55

Combinatorial Algorithms: An Update

HERBERT S. WILF
University of Pennsylvania

CBMS-NSF
REGIONAL CONFERENCE SERIES
IN APPLIED MATHEMATICS

SPONSORED BY
CONFERENCE BOARD OF
THE MATHEMATICAL SCIENCES

SUPPORTED BY
NATIONAL SCIENCE
FOUNDATION

Combinatorial Algorithms

CBMS-NSF REGIONAL CONFERENCE SERIES IN APPLIED MATHEMATICS

A series of lectures on topics of current research interest in applied mathematics under the direction of the Conference Board of the Mathematical Sciences, supported by the National Science Foundation and published by SIAM.

GARRETT BIRKHOFF, *The Numerical Solution of Elliptic Equations*
D. V. LINDLEY, *Bayesian Statistics, A Review*
R. S. VARGA, *Functional Analysis and Approximation Theory in Numerical Analysis*
R. R. BAHADUR, *Some Limit Theorems in Statistics*
PATRICK BILLINGSLEY, *Weak Convergence of Measures: Applications in Probability*
J. L. LIONS, *Some Aspects of the Optimal Control of Distributed Parameter Systems*
ROGER PENROSE, *Techniques of Differential Topology in Relativity*
HERMAN CHERNOFF, *Sequential Analysis and Optimal Design*
J. DURBIN, *Distribution Theory for Tests Based on the Sample Distribution Function*
SOL I. RUBINOW, *Mathematical Problems in the Biological Sciences*
P. D. LAX, *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves*
I. J. SCHOENBERG, *Cardinal Spline Interpolation*
IVAN SINGER, *The Theory of Best Approximation and Functional Analysis*
WERNER C. RHEINBOLDT, *Methods of Solving Systems of Nonlinear Equations*
HANS F. WEINBERGER, *Variational Methods for Eigenvalue Approximation*
R. TYRRELL ROCKAFELLAR, *Conjugate Duality and Optimization*
SIR JAMES LIGHTHILL, *Mathematical Biofluiddynamics*
GERARD SALTON, *Theory of Indexing*
CATHLEEN S. MORAWETZ, *Notes on Time Decay and Scattering for Some Hyperbolic Problems*
F. HOPPENSTEADT, *Mathematical Theories of Populations: Demographics, Genetics and Epidemics*
RICHARD ASKEY, *Orthogonal Polynomials and Special Functions*
L. E. PAYNE, *Improperly Posed Problems in Partial Differential Equations*
S. ROSEN, *Lectures on the Measurement and Evaluation of the Performance of Computing Systems*
HERBERT B. KELLER, *Numerical Solution of Two Point Boundary Value Problems*
J. P. LASALLE, *The Stability of Dynamical Systems* - Z. ARTSTEIN, *Appendix A: Limiting Equations and Stability of Nonautonomous Ordinary Differential Equations*
D. GOTTLIEB AND S. A. ORSZAG, *Numerical Analysis of Spectral Methods: Theory and Applications*
PETER J. HUBER, *Robust Statistical Procedures*
HERBERT SOLOMON, *Geometric Probability*
FRED S. ROBERTS, *Graph Theory and Its Applications to Problems of Society*
JURIS HARTMANIS, *Feasible Computations and Provable Complexity Properties*
ZOHAR MANNA, *Lectures on the Logic of Computer Programming*
ELLIS L. JOHNSON, *Integer Programming: Facets, Subadditivity, and Duality for Group and Semi-Group Problems*
SHMUEL WINOGRAD, *Arithmetic Complexity of Computations*
J. F. C. KINGMAN, *Mathematics of Genetic Diversity*
MORTON E. GURTIN, *Topics in Finite Elasticity*
THOMAS G. KURTZ, *Approximation of Population Processes*

JERROLD E. MARSDEN, *Lectures on Geometric Methods in Mathematical Physics*
BRADLEY EFRON, *The Jackknife, the Bootstrap, and Other Resampling Plans*
M. WOODROOFE, *Nonlinear Renewal Theory in Sequential Analysis*
D. H. SATTINGER, *Branching in the Presence of Symmetry*
R. TÉMAM, *Navier-Stokes Equations and Nonlinear Functional Analysis*
MIKLÓS CSÖRGO, *Quantile Processes with Statistical Applications*
J. D. BUCKMASTER AND G. S. S. LUDFORD, *Lectures on Mathematical Combustion*
R. E. TARJAN, *Data Structures and Network Algorithms*
PAUL WALTMAN, *Competition Models in Population Biology*
S. R. S. VARADHAN, *Large Deviations and Applications*
KIYOSI ITÔ, *Foundations of Stochastic Differential Equations in Infinite Dimensional Spaces*
ALAN C. NEWELL, *Solitons in Mathematics and Physics*
PRANAB KUMAR SEN, *Theory and Applications of Sequential Nonparametrics*
LÁSZLÓ LOVÁSZ, *An Algorithmic Theory of Numbers, Graphs and Convexity*
E. W. CHENEY, *Multivariate Approximation Theory: Selected Topics*
JOEL SPENCER, *Ten Lectures on the Probabilistic Method*
PAUL C. FIFE, *Dynamics of Internal Layers and Diffusive Interfaces*
CHARLES K. CHUI, *Multivariate Splines*
HERBERT S. WILF, *Combinatorial Algorithms: An Update*
HENRY C. TUCKWELL, *Stochastic Processes in the Neurosciences*
FRANK H. CLARKE, *Methods of Dynamic and Nonsmooth Optimization*
ROBERT B. GARDNER, *The Method of Equivalence and Its Applications*
GRACE WAHBA, *Spline Models for Observational Data*
RICHARD S. VARGA, *Scientific Computation on Mathematical Problems and Conjectures*
INGRID DAUBECHIES, *Ten Lectures on Wavelets*
STEPHEN F. MCCORMICK, *Multilevel Projection Methods for Partial Differential Equations*
HARALD NIEDERREITER, *Random Number Generation and Quasi-Monte Carlo Methods*
JOEL SPENCER, *Ten Lectures on the Probabilistic Method, Second Edition*
CHARLES A. MICCHELLI, *Mathematical Aspects of Geometric Modeling*
ROGER TÉMAM, *Navier–Stokes Equations and Nonlinear Functional Analysis, Second Edition*
GLENN SHAFER, *Probabilistic Expert Systems*
PETER J. HUBER, *Robust Statistical Procedures, Second Edition*
J. MICHAEL STEELE, *Probability Theory and Combinatorial Optimization*
WERNER C. RHEINBOLDT, *Methods for Solving Systems of Nonlinear Equations, Second Edition*
J. M. CUSHING, *An Introduction to Structured Population Dynamics*
TAI-PING LIU, *Hyperbolic and Viscous Conservation Laws*
MICHAEL RENARDY, *Mathematical Analysis of Viscoelastic Flows*
GÉRARD CORNUÉJOLS, *Combinatorial Optimization: Packing and Covering*

This page intentionally left blank

HERBERT S. WILF
University of Pennsylvania

Combinatorial Algorithms: An Update

SOCIETY FOR INDUSTRIAL AND APPLIED MATHEMATICS
PHILADELPHIA

10 9 8 7 6 5 4 3 2

Library of Congress Cataloging-in-Publication Data
Wilf, Herbert S., 1931–
Combinatorial algorithms—an update.

(CBMS-NSF regional conference series in applied mathematics ; 55)
Update ed. of: Combinatorial algorithms for computers and calculators / Albert Nijenhuis and Herbert S. Wilf, 2nd ed. 1978.
Based on a series of 10 lectures given at the CBMS-NSF Conference on Selection Algorithms for Combinatorial Objects, held in the summer of 1987 at the Baca Grande campus of Colorado College.
Bibliography: p.
Includes index.
1. Combinatorial analysis. 2. Algorithms.
I. Nijenhuis, Albert. Combinatorial algorithms for computers and calculators.
II. Title. III. Series.
QA164.W54 1989 511'.6 89-6070
ISBN 0-89871-231-9

Contents

This page intentionally left blank

Preface

This monograph is based on a series of ten lectures that I gave at the CBMS-NSF Conference on Selection Algorithms for Combinatorial Objects, held in the summer of 1987 at the beautiful place that is the Baca Grande campus of Colorado College. It is intended to be an updating of a book [NW] that Albert Nijenhuis and I first published in 1975 and revised in 1978.

That book was a collection of algorithms for listing and choosing at random combinatorial objects from given families, and of certain sizes within those families. For instance, how can we make a list of all of the partitions of the integer n? of all of the partitions of the *set* $[n]$? How can we choose uniformly at random a rooted tree from the set of all rooted trees of a given number n of vertices? Much has been done even in the ten years since the appearance of the second edition of our book. In these pages I will try to survey some of this new work.

Among the topics to be discussed are progress in Gray codes, in listing subsets of given size of a given universe, in listing rooted and free trees (a subject that had not existed in 1978), and in selecting free trees and unlabeled graphs uniformly at random. We also discuss ranking and unranking problems on unlabeled trees. It will, I hope, be apparent that the subject has continued to develop vigorously.

I wish to thank the United States Office of Naval Research for their support of the research that led to this monograph.

This page intentionally left blank

CHAPTER 1

The Original Gray Code

United States patent number 2,632,058, granted March 17, 1953 after having been applied for on November 13, 1947, to "Frank Gray, East Orange, N.J., assignor to Bell Telephone Laboratories, Incorporated, New York, N.Y., a corporation of New York" was for "pulse code communication" and it provided

> ... in a pulse code communication system, means for translating signal samples into reflected binary code groups of on-or-off pulses, means for transmitting pulse groups to a receiver station, means at said receiver station for converting each received group of said pulses into a conventional binary code pulse group, and means for translating said conventional binary code pulse groups into message signal samples.

The inventor, Frank Gray, is described in [Gr] as: "B.S., Purdue, 1911; Ph.D., University of Wisconsin, 1916. Western Electric Company, Engineering Department, 1919-1925. Bell Telephone Laboratories, 1925–. Dr. Gray has been engaged in work on electro-optical systems."

His idea was basically the following. Suppose you want to transmit a finite string of bits using an analogue transmission device. You might take the bit string, compute the integer whose bits they are, and transmit a signal of that strength (frequency, or whatever). The problem is that if a small error occurs in the signal strength, a small error will occur in the integer that is received, but a small change in an integer can cause a huge change in its bit string (e.g., between the strings of 31 and 32).

The question then arises as to how to associate integers with bit strings in such a way that small changes in the integers produce only small changes in the strings.

One way to do this is to list all finite strings of bits in a sequence with the property that each string differs from its successor in only a single bit position; that was the idea of the Gray code.

In Fig. 1.1 there is shown a list of all bit strings of length ≤ 4 in Gray code order. For each string we show its rank in the list, as an integer in the range [0,15], the binary digits of its rank, and the bit string itself.

0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

FIG. 1.1

We will now study some of the properties of the Gray code shown above, which is called the *standard reflected Gray code.*

First the list of strings has a simple recursive description. Let $\mathcal{L}_n$ denote the list of all n-bit strings, arranged in (standard reflected) Gray code order. Then we can generate all of these lists as follows.

1. $\mathcal{L}_0$ is the empty list.
2. For each $n = 1, 2, \ldots$, having formed $\mathcal{L}_{n-1}$, then
   (a) write out list $\mathcal{L}_{n-1}$ and prefix each entry with an additional bit "0";
   (b) write out list $\mathcal{L}_{n-1}$ in reverse order, and prefix each entry with an additional bit "1";
3. The list $\mathcal{L}_n$ is the result of concatenating the two lists that were formed in step 2.

The reader should take a moment to check that the list $\mathcal{L}_4$ that is shown in Fig. 1.1 can be formed in exactly the way that we have just described. If we denote the reversal of a list $\mathcal{L}_n$ by $\overline{\mathcal{L}}_n$, then the recipe given in steps 1-3 above can be summarized as

$$\mathcal{L}_n := 0 \oplus \mathcal{L}_{n-1}, 1 \oplus \overline{\mathcal{L}}_{n-1}. \tag{1.1}$$

This method of copying a list and its reversal, with some extra insertions going on, will be a common theme in some of the later constructions also, and it seems to be a good thing to think of when trying to construct some new kind of Gray code.

It is now a simple matter to prove by induction that if we start with the empty list and do (1.1) for $n \geq 1$ then we will indeed form lists that have the Gray code property: successive entries will differ in only a single bit position.

