are ordered so that $\text{PREV}(P) < P$ for all P , a faster test can be made in an obvious way depending on whether $P > Q$ or not, following the PREV links of the larger to see if the smaller is encountered.

11. A minuscule improvement in the speed of step C4 would be achieved by adding a new link field $\text{SIB}_1(P) \equiv \text{CHILD}(\text{PARENT}(P))$. More significantly, we could modify the CHILD and SIB links so that $\text{NAME}(\text{SIB}(P)) > \text{NAME}(P)$; this would speed up the search in step C3 considerably because it would require only one pass over each family to find the matching members. This change would therefore remove the only "search" present in Algorithm C. Algorithms A and C are readily modified for this interpretation, and the reader may find it an interesting exercise. (However, if we consider the relative frequency of MOVE CORRESPONDING statements and the usual size of family groups, the resulting speedup will not be terribly significant in the translation of actual COBOL programs.)

12. Leave steps B1, B2, B3 unchanged; change the other steps thus:

B4. Set $k \leftarrow k + 1$, $R \leftarrow \text{LINK}(P_k)$.

B5. If $R = \Lambda$, there is no match; set $P \leftarrow \text{PREV}(P)$ and go to B2. If $R < S \leq \text{SCOPE}(R)$, set $S \leftarrow R$ and go to B3. Otherwise set $R \leftarrow \text{PREV}(R)$ and repeat step B5. ■

This algorithm does not adapt to the PL/I convention of exercise 6.

13. Use the same algorithm, minus the operations that set NAME, PARENT, CHILD, and SIB. Whenever removing the top stack entry in step A5, set $\text{SCOPE}(P_1) \leftarrow Q - 1$. When the input is exhausted in step A2, simply set $L \leftarrow 0$ and continue, then terminate the algorithm if $L = 0$ in step A7.

14. The following algorithm, using an auxiliary stack, has steps numbered to show a direct correspondence with the text's algorithm.

C1. Set $P \leftarrow P_0$, $Q \leftarrow Q_0$, and set the stack contents empty.

- C2.** If $\text{SCOPE}(P) = P$ or $\text{SCOPE}(Q) = Q$, output (P, Q) as one of the desired pairs and go to C5. Otherwise put (P, Q) on the stack and set $P \leftarrow P + 1$, $Q \leftarrow Q + 1$.
- C3.** Determine if P and Q point to entries with the same name (see exercise 10(b)). If so, go to C2. If not, let (P_1, Q_1) be the entry at the top of the stack; if $\text{SCOPE}(Q) < \text{SCOPE}(Q_1)$, set $Q \leftarrow \text{SCOPE}(Q) + 1$ and repeat step C3.
- C4.** Let (P_1, Q_1) be the entry at the top of the stack. If $\text{SCOPE}(P) < \text{SCOPE}(P_1)$, set $P \leftarrow \text{SCOPE}(P) + 1$, $Q \leftarrow Q_1 + 1$, and go back to C3. If $\text{SCOPE}(P) = \text{SCOPE}(P_1)$, set $P \leftarrow P_1$, $Q \leftarrow Q_1$ and remove the top entry of the stack.
- C5.** If the stack is empty, the algorithm terminates. Otherwise go to C4. ■

SECTION 2.5

1. In such fortuitous circumstances, a stack-like operation may be used as follows: Let the memory pool area be locations 0 through $M - 1$, and let **AVAIL** point to the lowest free location. To reserve N words, report failure if $\text{AVAIL} + N \geq M$, otherwise set $\text{AVAIL} \leftarrow \text{AVAIL} + N$. To free these N words, just set $\text{AVAIL} \leftarrow \text{AVAIL} - N$.

Similarly, cyclic queue-like operation is appropriate for a first-in-first-out discipline.

2. The amount of storage space for an item of length l is $k \lceil l/(k-b) \rceil$, which has the average value $kL/(k-b) + (1-\alpha)k$, where α is assumed to be $1/2$, independent of k . This expression is a minimum (for real values of k) when $k = b + \sqrt{2bL}$. So choose k to be the integer just above or just below this value, whichever gives the lowest value of $kL/(k-b) + \frac{1}{2}k$. For example, if $b = 1$ and $L = 10$, we would choose $k \approx 1 + \sqrt{20} = 5$ or 6; both are equally good. For much greater detail about this problem, see JACM 12 (1965), 53–70.

4. $rI1 \equiv Q$, $rI2 \equiv P$.

