THE ART OF COMPUTER PROGRAMMING

Missing pages from select printings of Knuth, The Art of Computer Programming, Volume 4A
(ISBN-13: 9780201038040 / ISBN-10: 0201038048).

DONALD E. KNUTH *Stanford University*

ADDISON–WESLEY

Missing pages from select printings of Knuth, The Art of Computer Programming, Volume 4A
(ISBN-13: 9780201038040 / ISBN-10: 0201038048).

Volume 4A / **Combinatorial Algorithms, Part 1**

THE ART OF COMPUTER PROGRAMMING

Boston · Columbus · Indianapolis · New York · San Francisco
Amsterdam · Cape Town · Dubai · London · Madrid · Milan
Munich · Paris · Montréal · Toronto · Mexico City · São Paulo
Delhi · Sydney · Hong Kong · Seoul · Singapore · Taipei · Tokyo

Missing pages from select printings of Knuth, The Art of Computer Programming, Volume 4A
(ISBN-13: 9780201038040 / ISBN-10: 0201038048).

The poem on page 437 is quoted from *The Golden Gate* by Vikram Seth (New York: Random House, 1986), copyright © 1986 by Vikram Seth.

Library of Congress Cataloging-in-Publication Data

```
Knuth, Donald Ervin, 1938-
  The art of computer programming / Donald Ervin Knuth.
  xvi,883 p. 24 cm.
  Includes bibliographical references and index.
  Contents: v. 1. Fundamental algorithms. -- v. 2. Seminumerical
algorithms. -- v. 3. Sorting and searching. -- v. 4a. Combinatorial
algorithms, part 1.
  Contents: v. 4a. Combinatorial algorithms, part 1.
  ISBN 978-0-201-89683-1 (v. 1, 3rd ed.)
  ISBN 978-0-201-89684-8 (v. 2, 3rd ed.)
  ISBN 978-0-201-89685-5 (v. 3, 2nd ed.)
  ISBN 978-0-201-03804-0 (v. 4a)
  1. Electronic digital computers--Programming.  2. Computer
algorithms.   I. Title.
QA76.6.K64  1997
005.1--DC21                                                    97-2147
```

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

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

And see http://www-cs-faculty.stanford.edu/~knuth/mmix.html for basic information about the MMIX computer.

Electronic version by Mathematical Sciences Publishers (MSP), http://msp.org

ISBN-13 978-0-201-03804-0
ISBN-10 0-201-03804-8

Third digital release, March 2017

PREFACE

To put all the good stuff into one book is patently impossible,
and attempting even to be reasonably comprehensive
about certain aspects of the subject is likely to lead to runaway growth.

— GERALD B. FOLLAND, "Editor's Corner" (2005)

THE TITLE of Volume 4 is *Combinatorial Algorithms*, and when I proposed it I was strongly inclined to add a subtitle: *The Kind of Programming I Like Best*. My editors have decided to tone down such exuberance, but the fact remains that programs with a combinatorial flavor have always been my favorites.

On the other hand I've often been surprised to find that, in many people's minds, the word "combinatorial" is linked with computational difficulty. Indeed, Samuel Johnson, in his famous dictionary of the English language (1755), said that the corresponding noun "is now generally used in an ill sense." Colleagues tell me tales of woe, in which they report that "the combinatorics of the situation defeated us." Why is it that, for me, combinatorics arouses feelings of pure pleasure, yet for many others it evokes pure panic?

It's true that combinatorial problems are often associated with humongously large numbers. Johnson's dictionary entry also included a quote from Ephraim Chambers, who had stated that the total number of words of length 24 or less, in a 24-letter alphabet, is 1,391,724,288,887,252,999,425,128,493,402,200. The corresponding number for a 10-letter alphabet is 11,111,111,110; and it's only 3905 when the number of letters is 5. Thus a "combinatorial explosion" certainly does occur as the size of the problem grows from 5 to 10 to 24 and beyond.

Computing machines have become tremendously more powerful throughout my life. As I write these words, I know that they are being processed by a "laptop" whose speed is more than 100,000 times faster than the trusty IBM Type 650 computer to which I've dedicated these books; my current machine's memory capacity is also more than 100,000 times greater. Tomorrow's computers will be even faster and more capacious. But these amazing advances have not diminished people's craving for answers to combinatorial questions; quite the contrary. Our once-unimaginable ability to compute so rapidly has raised our expectations, and whetted our appetite for more — because, in fact, the size of a combinatorial problem can increase more than 100,000-fold when n simply increases by 1.

Combinatorial algorithms can be defined informally as techniques for the high-speed manipulation of combinatorial objects such as permutations or graphs. We typically try to find patterns or arrangements that are the best possible ways to satisfy certain constraints. The number of such problems is vast, and the art

of writing such programs is especially important and appealing because a single good idea can save years or even centuries of computer time.

Indeed, the fact that good algorithms for combinatorial problems can have a terrific payoff has led to terrific advances in the state of the art. Many problems that once were thought to be intractable can now be polished off with ease, and many algorithms that once were known to be good have now become better. Starting about 1970, computer scientists began to experience a phenomenon that we called "Floyd's Lemma": Problems that seemed to need n^3 operations could actually be solved in $O(n^2)$; problems that seemed to require n^2 could be handled in $O(n \log n)$; and $n \log n$ was often reducible to $O(n)$. More difficult problems saw a reduction in running time from $O(2^n)$ to $O(1.5^n)$ to $O(1.3^n)$, etc. Other problems remained difficult in general, but they were found to have important special cases that are much simpler. Many combinatorial questions that I once thought would never be answered during my lifetime have now been resolved, and those breakthroughs have been due mainly to improvements in algorithms rather than to improvements in processor speeds.

By 1975, such research was advancing so rapidly that a substantial fraction of the papers published in leading journals of computer science were devoted to combinatorial algorithms. And the advances weren't being made only by people in the core of computer science; significant contributions were coming from workers in electrical engineering, artificial intelligence, operations research, mathematics, physics, statistics, and other fields. I was trying to complete Volume 4 of *The Art of Computer Programming*, but instead I felt like I was sitting on the lid of a boiling kettle: I was confronted with a combinatorial explosion of another kind, a prodigious explosion of new ideas!

This series of books was born at the beginning of 1962, when I naïvely wrote out a list of tentative chapter titles for a 12-chapter book. At that time I decided to include a brief chapter about combinatorial algorithms, just for fun. "Hey look, most people use computers to deal with numbers, but we can also write programs that deal with patterns." In those days it was easy to give a fairly complete description of just about every combinatorial algorithm that was known. And even by 1966, when I'd finished a first draft of about 3000 handwritten pages for that already-overgrown book, fewer than 100 of those pages belonged to Chapter 7. I had absolutely no idea that what I'd foreseen as a sort of "salad course" would eventually turn out to be the main dish.

The great combinatorial fermentation of 1975 has continued to churn, as more and more people have begun to participate. New ideas improve upon the older ones, but rarely replace them or make them obsolete. So of course I've had to abandon any hopes that I once had of being able to surround the field, to write a definitive book that sets everything in order and provides one-stop shopping for everyone who has combinatorial problems to solve. The array of applicable techniques has mushroomed to the point where I can almost never discuss a subtopic and say, "Here's the final solution: end of story." Instead, I must restrict myself to explaining the most important principles that seem to underlie all of the efficient combinatorial methods that I've encountered so far.

At present I've accumulated more than twice as much raw material for Volume 4 as for all of Volumes 1–3 combined.

This sheer mass of material implies that the once-planned "Volume 4" must actually become several physical volumes. You are now looking at Volume 4A. Volumes 4B and 4C will exist someday, assuming that I'm able to remain healthy; and (who knows?) there may also be Volumes 4D, 4E, ...; but surely not 4Z.

My plan is to go systematically through the files that I've amassed since 1962 and to tell the stories that I believe are still waiting to be told, to the best of my ability. I can't aspire to completeness, but I do want to give proper credit to all of the pioneers who have been responsible for key ideas; so I won't scrimp on historical details. Furthermore, whenever I learn something that I think is likely to remain important 50 years from now, something that can also be explained elegantly in a paragraph or two, I can't bear to leave it out. Conversely, difficult material that requires a lengthy proof is beyond the scope of these books, unless the subject matter is truly fundamental.

OK, it's clear that the field of Combinatorial Algorithms is vast, and I can't cover it all. What are the most important things that I'm leaving out? My biggest blind spot, I think, is geometry, because I've always been much better at visualizing and manipulating algebraic formulas than objects in space. Therefore I don't attempt to deal in these books with combinatorial problems that are related to computational geometry, such as close packing of spheres, or clustering of data points in n-dimensional Euclidean space, or even the Steiner tree problem in the plane. More significantly, I tend to shy away from polyhedral combinatorics, and from approaches that are based primarily on linear programming, integer programming, or semidefinite programming. Those topics are treated well in many other books on the subject, and they rely on geometrical intuition. Purely combinatorial developments are easier for me to understand.