Sometimes we don't want to think of the *individual* lists $\mathcal{L}_n$, but of their "limit" as $n \to \infty$. Notice that the entry of rank m on some list $\mathcal{L}_n$ will

be identical with the entry of that rank on $\mathcal{L}_{n+1}$ except for an additional leading "0". Since additional leading "0" bits don't change the integer that is represented by the bit string, we can, if it is convenient, think of the entries of a Gray code list as *integers* encoded by their bit strings. Then the Gray code can be regarded as a permutation of the set of all positive integers that is the union of a family of disjoint bijections of the intervals $[2^j, 2^{j+1} - 1]$ $(j = 0, 1, \ldots)$.

From that point of view, the Gray code is the single infinite sequence

$$0, 1, 3, 2, 6, 7, 5, 4, 12, 13, 15, 14, 10, 11, 9, 8, \ldots.$$

Yet another manifestation of the code results from thinking of the entries as representing neither bit strings nor integers, but *sets*. The set that a bit string represents is the one whose members are the indices of the positions where the "1" bits appear in the string. Thus "00101011" is the set $\{1, 2, 4, 6\}$. In this state of mind, the list $\mathcal{L}_n$ is the list of all 2^n subsets of $[n]$, arranged so that each one can be obtained from its immediate predecessor by either a single adjunction of a singleton or a deletion of a single element.

As if that weren't enough, here's another pretty way to contemplate the world of Gray codes. Fix a positive integer n and imagine a graph G that has 2^n vertices, one for each possible string of n 0's and 1's. Now connect two of those vertices by an edge of G if and only if the two corresponding strings differ in only a single position. You are now looking at an n-dimensional cube.[1] In Fig. 1.2 we show the cube Q_3. As we follow down the Gray code list, we take a walk from one vertex of the cube to another, walking always on edges of the cube. The Gray code list $\mathcal{L}_n$ encodes a Hamilton walk (actually a circuit) on this graph.

The standard binary reflected Gray code that we have been talking about so far is certainly not the only Hamilton path on Q_n. There are a lot of them, although their exact number has not been determined (see [Gi]). In fact, the number of them has not even been determined asymptotically. This particular Hamilton path, however, does have a host of elegant properties which justify singling it out for discussion.

Hence a Gray code is a list of bit strings, or of integers, or of sets, or it is a Hamilton path on the cube.

Next we are going to study the relationships between the bit string of an integer m and the bit string that has rank m in the Gray code. Take $m = 9$, for instance. The string of rank 9 is 1101 (see Fig. 1.1), and the string of the integer 9 is 1001. Is there some simple rule from which we can construct one of these strings given the other? For instance, what is the string of rank 3982226135 in the Gray code?

We have the following remarkable theorem.

THEOREM 1.1. *Let $m = \sum \epsilon_i 2^i$ be the binary representation of the integer m, and let $\ldots e_3 e_2 e_1 e_0$ be the string of rank m in the Gray code. Then*

$$e_i \equiv \epsilon_i + \epsilon_{i+1} \pmod 2 \quad (i = 0, 1, 2, \ldots). \tag{1.2}$$

[1] Graph theorists call this graph Q_n ("Q" as in Qbe, after all).

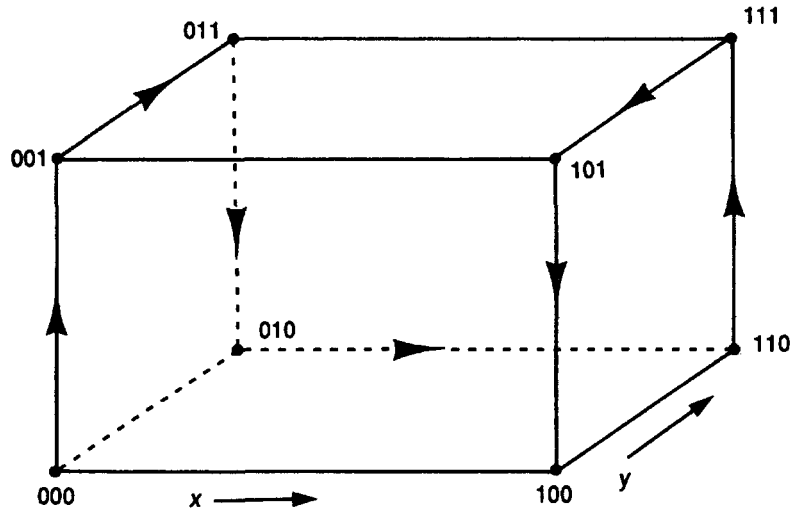

FIG. 1.2. *The Gray code $\mathcal{L}_3$ as a walk on Q_3.*

Proof. By induction on n we will show that the result holds for all m that occur on the list $\mathcal{L}_n$. This is extremely clear when $n = 0$. Suppose (1.2) holds for all strings on the list $\mathcal{L}_{n-1}$, and now consider the string of rank m on the $\mathcal{L}_n$. If $m \le 2^{n-1} - 1$ then there is nothing more to prove because the string of rank m is unchanged. Hence suppose $m \ge 2^{n-1}$, and write $m' = 2^n - 1 - m$. Then (1.2) holds for the integer m' and the string that has rank m' because $m' < 2^{n-1}$.

But the bits in the strings of ranks m and m' are related by

$$e_i(m) = e_i(m') \quad (i = 0, \ldots, n-2);\ e_{n-1}(m) = 1 + e_{n-1}(m'),$$

and the bits in the binary representations of the integers m and m' are related by

$$\epsilon_i(m) \equiv 1 + \epsilon_i(m') \pmod 2 \quad (i = 0, 1, \ldots, n-1).$$

It is now immediate to check that (1.2) continues to hold for the integer m, completing the proof. □

Exercise. Now, what is the string of rank 3982226135 in the Gray code?

It is easy to invert (1.2) in order to express the ϵ's in terms of the e's. We obtain

$$\epsilon_i = e_i + e_{i+1} + e_{i+2} + \ldots \quad (i = 0, 1, 2, \ldots). \tag{1.3}$$

We summarize with the following theorem.

THEOREM 1.2. *The bit string $\ldots e_3 e_2 e_1 e_0$ has rank m in the Gray code list, where the binary digits of m are given by* (1.3). *Conversely, if m is a given integer, then the bits in the string of rank m in the Gray code are given by* (1.2).

In general, for any list of combinatorial objects, the *ranking problem* is the question of determining the position in the list of some given object. The *unranking* problem gives the rank and asks for the object of that rank. Hence we may say that for the Gray code list, equation (1.3) solves the ranking problem and (1.2) solves the unranking problem.

Next, at the outset we stated that one of the motivations for the Gray code ordering was that if the ranks of two objects on the list are close then the objects themselves differ in rather few bit positions. By the *Hamming distance* between two bit strings we mean the number of bit positions in which they differ. Thus consecutive elements of the Gray code list have Hamming distance 1.

The question is this: Given a positive integer m, how close can two objects be to each other on the Gray code list if their Hamming distance is m? Symbolically, we are asking for

$$f(m) = \min_{dist(s,t)\geq m} \{|\rho(s) - \rho(t)|\} \quad (m \geq 1) \tag{1.4}$$

where ρ is the rank function.

For the given m, among all pairs s, t of strings that achieve the minimum in (1.4), choose a pair of minimum length, say of length L. Then the leftmost bit of s is different from the leftmost bit of t, for otherwise we could delete those bits and obtain shorter strings s', t' that are still at a distance m, while

$$|\rho(s') - \rho(t')| = |\rho(s) - \rho(t)| \tag{1.5}$$

which would be a contradiction.

Now let $q \geq 2$ be the first bit position, reading from the left, in which s and t have the same bit present. Suppose we construct s', t' by deleting that qth bit from each of s and t. Then the Hamming distance remains unchanged and the strings are shorter. To obtain a contradiction we must show that (1.5) still holds.

What happens to the rank of s when we delete the qth bit? Suppose that bit is 0. Then from (1.3), the binary digits of the rank of s are unaffected except that one of those digits has been deleted. The same happens for the rank of t, so the difference of their ranks does not change, and we have the contradiction. If the common bit in position q is a 1 then, according to (1.3) again, if we delete it then we don't affect any of the bits of the rank that are more significant than the qth, and every one of the bits that is less significant is reversed. Hence again both ranks change by the same amount under the deletion, and the difference is unaffected.

It follows that s and t differ in every bit position.

Hence in seeking the minimum (1.4) we may confine attention to *complementary pairs* s, t, that is, to pairs in which each of the L bits of the string t is the complement of the corresponding bit of s. We will write $t = \bar{s}$. Evidently we now have $L = m$ (recall that m is the Hamming distance between the two strings), so s is a string of length m.

We standardize by supposing that s is that one of the pair s, $\bar{s}$ that has a "1" in the leftmost bit position, i.e., s is the one of higher rank. Next we

look at (1.3) to see the relationship between the ranks of s and of $\bar{s}$. We see that the bits of their ranks are alternately the same and opposite! Precisely, we have (we let ρ be the rank function)

$$\rho(s) = 2^{m-1} + \epsilon_{m-2}2^{m-2} + \epsilon_{m-3}2^{m-3} + \dots$$

and

$$\rho(\bar{s}) = \epsilon_{m-2}2^{m-2} + (1 - \epsilon_{m-3})2^{m-3} + \dots$$

and therefore the difference between their ranks is

$$\begin{aligned}\rho(s) - \rho(\bar{s}) &= 2^{m-1} + (2\epsilon_{m-3} - 1)2^{m-3} + (2\epsilon_{m-5} - 1)2^{m-5} + \dots \\ &\geq 2^{m-1} - 2^{m-3} - 2^{m-5} - \dots\end{aligned}$$

where the terms continue as long as the exponents are nonnegative.

It is a simple matter to sum the above series and obtain the following theorem.

THEOREM 1.3. *Two strings on the Gray code list whose Hamming distance is $\geq m$ have ranks that differ by at least $\lceil 2^m/3 \rceil$. The bound is best possible.* □

The above result is essentially due to Yuen [Yu].

CHAPTER 2

Other Gray Codes

The idea of the Gray code has found applications in combinatorial families other than the family of subsets of a set. In any combinatorial family where we want to make a list of all of the objects in such a way that only the smallest possible change takes place as we go from each object to its successor, we can speak of a (generalized) Gray code.

It should be remarked that the concepts are meaningful only if we apply them to *encodings* of families of objects rather than to the underlying objects. More precisely, it may be possible (and usually is) to encode one and the same family of objects in several different ways. We might expect that a Gray code scheme will work on one of these encodings, but it might not provide minimal changes in any other one of the encodings.

A small example will illustrate the point. Suppose n is fixed and we consider the permutations of n letters. If we generate the list of all permutations of n letters by applying, at each stage, a single transposition to get from one to the next, then it is reasonable to expect that we will have a Gray code for permutations. Suppose, however, that instead of encoding permutations by their *values*, we choose to encode them by their *cycles*. Then as we apply a single transposition, large changes may take place that aren't at all in the spirit of the minimal changes that we expect from Gray coding. By the way, I can't resist pointing out that there might be an interesting Gray code for permutations in cycle form, but I don't know what it is.

In contrast to that example, let's think about subsets of a set again. The original Gray code was designed for the bit-string encoding of a subset. Suppose, instead, we use the list-of-members encoding. Do we still have the minimal change property? Indeed yes, because each set will be obtained from its predecessor by either deleting or adjoining a single member, and that is about the most minimal change we could expect. The point is that sometimes Gray codes have pleasant properties with respect to more than one encoding of a combinatorial family, but don't count on it..

For a first example of a Gray code applied to a family other than 0–1 strings, suppose we are given two positive integers n and k ($k < n$), and we want to make a list of all of the k-subsets of $[n]$. Let's adopt the encoding in which we simply display the list of members of each k-subset.

Now it's quite out of the question to have each item on the list differ from its predecessor by a single adjunction or deletion, because the cardinalities of the subsets are supposed to be kept fixed at k. The "Gray-est" thing we

can hope for, then, is to have each subset formed from its predecessor by *two* operations, a deletion *and* an insertion.

In [NW] such an algorithm was given, and many others are known. We called ours a "revolving door" (RD) algorithm, for presumably obvious reasons, and it goes like this.

Let $A(n,k)$ denote the list of all k-subsets of $[n]$, arranged in RD order, where the first set in the list is $\{1,2,\ldots,k\}$, and the last one is $\{1,2,\ldots,k-1,n\}$. Then we form the lists $A(n,k)$ recursively, as follows:

$$A(n,k) = A(n-1,k), \overline{A(n-1,k-1)} \oplus \{n\}. \tag{2.1}$$

In words, the list $A(n,k)$ is formed by writing down the list $A(n-1,k)$ and then following it with the list that is obtained from $A(n-1,k-1)$ by first writing it down in reverse order and then adjoining the singleton $\{n\}$ to each set on that list.