A1	LDA N	$rA \leftarrow N$.
	ENT2 AVAIL	$P \leftarrow \text{LOC}(\text{AVAIL})$.
A2A	ENT1 0,2	$Q \leftarrow P$.
A2	LD2 0,1(LINK)	$P \leftarrow \text{LINK}(Q)$.
	J2N OVERFLOW	If $P = \Lambda$, no room.
A3	CMPA 0,2(SIZE)	
	JG A2A	Jump if $N > \text{SIZE}(P)$.
A4	SUB 0,2(SIZE)	$rA \leftarrow N - \text{SIZE}(P) \equiv K$.
	JANZ *+3	Jump if $K \neq 0$.
	LDX 0,2(LINK)	
	STX 0,1(LINK)	$\text{LINK}(Q) \leftarrow \text{LINK}(P)$.
	STA 0,2(SIZE)	$\text{SIZE}(P) \leftarrow K$.
	LD1 0,2(SIZE)	Optional ending,
	INC1 0,2	sets $rI1 \leftarrow P + K$. ■

5. Probably not. The unavailable storage area just before location P will subsequently become available, and its length will be increased by the amount K ; an increase of 99 would not be negligible.

6. The idea is to try to search in different parts of the **AVAIL** list each time. We can use a “roving pointer,” called **ROVER** for example, which is treated as follows: In step A1, set $Q \leftarrow \text{ROVER}$. After step A4, set $\text{ROVER} \leftarrow \text{LINK}(Q)$. In step A2, when $P = \Lambda$ the first time during a particular execution of Algorithm A, set $Q \leftarrow \text{LOC}(\text{AVAIL})$ and repeat step A2. When $P = \Lambda$ the second time, the algorithm terminates unsuccessfully. In this way **ROVER** will tend to point to a random spot in the **AVAIL** list, and the sizes

will be more balanced. At the beginning of the program, set ROVER \leftarrow LOC(AVAIL); it is also necessary to set ROVER to LOC(AVAIL) everywhere else in the program where the block whose address equals the current setting of ROVER is taken out of the AVAIL list. (Sometimes, however, it is useful to have small blocks at the beginning, as in the strict first-fit method; for example, we might want to keep a sequential stack at the high end of memory. In such cases we can reduce the search time by using trees as suggested in exercise 6.2.3-30.)

7. 2000, 1000 with requests of sizes 800, 1300.

[An example where *worst-fit* succeeds, while *best-fit* fails, has been constructed by R. J. Weiland.]

8. In step A1, also set $M \leftarrow \infty$, $R \leftarrow \Lambda$. In step A2, if $P = \Lambda$ go to A6. In step A3, go to A5 instead of A4. Add new steps as follows:

A5. [Better fit?] If $M > \text{SIZE}(P)$, set $R \leftarrow Q$ and $M \leftarrow \text{SIZE}(P)$. Then set $Q \leftarrow P$ and return to A2.

A6. [Any found?] If $R = \Lambda$, the algorithm terminates unsuccessfully. Otherwise set $Q \leftarrow R$, $P \leftarrow \text{LINK}(Q)$, and go to A4. ■

9. Obviously if we are so lucky as to find $\text{SIZE}(P) = N$, we have a best fit and it is not necessary to search farther. (When there are only very few different block sizes, this occurs rather often.) If a "boundary tag" method like Algorithm C is being used, it is possible to maintain the AVAIL list in sorted order by size; so the length of search could be cut down to half the length of the list or less, on the average. But the best solution is to make the AVAIL list into a balanced tree structure as described in Section 6.2.3, if it is expected to be long.

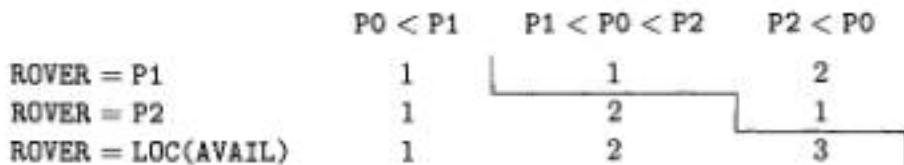
10. Make the following changes:

Step B2, for " $P > P_0$ " read " $P \geq P_0$ ".

Step B3, insert "If $P_0 + N > P$ and $P \neq \Lambda$, set $P \leftarrow \text{LINK}(P)$ and repeat step B3."

Step B4, for " $Q + \text{SIZE}(Q) = P_0$ ", read " $Q + \text{SIZE}(Q) \geq P_0$; and for " $\text{SIZE}(Q) \leftarrow \text{SIZE}(Q) + N$ " read " $\text{SIZE}(Q) \leftarrow P_0 + N - Q$ ".

11. If P_0 is greater than ROVER, we can set $Q \leftarrow \text{ROVER}$ instead of $Q \leftarrow \text{LOC(AVAIL)}$ in step B1. If there are n entries in the AVAIL list, the average number of iterations of step B2 is $(2n+3)(n+2)/6(n+1) = \frac{1}{3}n^2 + \frac{5}{6}n + O(\frac{1}{n})$. For example if $n = 2$ we get 9 equally probable situations, where P_1 and P_2 point to the two existing available blocks:



This chart shows the number of iterations needed in each case. The average is

$$\frac{1}{9} \left(\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \binom{3}{2} + \binom{2}{2} \right) = \frac{1}{9} \left(\binom{5}{3} + \binom{4}{3} \right) = \frac{14}{9}.$$

12. **A1.** Set $P \leftarrow \text{ROVER}$, $F \leftarrow 0$.