I also must confess a bias against algorithms that are efficient only in an asymptotic sense, algorithms whose superior performance doesn't begin to "kick in" until the size of the problem exceeds the size of the universe. A great many publications nowadays are devoted to algorithms of that kind. I can understand why the contemplation of ultimate limits has intellectual appeal and carries an academic cachet; but in *The Art of Computer Programming* I tend to give short shrift to any methods that I would never consider using myself in an actual program. (There are, of course, exceptions to this rule, especially with respect to basic concepts in the core of the subject. Some impractical methods are simply too beautiful and/or too insightful to be excluded; others provide instructive examples of what *not* to do.)

Furthermore, as in earlier volumes of this series, I'm intentionally concentrating almost entirely on *sequential* algorithms, even though computers are increasingly able to carry out activities in parallel. I'm unable to judge what ideas about parallelism are likely to be useful five or ten years from now, let alone fifty, so I happily leave such questions to others who are wiser than I. Sequential methods, by themselves, already test the limits of my own ability to discern what the artful programmers of tomorrow will want to know.

The main decision that I needed to make when planning how to present this material was whether to organize it by problems or by techniques. Chapter 5 in Volume 3, for example, was devoted to a single problem, the sorting of data into order; more than two dozen techniques were applied to different aspects of that problem. Combinatorial algorithms, by contrast, involve many different problems, which tend to be attacked with a smaller repertoire of techniques. I finally decided that a mixed strategy would work better than any pure approach. Thus, for example, these books treat the problem of finding shortest paths in Section 7.3, and problems of connectivity in Section 7.4.1; but many other sections are devoted to basic techniques, such as the use of Boolean algebra (Section 7.1), backtracking (Section 7.2.2), matroid theory (Section 7.6), or dynamic programming (Section 7.7). The famous Traveling Salesrep Problem, and other classic combinatorial tasks related to covering, coloring, and packing, have no sections of their own, but they come up several times in different places as they are treated by different methods.

I've mentioned great progress in the art of combinatorial computing, but I don't mean to imply that all combinatorial problems have actually been tamed. When the running time of a computer program goes ballistic, its programmers shouldn't expect to find a silver bullet for their needs in this book. The methods described here will often work a great deal faster than the first approaches that a programmer tries; but let's face it: Combinatorial problems get huge very quickly. We can even prove rigorously that a certain small, natural problem will *never* have a feasible solution in the real world, although it is solvable in principle (see the theorem of Stockmeyer and Meyer in Section 7.1.2). In other cases we cannot prove as yet that no decent algorithm for a given problem exists, but we know that such methods are unlikely, because any efficient algorithm would yield a good way to solve thousands of other problems that have stumped the world's greatest experts (see the discussion of NP-completeness in Section 7.9).

Experience suggests that new combinatorial algorithms will continue to be invented, for new combinatorial problems and for newly identified variations or special cases of old ones; and that people's appetite for such algorithms will also continue to grow. The art of computer programming continually reaches new heights when programmers are faced with challenges such as these. Yet today's methods are also likely to remain relevant.

Most of this book is self-contained, although there are frequent tie-ins with the topics discussed in Volumes 1–3. Low-level details of machine language programming have been covered extensively in those volumes, so the algorithms in the present book are usually specified only at an abstract level, independent of any machine. However, some aspects of combinatorial programming are heavily dependent on low-level details that didn't arise before; in such cases, all examples in this book are based on the `MMIX` computer, which supersedes the `MIX` machine that was defined in early editions of Volume 1. Details about `MMIX` appear in a paperback supplement to that volume called *The Art of Computer Programming*, Volume 1, Fascicle 1, containing Sections 1.3.1′, 1.3.2′, etc.; they're also available on the Internet, together with downloadable assemblers and simulators.

Another downloadable resource, a collection of programs and data called *The Stanford GraphBase*, is cited extensively in the examples of this book. Readers are encouraged to play with it, in order to learn about combinatorial algorithms in what I think will be the most efficient and most enjoyable way.

Incidentally, while writing the introductory material at the beginning of Chapter 7, I was pleased to note that it was natural to mention some work of my Ph.D. thesis advisor, Marshall Hall, Jr. (1910–1990), as well as some work of *his* thesis advisor, Oystein Ore (1899–1968), as well as some work of *his* thesis advisor, Thoralf Skolem (1887–1963). Skolem's advisor, Axel Thue (1863–1922), was already present in Chapter 6.

I'm immensely grateful to the hundreds of readers who have helped me to ferret out numerous mistakes that I made in the early drafts of this volume, which were originally posted on the Internet and subsequently printed in paperback fascicles. In particular, the extensive comments of Thorsten Dahlheimer, Marc van Leeuwen, and Udo Wermuth have been especially influential. But I fear that other errors still lurk among the details collected here, and I want to correct them as soon as possible. Therefore I will cheerfully award $2.56 to the first finder of each technical, typographical, or historical error. The `taocp` webpage cited on page iv contains a current listing of all corrections that have been reported to me.

Stanford, California D. E. K.
October 2010

In my preface to the first edition,
I begged the reader not to draw attention to errors.
I now wish I had not done so
and am grateful to the few readers who ignored my request.
— STUART SUTHERLAND, *The International Dictionary of Psychology* (1996)

Naturally, I am responsible for the remaining errors—
although, in my opinion, my friends could have caught a few more.
— CHRISTOS H. PAPADIMITRIOU, *Computational Complexity* (1994)

I like to work in a variety of fields
in order to spread my mistakes more thinly.
— VICTOR KLEE (1999)

A note on references. Several oft-cited journals and conference proceedings have special code names, which appear in the Index and Glossary at the close of this book. But the various kinds of *IEEE Transactions* are cited by including a letter code for the type of transactions, in boldface preceding the volume number. For example, '*IEEE Trans.* **C-35**' means the *IEEE Transactions on Computers*, volume 35. The IEEE no longer uses these convenient letter codes, but the codes aren't too hard to decipher: '**EC**' once stood for "Electronic Computers," '**IT**' for "Information Theory," '**SE**' for "Software Engineering," and '**SP**' for "Signal Processing," etc.; '**CAD**' meant "Computer-Aided Design of Integrated Circuits and Systems."

A cross-reference such as 'exercise 7.10–00' points to a future exercise in Section 7.10 whose number is not yet known.

A note on notations. Simple and intuitive conventions for the algebraic representation of mathematical concepts have always been a boon to progress, especially when most of the world's researchers share a common symbolic language. The current state of affairs in combinatorial mathematics is unfortunately a bit of a mess in this regard, because the same symbols are occasionally used with completely different meanings by different groups of people; some specialists who work in comparatively narrow subfields have unintentionally spawned conflicting symbolisms. Computer science — which interacts with large swaths of mathematics — needs to steer clear of this danger by adopting internally consistent notations whenever possible. Therefore I've often had to choose among a number of competing schemes, knowing that it will be impossible to please everyone. I have tried my best to come up with notations that I believe will be best for the future, often after many years of experimentation and discussion with colleagues, often flip-flopping between alternatives until finding something that works well. Usually it has been possible to find convenient conventions that other people have not already coopted in contradictory ways.

Appendix B is a comprehensive index to all of the principal notations that are used in the present book, inevitably including several that are not (yet?) standard. If you run across a formula that looks weird and/or incomprehensible, chances are fairly good that Appendix B will direct you to a page where my intentions are clarified. But I might as well list here a few instances that you might wish to watch for when you read this book for the first time:

- Hexadecimal constants are preceded by a number sign or hash mark. For example, `#123` means $(123)_{16}$.
- The "monus" operation $x \dot{-} y$, sometimes called dot-minus or saturating subtraction, yields $\max(0, x - y)$.
- The median of three numbers $\{x, y, z\}$ is denoted by $\langle xyz \rangle$.
- A set such as $\{x\}$, which consists of a single element, is often denoted simply by x in contexts such as $X \cup x$ or $X \setminus x$.
- If n is a nonnegative integer, the number of 1-bits in n's binary representation is νn. Furthermore, if $n > 0$, the leftmost and rightmost 1-bits of n are respectively $2^{\lambda n}$ and $2^{\rho n}$. For example, $\nu 10 = 2$, $\lambda 10 = 3$, $\rho 10 = 1$.
- The Cartesian product of graphs G and H is denoted by $G \mathbin{\square} H$. For example, $C_m \mathbin{\square} C_n$ denotes an $m \times n$ torus, because C_n denotes a cycle of n vertices.

NOTES ON THE EXERCISES

THE EXERCISES in this set of books have been designed for self-study as well as for classroom study. It is difficult, if not impossible, for anyone to learn a subject purely by reading about it, without applying the information to specific problems and thereby being encouraged to think about what has been read. Furthermore, we all learn best the things that we have discovered for ourselves. Therefore the exercises form a major part of this work; a definite attempt has been made to keep them as informative as possible and to select problems that are enjoyable as well as instructive.