The recursion has the same music as the Pascal triangle, i.e., the same three grid points (n,k), $(n-1,k)$, and $(n-1,k-1)$ are involved at each step. Hence the boundary conditions that are naturally needed for (2.1) to boot itself up will be the same kind that are needed by the binomial coefficients. Therefore we add to (2.1) the conditions that the list $A(n,k)$ is empty if $k < 0$ or if $k > n$. The reader should then work out a few of these lists, for instance, enough to verify that the list $A(4,6)$ of 4-subsets of [6] is as follows.

1 2 3 4
1 2 4 5
2 3 4 5
1 3 4 5
1 2 3 5
1 2 5 6
2 3 5 6
1 3 5 6
3 4 5 6
2 4 5 6
1 4 5 6
1 2 4 6
2 3 4 6
1 3 4 6
1 2 3 6

Here are some of the general principles of Gray code construction that already appear in this example. First, the general step is recursive, and is carried out by stitching together some earlier lists to make the next one. Second, there is the list reversal in the second part of the recurrence. The reason it is there is to smooth out the transition between the two portions of the recurrence. To put it a little more graphically, in the construction (2.1) we see a list, a comma, and another list. The hard part is at the comma.

Indeed, inductively we can suppose that the first list is already in some Gray code sequence, and the second sublist is also. The difficulty comes in

(1,2,3)	(2,3,4)
(1,2,4)	(2,3,5)
(1,2,5)	(2,3,6)
(1,2,6)	(2,5,6)
(1,5,6)	(2,4,6)
(1,4,6)	(2,4,5)
(1,4,5)	(3,4,5)
(1,3,5)	(3,4,6)
(1,3,6)	(3,5,6)
(1,3,4)	(4,5,6)

FIG. 2.1. *Strong revolving-door order.*

gluing them together so the transition from the first to the second also has just a minimal change; in this case one element enters and one leaves.

In order to *prove* that (2.1) does what it is supposed to do the reader will find that it is necessary to do more than just make the inductive assumption that the earlier lists $A(n-1,k)$ and $A(n-1,k-1)$ are in minimal change order. In order to prove that the new list $A(n,k)$ is also in Gray code order we have to know what the first subset and the last subset on the earlier lists are, so as to be able to prove that there is no bumpiness across the comma (again, what that means is that as we go from the last k-subset that appears on the first of the two sublists that appear on the right of (2.1) to the first of the k-subsets that appears in the second one of those two sublists, only one element gets deleted from the set and one element gets added).

Exercises.

(1) Find a simple and elegant nonrecursive implementation of the RD algorithm by describing a *successor* algorithm. Precisely, given a particular k-subset S of $[n]$, and given only that, find the next set in the list $A(n,k)$.

(2) Prove that (2.1) does indeed provide a recursive construction of the revolving door lists, by building into the inductive hypothesis not only that the earlier sets are Gray codes, but also what they begin and end with.

To "peek" at the answers, see page 28 of [NW].

Although the lists $A(n,k)$ satisfy the revolving door condition, there are minimal-change algorithms for k-subsets that are even more severely restricted. Suppose we specify a subset by giving its members in ascending order:

$$S: 1 \le a_1 < a_2 < \ldots < a_k \le n.$$

Then the subset S is represented by a k-vector whose components strictly increase.

Let's define the minimal-change condition by requiring that two k-subsets that are consecutive on our list may differ only in a single component of their membership vectors.

This is stronger than the revolving-door condition, but it can be satisfied. In Fig. 2.1 are the 20 3-subsets of [6] arranged in that way.

If $B(n,k)$ denotes this list of all of the k-subsets of $[n]$ in strong revolving door order, then the recursive construction of these lists, due to Eades and McKay [EM], is given by

$$B(n,k) := B(n-2,k-2)\oplus(n-1,n), \overline{B(n-2,k-1)}\oplus(n), B(n-1,k).$$

Lest we should think that Gray coding questions that sound reasonable always have affirmative answers, here is yet a third way to think about making a Gray code for k-subsets of a set, where the answer turns out to be complicated. Suppose we are permitted to alter the status only of two *adjacent* elements. That is, at each stage we can delete i and adjoin $i+1$, or we can delete $i+1$ and adjoin i, and furthermore we regard elements 1 and n as being adjacent. Then is it possible to construct the lists?

This problem has been studied by Joichi and White [JW], Buck and Wiedemann [BW], Eades, Hickey, and Read [EHR], and Ruskey and Miller [RM], and the answer is given by the following theorem.

THEOREM 1.4. *There is an adjacent interchange Gray code for k-subsets of $[n]$ if and only if either k has one of the four values 0, 1, $n-1$, n or else n is even and k is odd.*

Next we leave the subject of k-subsets of a set, and turn to that of the compositions of an integer. If, once more, n and k are given positive integers, then by a composition of n into k parts we mean an *ordered* representation

$$n = r_1 + r_2 + \ldots + r_k \quad (\forall i : r_i \geq 0)$$

of n as a sum of exactly k nonnegative integers.

For our present purposes it is convenient to adopt the language of balls and boxes. Hence suppose we have k *labeled* boxes and n *unlabeled* balls. Then there is obviously a bijection between compositions of n into k parts and arrangements of the balls in the boxes.

It is well known, and easy to see, that there are exactly $\binom{n+k-1}{n}$ such arrangements, and it isn't very hard to figure out ways of making lists of all of them. But now let's add the Gray code restriction to make things a little more interesting.

By a Gray code for compositions of n into k parts we mean a list of all of those compositions, sequenced so that each arrangement of n balls in k boxes arises from its immediate predecessor by moving *exactly one ball from one box to another.*

In the first edition of [NW] we asked if such an algorithm could be constructed, and Donald Knuth [Kn2] found the elegant solution that we will now describe.

Let $\mathcal{L}(n,k)$ denote the desired lists. We will construct them recursively, for n fixed, and $k = 1,2,\ldots$. First, the list $\mathcal{L}(n,1)$ has just one entry, namely, the composition $n = n$ of n into one part.

If, inductively, the lists $\mathcal{L}(n,j)$ $(j = 1,2,\ldots k)$ have been constructed, then

$$\mathcal{L}(n,k+1) = \mathcal{L}(n,k)\oplus\{0\}, \overline{\mathcal{L}(n-1,k)}\oplus\{1\}, \mathcal{L}(n-2,k)\oplus\{2\}, \overline{\mathcal{L}(n-3,k)}\oplus\{3\},\ldots$$

where the last term will be the one that contains $\mathcal{L}(0, k)$.

The general approach that we described earlier for building Gray codes is shown here in full flower. The flip-flopping of the lists ensures smooth transitions across the commas, and the rest is taken care of inductively. Here is the list of the 21 compositions of 5 into three parts as they appear on the list $\mathcal{L}(5, 3)$. Can you follow the bouncing ball? Just to the right of each composition on the list we show the jump that the ball made in the form of the number of its origin and destination boxes, starting from the composition 5+0+0.

4	1	0	$1 \rightarrow 2$
3	2	0	$1 \rightarrow 2$
2	3	0	$1 \rightarrow 2$
1	4	0	$1 \rightarrow 2$
0	5	0	$1 \rightarrow 2$
0	4	1	$2 \rightarrow 3$
1	3	1	$2 \rightarrow 1$
2	2	1	$2 \rightarrow 1$
3	1	1	$2 \rightarrow 1$
4	0	1	$2 \rightarrow 1$
3	0	2	$1 \rightarrow 3$
2	1	2	$1 \rightarrow 2$
1	2	2	$1 \rightarrow 2$
0	3	2	$1 \rightarrow 2$
0	2	3	$2 \rightarrow 3$
1	1	3	$2 \rightarrow 1$
2	0	3	$2 \rightarrow 1$
1	0	4	$1 \rightarrow 3$
0	1	4	$1 \rightarrow 2$
0	0	5	$2 \rightarrow 3$

Here is another example of the genre, whose origins are similar. That is to say, Nijenhuis and I asked if, for n fixed, all of the partitions of a *set* of n elements could be arranged in Gray code sequence, in the sense that as we pass down the list from a set partition to its successor, exactly one element moves from one equivalence class to another (the rule permits an element to leave a certain class and start a singleton class of its own).

The answer was again found by Knuth, in a recursive form that will now be described.

A set partition is described by giving a list of the members of each of its classes. We will suppose that the classes are given in ascending order of their smallest elements. For a fixed partition P of $[n]$ we construct the list of its *children*. These are the partitions of $[n+1]$ that are obtained by successively placing the letter $n+1$ into each of the classes of P in turn, and then placing $n+1$ into its own singleton class. Thus, the children of the partition $(1, 3)(2, 4, 5)$ of [5] are the three partitions $(1, 3, 6)(2, 4, 5)$, $(1, 3)(2, 4, 5, 6)$, and $(1, 3)(2, 4, 5)(6)$ of [6].

Inductively, suppose we have before us the list of all partitions of the set $[n]$ arranged in a Gray code order so each partition is obtainable from its predecessor by a single move of one letter from one class to another. We make the corresponding list of partitions of $[n+1]$ as follows.

Write down the children of the first partition on the list. Then write down in reverse order the children of the next partition. Then create the children of the next partition, then the reverse ... etc., and that's it.

For example the partitions of [2] are $(1,2)$ and $(1)(2)$. Hence the list of partitions of [3] that we would create is $(1,2,3)$, $(1,2)(3)$, $(1)(2)(3)$, $(1)(2,3)$, $(1,3)(2)$.

It should be noted that when the complete lists are needed, the recursive form of these algorithms is sufficient. If, however, we want to produce the objects one at a time without storing them all at once, then nonrecursive versions may be helpful. Nonrecursive implementations of Knuth's algorithms have been given by Klingsberg [Kl] and Kaye [Ka].

Gray code constructions don't always work so smoothly. Consider, for a fixed positive integer n, the problem of arranging all of the partitions of the *integer n* in Gray code order. What we mean is that each partition should be obtainable from its predecessor by diminishing one part by 1 and adding 1 to another part (the latter to include creating a new part = 1).

This problem was treated by Yoshimura [Yo], who gave such lists for $n = 1, 2, \ldots, 11$. It has been completely solved by Carla D. Savage [*J. Algorithms*, to appear]. Here, for instance, is a list for $n = 10$:

10, 9+1, 8+2, 7+3, 6+4, 5+5, 5+4+1, 4+4+1+1, 4+4+2,
4+3+3, 3+3+3+1, 3+3+2+2, 3+3+2+1+1, 4+3+2+1, 5+3+2,
5+3+1+1, 6+3+1, 6+2+2, 5+2+2+1, 4+2+2+2, 3+2+2+2+1,
2+2+2+2+2, 2+2+2+2+1+1, 2+2+2+1+1+1+1, 3+2+2+1+1+1,
4+2+2+1+1, 4+3+1+1+1, 3+3+1+1+1+1, 3+2+1+1+1+1+1,
4+2+1+1+1+1, 5+2+1+1+1, 6+2+1+1, 7+2+1, 8+1+1, 7+1+1+1,
6+1+1+1+1, 5+1+1+1+1+1, 4+1+1+1+1+1+1,
3+1+1+1+1+1+1+1, 2+2+1+1+1+1+1+1, 2+1+1+1+1+1+1+1+1,
1+1+1+1+1+1+1+1+1+1.

Here is one last Gray coding problem that has received attention recently. Consider, for n fixed, the triangulations of a labeled n-gon. This is a well-known Catalan-countable family of objects. In that family we will now define an elementary operation, called a *flip*, which maps a triangulation onto another one. To do a flip, select a quadrilateral, delete the diagonal that it has, and draw the other one. In Fig. 2.2 we show a flip in a triangulation of a hexagon.

The question, of course, is whether it is possible to arrange the set of all C_{n-1} of these triangulations in a sequence with the property that each member of the sequence is a flip of its predecessor. Such an arrangement of the 5 triangulations of a pentagon is shown in Fig. 2.3, where, in each case, the edge that will get flipped is marked.