A2. If $P = \text{LOC(AVAIL)}$ and $F = 0$, set $P \leftarrow \text{AVAIL}$, $F \leftarrow 1$, and repeat step A2. If $P = \text{LOC(AVAIL)}$ and $F \neq 0$, the algorithm terminates unsuccessfully.

A3. If $\text{SIZE}(P) \geq N$, go to A4; otherwise set $P \leftarrow \text{LINK}(P)$ and return to A2.

Clearly the situation of (2), (3), (4) can't occur here; the only real effect on storage allocation is that the search here will tend to be longer than in exercise 12, and sometimes K will be less than c although there is really another available block preceding this one that we do not know about.

(An alternative is to take the collapsing out of the inner loop A3, and to do the collapsing only in step A4 before the final allocation or in the inner loop when the algorithm would otherwise have terminated unsuccessfully. This alternative requires a simulation study to see if it is an improvement or not.)

[This method, with a few refinements, has proved to be quite satisfactory in the implementations of TeX and METAFONT. See *TeX: The Program* (Addison-Wesley, 1986), §125.]

20. When a buddy is found to be available, during the collapsing loop, we want to remove that block from its $\text{AVAIL}[k]$ list, but we do not know which links to update unless (i) we do a possibly long search, or (ii) the list is doubly linked.

21. If $n = 2^k\alpha$, where $1 \leq \alpha \leq 2$, a_n is $2^{2k+1}(\alpha - \frac{2}{3}) + \frac{1}{3}$, and b_n is $2^{2k-1}\alpha^2 + 2^{k-1}\alpha$. The ratio a_n/b_n for large n is essentially $4(\alpha - \frac{2}{3})/\alpha^2$, which takes its minimum value $\frac{4}{3}$ when $\alpha = 1$ and 2, and its maximum value $\frac{1}{2}$ when $\alpha = 1\frac{1}{2}$. So a_n/b_n approaches no limit; it oscillates between these two extremes. The averaging methods of Section 4.2.4 do, however, yield an average ratio of $4(\ln 2)^{-1} \int_1^2 (\alpha - \frac{2}{3}) d\alpha / \alpha^3 = (\ln 2)^{-1} \approx 1.44$.

22. This idea requires a TAG field in several words of the 11-word block, not only in the first word. It is a workable idea, if those extra TAG bits can be spared, and it would appear to be especially suitable for use in computer hardware.

23. 011011110100; 011011100000.

24. This would introduce a bug in the program; we may get to step S1 when $\text{TAG}(0) = 1$, since S2 may return to S1. To make it work, add "TAG(L) $\leftarrow 0$ " after "L $\leftarrow P$ " in step S2. (It is easier to assume instead that $\text{TAG}(2^m) = 0$.)

25. The idea is absolutely correct. (Criticism need not be negative.) The list heads $\text{AVAIL}[k]$ may be eliminated for $n < k \leq m$; the algorithms of the text may be used if " m " is changed to " n " in steps R1, S1. The initial conditions (13) and (14) should be changed to indicate 2^{m-n} blocks of size 2^n instead of one block of size 2^m .

26. Using the binary representation of M , we can easily modify the initial conditions (13), (14) so that all memory locations are divided into blocks whose size is a power of two, with blocks in decreasing order of size. In Algorithm S, $\text{TAG}(P)$ should be regarded as 0 whenever $P \geq M - 2^k$.

27. r11 $\equiv k$, r12 $\equiv j$, r13 $\equiv j - k$, r14 $\equiv L$, $\text{LOC}(\text{AVAIL}[j]) = \text{AVAIL} + j$; assume that there is an auxiliary table $\text{TWO}[j] = 2^j$, stored in location $\text{TWO} + j$, for $0 \leq j \leq m$. Assume further that "+" and "-" represent tags of 0 and 1, and that $\text{TAG}(\text{LOC}(\text{AVAIL}[j])) = "-";$ but $\text{TAG}(\text{LOC}(\text{AVAIL}[m+1])) = "+"$ is a sentinel.

00	KVAL	EQU	5:5		
01	TAG	EQU	0:0		
02	LINKF	EQU	1:2		
03	LINKB	EQU	3:4		
04	TLNKF	EQU	0:2		
05	R1	LD1	K	1	<u>R1. Find block.</u>
06		ENT2	0,1	1	$j \leftarrow k$.
07		ENT3	0	1	
08		LD4	AVAIL,2(LINKF)	1	

23	ST2	0,4(LINKF)	1	LINKF(L) \leftarrow AVAILF[k].
24	ST1	0,4(KVAL)	1	KVAL(L) \leftarrow k.
25	ST4	0,2(LINKB)	1	LINKB(AVAILF[k]) \leftarrow L.
26	ST4	AVAIL,1(LINKF)	1	AVAIL[k] \leftarrow L. ■

29. Yes, but only at the expense of some searching, or (better) an additional table of TAG bits packed somehow. (It is tempting to suggest that buddies not be joined together during Algorithm S, but only in Algorithm R if there is no block large enough to meet the request; but that would probably lead to a badly fragmented memory.)