In many books, easy exercises are found mixed randomly among extremely difficult ones. A motley mixture is, however, often unfortunate because readers like to know in advance how long a problem ought to take — otherwise they may just skip over all the problems. A classic example of such a situation is the book *Dynamic Programming* by Richard Bellman; this is an important, pioneering work in which a group of problems is collected together at the end of some chapters under the heading "Exercises and Research Problems," with extremely trivial questions appearing in the midst of deep, unsolved problems. It is rumored that someone once asked Dr. Bellman how to tell the exercises apart from the research problems, and he replied, "If you can solve it, it is an exercise; otherwise it's a research problem."

Good arguments can be made for including both research problems and very easy exercises in a book of this kind; therefore, to save the reader from the possible dilemma of determining which are which, *rating numbers* have been provided to indicate the level of difficulty. These numbers have the following general significance:

Rating *Interpretation*

00 An extremely easy exercise that can be answered immediately if the material of the text has been understood; such an exercise can almost always be worked "in your head," unless you're multitasking.

10 A simple problem that makes you think over the material just read, but is by no means difficult. You should be able to do this in one minute at most; pencil and paper may be useful in obtaining the solution.

20 An average problem that tests basic understanding of the text material, but you may need about fifteen or twenty minutes to answer it completely. Maybe even twenty-five.

30 A problem of moderate difficulty and/or complexity; this one may involve more than two hours' work to solve satisfactorily, or even more if the TV is on.

40 Quite a difficult or lengthy problem that would be suitable for a term project in classroom situations. A student should be able to solve the problem in a reasonable amount of time, but the solution is not trivial.

50 A research problem that has not yet been solved satisfactorily, as far as the author knew at the time of writing, although many people have tried. If you have found an answer to such a problem, you ought to write it up for publication; furthermore, the author of this book would appreciate hearing about the solution as soon as possible (provided that it is correct).

By interpolation in this "logarithmic" scale, the significance of other rating numbers becomes clear. For example, a rating of *17* would indicate an exercise that is a bit simpler than average. Problems with a rating of *50* that are subsequently solved by some reader may appear with a *40* rating in later editions of the book, and in the errata posted on the Internet (see page iv).

The remainder of the rating number divided by 5 indicates the amount of detailed work required. Thus, an exercise rated *24* may take longer to solve than an exercise that is rated *25*, but the latter will require more creativity. All exercises with ratings of *46* or more are open problems for future research, rated according to the number of different attacks that they've resisted so far.

The author has tried earnestly to assign accurate rating numbers, but it is difficult for the person who makes up a problem to know just how formidable it will be for someone else to find a solution; and everyone has more aptitude for certain types of problems than for others. It is hoped that the rating numbers represent a good guess at the level of difficulty, but they should be taken as general guidelines, not as absolute indicators.

This book has been written for readers with varying degrees of mathematical training and sophistication; as a result, some of the exercises are intended only for the use of more mathematically inclined readers. The rating is preceded by an M if the exercise involves mathematical concepts or motivation to a greater extent than necessary for someone who is primarily interested only in programming the algorithms themselves. An exercise is marked with the letters "*HM*" if its solution necessarily involves a knowledge of calculus or other higher mathematics not developed in this book. An "*HM*" designation does *not* necessarily imply difficulty.

Some exercises are preceded by an arrowhead, "▸"; this designates problems that are especially instructive and especially recommended. Of course, no reader/student is expected to work *all* of the exercises, so those that seem to be the most valuable have been singled out. (This distinction is not meant to detract from the other exercises!) Each reader should at least make an attempt to solve all of the problems whose rating is *10* or less; and the arrows may help to indicate which of the problems with a higher rating should be given priority.

Several sections have more than 100 exercises. How can you find your way among so many? In general the sequence of exercises tends to follow the sequence of ideas in the main text. Adjacent exercises build on each other, as in the pioneering problem books of Pólya and Szegő. The final exercises of a section often involve the section as a whole, or introduce supplementary topics.

Solutions to most of the exercises appear in the answer section. Please use them wisely; do not turn to the answer until you have made a genuine effort to solve the problem by yourself, or unless you absolutely do not have time to work this particular problem. *After* getting your own solution or giving the problem a decent try, you may find the answer instructive and helpful. The solution given will often be quite short, and it will sketch the details under the assumption that you have earnestly tried to solve it by your own means first. Sometimes the solution gives less information than was asked; often it gives more. It is quite possible that you may have a better answer than the one published here, or you may have found an error in the published solution; in such a case, the author will be pleased to know the details. Later printings of this book will give the improved solutions together with the solver's name where appropriate.

When working an exercise you may generally use the answers to previous exercises, unless specifically forbidden from doing so. The rating numbers have been assigned with this in mind; thus it is possible for exercise $n + 1$ to have a lower rating than exercise n, even though it includes the result of exercise n as a special case.

Summary of codes:			
		00	Immediate
		10	Simple (one minute)
		20	Medium (quarter hour)
▶	Recommended	*30*	Moderately hard
M	Mathematically oriented	*40*	Term project
HM	Requiring "higher math"	*50*	Research problem

EXERCISES

▶ **1.** [*00*] What does the rating "*M15*" mean?

2. [*10*] Of what value can the exercises in a textbook be to the reader?

3. [*HM45*] Prove that every simply connected, closed 3-dimensional manifold is topologically equivalent to a 3-dimensional sphere.

Art derives a considerable part of its beneficial exercise
from flying in the face of presumptions.

— HENRY JAMES, "The Art of Fiction" (1884)

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

— PETER J. GOMES, *The Good Book* (1996)

I at last deliver to the world a Work which I have long promised,
and of which, I am afraid, too high expectations have been raised.
The delay of its publication must be imputed, in a considerable degree,
to the extraordinary zeal which has been shown by distinguished persons
in all quarters to supply me with additional information.

— JAMES BOSWELL, *The Life of Samuel Johnson, LL.D.* (1791)

The author is especially grateful to the Addison–Wesley Publishing Company
for its patience in waiting a full decade for this manuscript
from the date the contract was signed.

— FRANK HARARY, *Graph Theory* (1969)

The average boy who abhors square root or algebra
finds delight in working puzzles which involve similar principles,
and may be led into a course of study
which would develop the mathematical and inventive bumps
in a way to astonish the family phrenologist.

— SAM LOYD, *The World of Puzzledom* (1896)

Bitte ein Bit!

— Slogan of Bitburger Brauerei (1951)

CONTENTS

Hommage à Bach.

Missing pages from select printings of Knuth, The Art of Computer Programming, Volume 4A
(ISBN-13: 9780201038040 / ISBN-10: 0201038048).

CHAPTER SEVEN

COMBINATORIAL SEARCHING

You shall seeke all day ere you finde them,
& when you have them, they are not worth the search.

— BASSANIO, in *The Merchant of Venice* (Act I, Scene 1, Line 117)

Amid the action and reaction of so dense a swarm of humanity,
every possible combination of events may be expected to take place,
and many a little problem will be presented which may be striking and bizarre.

— SHERLOCK HOLMES, in *The Adventure of the Blue Carbuncle* (1892)

The field of combinatorial algorithms is too vast to cover
in a single paper or even in a single book.

— ROBERT E. TARJAN (1976)

While jostling against all manner of people
it has been impressed upon my mind that the successful ones
are those who have a natural faculty for solving puzzles.
Life is full of puzzles, and we are called upon
to solve such as fate throws our way.

— SAM LOYD, JR. (1926)

COMBINATORICS is the study of the ways in which discrete objects can be arranged into various kinds of patterns. For example, the objects might be $2n$ numbers $\{1, 1, 2, 2, \ldots, n, n\}$, and we might want to place them in a row so that exactly k numbers occur between the two appearances of each digit k. When $n = 3$ there is essentially only one way to arrange such "Langford pairs," namely 231213 (and its left-right reversal); similarly, there's also a unique solution when $n = 4$. Many other types of combinatorial patterns are discussed below.

Five basic types of questions typically arise when combinatorial problems are studied, some more difficult than others.

i) Existence: Are there any arrangements X that conform to the pattern?
ii) Construction: If so, can such an X be found quickly?
iii) Enumeration: How many different arrangements X exist?
iv) Generation: Can all arrangements X_1, X_2, ... be visited systematically?
v) Optimization: What arrangements maximize or minimize $f(X)$, given an objective function f?

Each of these questions turns out to be interesting with respect to Langford pairs.

Missing pages from select printings of Knuth, The Art of Computer Programming, Volume 4A
(ISBN-13: 9780201038040 / ISBN-10: 0201038048).

For example, consider the question of existence. Trial and error quickly reveals that, when $n = 5$, we cannot place $\{1, 1, 2, 2, \ldots, 5, 5\}$ properly into ten positions. The two 1s must both go into even-numbered slots, or both into odd-numbered slots; similarly, the 3s and 5s must choose between two evens or two odds; but the 2s and 4s use one of each. Thus we can't fill exactly five slots of each parity. This reasoning also proves that the problem has no solution when $n = 6$, or in general whenever the number of odd values in $\{1, 2, \ldots, n\}$ is odd.