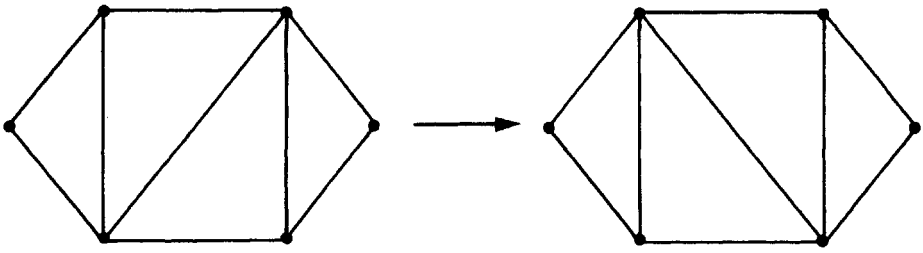

FIG. 2.2. *A flip.*

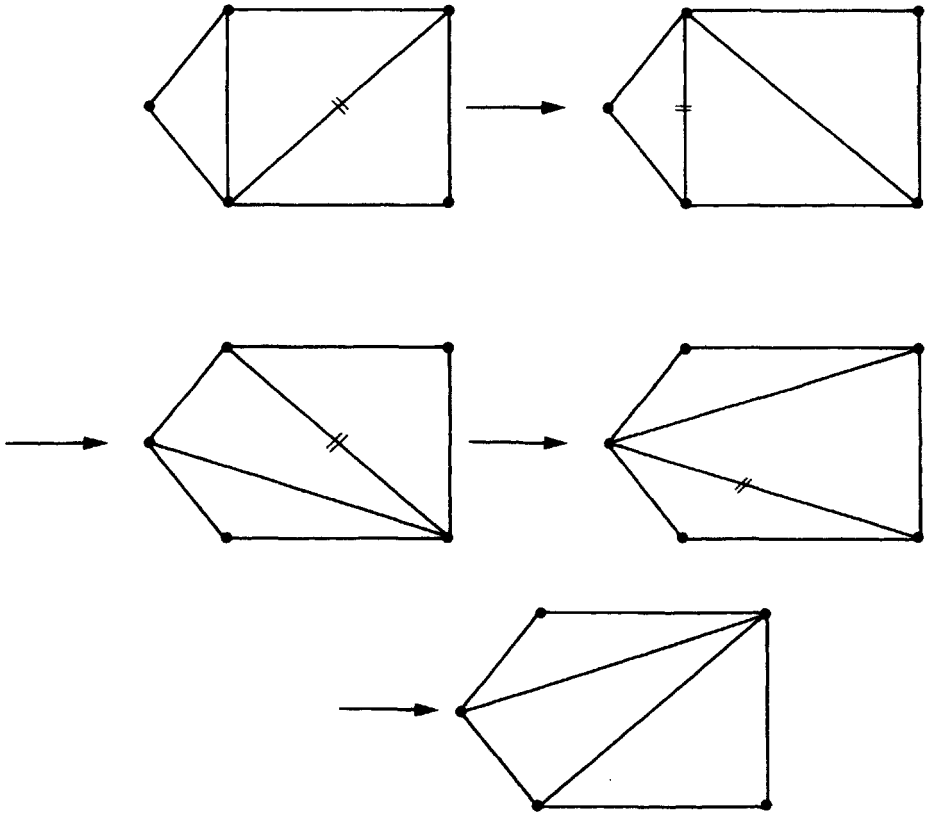

FIG. 2.3. *Flipping through the triangulations.*

This question is not only pretty, but it relates, via the familiar family of bijections that map Catalan-countable problems onto each other, to other questions about other families. In the case of binary trees, it turns out that the motion of flipping triangulations maps into an operation called *rotation* of a binary tree, and so the question concerns the possibility of arranging all binary trees of n internal nodes in sequence so that each can be obtained from its predecessor by a single rotation.

Here is a precise description of a rotation of a binary tree with respect to an edge (caution: it does *not* mean interchange of the right and left subtrees at some vertex). In Fig. 2.4 we show a binary tree before and after a flip with respect to the edge $x - y$. The symbols T_1, T_2, T_3 denote the left and right subtrees at y and the right subtree at x, respectively. It may (or may not) help to visualize the operation by thinking of the tree as a mobile that hangs from a point of attachment at x. Then pick it up at y, lifting y above x. Then let the subtree T_2 slide down the string to become the left subtree of x, which wouldn't otherwise have one.

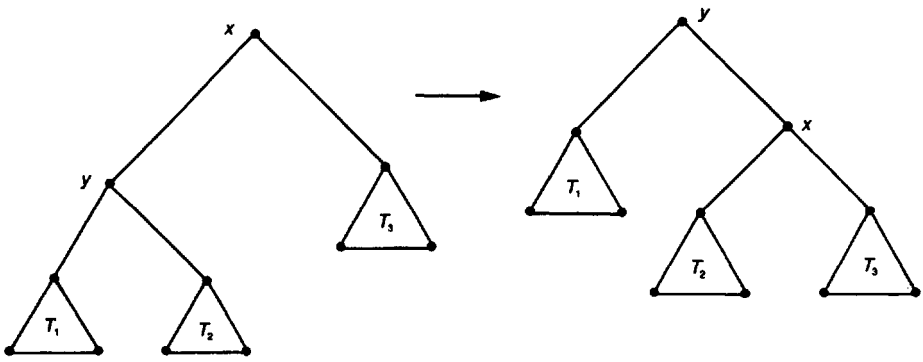

FIG. 2.4. *A rotation in a binary tree.*

Anyway, the operation of tree rotation is used in computer science to balance trees that are being used as data structures.

The connection between the flips of triangulations and rotations of binary trees appears in the thesis of Lucas [Luc], who also gave the first proof of the existence of these Gray codes. Her successful solution of this problem involved overcoming a number of subtle difficulties.

Weight sequences for binary trees were introduced by Pallo. They associate with a binary tree of n internal nodes, a sequence $(w_1, \ldots, w_n)$ that is constructed as follows: visit the internal nodes of the tree in *inorder*, which is to say, visit the left subtree, the root, and the right subtree, in that order. For each $i = 1, n$, as the ith internal node is visited, let w_i denote the number of *leaves* in the left subtree at that node. A binary tree and its weight sequence are shown in Fig. 2.5. Observe that if a binary tree is obtained from another

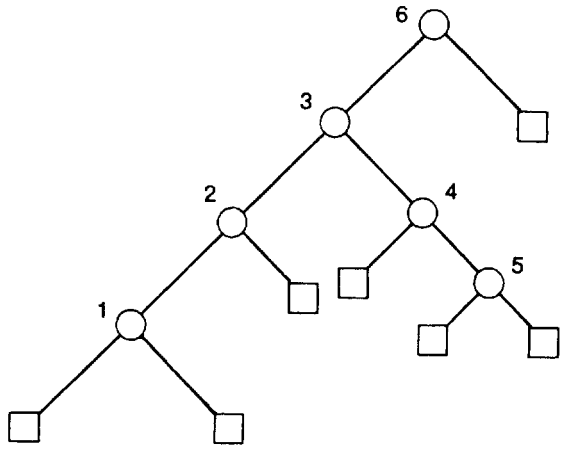

FIG. 2.5. *A binary tree of weight sequence* (123116).

one by a single rotation, then its weight sequence differs from the original in only a single component.

This page intentionally left blank

CHAPTER 3

Variations on the Theme

Here is a variant of the idea of a Gray code. For fixed n, we want to list the permutations of S_n in such a way that each permutation will be totally different from its immediate predecessor. What that means is that if σ and τ are consecutive on the list, then for all $i : \sigma(i) \neq \tau(i)$.

We can restate the condition by saying that whenever σ and τ are consecutive, then $\sigma^{-1}\tau$ has no fixed points. Again, we may rephrase the problem by saying that we want each permutation on the list to be obtained from its predecessor by multiplication with a fixed-point free permutation.

Let F_n be the set of all fixed-point free permutations of n letters. Imagine the Cayley graph G_n that is formed out of the vertex set S_n, with an undirected edge between σ and τ whenever $\sigma^{-1}\tau \in F$.

We are asking if G_n is Hamiltonian (i.e., if it has a Hamilton path).

Before we get to that question we deal with an easier one. Is G_n connected? This is the same as asking if S_n is generated by its fixed-point free permutations F_n.

In the theory of Hamiltonian graphs there are a number of theorems that permit one to conclude that a certain graph is Hamiltonian, provided that the graph has a great many edges. Typically, if the degree of each of the vertices of the graph G is at least about $n/2$, where $n = |V(G)|$, then G will have a Hamilton path or cycle.

In this problem, such a theorem really helps. There are exactly $\langle n!/e \rangle$ fixed-point free permutations of n letters, for each $n \geq 1$, where "$\langle \cdot \rangle$" denotes the "nearest-integer" function. There is a theorem of Jackson[2] [Ja1] which asserts that a 2-connected graph of N vertices in which each vertex has degree $\geq N/3$ is Hamiltonian.

It is easy to check (see below) that the Cayley graph of the symmetric group with respect to the set of fixed-point free permutations is connected for all $n \neq 1, 3$ and is 2-connected[3] [Im] for all $n \geq 4$. Further since

$$\langle n!/e \rangle \geq n!/3 \qquad (n \geq 2)$$

the hypotheses of Jackson's theorem are fulfilled for $n \geq 4$, and a Hamiltonian path exists. We do not know of any simple algorithm for an algorithmic description of such a path.

---

[2] My thanks to Dr. Linda Lesniak for this reference.

[3] I thank Dr. Yahya Hamidoune for this reference.

In the case $n = 4$, here is a list of the 24 permutations in which each one has no common values with its predecessor: 1234, 2143, 3214, 4321, 3412, 2341, 3124, 1432, 2314, 3421, 4132, 1423, 2134, 3241, 4312, 1243, 2431, 3142, 4213, 1324, 2413, 4231, 1342, 4123.

Most of the interesting questions about Cayley graphs and their Hamiltonicity involve extremely sparse graphs, and for those we have no general theorems. The redeeming quality that these graphs do have, however, is their *symmetry*.

Hence, what the world needs are some theorems to the effect that if a graph is somehow *symmetrical* enough, then it has a Hamilton path.

A question that was raised in 1969 by Lovász [Lov] addresses this need directly. Say that a graph G is *vertex-transitive* if for every pair v, w of vertices of G there is an automorphism σ of G that carries v into w (the vertices of G are "all alike"). The question asks if it is true that *every connected, vertex-transitive undirected graph has a Hamilton path.* The graph G_n in our little example above is certainly vertex-transitive, so if we can show that it is connected (we can), then an affirmative answer to Lovász's question would imply that it is Hamiltonian also. In fact, it would imply that every Cayley graph with respect to a set of generators that is undirected, in that the inverse of each generator is also a generator, is Hamiltonian.

The question that Lovász asked has not been answered, but progress has been made, nonetheless. A survey article of Alspach [Al] reviews some of these partial results. Also, Babai [Ba] proved that every connected vertex-transitive graph on $n \geq 4$ vertices has a cycle longer than $\sqrt{3n}$. Jackson [Ja2] showed that every 3-connected cubic graph has a cycle of length $|V(G)|^t$, where $t = .69...$. Watkins [Wat] proved that if $\rho(G)$ is the minimum vertex degree of G, and $\kappa(G)$ is its vertex connectivity, then l.u.b. $\rho(G)/\kappa(G)$ is exactly 3/2, where the l.u.b. is taken over all connected, vertex-transitive graphs G.

In [NW], by the way, we described a *directed* Cayley graph that is *not* Hamiltonian. Consider the symmetric group as being generated by just two generators: a cyclic right-shift of one unit, and an interchange of the first and second values. We found that the Cayley graph here is Hamiltonian if $n = 1, 2, 3, 4$, but not if $n = 5$. The status for $n \geq 6$ is unknown.

THEOREM 3.1. *The symmetric group S_n is generated by the fixed-point free permutations if and only if $n \neq 1, 3$.*

Proof. Since S_n is generated by the transpositions, it will be enough to decide when the transpositions are generated by the fixed-point free permutations. But if any single transposition is so generated, then they all are. Hence S_n is generated by the fixed-point free permutations if and only if the single transposition

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & \dots & n \\ 2 & 1 & 3 & 4 & \dots & n \end{pmatrix}$$

is a product of fixed-point free permutations.

First, let n be odd, $n \geq 5$, and let

$$\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & \dots & n \\ 2 & 1 & 4 & 5 & 6 & 7 & \dots & 3 \end{pmatrix}.$$

Then $\tau^{n-2} = \sigma$, and we are finished in this case since $\tau \in F_n$.

Next, let n be even, $n \geq 6$. With τ as in the previous case, define

$$\omega = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & \dots & n-1 & n \\ 2 & 1 & n-1 & n & 3 & 4 & \dots & n-3 & n-2 \end{pmatrix}.$$

Then $\omega\tau^{n-4} = \sigma$, which completes this case.

If $n = 4$ let ω_1 in cycle form be (1432) and let ω_2 in cycle form be (1324). Then $\omega_2\omega_1^2 = \sigma$.

The case $n = 2$ is trivial. By drawing G_3, we see that it has two connected components, so all assertions have now been proved. □

The following result is stronger and is also easier to prove.