31. See David L. Russell, *SICOMP* 6 (1977), 607–621.

33. G1. [Clear LINKs.] Set $P \leftarrow 1$, and repeat the operation $\text{LINK}(P) \leftarrow \Lambda$, $P \leftarrow P + \text{SIZE}(P)$ until $P = \text{AVAIL}$. (This merely sets the LINK field in the first word of each node to Λ ; we may assume in most cases that this step is unnecessary, since $\text{LINK}(P)$ is set to Λ in step G9 below and it can be set to Λ by the storage allocator.)

G2. [Initialize marking phase.] Set $\text{TOP} \leftarrow \text{USE}$, $\text{LINK}(\text{TOP}) \leftarrow \text{AVAIL}$, $\text{LINK}(\text{AVAIL}) \leftarrow \Lambda$. (TOP points to the top of a stack as in Algorithm 2.3.5D.)

G3. [Pop up stack.] Set $P \leftarrow \text{TOP}$, $\text{TOP} \leftarrow \text{LINK}(\text{TOP})$. If $\text{TOP} = \Lambda$, go to G5.

G4. [Put new links on stack.] For $1 \leq k \leq T(P)$, do the following operations: Set $Q \leftarrow \text{LINK}(P + k)$; then if $Q \neq \Lambda$ and $\text{LINK}(Q) = \Lambda$, set $\text{LINK}(Q) \leftarrow \text{TOP}$, $\text{TOP} \leftarrow Q$. Then go back to G3.

G5. [Initialize next phase.] (Now $P = \text{AVAIL}$, and the marking phase has been completed so that the first word of each accessible node has a nonnull LINK. Our next goal is to combine adjacent inaccessible nodes, for speed in later steps, and to assign new addresses to the accessible nodes.) Set $Q \leftarrow 1$, $\text{LINK}(\text{AVAIL}) \leftarrow Q$, $\text{SIZE}(\text{AVAIL}) \leftarrow 0$, $P \leftarrow 1$. (Location AVAIL is being used as a sentinel to signify the end of a loop in subsequent phases.)

G6. [Assign new addresses.] If $\text{LINK}(P) = \Lambda$, go to G7. Otherwise if $\text{SIZE}(P) = 0$, go to G8. Otherwise set $\text{LINK}(P) \leftarrow Q$, $Q \leftarrow Q + \text{SIZE}(P)$, $P \leftarrow P + \text{SIZE}(P)$, and repeat this step.

G7. [Collapse available areas.] If $\text{LINK}(P + \text{SIZE}(P)) = \Lambda$, increase $\text{SIZE}(P)$ by $\text{SIZE}(P + \text{SIZE}(P))$ and repeat this step. Otherwise set $P \leftarrow P + \text{SIZE}(P)$ and return to G6.

G8. [Translate all links.] (Now the LINK field in the first word of each accessible node contains the address to which the node will be moved.) Set $\text{USE} \leftarrow \text{LINK}(\text{USE})$, and $\text{AVAIL} \leftarrow Q$. Then set $P \leftarrow 1$, and repeat the following operation until $\text{SIZE}(P) = 0$: If $\text{LINK}(P) \neq \Lambda$, set $\text{LINK}(Q) \leftarrow \text{LINK}(\text{LINK}(Q))$ for all Q such that $P < Q \leq P + T(P)$ and $\text{LINK}(Q) \neq \Lambda$; then regardless of the value of $\text{LINK}(P)$, set $P \leftarrow P + \text{SIZE}(P)$.

G9. [Move.] Set $P \leftarrow 1$, and repeat the following operation until $\text{SIZE}(P) = 0$: Set $Q \leftarrow \text{LINK}(P)$, and if $Q \neq \Lambda$ set $\text{LINK}(P) \leftarrow \Lambda$ and $\text{NODE}(Q) \leftarrow \text{NODE}(P)$; then whether $Q = \Lambda$ or not, set $P \leftarrow P + \text{SIZE}(P)$. (The operation $\text{NODE}(Q) \leftarrow \text{NODE}(P)$ implies the movement of $\text{SIZE}(P)$ words; we always have $Q \leq P$, so it is safe to move the words in order from smallest location to largest.) ■

[This method is called the “LISP 2 garbage collector.” An interesting alternative, which does not require the LINK field at the beginning of a node, can be based on the idea of linking together all pointers that point to each node—see Lars-Erik Thorelli, *BIT*

37. First sixteen single males enter, and she seats them. There are 17 gaps of empty seats between the occupied seats, counting one gap at each end, with a gap of length zero assumed between adjacent occupied seats. The total number of empty seats, namely the sum of all seventeen gaps, is 6. Suppose x of the gaps are of odd length; then $6 - x$ spaces are available to seat pairs. (Note that $6 - x$ is even and ≥ 0 .) Now each of the customers 1, 3, 5, 7, 9, 11, 13, 15, from left to right, who has an even gap on both sides, finishes his lunch and walks out. Each odd gap prevents at most one of these eight diners from leaving, hence at least $8 - x$ people leave. There still are only $6 - x$ spaces available to seat pairs. But now $(8 - x)/2$ pairs enter.