In other words, Langford pairings can exist only when $n = 4m-1$ or $n = 4m$, for some integer m. Conversely, when n does have this form, Roy O. Davies has found an elegant way to construct a suitable placement (see exercise 1).

How many essentially different pairings, L_n, exist? Lots, when n grows:

$$\begin{aligned} L_3 &= 1; & L_4 &= 1; \\ L_7 &= 26; & L_8 &= 150; \\ L_{11} &= 17{,}792; & L_{12} &= 108{,}144; \\ L_{15} &= 39{,}809{,}640; & L_{16} &= 326{,}721{,}800; \\ L_{19} &= 256{,}814{,}891{,}280; & L_{20} &= 2{,}636{,}337{,}861{,}200; \\ L_{23} &= 3{,}799{,}455{,}942{,}515{,}488; & L_{24} &= 46{,}845{,}158{,}056{,}515{,}936. \end{aligned} \tag{1}$$

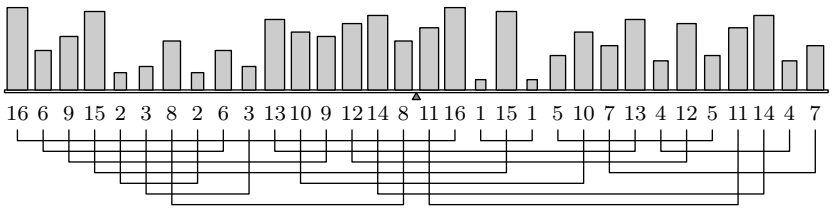
[The values of L_{23} and L_{24} were determined by M. Krajecki, C. Jaillet, and A. Bui in 2004 and 2005; see *Studia Informatica Universalis* **4** (2005), 151–190.] A seat-of-the-pants calculation suggests that L_n might be roughly of order $(4n/e^3)^{n+1/2}$ when it is nonzero (see exercise 5); and in fact this prediction turns out to be basically correct in all known cases. But no simple formula is apparent.

The problem of Langford arrangements is a simple special case of a general class of combinatorial challenges called *exact cover problems*. In Section 7.2.2.1 we shall study an algorithm called "dancing links," which is a convenient way to generate all solutions to such problems. When $n = 16$, for example, that method needs to perform only about 3200 memory accesses for each Langford pair arrangement that it finds. Thus the value of L_{16} can be computed in a reasonable amount of time by simply generating all of the pairings and counting them.

Notice, however, that L_{24} is a *huge* number — roughly 5×10^{16}, or about 1500 MIP-years. (Recall that a "MIP-year" is the number of instructions executed per year by a machine that carries out a million instructions per second, namely 31,556,952,000,000.) Therefore it's clear that the exact value of L_{24} was determined by some technique that did *not* involve generating all of the arrangements. Indeed, there is a much, much faster way to compute L_n, using polynomial algebra. The instructive method described in exercise 6 needs $O(4^n n)$ operations, which may seem inefficient; but it beats the generate-and-count method by a whopping factor of order $((n/e^3)^{n-1/2})$, and even when $n = 16$ it runs about 20 times faster. On the other hand, the exact value of L_{100} will probably never be known, even as computers become faster and faster.

We can also consider Langford pairings that are *optimum* in various ways. For example, it's possible to arrange sixteen pairs of weights $\{1, 1, 2, 2, \ldots, 16, 16\}$ that satisfy Langford's condition and have the additional property of being "well-

balanced," in the sense that they won't tip a balance beam when they are placed in the appropriate order:



(2)

In other words, $15.5\cdot16+14.5\cdot6+\cdots+0.5\cdot8 = 0.5\cdot11+\cdots+14.5\cdot4+15.5\cdot7$; and in this particular example we also have another kind of balance, $16+6+\cdots+8 = 11+16+\cdots+7$, hence also $16\cdot16+15\cdot6+\cdots+1\cdot8 = 1\cdot11+\cdots+15\cdot4+16\cdot7$.

Moreover, the arrangement in (2) has *minimum width* among all Langford pairings of order 16: The connecting lines at the bottom of the diagram show that no more than seven pairs are incomplete at any point, as we read from left to right; and one can show that a width of six is impossible. (See exercise 7.)

What arrangements $a_1a_2\ldots a_{32}$ of $\{1,1,\ldots,16,16\}$ are the *least* balanced, in the sense that $\sum_{k=1}^{32} ka_k$ is maximized? The maximum possible value turns out to be 5268. One such pairing — there are 12,016 of them — is

$$2\ 3\ 4\ 2\ 1\ 3\ 1\ 4\ 16\ 13\ 15\ 5\ 14\ 7\ 9\ 6\ 11\ 5\ 12\ 10\ 8\ 7\ 6\ 13\ 9\ 16\ 15\ 14\ 11\ 8\ 10\ 12. \quad (3)$$

A more interesting question is to ask for the Langford pairings that are smallest and largest in lexicographic order. The answers for $n = 24$ are

$$\{\texttt{abacbdecfgdoersfpgqtuwxvjklonhmirpsjqkhltiunmwvx},\ \texttt{xvwsquntkigrdapaodgiknqsvxwutmrpohljcfbecbhmfejl}\} \quad (4)$$

if we use the letters `a`, `b`, ..., `w`, `x` instead of the numbers 1, 2, ..., 23, 24.

We shall discuss many techniques for combinatorial optimization in later sections of this chapter. Our goal, of course, will be to solve such problems without examining more than a tiny portion of the space of all possible arrangements.

Orthogonal latin squares. Let's look back for a moment at the early days of combinatorics. A posthumous edition of Jacques Ozanam's *Recreations mathematiques et physiques* (Paris: 1725) included an amusing puzzle in volume 4, page 434: "Take all the aces, kings, queens, and jacks from an ordinary deck of playing cards and arrange them in a square so that each row and each column contains all four values and all four suits." Can you do it? Ozanam's solution, shown in Fig. 1 on the next page, does even more: It exhibits the full panoply of values and of suits also on both main diagonals. (Please don't turn the page until you've given this problem a try.)

By 1779 a similar puzzle was making the rounds of St. Petersburg, and it came to the attention of the great mathematician Leonhard Euler. "Thirty-six officers of six different ranks, taken from six different regiments, want to march in a 6×6 formation so that each row and each column will contain one officer of each rank and one of each regiment. How can they do it?" Nobody was able to

Fig. 1. Disorder in the court cards: No agreement in any line of four. (This configuration is one of many ways to solve a popular eighteenth-century problem.)

find a satisfactory marching order. So Euler decided to resolve the riddle — even though he had become nearly blind in 1771 and was dictating all of his work to assistants. He wrote a major paper on the subject [eventually published in *Verhandelingen uitgegeven door het Zeeuwsch Genootschap der Wetenschappen te Vlissingen* **9** (1782), 85–239], in which he constructed suitable arrangements for the analogous task with n ranks and n regiments when $n = 1$, 3, 4, 5, 7, 8, 9, 11, 12, 13, 15, 16, ...; only the cases with $n \bmod 4 = 2$ eluded him.

There's obviously no solution when $n = 2$. But Euler was stumped when $n = 6$, after having examined a "very considerable number" of square arrangements that didn't work. He showed that any actual solution would lead to many others that look different, and he couldn't believe that all such solutions had escaped his attention. Therefore he said, "I do not hesitate to conclude that one cannot produce a complete square of 36 cells, and that the same impossibility extends to the cases $n = 10$, $n = 14$ ... in general to all oddly even numbers."

Euler named the 36 officers $a\alpha$, $a\beta$, $a\gamma$, $a\delta$, $a\epsilon$, $a\zeta$, $b\alpha$, $b\beta$, $b\gamma$, $b\delta$, $b\epsilon$, $b\zeta$, $c\alpha$, $c\beta$, $c\gamma$, $c\delta$, $c\epsilon$, $c\zeta$, $d\alpha$, $d\beta$, $d\gamma$, $d\delta$, $d\epsilon$, $d\zeta$, $e\alpha$, $e\beta$, $e\gamma$, $e\delta$, $e\epsilon$, $e\zeta$, $f\alpha$, $f\beta$, $f\gamma$, $f\delta$, $f\epsilon$, $f\zeta$, based on their regiments and ranks. He observed that any solution would amount to having two *separate* squares, one for Latin letters and another for Greek. Each of those squares is supposed to have distinct entries in rows and columns; so he began by studying the possible configurations for $\{a, b, c, d, e, f\}$, which he called *Latin squares*. A Latin square can be paired up with a Greek square to form a "Græco-Latin square" only if the squares are *orthogonal* to each other, meaning that no (Latin, Greek) pair of letters can be found together in more than one place when the squares are superimposed. For example, if we let $a = \texttt{A}$, $b = \texttt{K}$, $c = \texttt{Q}$, $d = \texttt{J}$, $\alpha = \clubsuit$, $\beta = \spadesuit$, $\gamma = \diamondsuit$, and $\delta = \heartsuit$, Fig. 1 is equivalent

to the Latin, Greek, and Græco-Latin squares

$$\begin{pmatrix} d & a & b & c \\ c & b & a & d \\ a & d & c & b \\ b & c & d & a \end{pmatrix}, \quad \begin{pmatrix} \gamma & \delta & \beta & \alpha \\ \beta & \alpha & \gamma & \delta \\ \alpha & \beta & \delta & \gamma \\ \delta & \gamma & \alpha & \beta \end{pmatrix}, \quad \text{and} \quad \begin{pmatrix} d\gamma & a\delta & b\beta & c\alpha \\ c\beta & b\alpha & a\gamma & d\delta \\ a\alpha & d\beta & c\delta & b\gamma \\ b\delta & c\gamma & d\alpha & a\beta \end{pmatrix}. \tag{5}$$

Of course we can use *any* n distinct symbols in an $n \times n$ Latin square; all that matters is that no symbol occurs twice in any row or twice in any column. So we might as well use numeric values $\{0, 1, \ldots, n-1\}$ for the entries. Furthermore we'll just refer to "latin squares" (with a lowercase "l"), instead of categorizing a square as either Latin or Greek, because orthogonality is a symmetric relation.