THEOREM 3.2. *Fix $n \geq 5$. Let C be a fixed conjugacy class of the symmetric group S_n. Then C generates S_n if and only if C contains an odd permutation.*

Proof.[4] Consider the subgroup H that C generates. Then clearly H is normal in S_n. Suppose H is a proper subgroup. If $n \geq 5$ the only proper normal subgroup that S_n has is the alternating group A_n. If C contains an odd permutation then H cannot be A_n so it must be all of S_n. Conversely, if C consists of even permutations only, then so does H, and so it cannot be the full symmetric group. □

[4] This pretty proof was found independently by two of my colleagues, Drs. Gerstenhaber and Nijenhuis.

This page intentionally left blank

CHAPTER 4

Choosing 2-Samples

This chapter contains a small technical procedure that will be of use in Chapter 6, where we will be selecting random free trees. The problem is quite simple to state.

Given a device that will produce, on demand, one of N given objects, uniformly at random (u.a.r.), it is required to produce, u.a.r., one of the $\binom{N+1}{2}$ unordered pairs (= "2-samples") of the objects. The objects are to be thought of as complicated, so it may be expensive to ask if two of them are different or the same.

We will give two methods, of which the first is really here only to define the problem a little better, and the second is the one that works.

Method 1. Choose two objects independently u.a.r. and output the pair. If we use this method then

$$prob(\{A, A\}) = 1/N^2,$$

while for $A \neq B$,

$$prob(\{A, B\}) = 2/N^2$$

so pairs of distinct elements are chosen twice as often as the others. If we attempt to rectify this by rejecting such pairs occasionally, then we run into the problem of recognizing these pairs at all, which we have hypothesized as being expensive to carry out.

Method 2. For a fixed real number p,

(A1) with probability p do
choose an object ω and output $\{\omega, \omega\}$ or

(A2) with probability $1 - p$ do
choose 2 objects (ω', ω'') independently u.a.r. and (without examining them further!) output the pair $\{\omega', \omega''\}$. □

Now let A be some fixed object. What is the probability that Method 2 produces the output $\{A, A\}$ (note that either (A1) or (A2) can give this result)? It is

$$p \cdot (1/N) + (1 - p) \cdot (1/N^2).$$

Next suppose that A, B are two *distinct* objects. Then what is the probability that the unordered pair $\{A, B\}$ will be output? It is

$$p \cdot 0 + (1 - p) \cdot (2/N^2) = 2(1 - p)/N^2$$

These two probabilities are equal if we use the p that satisfies

$$p/N + (1 - p)/N^2 = 2(1 - p)/N^2,$$

i.e., if $p = 1/(N+1)$, and we have the following algorithm.

ALGORITHM 2-SAMPLE.
With probability $1/(N+1)$, choose an object ω u.a.r. and output the pair $\{\omega, \omega\}$, else,
With probability $N/(N+1)$, choose two objects ω', ω'' independently u.a.r. and output the pair $\{\omega', \omega''\}$. □

CHAPTER 5

Listing Rooted Trees

In the years since [NW] appeared there has been a great deal of work on algorithms for selection of trees. In particular, there are now efficient algorithms for listing all rooted trees of given size [BH], listing all free trees of given size ([Re], [DZ], [Ko], [WROM]), choosing uniformly at random a free tree of a given number of vertices [Wi], and ranking and unranking rooted and free trees ([WY], [Yo]).

In this chapter we discuss the problem of listing rooted trees in constant average time. The results are due to Beyer and Hedetniemi [BH].

If n is fixed, their algorithm will produce a list of all rooted trees of n vertices where, in a certain encoding, the average labor per tree is uniformly bounded, as a function of n. In their scheme, rooted trees are encoded by means of their *level sequences.*

To define the level sequence of a tree we must first define it for an *ordered* tree. In that case we begin by visiting its vertices in *preorder*, which means

(a) visit the root, and then

(b) visit the subtrees in order.

While the visitation is going on we must note the *level* on which each vertex lives at the time we visit it. The root is defined to live on level 1. In general, the level of a vertex is equal to the number of vertices in the path that joins it to the root.

The result of visiting the vertices in preorder and writing down the levels on which they live is the level sequence of the given ordered tree.

In Fig. 5.1 we show an ordered tree and its level sequence. The vertices of the tree are unlabeled but we show next to each vertex the epoch at which it gets visited.

The appearances of the 2's in a level sequence of a tree T are important. If a 2 appears in a certain position, then we have arrived at the level just below the root, i.e., we are at the root of one of the subtrees at the root of T. Hence all entries in the level sequence, beginning with that 2, and ending just before the next 2, comprise the level sequence of a single root subtree of T.

An ordered tree of course comes equipped with an ordering of its subtrees. The level sequence provides another, natural, ordering of all subtrees. Precisely, let U and V be subtrees of an ordered tree T. The level subsequence $L(U)$ of U is the (consecutive) subsequence of $L(T)$ that belongs to vertices of U. The level subsequences of all subtrees of T can be ordered lexicograph-

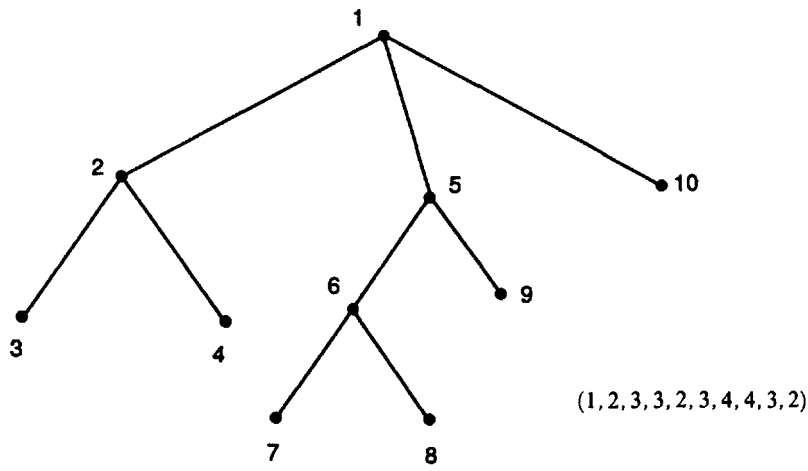

FIG. 5.1. *An ordered, rooted tree, with its visitation and level sequences.*

ically. Then we can say that $U < V$ if $L(U) < L(V)$ in that ordering.

In general, if T is some given rooted tree, then there will be many ordered trees T' such that T' is an ordering of T. We will now specify a single ordered tree T^* to be the *canonical* ordered tree that corresponds to T. It is the one ordered tree, among all of those that correspond to T, whose level sequence is the largest, in the lexicographic ordering of level sequences.

In Fig. 5.2 we show two ordered trees that have the same underlying rooted tree T, and in fact, they are the only two ordered trees that T has. Their level sequences are, respectively, (1,2,3,3,2) and (1,2,2,3,3), of which the first is the larger. Hence T_1 is the canonical ordered tree that corresponds to the rooted tree T, and we write $T_1 = T^*$.

FIG. 5.2. *T corresponds to these two ordered trees.*

We say that two subtrees of an ordered tree are *adjacent* if their roots are two consecutive children of the same parent vertex. In a level sequence $L(T)$, the subsequences that belong to two adjacent subtrees are two consecutive blocks.

We say that a level sequence of a rooted ordered tree is *regular* if whenever U and V are two consecutive (in that order) adjacent subtrees then $L(U) \geq L(V)$. It is then easy to prove the following proposition.

PROPOSITION. *An ordered tree is the canonical ordering of a given rooted tree if and only if its level sequence is regular. In that case, if two consecutive elements of the level sequence are equal to 2, then all remaining elements are equal to 2.* □

Now we're just about ready to write down the algorithm for generating rooted trees of given size. If n is given, then what we are going to do is to write down the level sequences of all canonical ordered trees that correspond to rooted trees of n vertices. In other words, the *encoding* of trees that is used is that a rooted tree is encoded by the level sequence of its associated canonical ordering.

What we would like to avoid is writing down every possible level sequence and discarding the ones that aren't regular. Instead, Beyer and Hedetniemi [BH] found a way to hop directly from one of those level sequences to its successor. It hinges on the positions of the 2's in the original sequence.

Let a level sequence $L = (l_1, l_2, \ldots, l_n)$ be given. If L contains no entry > 2 then the algorithm halts and all trees of n vertices have been generated.

Else, let l_p be the rightmost entry of L that is > 2. Let l_q be the rightmost position preceding p such that $l_q = l_p - 1$ (vertex q is the parent of vertex p).

The successor $s(L)$ of the level sequence L is defined by

$$s(L) = (s_1, \ldots, s_n)$$

where

$$s_i = \begin{cases} l_i & \text{for } i = 1, 2, \ldots, p-1, \\ s_{i-(p-q)} & \text{for } i = p, \ldots, n. \end{cases}$$

What this amounts to is that we take a certain subtree and make enough copies of it to fill in the n entries of the level vector, including possibly a partial copy at the end.

We show in Fig. 5.3 some examples of canonical level sequences and their successors.

L(T)	s(L(T))
(1,2,3,2,2,2)	(1,2,2,2,2,2)
(1,2,3,4,2,2)	(1,2,3,3,3,3)
(1,2,3,4,5,5,2,2,2,2)	(1,2,3,4,5,4,5,4,5,4)
(1,2,3,3,2,3,3)	(1,2,3,3,3,3,3)

FIG. 5.3. *Examples of successors.*

This page intentionally left blank

CHAPTER 6

Random Selection of Free Trees

In [NW] we discussed how to choose, uniformly at random, a rooted tree of a given number n of vertices. It isn't hard to extend this to an algorithm for choosing, u.a.r., a *free* (i.e., unlabeled and unrooted) tree of n vertices, so we will discuss that extension here, following [Wi]. Hence we assume that Algorithm Ranrut of [NW] is available, and we will use it here as a subroutine.

The main point is that free trees have hidden "roots," called *centroids*. It turns out that every free tree has exactly one or two of these distinguished vertices, and that they can serve as roots for Algorithm Ranrut.

We remark here that Algorithm Ranrut has unknown (to me, anyway) average complexity. It seems that it runs in linear or just slightly superlinear time, but the analysis appears to be difficult. I am willing to make a formal conjecture that, for every $\epsilon > 0$, Algorithm Ranrut will select a rooted tree of n vertices in average time $O(n^{1+\epsilon})$, and perhaps even $\epsilon = 0$.

To get back to free trees, we will first define a *centroid*. For each vertex v of a free tree T we let $w(v)$, the *weight* of v, denote the size of the largest subtree that is rooted at v. A centroid v^* is a vertex of minimum weight. The main properties of centroids are (see [Kn1] for proofs) the following:

(1) Every tree T has exactly one or exactly two centroids;
(2) T has two centroids if and only if these are joined by an edge of T, and removal of that edge would leave two trees of equal sizes;
(3) A vertex X is the unique centroid of a tree T of n vertices if and only if $w(X) \leq (n-1)/2$.

The algorithm that we will describe for choosing a free tree of n vertices u.a.r. will first decide, with the correct probabilities, whether it is going to output a tree of one centroid or a tree that has two centroids; then it will construct a tree of the selected type. We consider the two cases separately.

Suppose first that the output tree has been decreed to have two centroids. Then (n is even and) it will consist of two rooted trees of $n/2$ vertices each, with their roots joined by a new edge.

We need to discuss the probabilities with which the various pairs of rooted trees are to be produced. For example, a set of three rooted trees A, B, C of the same size will generate six bicentroidal trees of twice that size, namely one corresponding to each of the 2-samples

$$\{A,A\},\{A,B\},\{A,C\},\{B,B\},\{B,C\},\{C,C\}.$$

The procedure of Chapter 4, which produces a 2-sample, in this case of

rooted trees of size $n/2$, is the correct one for this application. Hence to choose a bicentroidal tree of n vertices u.a.r., let t_n be the number of rooted trees of n vertices. Using Algorithm 2-Sample, of Chapter 4, choose a 2-sample of rooted trees of $n/2$ vertices, with Algorithm Ranrut. Connect their roots by an edge and output the resulting tree of n vertices.

Next we will describe how to choose, u.a.r., a tree of n vertices that has just one centroid. This is clearly equivalent to choosing a rooted tree from among those rooted trees whose centroid and root are identical. Again, by property (3) of centroids, above, what we must do is choose a rooted tree of n vertices u.a.r. from among those whose root subtrees are all of sizes $\leq (n-1)/2$. Equivalently, once more, we may produce a rooted *forest* of $n-1$ vertices u.a.r. from among those such that each connected component has $\leq (n-1)/2$ vertices.

Hence we consider the following more general problem. Given integers m, q, produce u.a.r., a rooted forest of m vertices whose connected components each contain $\leq (q-1)/2$ vertices, from the set of $\mathcal{F}(m,q)$ of all such.