38. The arguments generalize readily; $N(n, 2) = \lfloor (3n - 1)/2 \rfloor$ for $n \geq 1$. [When the hostess uses a first-fit strategy instead of an optimal one, Robson has proved that the necessary and sufficient number of seats is $\lfloor (5n - 2)/3 \rfloor$.]

39. Divide memory into three independent regions of sizes $N(n_1, m)$, $N(n_2, m)$, and $N(2m - 2, m)$. To process a request for space, put each block into the first region for which the stated capacity is not exceeded, using the relevant optimum strategy for that region. This cannot fail, for if we were unable to fill a request for x locations we must have at least $(n_1 - x + 1) + (n_2 - x + 1) + (2m - x - 1) > n_1 + n_2 - x$ locations already occupied.

Now if $f(n) = N(n, m) + N(2m - 2, m)$, we have the subadditive law $f(n_1 + n_2) \leq f(n_1) + f(n_2)$. Hence $\lim f(n)/n$ exists. (Proof: $f(a + bc) \leq f(a) + bf(c)$; hence $\limsup_{n \rightarrow \infty} f(n)/n = \max_{0 \leq a < c} \limsup_{b \rightarrow \infty} f(a + bc)/(a + bc) \leq f(c)/c$ for all c ; hence $\limsup_{n \rightarrow \infty} f(n)/n \leq \liminf_{n \rightarrow \infty} f(n)/n$.) Therefore $\lim N(n, m)/n$ exists.

[From exercise 38 we know that $N(2) = \frac{3}{2}$. The value $N(m)$ is not known for any $m > 2$. It is not difficult to show that the multiplicative factor for just two block sizes, 1 and b , is $2 - 1/b$; hence $N(3) \geq 1\frac{2}{3}$. Robson's methods imply that $N(3) \leq 1\frac{11}{12}$, and $2 \leq N(4) \leq 2\frac{1}{6}$.]

40. Robson has proved that $N(2^r) \leq 1 + r$, by using the following strategy: Allocate to each block of size k , where $2^m \leq k < 2^{m+1}$, the first available block of k locations starting at a multiple of 2^m .

Let $N(\{b_1, b_2, \dots, b_n\})$ denote the multiplicative factor when all block sizes are constrained to lie in the set $\{b_1, b_2, \dots, b_n\}$, so that $N(n) = N(\{1, 2, \dots, n\})$. Robson and S. Krogdahl have discovered that $N(\{b_1, b_2, \dots, b_n\}) = n - (b_1/b_2 + \dots + b_{n-1}/b_n)$ whenever b_i is a multiple of b_{i-1} for $1 < i \leq n$; indeed, Robson has established the exact formula $N(2^r m, \{1, 2, 4, \dots, 2^r\}) = 2^r m(1 + \frac{1}{2}r) - 2^r + 1$. Thus in particular, $N(n) \geq 1 + \frac{1}{2}\lfloor \lg n \rfloor$. He also has derived the upper bound $N(n) \leq 1.1825 \ln n + O(1)$, and he conjectures tentatively that $N(n) = H_n$. This conjecture would follow if $N(\{b_1, b_2, \dots, b_n\})$ were equal to $n - (b_1/b_2 + \dots + b_{n-1}/b_n)$ in general, but this is unfortunately not the case since Robson has proved that $N(\{3, 4\}) \geq 1\frac{4}{15}$. (See Inf. Proc. Letters 2 (1973), 96-97; JACM 21 (1974), 491-499.)

41. Consider maintaining the blocks of size 2^k : The requests for sizes 1, 2, 4, ..., 2^{k-1} will periodically call for a new block of size 2^k to be split, or a block of that size will be returned. We can prove by induction on k that the total storage consumed by such split blocks never exceeds kn ; for after every request to split a block of size 2^{k+1} , we are using at most kn locations in split 2^k -blocks and at most n locations in unsplit ones.