Euler's assertion that two 6×6 latin squares cannot be orthogonal was verified by Thomas Clausen, who reduced the problem to an examination of 17 fundamentally different cases, according to a letter from H. C. Schumacher to C. F. Gauss dated 10 August 1842. But Clausen did not publish his analysis. The first demonstration to appear in print was by G. Tarry [*Comptes rendus, Association française pour l'avancement des sciences* **29**, part 2 (1901), 170–203], who discovered in his own way that 6×6 latin squares can be classified into 17 different families. (In Section 7.2.3 we shall study how to decompose a problem into combinatorially inequivalent classes of arrangements.)

Euler's conjecture about the remaining cases $n = 10$, $n = 14$, ... was "proved" three times, by J. Petersen [*Annuaire des mathématiciens* (Paris: 1902), 413–427], by P. Wernicke [*Jahresbericht der Deutschen Math.-Vereinigung* **19** (1910), 264–267], and by H. F. MacNeish [*Annals of Math.* (2) **23** (1922), 221–227]. Flaws in all three arguments became known, however; and the question was still unsettled when computers became available many years later. One of the very first combinatorial problems to be tackled by machine was therefore the enigma of 10×10 Græco-Latin squares: Do they exist or not?

In 1957, L. J. Paige and C. B. Tompkins programmed the SWAC computer to search for a counterexample to Euler's prediction. They selected one particular 10×10 latin square "almost at random," and their program tried to find another square that would be orthogonal to it. But the results were discouraging, and they decided to shut the machine off after five hours. Already the program had generated enough data for them to predict that at least 4.8×10^{11} hours of computer time would be needed to finish the run!

Shortly afterwards, three mathematicians made a breakthrough that put latin squares onto page one of major world newspapers: R. C. Bose, S. S. Shrikhande, and E. T. Parker found a remarkable series of constructions that yield orthogonal $n \times n$ squares for all $n > 6$ [*Proc. Nat. Acad. Sci.* **45** (1959), 734–737, 859–862; *Canadian J. Math.* **12** (1960), 189–203]. Thus, after resisting attacks for 180 years, Euler's conjecture turned out to be almost entirely wrong.

Their discovery was made without computer help. But Parker worked for UNIVAC, and he soon brought programming skills into the picture by solving the problem of Paige and Tompkins in less than an hour, on a UNIVAC 1206 Military Computer. [See *Proc. Symp. Applied Math.* **10** (1960), 71–83; **15** (1963), 73–81.]

Let's take a closer look at what the earlier programmers did, and how Parker dramatically trumped their approach. Paige and Tompkins began with the following 10×10 square L and its unknown orthogonal mate(s) M:

$$L = \begin{pmatrix} 0&1&2&3&4&5&6&7&8&9\\ 1&8&3&2&5&4&7&6&9&0\\ 2&9&5&6&3&0&8&4&7&1\\ 3&7&0&9&8&6&1&5&2&4\\ 4&6&7&5&2&9&0&8&1&3\\ 5&0&9&4&7&8&3&1&6&2\\ 6&5&4&7&1&3&2&9&0&8\\ 7&4&1&8&0&2&9&3&5&6\\ 8&3&6&0&9&1&5&2&4&7\\ 9&2&8&1&6&7&4&0&3&5 \end{pmatrix} \quad \text{and} \quad M = \begin{pmatrix} 0&␣&␣&␣&␣&␣&␣&␣&␣&␣\\ 1&␣&␣&␣&␣&␣&␣&␣&␣&␣\\ 2&␣&␣&␣&␣&␣&␣&␣&␣&␣\\ 3&␣&␣&␣&␣&␣&␣&␣&␣&␣\\ 4&␣&␣&␣&␣&␣&␣&␣&␣&␣\\ 5&␣&␣&␣&␣&␣&␣&␣&␣&␣\\ 6&␣&␣&␣&␣&␣&␣&␣&␣&␣\\ 7&␣&␣&␣&␣&␣&␣&␣&␣&␣\\ 8&␣&␣&␣&␣&␣&␣&␣&␣&␣\\ 9&␣&␣&␣&␣&␣&␣&␣&␣&␣ \end{pmatrix}. \quad (6)$$

We can assume without loss of generality that the rows of M begin with 0, 1, ..., 9, as shown. The problem is to fill in the remaining 90 blank entries, and the original SWAC program proceeded from top to bottom, left to right. The top left ␣ can't be filled with 0, since 0 has already occurred in the top row of M. And it can't be 1 either, because the pair $(1, 1)$ already occurs at the left of the next row in (L, M). We can, however, tentatively insert a 2. The digit 1 can be placed next; and pretty soon we find the lexicographically smallest top row that might work for M, namely 0214365897. Similarly, the smallest rows that fit below 0214365897 are 1023456789 and 2108537946; and the smallest legitimate row below them is 3540619278. Now, unfortunately, the going gets tougher: There's no way to complete another row without coming into conflict with a previous choice. So we change 3540619278 to 3540629178 (but that doesn't work either), then to 3540698172, and so on for several more steps, until finally 3546109278 can be followed by 4397028651 before we get stuck again.

In Section 7.2.2 we'll study ways to estimate the behavior of such searches, without actually performing them. Such estimates tell us in this case that the Paige–Tompkins method essentially traverses an implicit search tree that contains about 2.5×10^{18} nodes. Most of those nodes belong to only a few levels of the tree; more than half of them deal with choices on the right half of the sixth row of M, after about 50 of the 90 blanks have been tentatively filled in. A typical node of the search tree probably requires about 75 mems (memory accesses) for processing, to check validity. Therefore the total running time on a modern computer would be roughly the time needed to perform 2×10^{20} mems.

Parker, on the other hand, went back to the method that Euler had originally used to search for orthogonal mates in 1779. First he found all of the so-called *transversals* of L, namely all ways to choose some of its elements so that there's exactly one element in each row, one in each column, and one of each value. For example, one transversal is 0859734216, in Euler's notation, meaning that we choose the 0 in column 0, the 8 in column 1, ..., the 6 in column 9. Each transversal that includes the k in L's leftmost column represents a legitimate way to place the ten k's into square M. The task of finding transversals is, in fact, rather easy, and the given matrix L turns out to have exactly 808 of them; there are respectively $(79, 96, 76, 87, 70, 84, 83, 75, 95, 63)$ transversals for $k = (0, 1, \ldots, 9)$.

Once the transversals are known, we're left with an exact cover problem of 10 stages, which is much simpler than the original 90-stage problem in (6). All we need to do is cover the square with ten transversals that don't intersect — because every such set of ten is equivalent to a latin square M that is orthogonal to L.

The particular square L in (6) has, in fact, exactly one orthogonal mate:

$$\begin{pmatrix} 0&1&2&3&4&5&6&7&8&9\\ 1&8&3&2&5&4&7&6&9&0\\ 2&9&5&6&3&0&8&4&7&1\\ 3&7&0&9&8&6&1&5&2&4\\ 4&6&7&5&2&9&0&8&1&3\\ 5&0&9&4&7&8&3&1&6&2\\ 6&5&4&7&1&3&2&9&0&8\\ 7&4&1&8&0&2&9&3&5&6\\ 8&3&6&0&9&1&5&2&4&7\\ 9&2&8&1&6&7&4&0&3&5 \end{pmatrix} \perp \begin{pmatrix} 0&2&8&5&9&4&7&3&6&1\\ 1&7&4&9&3&6&5&0&2&8\\ 2&5&6&4&8&7&0&1&9&3\\ 3&6&9&0&4&5&8&2&1&7\\ 4&8&1&7&5&3&6&9&0&2\\ 5&1&7&8&0&2&9&4&3&6\\ 6&9&0&2&7&1&3&8&4&5\\ 7&3&5&1&2&0&4&6&8&9\\ 8&0&2&3&6&9&1&7&5&4\\ 9&4&3&6&1&8&2&5&7&0 \end{pmatrix}. \tag{7}$$

The dancing links algorithm finds it, and proves its uniqueness, after doing only about 1.7×10^8 mems of computation, given the 808 transversals. Furthermore, the cost of the transversal-finding phase, about 5 million mems, is negligible by comparison. Thus the original running time of 2×10^{20} mems — which once was regarded as the inevitable cost of solving a problem for which there are 10^{90} ways to fill in the blanks — has been reduced by a further factor of more than 10^{12}(!).