For fixed q, the set $\mathcal{F}(q)$ of all rooted forests, of unrestricted numbers of vertices, whose trees each contain $\leq q$ vertices, forms a prefab (see [NW] for definitions) itself. That is, the primes are all trees of $\leq q$ vertices, and the objects are all possible forests formed freely from those trees. In this prefab $\mathcal{F}(q)$, the set $\mathcal{F}(m,q)$ is the set of all elements of order m.

Thus, the algorithm of [NW] that chooses u.a.r. an object of given order from a prefab applies here without modification. First the counting function $\alpha(m,q) = |\mathcal{F}(m,q)|$ satisfies

$$\sum_{m\geq 0} \alpha(m,q)x^m = \prod_{j=1}^{q}(1-x^j)^{-t_j}.$$

The procedure specializes, in this case, to the following.

ALGORITHM FOREST (m,q).
[m,q are given; returns a rooted forest of m vertices, each of whose trees has $\leq q$ vertices, selected u.a.r.]
If $m=0$ then do Forest := $\emptyset$, else do
(F1) Choose a pair of integers (j,d) such that
$$Prob(j,d) = d\alpha(m-jd,q)t_d/(m\alpha(m,q)) \quad (j\geq 1; 1\leq d\leq q)$$
(F2) Recursively put
$\mathcal{F}'$:= Forest$(m-jd,q)$;
T' := Ranrut](d);
Forest := $\mathcal{F}'$ together with j copies of T'
End. □

It is worth remarking that, in the recursive step (F2), the index q never changes. This shows that we remain inside the same prefab $\mathcal{F}(q)$ at all times, while dealing with objects of different orders inside that prefab.

Finally, let a_j denote the number of free trees of j vertices ($j=1,2,\ldots$). To select a free tree we have the following algorithm.

ALGORITHM FREE(n).
[*Returns a free tree of n vertices, chosen u.a.r.*]
(T1) If n is odd then $p := 0$ else $p := \binom{1+t_{n/2}}{2}/a_n$
(T2) With probability p **do**
[*the output tree will have two centroids*]
Use Algorithm 2-Sample of Chapter 4 to choose a pair of rooted trees of $n/2$ vertices;
Draw a new edge between their roots to obtain the output tree Free.
else do
[*the output tree will have one centroid only*]
$\mathcal{F}$:=Forest($n-1,(n-1)/2$);
Free:=the tree obtained from $\mathcal{F}$ by adjoining a new vertex and joining it to all of the roots of trees of $\mathcal{F}$; □

This page intentionally left blank

CHAPTER 7

Listing Free Trees

Although the *centroid* played the key role in the random selection of free trees, it is the related, but different, idea of the *center* that turns out to be most helpful in listing them. In Chapter 5 we discussed a method of listing rooted trees of given size in constant average time. In this chapter we will follow [WROM] to solve the same problem for free trees.

First, in a tree T, a vertex z is a *center* if its maximum distance to any vertex of T is minimum. Consider the tree of Fig. 7.1.

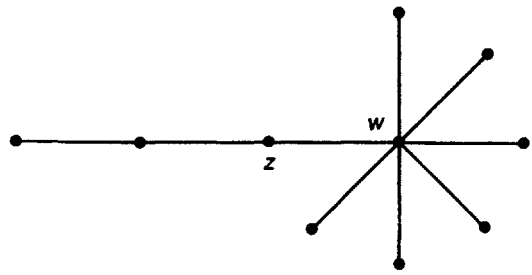

FIG. 7.1. *Center z, centroid w.*

There the vertex z is the center of T since it is at a distance ≤ 2 from every other vertex of T whereas each other vertex is at a distance ≥ 3 from some vertex of T. On the other hand z is not a centroid of T because it has a subtree of size 7 ($> (n-1)/2 = 4.5$) whereas at w all of the subtrees are of size ≤ 3. Thus z is the unique center and w is the unique centroid.

The main facts about centers of trees that we need are

(i) every tree has exactly one or exactly two centers;
(ii) if T is bicentral then its centers are adjacent;
(iii) z is a center of T if and only if it is a center of every path of maximum length in T.

The method of [WROM] also uses the representation of trees by their level sequences, as in [BH]. Recall that the essence of the method of Chapter 5 was that we proceed from the level sequence of a rooted tree to the level sequence of its successor by a simple algorithm which most of the time does very little work, and is fast on average.

The method of [WROM] extends and refines this idea. In it we take pos-

sibly a larger jump from one level sequence, representing a certain *free* tree, to its successor, in such a way that we leap over all but exactly one rooted version of each free tree.

Let T be a free tree, and contemplate all possible rooted trees $\tilde{T}$ that might arise from T by distinguishing some vertex as a root. Furthermore, we require that the root must be a center of T. Hence, associated with each free tree there are either one or two rooted trees $\tilde{T}$.

If T is a tree with just one center then we will associate with T the unique rooted tree $\tilde{T}$ whose root is that center.

Now suppose T is bicentral. We want to provide an unambiguous rule for choosing one of its centers to be the root of $\tilde{T}$. To do this we begin by defining a certain linear ordering of the set of all rooted trees of all sizes.

Let T' and T'' be two rooted trees. Say that $T' \prec T''$ if either

(a) $|V(T')| < |V(T'')|$ or

(b) $|V(T')| = |V(T'')|$ and the canonical level sequence (see Chapter 4) of T' strictly lexicographically precedes that of T''.

Here is how we make that unique choice of a root for a free tree T:

(a) if T has just one center then root T at that center, else

(b) if T has two centers then delete the edge that joins them, obtaining two rooted subtrees T', T''. The rooted version $\tilde{T}$ of T is created by placing the root at the root of T'' if $T' \prec T''$ or at T' if $T'' \preceq T'$.

We will call the rooted tree $\tilde{T}$ that has been uniquely associated with each free tree T, the *primary* rooted version of T.

Imagine that the set of all canonical level sequences of all rooted trees of n vertices is displayed in a certain list. The algorithm of [BH] showed how to march from each element of this list to its successor.

Imagine next that on the list only those sequences that correspond to primary rooted versions of free trees have each been marked with an asterisk.

The question before us is this: how do we refine the algorithm of [BH] so that instead of taking us from one member of the list to its successor, it will take us from a starred member of the list to the next starred member of the list? When that question has been answered we will obviously have an algorithm for listing all free trees of n vertices.

The strategy of [WROM] is the following. If T is a free tree, and if $\tilde{T}$ is its primary rooted version, then let $L(\tilde{T})$ be the canonical level sequence of $\tilde{T}$. We will call $L(\tilde{T})$ the *primary canonical level sequence* of the free tree T (these are the starred level sequences on the list that we were imagining above).

Example. Let $n = 5$. There are 3 free trees of 5 vertices, as shown in Fig. 7.2. These 3 trees give rise to 9 rooted trees, as shown in Fig. 7.3.

The canonical level sequences of these 9 rooted trees are shown in Fig. 7.4, where we have listed them in descending lexicographic order, as they would have been generated by the algorithm of [BH].

Now let's find which 3 of these 9 rooted trees are the primary rooted versions of the 3 free trees of Fig. 7.2.

First, T_1 has just one center, so its primary root is that center, and $T_{13} = \tilde{T}_1$.

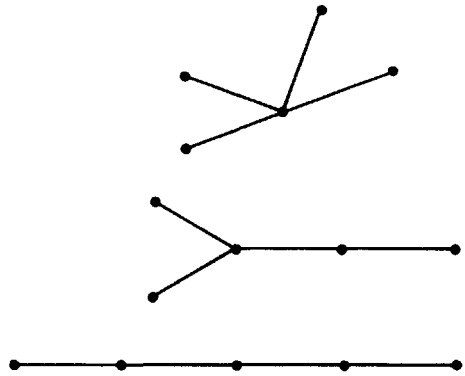

FIG. 7.2. *The free trees of five vertices.*

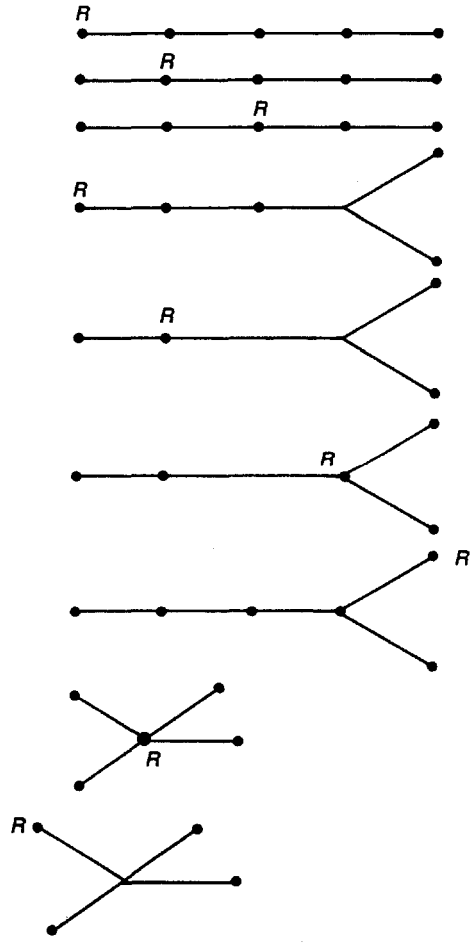

FIG. 7.3. *Nine ways to root three free trees.*

T_{11} : 12345
T_{21} : 12344
T_{24} : 12343
T_{12} : 12342
T_{32} : 12333
T_{22} : 12332
T_{13} : 12323
T_{23} : 12322
T_{31} : 12222

FIG. 7.4. *Level sequences.*

Further, T_3 has just one center, and $T_{31} = \tilde{T}_3$.

Finally, T_2 is bicentral, so we have to work a little harder. If we delete the edge that joins the centers of T_2 we obtain the trees

T' T''

Since $|V(T')| = 2 < |V(T'')| = 3$, we have $T_2' \prec T_2''$, and therefore we choose the root of T_2'' for the primary root of T_2. Thus $\tilde{T}_2 = T_{23}$.

We are now in a position to look at the list that we had previously only imagined. It shows (see Fig. 7.5), for $n = 5$, the canonical level sequences of all rooted trees of 5 vertices, and marked with asterisks are the ones that are the primary sequences of free trees.

12345
12344
12343
12342
12333
12322
*12323 $\tilde{T}_1$
*12322 $\tilde{T}_2$
*12222 $\tilde{T}_3$

FIG. 7.5. *Canonical and primary-canonical sequences for* n=5.

One visual impression here turns out to be quite relevant. The primary sequences, in this case, form *one contiguous block.* They don't always form one contiguous block, but they *do* always form relatively few contiguous blocks, and that is what is responsible for the speed of the [WROM] algorithm, and of course, that is why the choice of primary rooting was made the way it was. □

Indeed, if it is true that the primary sequences in general comprise relatively few blocks, then "most of the time" the succession algorithm of [BH], which was designed for rooted trees, will produce the next primary sequence that follows a given one. It will fail to do so only when the given one is at the end of a contiguous block of primary sequences.

So here is a summary of how the [WROM] algorithm works.

Begin with the primary canonical sequence of the n–path rooted at a center.

In general, having arrived at a primary canonical level sequence L of some tree T, test it to see if its successor, $s(L)$, will be primary, i.e., to see if its successor still has an asterisk. If so, go to that successor. If not, go to the top of the next contiguous block of asterisks that lie below.

That raises two questions. How do we test for the bottom of a starred block, and how do we find the top of the next such block?

First, let T be a rooted tree, z its root, having n vertices and canonical level sequence $L = (l_1, \ldots, l_n)$. By the *principal subsequences* of L we mean the subsequences that correspond to the root subtrees of T. These each begin with a 2, contain no other 2's, are consecutive in L, and are maximal with respect to those properties.

For $i = 1, 2, \ldots$ let h_i denote the position of the first occurrence of the highest level number in the ith principal subsequence of L. As an example, in the level sequence of $\tilde{T}_1$ of Fig. 7.4, the sequence is 12323, the principal subsequences are 23 and 23, and we have $h_1 = 3$, $h_2 = 5$. Next let m denote the position at which the second principal subsequence begins ($m = 4$ in the example) in L.

THEOREM 7.1. *A canonical level sequence $L = (l_1, l_2, \ldots, l_n)$ of a rooted tree is a primary canonical level sequence of the underlying free tree if and only if*

(a) *h_2 exists (i.e., there is more than one principal subsequence), and*

(b) *$l_{h_2} \geq l_{h_1} - 1$, and*

(c) *if equality holds in* (b) *then $m - 2 \leq n - m + 2$, and*

(d) *if equality holds in* (b) *and* (c) *then let L_1 be the first principal subsequence of L, with all of its entries reduced by* 1, *and let L_2 be the rest of L, i.e., the initial* 1 *and all entries to the right of L_1. We must have $L_1 \preceq L_2$ in the ordering defined above.* □

Example. Consider the level sequence $L(\tilde{T}_2) = (1, 2, 3, 2, 2)$. Here $h_1 = 3$, $h_2 = 4$, $n = 5$, $m = 4$, $L_1 = (1, 2)$, $L_2 = (1, 2, 2)$. Then (a) holds. For (b) we check that $l_4 \geq l_3 - 1$. Since equality holds, we check, in (c), that $4 - 2 \leq 5 - 4 + 2$, and since the inequality is strict the sequence is primary.