This argument can be strengthened to show that $a_r n$ cells suffice, where $a_0 = 1$ and $a_k = 1 + a_{k-1}(1 - 2^{-k})$; we have

$k =$	0	1	2	3	4	5
$a_k =$	1	$1\frac{1}{2}$	$2\frac{1}{8}$	$2\frac{55}{64}$	$3\frac{697}{1024}$	$4\frac{18535}{32768}$

Table 2

QUANTITIES THAT ARE FREQUENTLY USED IN STANDARD SUBROUTINES
AND IN ANALYSIS OF COMPUTER PROGRAMS (45 OCTAL PLACES)

The names at the left of the "=" signs are given in decimal notation.

0.1 =	0.06314 63146 31463 14631 46314 63146 31463 14631 46315-
0.01 =	0.00507 53412 17270 24365 60507 53412 17270 24365 60510-
0.001 =	0.00040 61115 64570 65176 76355 44264 16254 02030 44672+
0.0001 =	0.00003 21556 13530 70414 54512 75170 33021 15002 35223-
0.00001 =	0.00000 24761 32610 70664 36041 06077 17401 56063 34417-
0.000001 =	0.00000 02061 57364 05536 66151 55323 07746 44470 26033+
0.0000001 =	0.00000 00153 27745 15274 53644 12741 72312 20354 02151+
0.00000001 =	0.00000 00012 57143 56106 04303 47374 77341 01512 63327+
0.000000001 =	0.00000 00001 04560 27640 46655 12262 71426 40124 21742+
0.0000000001 =	0.00000 00000 06676 33766 35367 55653 37265 34642 01627-
$\sqrt{2}$ =	1.32404 74631 77167 46220 42627 66115 46725 12575 17435+
$\sqrt{3}$ =	1.56663 65641 30231 25163 54453 50265 60361 34073 42223-
$\sqrt{5}$ =	2.17067 36394 57722 47602 57471 63003 00563 55620 32021-
$\sqrt{10}$ =	3.12305 40726 64555 22444 02242 57101 41466 33775 22532+
$\sqrt[3]{2}$ =	1.20505 05746 15345 05342 10756 65334 25574 22415 03024+
$\sqrt[3]{3}$ =	1.34233 50444 22175 73134 67363 76133 05334 31147 60121-
$\sqrt[4]{2}$ =	1.14067 74050 61556 12455 72152 64430 60271 02755 73136+
$\ln 2$ =	0.54271 02775 75071 73632 57117 07316 30007 71366 53640+
$\ln 3$ =	1.06237 24752 55006 05227 32440 63065 25012 35574 55337+
$\ln 10$ =	2.23273 06735 52524 25405 56512 66542 56026 46050 50705+
$1/\ln 2$ =	1.34252 16624 53405 77027 35750 37766 40644 35175 04353+
$1/\ln 10$ =	0.33626 75425 11562 41614 52325 33525 27655 14756 06220-
π =	3.11037 55242 10264 30215 14230 63050 56006 70163 21122+
$1^\circ = \pi/180$ =	0.01073 72152 11224 72344 25603 54276 63351 22056 11544+
$1/\pi$ =	0.24276 30155 62344 20251 23760 47257 50765 15156 70067-
π^2 =	11.67517 14467 62135 71322 25561 15466 30021 40654 34103-
$\sqrt{\pi} = \Gamma(1/2)$ =	1.61337 61106 64736 65247 47035 40510 15273 34470 17762-
$\Gamma(1/3)$ =	2.53347 35234 51013 61316 73106 47644 54653 00106 66046-
$\Gamma(2/3)$ =	1.26523 57112 14154 74312 54572 37655 60126 23231 02452+
e =	2.55760 52130 50535 51246 52773 42542 00471 72363 61661+
$1/e$ =	0.27426 53066 13167 46761 52726 75436 02440 52371 03355+
e^2 =	7.30714 45615 23355 33460 63507 35040 32664 25356 50217+
γ =	0.44742 14770 67666 06172 23215 74376 01002 51313 25521-
$\ln \pi$ =	1.11206 40443 47503 36413 65374 52661 52410 37511 46057+
ϕ =	1.47433 57156 27751 23701 27634 71401 40271 66710 15010+
e^γ =	1.61772 13452 61152 65761 22477 36553 53327 17554 21260+
$e^{\pi/4}$ =	2.14275 31512 16162 52370 35530 11342 53525 44307 02171-
$\sin 1$ =	0.65665 24436 04414 73402 03067 23644 11612 07474 14505-
$\cos 1$ =	0.42450 50037 32406 42711 07022 14666 27320 70675 12321+
$-\zeta'(2)$ =	0.74001 45144 53253 42362 42107 23350 50074 46100 27706+
$\zeta(3)$ =	1.14735 00023 60014 20470 15613 42561 31715 10177 06614+
$\ln \phi$ =	0.36630 26256 61213 01145 13700 41004 52264 30700 40646+
$1/\ln \phi$ =	2.04776 60111 17144 41512 11436 16575 00355 43630 40651+
$-\ln \ln 2$ =	0.27351 71233 67265 63650 17401 56637 26334 31455 57005-

Interlock time: Delay of one part of a system while another part is busy completing some action.