We will see later that advances have also been made in methods for solving 90-level problems like (6). Indeed, (6) turns out to be representable directly as an exact cover problem (see exercise 17), which the dancing links procedure of Section 7.2.2.1 solves after expending only 1.3×10^{11} mems. Even so, the Euler–Parker approach remains about a thousand times better than the Paige–Tompkins approach. By "factoring" the problem into two separate phases, one for transversal-finding and one for transversal-combining, Euler and Parker essentially reduced the computational cost from a product, T_1T_2, to a sum, T_1+T_2.

The moral of this story is clear: Combinatorial problems might confront us with a huge universe of possibilities, yet we shouldn't give up too easily. A single good idea can reduce the amount of computation by many orders of magnitude.

Puzzles versus the real world. Many of the combinatorial problems we shall study in this chapter, like Langford's problem of pairs or Ozanam's problem of the sixteen honor cards, originated as amusing puzzles or "brain twisters." Some readers might be put off by this emphasis on recreational topics, which they regard as a frivolous waste of time. Shouldn't computers really be doing useful work? And shouldn't textbooks about computers be primarily concerned with significant applications to industry and/or world progress?

Well, the author of the textbook you are reading has absolutely no objections to useful work and human progress. But he believes strongly that a book such as this should stress *methods* of problem solving, together with mathematical ideas and *models* that help to solve many different problems, rather than focusing on the reasons why those methods and models might be useful. We shall learn many beautiful and powerful ways to attack combinatorial problems, and the elegance

of those methods will be our main motivation for studying them. Combinatorial challenges pop up everywhere, and new ways to apply the techniques discussed in this chapter arise every day. So let's not limit our horizons by attempting to catalog in advance what the ideas are good for.

For example, it turns out that orthogonal latin squares are enormously useful, particularly in the design of experiments. Already in 1788, François Cretté de Palluel used a 4×4 latin square to study what happens when sixteen sheep — four each from four different breeds — were fed four different diets and harvested at four different times. [*Mémoires d'Agriculture* (Paris: Société Royale d'Agriculture, trimestre d'été, 1788), 17–23.] The latin square allowed him to do this with 16 sheep instead of 64; with a Græco-Latin square he could also have varied another parameter by trying, say, four different quantities of food or four different grazing paradigms.

But if we had focused our discussion on his approach to animal husbandry, we might well have gotten bogged down in details about breeding, about root vegetables versus grains and the costs of growing them, etc. Readers who aren't farmers might therefore have decided to skip the whole topic, even though latin square designs apply to a wide range of studies. (Think about testing five kinds of pills, on patients in five stages of some disease, five age brackets, and five weight groups.) Moreover, a concentration on experimental design could lead readers to miss the fact that latin squares also have important applications to discrete geometry and error-correcting codes (see exercises 18–24).

Even the topic of Langford pairing, which seems at first to be purely recreational, turns out to have practical importance. T. Skolem used Langford sequences to construct Steiner triple systems, which we have applied to database queries in Section 6.5 [see *Math. Scandinavica* **6** (1958), 273–280]; and in the 1960s, E. J. Groth of Motorola Corporation applied Langford pairs to the design of circuits for multiplication. Furthermore, the algorithms that efficiently find Langford pairs and latin square transversals, such as the method of dancing links, apply to exact cover problems in general; and the problem of exact covering has great relevance to crucial problems such as the equitable apportionment of voter precincts to electoral districts, etc.

The applications are not the most important thing, and neither are the puzzles. Our primary goal is rather to get basic concepts into our brains, like the notions of latin squares and exact covering. Such notions give us the building blocks, vocabulary, and insights that *tomorrow's* problems will need.

Still, it's foolish to discuss problem solving without actually solving any problems. We need good problems to stimulate our creative juices, to light up our grey cells in a more or less organized fashion, and to make the basic concepts familiar. Mind-bending puzzles are often ideal for this purpose, because they can be presented in a few words, needing no complicated background knowledge.

Václav Havel once remarked that the complexities of life are vast: "There is too much to know... We have to abandon the arrogant belief that the world is merely a puzzle to be solved, a machine with instructions for use waiting to be discovered, a body of information to be fed into a computer." He called

for an increased sense of justice and responsibility; for taste, courage, and compassion. His words were filled with great wisdom. Yet thank goodness we do also have puzzles that *can* be solved! Puzzles deserve to be counted among the great pleasures of life, to be enjoyed in moderation like all other treats.

Of course, Langford and Ozanam directed their puzzles to human beings, not to computers. Aren't we missing the point if we merely shuffle such questions off to machines, to be solved by brute force instead of by rational thought? George Brewster, writing to Martin Gardner in 1963, expressed a widely held view as follows: "Feeding a recreational puzzle into a computer is no more than a step above dynamiting a trout stream. Succumbing to instant recreation."

Yes, but that view misses another important point: Simple puzzles often have generalizations that go beyond human ability and arouse our curiosity. The study of those generalizations often suggests instructive methods that apply to numerous other problems and have surprising consequences. Indeed, many of the key techniques that we shall study were born when people were trying to solve various puzzles. While writing this chapter, the author couldn't help relishing the fact that puzzles are now more fun than ever, as computers get faster and faster, because we keep getting more powerful dynamite to play with. [Further comments appear in the author's essay, "Are toy problems useful?", originally written in 1976; see *Selected Papers on Computer Science* (1996), 169–183.]

Puzzles do have the danger that they can be *too* elegant. Good puzzles tend to be mathematically clean and well-structured, but we also need to learn how to deal systematically with the messy, chaotic, organic stuff that surrounds us every day. Indeed, some computational techniques are important chiefly because they provide powerful ways to cope with such complexities. That is why, for example, the arcane rules of library-card alphabetization were presented at the beginning of Chapter 5, and an actual elevator system was discussed at length to illustrate simulation techniques in Section 2.2.5.

A collection of programs and data called the Stanford GraphBase (SGB) has been prepared so that experiments with combinatorial algorithms can readily be performed on a variety of real-world examples. SGB includes, for example, data about American highways, and an input-output model of the U.S. economy; it records the casts of characters in Homer's *Iliad*, Tolstoy's *Anna Karenina*, and several other novels; it encapsulates the structure of Roget's *Thesaurus* of 1879; it documents hundreds of college football scores; it specifies the gray-value pixels of Leonardo da Vinci's *Gioconda* (Mona Lisa). And perhaps most importantly, SGB contains a collection of five-letter words, which we shall discuss next.

The five-letter words of English. Many of the examples in this chapter will be based on the following list of five-letter words:

`aargh`, `abaca`, `abaci`, `aback`, `abaft`, `abase`, `abash`, ..., `zooms`, `zowie`. (8)

(There are 5757 words altogether — too many to display here; but those that are missing can readily be imagined.) It's a personal list, collected by the author between 1972 and 1992, beginning when he realized that such words would make `ideal` data for testing many kinds of combinatorial algorithms.

The list has intentionally been restricted to words that are `truly` part of the English language, in the sense that the author has encountered them in actual use. Unabridged dictionaries contain thousands of entries that are much more esoteric, like `aalii`, `abamp`, ..., `zymin`, and `zyxst`; words like that are useful primarily to SCRABBLE® players. But unfamiliar words tend to `spoil` the fun for anybody who doesn't know them. Therefore, for twenty years, the author systematically took note of all `words` that seemed `right` for the expository `goals` of *The Art of Computer Programming*.

Finally it was necessary to freeze the collection, in order to have a `fixed point` for reproducible experiments. The English language will always be evolving, but the 5757 SGB words will therefore always stay the same — even though the author has been tempted at times to add a few words that he didn't know in 1992, such as `chads`, `stent`, `blogs`, `ditzy`, `phish`, `bling`, and possibly `tetch`. No; `noway`. The time for any changes to SGB has long since `ended`: `finis`.