The first step in the [WROM] succession algorithm is to compute the successor $s(L)$ of the current primary sequence L, using the succession rule of [BH], as in the case of rooted trees.

Next, check that $s(L)$ satisfies the conditions of Theorem 7.1. If so, then output $s(L)$ as the next free tree, encoded as a level sequence. If not, then we take a long jump down the list of rooted trees, as follows:

(i) Begin with L;

(ii) Save its value of m; call it m^*;

(iii) Do $L := s(L)$;

(iv) If $l_{m^*-1} > 3$ then replace the final $h_1 - 1$ entries of L with $2, 3, \ldots, h_1$.

The algorithm starts with the level sequence of an n-path rooted at a center, which is (if $n \geq 4$)

$$L = (1, 2, \ldots, k, 2, 3, \ldots, n - k + 1)$$

where $k = \lfloor n/2 \rfloor + 1$.

The complexity of this procedure is that, on the average, it operates in constant time per tree generated. The step from one *rooted* tree to the next is, as we saw in Chapter 5, done in average constant time. Hence the complexity analysis in the free tree case must show that testing for exceptional cases and carrying out remedial action also can be done in average constant time. Some of the details of this analysis are quite delicate, and the interested reader should consult the original paper [WROM].

CHAPTER 8

Generating Random Graphs

In the previous chapters there has been, as the reader has no doubt noticed, a definite correlation between the ease with which various combinatorial objects can be chosen uniformly at random and the ease with which they can be counted. Sets, subsets, partitions, and the like can be counted rather pleasantly, and fairly transparent recursive proofs can be given for the resulting counting formulas. As soon as we have such proofs along with "really" combinatorial proofs of them, the way is open for a random selection algorithm.

By the time we get to rooted and free trees, the enumeration techniques begin to get a little bit sophisticated, and the corresponding algorithms become somewhat less transparent. In this chapter we will talk about the uniform selection of graphs. Here we enter a subject where the enumeration answers aren't known in many of the most important cases. For instance, we don't know how many planar graphs there are on n labeled vertices, so we don't expect to find a random selection algorithm for planar graphs, not, at least, such an algorithm that models directly from a counting process. The same is true, say, for Hamiltonian graphs, for 3-colorable graphs, and many other graph families that are of great theoretical importance.

In this chapter we will push the methods of the previous chapters a little further by looking at unlabeled graphs. For results on labeled regular graphs, and labeled bipartite graphs with given degree sequence, see Wormald [Wo].

The methods of this chapter will still depend on our ability to count the graphs that we are trying to sample, but new kinds of problems will surface whose resolution seems instructive. Very recently attention has been paid to the random selection of combinatorial objects where we do not know how to enumerate the objects themselves in any efficient way. In such cases we may settle for "almost uniform" distribution of the objects produced as the price we pay for our lack of information on the numbers of objects that are in the family. One way to do this is to construct a small family of basic transformations that generate the whole set of objects from which we want to sample. Then begin with some particular object and carry out some large number of randomly chosen basic transformations. It will often happen that enough mixing takes place that the resulting object will have a very nearly uniform distribution, but the number of moves that need to be executed may well be too large to permit efficient use of the method.

As an introduction to the problems of unlabeled counting and random selection, consider the famous problem of the "necklaces." We fix two positive

integers n, k, and consider the set of all circular arrangements of n beads, where each of the beads is colored in one of k given colors. We regard two such necklaces as identical if a rotation of one of them carries it into the other. We do not regard them as identical if a flip of one necklace results in the other. We could carry out this analysis using the dihedral group of symmetries, but the cyclic group will be quite sufficient to show the ideas involved.

A standard question in many combinatorics texts asks for the number of such necklaces, and it turns out to be

$$f(n,k) = \frac{1}{n}\sum_{d|n} \phi(n/d)k^d \tag{8.1}$$

where ϕ is Euler's function. The appearance of the "$1/n$" in front of this formula means that the sum by itself does overcounting that needs to be compensated for. Consequently, it will be hard to use the derivation of this formula as a model for an algorithm that would produce a list of all (n,k) necklaces. Indeed, such an algorithm would involve examining each necklace as it is produced and checking somehow to see if it had previously been produced. Hence there are considerable difficulties involved in *listing* necklaces of given type. As far as I know there are no known *loop-free* algorithms for doing so. That is, they all engage in some following of blind alleys with backtracking, rejection, and ultimate selection.

The random selection process is a lot easier because the factor of $1/n$ that caused the troubles in listing is not a bit troublesome for random selection since the same factor is common to all necklaces, and it doesn't affect the uniformity of the distribution. Thus there are various ways of sampling necklaces with equal probabilities. We would like now to describe one of these, but in a setting that is a little more general than the necklace problem. Just as in the problem of counting the necklaces, where one can use very special methods or one can invoke Burnside's lemma, we choose here to give an algorithm that randomly selects an orbit of the action of a group of permutations on a set. That is, the algorithm will apply to precisely the situations that Burnside's lemma covers. The discussion follows [DW].

When a finite group G acts as a group of permutations on a set Ω it induces an equivalence relation on Ω, in which the classes are called the *orbits* of the action of G. For every $g \in G$ we let $Fix(g)$ be the set of objects (elements of Ω) that are fixed by g. Then classical Burnside theory gives us the following facts.

(a) The number of orbits is

$$m = \frac{1}{|G|}\sum_{g\in G} |Fix(g)|. \tag{8.2}$$

(b) For each orbit ω, define

$$\Gamma_\omega = \{(g,\alpha) \in G \times \Omega | \alpha \in \omega \cap Fix(g)\}. \tag{8.3}$$

Then $|\Gamma_\omega| = |G|$ for all orbits ω.

(c) For each orbit ω and for each conjugacy class C of G, every element of C has the same number of fixed points in ω. In particular, each element of G has the same number of fixed points.

Property (b) above is the basis for the algorithm that selects an orbit of the action uniformly at random. Indeed, it implies that the multiset $\cup_{g\in G} Fix(g)$ contains exactly $|G|$ representatives of each orbit, and consequently if we choose an element from that multiset u.a.r., then the orbit in which the element lives is distributed u.a.r.

The algorithm that follows uses that idea, but economizes it by working with conjugacy classes rather than with the elements of G themselves, as permitted by property (c) above.

Let $C_1, \ldots, C_r$ be the conjugacy classes of G. Let $h_j = |C_j|$ and let g_j be a representative of C_j, for each $j = 1, \ldots, r$. Further, let the *weight* of a conjugacy class C_j be $w_j = h_j|Fix(g_j)|$, and note that from property (a) above, we have $\sum_j w_j = m|G|$.

In terms of these parameters we can now state the algorithm. We will then give two examples of its operation. The first will be to the necklace problem, where the group, the conjugacy classes, etc., are all very transparent, and the second will be to the selection of unlabeled graphs, where life is more complicated.

ALGORITHM RANDOM ORBIT.

[Input is a group G that acts on a set Ω. Output is an orbit ω chosen u.a.r. from the orbits of the group action.]

RO1 Choose a conjugacy class of G so that the probability of choosing C_j is $Prob(C_j) = w_j/(m|G|)$ $(j = 1, \ldots, r)$, where m is the number of orbits, given by (8.1).

RO2 Let C_j be the conjugacy class that was chosen in step RO1, and let g_j be its representative. Then choose an element α u.a.r. from $Fix(g_j)$.

RO3 Return the orbit ω that contains α. □

The following computation shows that each orbit ω has the same a priori probability $(= 1/m)$ of being chosen by this algorithm:

$$
\begin{aligned}
Prob(\omega) &= \sum_{j=1}^{r} \frac{w_j}{m|G|} \frac{|Fix(g_j) \cap \omega|}{|Fix(g_j)|} \\
&= \sum_{j=1}^{r} \frac{h_j|Fix(g_j)|}{m|G|} \frac{|Fix(g_j) \cap \omega|}{|Fix(g_j)|} \\
&= \frac{1}{m|G|} \sum_{g\in G} |Fix(g) \cap \omega| \qquad \text{(by (c))} \\
&= \frac{|\Gamma_\omega|}{m|G|} \\
&= \frac{1}{m} \qquad \text{(by (b))}.
\end{aligned}
$$

Example 1. Choosing necklaces at random. In this example we fix two integers $1 \leq k \leq n$, and take the set Ω of objects to be the set of k^n necklaces of n *labeled* beads, where each bead is colored in one of the k colors $1, 2, \ldots, k$. The beads are labeled with $1, 2, \ldots, n$, so each bead has a label that gives its position on the necklace, and a color.

The group G is the cyclic group of order n, which we take in the form

$$G = \{c, c^2, c^3, \ldots, c^{n-1}, c^n = 1\}$$

where c is a cyclic right shift of one unit.

For each j, $1 \leq j \leq n$, the mapping c^j has $gcd(n, j)$ cycles, each of length $n/gcd(n, j)$. Since the cycle length set is the only conjugacy invariant, there is a conjugacy class C_d for each divisor d of n. There are $r = d(n)$ conjugacy classes altogether, and the class C_d contains exactly $h_d = |C_d| = \phi(n/d)$ elements of G, for each $d|n$.

A representative of C_d is $g_d = c^d$ and

$$|Fix(g_d)| = |Fix(c^d)| = k^{\#\text{ cycles of } c^d} = k^d.$$

It is now trivial to check that Burnside's lemma gives the number of orbits m as in (8.1) above. The weight w_d of the class C_d is $\phi(n/d)k^d$, and Algorithm Random Orbit boils down to the following:

1. Choose a class C_d with probability

$$Prob(C_d) = \frac{\phi(n/d)k^d}{\sum_{d'|n} \phi(n/d')k^{d'}} \qquad (d|n).$$

2. Choose d colors from the available set $\{1, 2, \ldots, k\}$ independently u.a.r. Then color beads $1, d+1, 2d+1, \ldots$ in the first chosen color, and beads $2, d+2, 2d+2, \ldots$ in the second color chosen, etc. until all d chosen colors have been used (and all beads have been colored!). We have now chosen an element α u.a.r. from $Fix(c^d)$.

3. Erase the labels of the beads, leaving only their colors, and output the resulting necklace.

Example 2. Choosing unlabeled graphs at random. The setting is now the following: A positive integer n is fixed. The set Ω of objects is the set of all vertex-labeled graphs of n vertices. The group that acts on Ω is the symmetric group S_n and it acts by permuting vertex labels.

Now consider, for a fixed permutation $g \in S_n$, the set $Fix(g)$ of graphs that are fixed by g. We associate with g another permutation g^* which acts not on the n letters $1, 2, \ldots, n$ but on the $\binom{n}{2}$ unordered pairs of those letters. Alternatively, we can say that g^* acts on the *edges* of the complete graph K_n. Its action is just this: $g^*\{i, j\} = \{gi, gj\}$. Now $Fix(g)$ can be described easily. It is the set of all graphs whose edge sets are mapped into themselves by g^*. Again, a graph H is fixed by g if and only if for each cycle q of the induced permutation g^* either all pairs in q are edges of H or none of them are.

It is easy to count the graphs that g fixes. For each cycle of g^* we must either take all of the edges in the cycle or none of them, so there are exactly $2^{c(g)}$ graphs in $Fix(g)$, where $c(g)$ is the number of cycles of the pair permutation g^*.

Next we want to express these counts in a more quantitative nuts-and-bolts fashion. Hence we let $(k_1, k_2, \ldots, k_n)$ denote the partition of the integer n that has exactly k_i parts of size i, for all $i \geq 1$. We let $[k_1, k_2, \ldots, k_n]$ denote the corresponding conjugacy class of S_n, which consists of all permutations that have k_i cycles of size i for each $i \geq 1$.

A well-known and elementary counting argument shows that the number of permutations in the conjugacy class $[k_1, k_2, \ldots, k_n]$ is exactly

$$\frac{n!}{\prod_i (i^{k_i} k_i!)}.$$

The second fact that we need is less well known. It states the following.

THEOREM 8.1. *The number of cycles in the induced pair permutation g^* can be expressed in terms of the cycle structure of g itself as*

$$c(g) = \sum_{i<j} k_i k_j gcd(i,j) + \sum_i i \left(k_{2i} + k_{2i+1} + \binom{k_i}{2} \right)$$
$$= \frac{1}{2} \left\{ \sum_{i=1}^{n} l(i)^2 \phi(i) - l(1) + l(2) \right\}$$

where ϕ is Euler's function and $l(i) = \sum_{i|j} k_j$.