Internal nodes, 400–406.

Internal path length, 400, 402, 405.

Internet, iv, xvi.

Interpreter (interpretive routine), 200–202, 230, 340.

Interrupt, 228.

Intervals, notation for, 21.

Invariants, 17.

Inverse modulo m , 42.

Inverse of a matrix, 37–38, 73, 307.

Inverse of a permutation, 106, 175–178, 182.

Inversion problem, 63–64.

Inversions of a permutation, 542, 557, 577.

Inverting a linked list, 269, 279.

I/O: Input or output, 215.

IOC (input-output control), 137.

IPL, 230, 458–459, 460–461, 552.

Irreflexive relation, 261.

Isaacs, Irving Martin, 601.

Isolated vertex, 374.

Itai, Alon (אילן איטאי), 534.

Iverson, Kenneth Eugene, 33, 39, 459–460 convention, 32–33, 61, 103.

J-register of MIX, 125, 143, 186, 189, 212–214.

J1N (jump if r11 negative), 135, 210.

J1NN (jump if r11 nonnegative), 135, 210.

J1NP (jump if r11 nonpositive), 135, 210.

J1NZ (jump if r11 nonzero), 135, 210.

J1P (jump if r11 positive), 135, 210.

J1Z (jump if r11 zero), 135, 210.

JACM: Journal of the ACM, a publication of the Association for Computing Machinery since 1954.

Jacob, Simon, 81.

Jacquard, Joseph Marie, 229.

JAN (jump if rA negative), 135, 210.

JANN (jump if rA nonnegative), 135, 210.

JANP (jump if rA nonpositive), 135, 210.

JANZ (jump if rA nonzero), 135, 210.

JAP (jump if rA positive), 135, 210.

Jarden, Dov (דב גורדן), 85, 494.

JAZ (jump if rA zero), 135, 210.

JBUS (jump if busy), 137, 157, 212, 216.

JE (jump if equal), 135, 209.

Jeffrey, David John, 395.

Jenkins, D. P., 460.

JG (jump if greater), 135, 209.

JGE (jump if greater or equal), 135, 209.

JL (jump if less), 135, 209.

JLE (jump if less or equal), 135, 209.

JMP (jump), 134, 187, 209, 288.

JNE (jump if not equal), 135, 209.

JNOV (jump if no overflow), 134, 142, 209.

Jodeit, Jane Griffin, 462.

Johnson, Lyle Robert, 459–460.

Johnstone, Mark Stuart, 452.

Jokes, 54, 200.

Jones, Clifford Bryn, 18.

Jones, Mary Whitmore, 378.

Jonkers, Henricus (= Hans) Bernardus Maria, 614.

Jordan, Marie Ennemond Camille, 388, 406.

Josephus, Flavius, problem, 162, 184.

JOV (jump if overflow), 134, 142, 209.

Joyal, André, 395.

JRED (jump if ready), 137, 222–223.

JSJ (jump saving rJ), 134, 189, 210, 531.

Jump operators of MIX, 134–135, 209.

Jump trace, 214, 296, 528.

JXN (jump if rX negative), 135, 210.

JXNN (jump if rX nonnegative), 135, 210.

JXNP (jump if rX nonpositive), 135, 210.

JXNZ (jump if rX nonzero), 135, 210.

JXP (jump if rX positive), 135, 210.

JXZ (jump if rX zero), 135, 210.

Kahn, Arthur B., 268.

Kahrimanian, Harry George, 459.

Kallick, Bruce, 404.

Kaplansky, Irving, 184.

Karamata, Jovan, 66.

Karp, Richard Manning, 406.

Katz, Leo, 590.

Kaucký, Josef, 63.

Keller, Helen Adams, 123.

Kepler, Johann, 80, 81.

Kilmer, Alfred Joyce, 232.

King, James Cornelius, 20.

Kirchhoff, Gustav Robert, 406, 583.

law of conservation of flow, 97, 170–171, 268, 278, 364–370, 380.

Kirkman, Thomas Penyngton, 408.

Kirschenhofer, Peter, 506.

Klarner, David Anthony, 86.

Kleitman, Daniel J (Isaiah Solomon), 547, 596.

Knopp, Konrad Hermann Theodor, 48, 498.

Knotted lists, 459.

Knowlton, Kenneth Charles, 462.

Knuth, Donald Ervin (高德纳), ii, iv, xi, 11, 33, 66, 120, 193, 201, 202, 296, 297, 395, 457, 461, 471, 484, 499, 504, 523, 525, 565, 579, 580, 584, 592, 596, 631, 650.

Knuth, Nancy Jill Carter (高精蘭), x, xx.

Kolmogorov, Andrei Nikolaevich (Колмогоров, Андрей Николаевич), 104, 105, 464.

König, Dénes, 382, 406, 588.