> *The following Glossary is intended to contain all well-known English words ... which may be used in good Society, and which can serve as Links. ... There must be a stent to the admission of spick words.*
>
> — LEWIS CARROLL, *Doublets: A Word-Puzzle* (1879)

> *If there is such a verb as to tetch, Mr. Lillywaite tetched.*
>
> — ROBERT BARNARD, *Corpse in a Gilded Cage* (1984)

Proper names like `Knuth` are not considered to be legitimate words. But `gauss` and `hardy` are `valid`, because "gauss" is a unit of magnetic induction and "hardy" is hardy. In fact, SGB words are composed entirely of ordinary lowercase letters; the list contains no hyphenated words, contractions, or terms like `blasé` that require an accent. Thus each word can also be regarded as a vector, which has five components in the range $[0\,..\,26)$. In the vector sense, the words `yucca` and `abuzz` are furthest apart: The Euclidean distance between them is

$$\|(24,20,2,2,0)-(0,1,20,25,25)\|_2=\sqrt{24^2+19^2+18^2+23^2+25^2}=\sqrt{2415}.$$

The entire Stanford GraphBase, including all of its programs and data sets, is easy to download from the author's website (see page iv). And the list of all SGB words is even easier to obtain, because it is in the file '`sgb-words.txt`' at the same place. That file contains 5757 lines with one word per line, beginning with '`which`' and ending with '`pupal`'. The words appear in a default order, corresponding to frequency of usage; for example, the words of rank 1000, 2000, 3000, 4000, and 5000 are respectively `ditch`, `galls`, `visas`, `faker`, and `pismo`. The notation '`WORDS`(n)' will be used in this chapter to stand for the n most common words, according to this ranking.

Incidentally, five-letter words include many plurals of *four-letter words*, and it should be noted that no Victorian-style censorship was done. Potentially offensive vocabulary has been expurgated from *The Official SCRABBLE® Players Dictionary*, but not from the SGB. One way to ensure that semantically unsuitable

terms will not appear in a professional paper based on the SGB wordlist is to restrict consideration to WORDS(n) where n is, say, 3000.

Exercises 26–37 below can be used as warmups for initial explorations of the SGB words, which we'll see in many different combinatorial contexts throughout this chapter. For example, while covering problems are still on our minds, we might as well note that the four words '`third flock began jumps`' cover 20 of the first 21 letters of the alphabet. Five words can, however, cover at most 24 different letters, as in {`becks`, `fjord`, `glitz`, `nymph`, `squaw`} — unless we resort to a rare non-SGB word like `waqfs` (Islamic endowments), which can be combined with {`gyved`, `bronx`, `chimp`, `klutz`} to cover 25.

Simple words from WORDS(400) suffice to make a *word square*:

```
class
light
agree .
sheep
steps
```
(9)

We need to go almost to WORDS(3000), however, to obtain a *word cube*,

```
types  yeast  pasta  ester  start
yeast  earth  armor  stove  three
pasta  armor  smoke  token  arena ,
ester  stove  token  event  rents
start  three  arena  rents  tease
```
(10)

in which every 5×5 "slice" is a word square. With a simple extension of the basic dancing links algorithm (see Section 7.2.2.1), one can show after performing about 390 billion mems of computation that WORDS(3000) supports only three symmetric word cubes such as (10); exercise 36 reveals the other two. Surprisingly, 83,576 symmetrical cubes can be made from the full set, WORDS(5757).

Graphs from words. It's interesting and important to arrange objects into rows, squares, cubes, and other designs; but in practical applications another kind of combinatorial structure is even *more* interesting and important, namely a *graph*. Recall from Section 2.3.4.1 that a graph is a set of points called *vertices*, together with a set of lines called *edges*, which connect certain pairs of vertices. Graphs are ubiquitous, and many beautiful graph algorithms have been discovered, so graphs will naturally be the primary focus of many sections in this chapter. In fact, the Stanford GraphBase is primarily about graphs, as its name implies; and the SGB words were collected chiefly because they can be used to define interesting and instructive graphs.

Lewis Carroll blazed the trail by inventing a game that he called Word-Links or Doublets, at the end of 1877. [See Martin Gardner, *The Universe in a Handkerchief* (1996), Chapter 6.] Carroll's idea, which soon became quite popular, was to transform one word to another by changing a letter at a time:

`tears` — `sears` — `stars` — `stare` — `stale` — `stile` — `smile`. (11)

The shortest such transformation is the shortest *path* in a graph, where the vertices of the graph are English words and the edges join pairs of words that have "Hamming distance 1" (meaning that they disagree in just one place).

When restricted to SGB words, Carroll's rule produces a graph of the Stanford GraphBase whose official name is $words(5757,0,0,0)$. Every graph defined by SGB has a unique identifier called its *id*, and the graphs that are derived in Carrollian fashion from SGB words are identified by *id*s of the form $words(n,l,t,s)$. Here n is the number of vertices; l is either 0 or a list of weights, used to emphasize various kinds of vocabulary; t is a threshold so that low-weight words can be disallowed; and s is the seed for any pseudorandom numbers that might be needed to break ties between words of equal weight. The full details needn't concern us, but a few examples will give the general idea:

- $words(n,0,0,0)$ is precisely the graph that arises when Carroll's idea is applied to $\texttt{WORDS}(n)$, for $1 \le n \le 5757$.
- $words(1000,\{0,0,0,0,0,0,0,0,0,0\},0,s)$ contains 1000 randomly chosen SGB words, usually different for different values of s.
- $words(766,\{0,0,0,0,0,0,0,1,0\},1,0)$ contains all of the five-letter words that appear in *The TEXbook* and *The METAFONTbook*.

There are only 766 words in the latter graph, so we can't form very many long paths like (11), although

$$\texttt{basic}\text{ — }\texttt{basis}\text{ — }\texttt{bases}\text{ — }\texttt{based}\text{ — }\texttt{baked}\text{ — }\texttt{naked}\text{ — }\texttt{named}\text{ — }\texttt{names}\text{ — }\texttt{games} \tag{12}$$

is one noteworthy example.

Of course there are many other ways to define the edges of a graph when the vertices represent five-letter words. We could, for example, require the Euclidean distance to be small, instead of the Hamming distance. Or we could declare two words to be adjacent whenever they share a subword of length four; that strategy would substantially enrich the graph, making it possible for `chaos` to yield `peace`, even when confined to the 766 words that are related to TEX:

$$\texttt{chaos}\text{ — }\texttt{chose}\text{ — }\texttt{chore}\text{ — }\texttt{score}\text{ — }\texttt{store}\text{ — }\texttt{stare}\text{ — }\texttt{spare}\text{ — }\texttt{space}\text{ — }\texttt{peace}. \tag{13}$$

(In this rule we remove a letter, then insert another, possibly in a different place.) Or we might choose a totally different strategy, like putting an edge between word vectors $a_1a_2a_3a_4a_5$ and $b_1b_2b_3b_4b_5$ if and only if their dot product $a_1b_1+a_2b_2+a_3b_3+a_4b_4+a_5b_5$ is a multiple of some parameter m. Graph algorithms thrive on different kinds of data.

SGB words lead also to an interesting family of *directed* graphs, if we write $a_1a_2a_3a_4a_5 \to b_1b_2b_3b_4b_5$ when $\{a_2,a_3,a_4,a_5\} \subseteq \{b_1,b_2,b_3,b_4,b_5\}$ as multisets. (Remove the first letter, insert another, and rearrange.) With this rule we can, for example, transform `words` to `graph` via a shortest oriented path of length six:

$$\texttt{words} \to \texttt{dross} \to \texttt{soars} \to \texttt{orcas} \to \texttt{crash} \to \texttt{sharp} \to \texttt{graph}. \tag{14}$$

Theory is the first term in the Taylor series of practice.
— THOMAS M. COVER (1992)

The number of systems of terminology presently used in graph theory is equal, to a close approximation, to the number of graph theorists.
— RICHARD P. STANLEY (1986)

Graph theory: The basics. A graph G consists of a set V of vertices together with a set E of edges, which are pairs of distinct vertices. We will assume that V and E are *finite* sets unless otherwise specified. We write $u \mathrel{—} v$ if u and v are vertices with $\{u, v\} \in E$, and $u \mathrel{+\!\!\!—} v$ if u and v are vertices with $\{u, v\} \notin E$. Vertices with $u \mathrel{—} v$ are called "neighbors," and they're also said to be "adjacent" in G. One consequence of this definition is that we have $u \mathrel{—} v$ if and only if $v \mathrel{—} u$. Another consequence is that $v \mathrel{+\!\!\!—} v$, for all $v \in V$; that is, no vertex is adjacent to itself. (We shall, however, discuss multigraphs below, in which loops from a vertex to itself are permitted, and in which repeated edges are allowed too.)

The graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. It's a *spanning* subgraph of G if, in fact, $V' = V$. And it's an *induced* subgraph of G if E' has as many edges as possible, when V' is a given subset of the vertices. In other words, when $V' \subseteq V$ the subgraph of $G = (V, E)$ induced by V' is $G' = (V', E')$, where

$$E' \;=\; \big\{\, \{u, v\} \;\big|\; u \in V',\, v \in V', \text{ and } \{u, v\} \in E \,\big\}. \tag{15}$$

This subgraph G' is denoted by $G \,|\, V'$, and often called "G restricted to V'." In the common case where $V' = V \setminus \{v\}$, we write simply $G \setminus v$ ("G minus vertex v") as an abbreviation for $G \mid (V \setminus \{v\})$. The similar notation $G \setminus e$ is used when $e \in E$ to denote the subgraph $G' = (V, E \setminus \{e\})$, obtained by removing an edge instead of a vertex. Notice that all of the SGB graphs known as $\mathit{words}(n, l, t, s)$, described earlier, are induced subgraphs of the main graph $\mathit{words}(5757, 0, 0, 0)$; only the vocabulary changes in those graphs, not the rule for adjacency.