The following proof is from Robinson [Ro]. Consider separately the edges of the complete graph K_n that join distinct cycles of g and those that join points of the same cycle of g. There are ij edges in K_n that join distinct cycles of length i and length j of g, and g^* acts on these to produce $gcd(i,j)$ cycles of $lcm(i,j)$ points each. On the other hand, the edges of K_n that join points of a single cycle of g whose length is $2i+1$ are permuted into i cycles of length $2i+1$, while the edges in a single cycle of length $2i$ are permuted into $i-1$ cycles of length $2i$ and one of length i, and the formula results immediately. □

The orbits of the action of S_n on Ω are the unlabeled graphs of n vertices. Hence, Algorithm Random Orbit applies to this situation and will produce unlabeled graphs of n vertices with equal a priori probabilities.

The weight of the conjugacy class $[k_1, \ldots, k_n]$ is

$$w([k_1, \ldots, k_n]) = \frac{n! 2^{c(g)}}{\prod_i (i^{k_i} k_i!)}$$

where $c(g)$ is computed from the theorem above.

ALGORITHM RANDOM GRAPH.

RG1 Choose a partition of the integer n such that the probability of choosing $\pi = (k_1, \ldots, k_n)$ is

$$Prob(\pi) = w([k_1, \ldots, k_n])/n!g_n.$$

RG2 Take a representative g from the class that corresponds to the partition π, and choose u.a.r. a (labeled) graph H from $Fix(g)$.

RG3 Erase the labels and output the unlabeled graph H. □

Concerning the actual implementation of this algorithm, consider step RG2 first. To choose the representative of the conjugacy class, all we need is any permutation whose cycle partition is the given one. That's easy to do. A permutation for which $[k_1, k_2, k_3] = [2, 2, 1]$, for example, is (1)(2)(34)(56)(789).

Next we have to choose a graph that is fixed by the induced permutation g^*. To do that, just follow around all of the cycles of g^*, and then decide, independently u.a.r. for each cycle, whether to admit all of its edges into the graph H or to admit none of them. The time required to trace the cycles of g^* and make these decisions is clearly $O(n^2)$.

In step RG1 we must select a partition of the integer n with certain given probabilities attached to each partition. There are a great many partitions of n, around $e^{K\sqrt{n}}$ of them, so we do not want to have to look at very many of them, on average, before finding the winner. Hence it is important to know which of the partitions are most likely to occur, and to arrange the calculation so we examine the partitions in roughly decreasing order of their likelihoods of being chosen. In that way we'll be more likely to get the game over quickly.

The fact is that the partitions of n that have many parts = 1 are the ones that are most likely to be chosen. Indeed the one partition whose parts are *all* ones has by far the largest probability. Its weight is $2^{\binom{n}{2}}/n!$, and that one term already gives the correct asymptotic behavior of the number of unlabeled n-graphs.

It was shown by Oberschelp [Ob] that the total probability of all partitions of n that have no more than r 1's is $O(n^r 2^{-nr/2})$, so the probabilities decrease geometrically as the number of parts = 1 decreases. A careful analysis shows that the average number of partitions that need to be looked at before selecting one is *no more than 3* if they are examined in a sequence that respects the number of 1's in the partition. For the details, see [DW].

The result is that with a careful implementation we can choose unlabeled n-graphs u.a.r. in average time $O(n^2)$ per graph, which is surely the best that could be expected.

Bibliography

[Al] BRIAN ALSPACH, *The search for long paths and cycles in vertex-transitive graphs and digraphs*; in Combinatorial Mathematics VIII, Proc. Eighth Australian Conference on Combinatorial Math., Lecture Notes in Mathematics 884, Springer-Verlag, Berlin, 1981, pp. 14-22.

[Ar] BENJAMIN ARAZI, *An approach for generating different types of Gray codes*, Inform. and Control, 63 (1984), pp. 1-10.

[Ba] L. BABAI, *Long cycles in vertex-transitive graphs*, J. Graph Theory, 3 (1979), pp. 301-304.

[BER] JAMES R. BITNER, GIDEON EHRLICH, AND EDWARD M. REINGOLD, *Efficient generation of the binary reflected Gray code and its applications*, ACM Comm., 19 (1976), pp. 517-521.

[BH] TERRY BEYER AND SANDRA MITCHELL HEDETNIEMI, *Constant time generation of rooted trees*, SIAM J. Comput., 9 (1980), pp. 706-712.

[BW] M. BUCK AND D. WIEDEMANN, *Gray codes with restricted density*, Discrete Math., 48 (1984), pp. 19-29.

[CLD] C. C. CHANG, R. C. T. LEE, AND M. W. DU, *Symbolic Gray code as a perfect multiattribute hashing scheme for partial match queries*, IEEE Trans. Software Engrg., 8 (1982), pp. 235-249.

[DW] JOHN D. DIXON AND HERBERT S. WILF, *The random selection of unlabeled graphs*, J. Algorithms, 4 (1983), pp. 205-212.

[DZ] E. A. DINITS AND M. A. ZAITSEV, *Algorithm for the generation of nonisomorphic trees*, Avtomat. i Telemekh., 4 (1977), pp. 121-126; Automat. Remote Control, 38 (1977), pp. 554-558.

[EHR] P. EADES, M. HICKEY, AND R. C. READ, *Some Hamilton paths and a minimal change algorithm*, J. Assoc. Comput. Mach., 31 (1984), pp. 19-29.

[EM] PETER EADES AND BRENDAN MCKAY, *An algorithm for generating subsets of a fixed size with a string minimal change property*, preprint,1982.

[Er] M. C. ER, *Two recursive algorithms for generating the binary reflected Gray code*, J. Inform. Optim. Sci., 6 (1985), pp. 213-216.

[Fr] MICHAEL FREDMAN, *Observations on the complexity of generating quasi-Gray codes*, SIAM J. Comput., 7 (1978), pp. 134-146.

[FR] P. FLAJOLET AND LYLE RAMSHAW, *A note on Gray code and odd-even merge*, SIAM J. Comput., 9 (1980), pp. 142-158.

[Gi] E. N. GILBERT, *Gray codes and paths on the n-cube*, Bell System Tech. J., 37 (1958), pp. 815-826.

[Gr] Bell System Tech. J., 18 (1939), p. 252.

[Ha] RICHARD W. HAMMING, *Coding and Information Theory*, Prentice–Hall, Englewood Cliffs, N.J., 1980.

[Im] W. IMRICH, *On the connectivity of Cayley graphs*, J. Combin. Theory Ser. B, 26 (1979), pp. 323-326.

[Ja1] BILL JACKSON, *Hamilton cycles in regular 2-connected graphs*, J. Combin. Theory Ser. B, 29 (1980), pp. 27-46.

[Ja2] ———, *Longest cycles in 3-connected cubic graphs*, J. Combin. Theory Ser. B, 41 (1986), pp. 17-26.

[JW] J. T. JOICHI AND DENNIS E. WHITE, *Gray codes in graphs of subsets*, Discrete Math., 31 (1980), pp. 29-41.

[JWW] J. T. JOICHI, DENNIS E. WHITE, AND S. G. WILLIAMSON, *Combinatorial Gray codes*, SIAM J. Comput., 9 (1980), pp. 130-141.

[Ka] RICHARD KAYE, *A Gray code for set partitions*, Inform. Process. Lett., 5 (1976), pp. 171-173.

[Kl] PAUL KLINGSBERG, *A Gray code for compositions*, J. Algorithms, 3 (1982), pp. 41-44.

[Kn1] DONALD E. KNUTH, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, Addison–Wesley, Reading, MA, 1968.

[Kn2] ———, personal communication, 1988.

[Ko] A. V. KOZINA, *Coding and generation of nonisomorphic trees*, Cybernetics, 15 (1975), pp. 645-651. Kibernetika, 5 (1979), pp. 38-43. (In Russian.)

[KP] P. KIRSCHENHOFER AND H. PRODINGER, *Subblock occurrences in positional number systems and Gray code representation*, J. Inform. Optim. Sci., 5 (1984), pp. 29-42.

[Lov] L. LOVÁSZ, *Problem 11, Combinatorial structures and their applications*, Gordon and Breach, New York, 1970.

[Lu] HEINZ LUNEBERG, *Gray codes*, Abh. Math. Sem. Univ. Hamburg, 52 (1982), pp. 208-227.

[Luc] J. LUCAS, *The rotation graph of binary trees is Hamiltonian*, Tech. Report TR-021, Dept. of Computer Science, Princeton University, Princeton, N.J., 1986.

[NW] A. NIJENHUIS AND H. S. WILF, *Combinatorial algorithms*, second edition, Academic Press, New York, 1978.

[Ob] W. OBERSCHELP, *Kombinatorische anzahlbestimmungen in Relationen*, Math. Annalen, 174 (1967), pp. 53-58.

[Pr] HELMUT PRODINGER, *Nonrepetitive sequences and Gray code*, Discrete Math., 43 (1983), pp. 113-116.

[PR] ANDRZEJ PROSKUROWSKI AND FRANK RUSKEY, *Binary tree Gray codes*, J. Algorithms, 6 (1985), pp. 225-238.

[Re] R. C. READ, *How to grow trees*, in Combinatorial Structures and Their Applications, Gordon and Breach, New York, 1970.

[RH] F. RUSKEY AND T. C. HU, *Generating binary trees lexicographically*, SIAM J. Comput., 6 (1977), pp. 745-758.

[Ri] DANA RICHARDS, *Data compression and Gray-code sorting*, Inform. Process. Lett., 22 (1986), pp. 201-205.

[RM] F. RUSKEY AND D. J. MILLER, *Adjacent interchange generation of combinations and trees*, University of Victoria, British Colombia, Canada, DCS-44-IR, 1984.

[Ro] ROBERT W. ROBINSON, *Enumeration of Euler graphs*, in Proof Techniques in Graph Theory, F. Harary, ed., Academic Press, New York, 1969, pp. 147-153.

[RP] FRANK RUSKEY AND ANDRZEJ PROSKUROWSKI, *Generating binary trees by transpositions*, preprint, 1987.

[Ru] FRANK RUSKEY, *Generating t-ary trees lexicographically*, SIAM J. Comput., 7 (1978), pp. 424-439.

[Ru2] ———, *Adjacent interchange generation of combinations*, J. Algorithms, to appear.

[SK1] B. D. SHARMA AND R. K. KHANNA, *J. Combin. Inform. System Sci.*, 4 (1979), pp. 227-236.

[SK2] ———, *On level weight studies of binary and m-ary Gray codes*, Inform. Sci., 21 (1980), pp. 179-186.

[SK3] ———, *On m-ary Gray codes*, Inform. Sci., 15 (1978), pp. 31-43.

[Tr] A. TROJANOWSKI, *Ranking and listing algorithms for k-ary trees*, SIAM J. Comput., 7 (1978), pp. 492-509.

[Wat] MARK E. WATKINS, *Connectivity of vertex transitive graphs*, J. Combin. Theory, 8 (1970), pp. 23-29.

[Wi] H. S. WILF, *The uniform selection of free trees*, J. Algorithms, 2 (1981), pp. 204-207.

[Wo] NICOLAS C. WORMALD, *Generating random regular graphs*, J. Algorithms, 5 (1984), pp. 247-280.

[WROM] ROBERT ALAN WRIGHT, BRUCE RICHMOND, ANDREW ODLYZKO, AND BRENDAN D. MCKAY, *Constant time generation of free trees*, SIAM J. Comput., 15 (1986), pp. 540-548.

[WY] HERBERT S. WILF AND NANCY A. YOSHIMURA, *Ranking rooted trees, and a graceful application*, in Perspectives in Computing, Proc. Japan-U.S. Joint Seminar in Discrete Algorithms and Complexity, June 4-6, 1986, Kyoto, Japan, Academic Press, 1987, pp. 341-350.

[Yo] NANCY ALLISON YOSHIMURA, *Ranking and unranking algorithms for trees and other combinatorial objects*, Ph.D. thesis, Computer and Information Science, University of Pennsylvania, Philadelphia, 1987.

[Yu] C. K. YUEN, *The separability of Gray code*, IEEE Trans. Inform. Theory (1974), p. 668.

[Za1] S. ZAKS, *Lexicographic generation of ordered trees*, Theoret. Comput. Sci., 10 (1980), pp. 63-82.

[Za2] ———, *Generating k-ary trees lexicographically*, Tech. Report UICSCS-R77-901, University of Illinois, Urbana, IL, 1977.

This page intentionally left blank

Index