Koster, Cornelis (= Kees) Hermanus Antonius, 461.

Kozelka, Robert Marvin, 544.

Kramp, Christian, 49.

Krattenthaler, Christian, 39.

Kreweras, Germain, 598.

corresponding to C by the rule " X is an ancestor of Y , and Y is a descendant of X , if and only if $X \supset Y$." Each equivalence class of C corresponds to an oriented tree, which is an oriented forest with $X \equiv Y$ for all X, Y . (We thereby have generalized the definitions of forest and tree that were given for finite collections.) In these terms, we may define the *level* of X as the cardinal number of ancestors(X). Similarly, the *degree* of X is the cardinal number of equivalence classes in the nested collection descendants(X). We say X is the *parent* of Y , and Y is a *child* of X , if X is an ancestor of Y but there is no Z such that $X \supset Z \supset Y$. (It is possible for X to have descendants but no children, ancestors but no parent.) To get *ordered* trees and forests, order the equivalence classes mentioned above in some ad hoc manner, for example by embedding the relation \subseteq into linear order as in exercise 2.2.3-14.

Example (a): Let $S_{\alpha k} = \{x \mid x = .d_1 d_2 d_3 \dots \text{ in decimal notation, where } \alpha = .e_1 e_2 e_3 \dots \text{ in decimal notation, and } d_j = e_j \text{ if } j \bmod 2^k \neq 0\}$. The collection $C = \{S_{\alpha k} \mid k \geq 0, 0 < \alpha < 1\}$ is nested, and gives a tree with infinitely many levels and uncountable degree for each node.

Example (b), (c): It is convenient to define this set in the plane, instead of in terms of real numbers, and this is sufficient since there is a one-to-one correspondence between the plane and the real numbers. Let $S_{\alpha m n} = \{(\alpha, y) \mid m/2^n \leq y < (m+1)/2^n\}$, and let $T_\alpha = \{(x, y) \mid x \leq \alpha\}$. The collection $C = \{S_{\alpha m n} \mid 0 < \alpha < 1, n \geq 0, 0 \leq m < 2^n\} \cup \{T_\alpha \mid 0 < \alpha < 1\}$ is easily seen to be nested. The children of $S_{\alpha m n}$ are $S_{\alpha(2m)(n+1)}$ and $S_{\alpha(2m+1)(n+1)}$, and T_α has the child $S_{\alpha 0 0}$ plus the subtree $\{S_{\beta m n} \mid \beta < \alpha\} \cup \{T_\beta \mid \beta < \alpha\}$. So each node has degree 2, and each node has uncountably many ancestors of the form T_α . This construction is due to R. Bigelow.

Note: If we take a suitable well-ordering of the real numbers, and if we define $T_\alpha = \{(x, y) \mid x > \alpha\}$, we can improve this construction slightly, obtaining a nested collection where each node has uncountable level, degree 2, and two children.

12. We impose an additional condition on the partial ordering (analogous to that of "nested sets") to ensure that it corresponds to a forest: If $x \preceq y$ and $x \preceq z$ then either $y \preceq z$ or $z \preceq y$. In other words, the elements larger than any given element are linearly ordered. To make a tree, also assert the existence of a largest element r such that $x \preceq r$ for all x . A proof that this gives an unordered tree as defined in the text, when the number of nodes is finite, runs like the proof for nested sets in exercise 10.

13. $a_1, a_1.a_2, \dots, a_1.a_2.\dots.a_k$.

14. Since S is nonempty, it contains an element $1.a_1.\dots.a_k$ where k is as small as possible; if $k > 0$ we also take a_k as small as possible in S , and we immediately see that k must be 0. In other words, S must contain the element 1. Let 1 be the root. All other elements have $k > 0$, and so the remaining elements of S can be partitioned into sets $S_j = \{1.j.a_2.\dots.a_k\}, 1 \leq j \leq m$, for some $m \geq 0$. If $m \neq 0$ and S_m is nonempty, we deduce by reasoning as above that $1.j$ is in S_j for each S_j ; hence each S_j is nonempty. Then it is easy to see that the sets $S'_j = \{1.a_2.\dots.a_k \mid 1.j.a_2.\dots.a_k \text{ is in } S_j\}$ satisfy the same condition as S did. By induction, each of the S_j forms a tree.

15. Let the root be 1, and let the roots of the left and right subtrees of α be $\alpha.0$ and $\alpha.1$, respectively, when such roots exist. For example, King Christian IX appears in two positions of Fig. 18(a), namely 1.0.0.0.0 and 1.1.0.0.1.0. For brevity we may drop the decimal points and write merely 10000 and 110010. Note: This notation is due to Francis Galton; see *Natural Inheritance* (Macmillan, 1889), 249. For pedigrees, it is more mnemonic to use F and M in place of 0 and 1 and to drop the initial 1; thus Christian IX is Charles's MFFMF, his mother's father's father's mother's father.