A graph with n vertices and e edges is said to have *order* n and *size* e. The simplest and most important graphs of order n are the *complete graph* K_n, the *path* P_n, and the *cycle* C_n. Suppose the vertices are $V = \{1, 2, \ldots, n\}$. Then

- K_n has $\binom{n}{2} = \frac{1}{2}n(n-1)$ edges $u \mathrel{—} v$ for $1 \le u < v \le n$; every n-vertex graph is a spanning subgraph of K_n.
- P_n has $n-1$ edges $v \mathrel{—} (v{+}1)$ for $1 \le v < n$, when $n \ge 1$; it is a path of length $n{-}1$ from 1 to n.
- C_n has n edges $v \mathrel{—} ((v \bmod n){+}1)$ for $1 \le v \le n$, when $n \ge 1$; it is a graph only when $n \ge 3$ (but C_1 and C_2 are multigraphs).

We could actually have defined K_n, P_n, and C_n on the vertices $\{0, 1, \ldots, n{-}1\}$, or on *any* n-element set V instead of $\{1, 2, \ldots, n\}$, because two graphs that differ only in the names of their vertices but not in the structure of their edges are combinatorially equivalent.

Formally, we say that graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there is a one-to-one correspondence φ from V to V' such that $u \mathrel{—} v$ in G if

and only if $\varphi(u) \mathrel{—} \varphi(v)$ in G'. The notation $G \cong G'$ is often used to indicate that G and G' are isomorphic; but we shall often be less precise, by treating isomorphic graphs as if they were equal, and by occasionally writing $G = G'$ even when the vertex sets of G and G' aren't strictly identical.

Small graphs can be defined by simply drawing a diagram, in which the vertices are small circles and the edges are lines between them. Figure 2 illustrates several important examples, whose properties we will be studying later. The Petersen graph in Figure 2(e) is named after Julius Petersen, an early graph theorist who used it to disprove a plausible conjecture [L'*Intermédiaire des Mathématiciens* **5** (1898), 225–227]; it is, in fact, a remarkable configuration that serves as a counterexample to many optimistic predictions about what might be true for graphs in general. The Chvátal graph, Figure 2(f), was introduced by Václav Chvátal in *J. Combinatorial Theory* **9** (1970), 93–94.

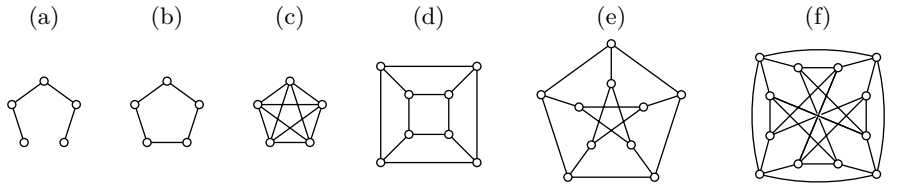
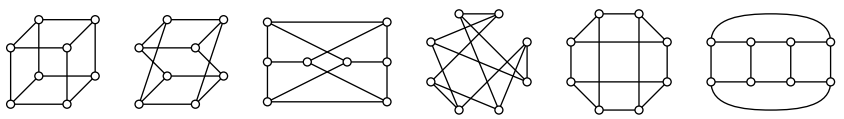


Fig. 2. Six example graphs, which have respectively $(5, 5, 5, 8, 10, 12)$ vertices and $(4, 5, 10, 12, 15, 24)$ edges.

The lines of a graph diagram are allowed to cross each other at points that aren't vertices. For example, the center point of Fig. 2(f) is *not* a vertex of Chvátal's graph. A graph is called *planar* if there's a way to draw it without any crossings. Clearly P_n and C_n are always planar; Fig. 2(d) shows that the 3-cube is also planar. But K_5 has too many edges to be planar (see exercise 46).

The *degree* of a vertex is the number of neighbors that it has. If all vertices have the same degree, the graph is said to be *regular*. In Fig. 2, for example, P_5 is irregular because it has two vertices of degree 1 and three of degree 2. But the other five graphs are regular, of degrees $(2, 4, 3, 3, 4)$ respectively. A regular graph of degree 3 is often called "cubic" or "trivalent."

There are many ways to draw a given graph, some of which are much more perspicuous than others. For example, each of the six diagrams



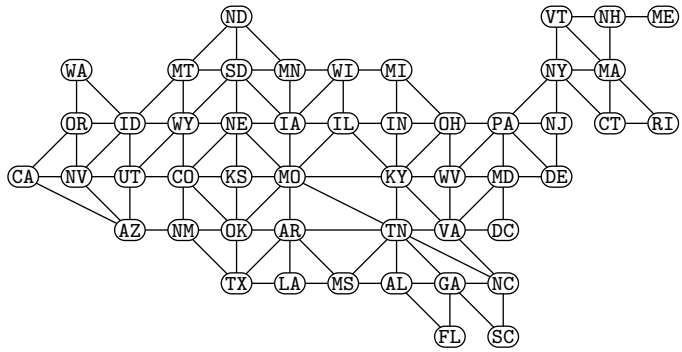
(16)

is isomorphic to the 3-cube, Fig. 2(d). The layout of Chvátal's graph that appears in Fig. 2(f) was discovered by Adrian Bondy many years after Chvátal's paper was published, thereby revealing unexpected symmetries.

The symmetries of a graph, also known as its *automorphisms*, are the permutations of its vertices that preserve adjacency. In other words, the permutation φ is an automorphism of G if we have $\varphi(u) \mathrel{—} \varphi(v)$ whenever $u \mathrel{—} v$ in G. A

well-chosen drawing like Fig. 2(f) can reveal underlying symmetry, but a single diagram isn't always able to display all the symmetries that exist. For example, the 3-cube has 48 automorphisms, and the Petersen graph has 120. We'll study algorithms that deal with isomorphisms and automorphisms in Section 7.2.3. Symmetries can often be exploited to avoid unnecessary computations, making an algorithm almost k times faster when it operates on a graph that has k automorphisms.

Graphs that have evolved in the real world tend to be rather different from the mathematically pristine graphs of Figure 2. For example, here's a familiar graph that has no symmetry whatsoever, although it does have the virtue of being planar:



(17)

It represents the contiguous United States of America, and we'll be using it later in several examples. The 49 vertices of this diagram have been labeled with two-letter postal codes for convenience, instead of being reduced to empty circles.

Paths and cycles. A spanning path P_n of a graph is called a *Hamiltonian path*, and a spanning cycle C_n is called a *Hamiltonian cycle*, because W. R. Hamilton invented a puzzle in 1856 whose goal was to find such paths and cycles on the edges of a dodecahedron. T. P. Kirkman had independently studied the problem for polyhedra in general, in *Philosophical Transactions* **146** (1856), 413–418; **148** (1858), 145–161. [See *Graph Theory 1736–1936* by N. L. Biggs, E. K. Lloyd, and R. J. Wilson (1998), Chapter 2.] The task of finding a spanning path or cycle is, however, much older — indeed, we can legitimately consider it to be the oldest problem of graph theory, because paths and tours of a knight on a chessboard have a continuous history going back to ninth-century India (see Section 7.2.2.4). A graph is called *Hamiltonian* if it has a Hamiltonian cycle. (The Petersen graph, incidentally, is the smallest 3-regular graph that is neither planar nor Hamiltonian; see C. de Polignac, *Bull. Soc. Math. de France* **27** (1899), 142–145.)

The *girth* of a graph is the length of its shortest cycle; the girth is infinite if the graph is acyclic (containing no cycles). For example, the six graphs of Fig. 2 have girths $(\infty, 5, 3, 4, 5, 4)$, respectively. It's not difficult to prove that a graph of minimum degree k and girth 5 must have at least $k^2 + 1$ vertices. Further analysis shows in fact that this minimum value is achievable only if $k = 2$ (C_5), $k = 3$ (Petersen), $k = 7$, or perhaps $k = 57$. (See exercises 63 and 65.)

The *distance* $d(u,v)$ between two vertices u and v is the minimum length of a path from u to v in the graph; it is infinite if there's no such path. Clearly $d(v,v)=0$, and $d(u,v)=d(v,u)$. We also have the triangle inequality

$$d(u,v)+d(v,w)\ \ge\ d(u,w). \tag{18}$$

For if $d(u,v)=p$ and $d(v,w)=q$ and $p<\infty$ and $q<\infty$, there are paths

$$u=u_0\mathrel{—}u_1\mathrel{—}\cdots\mathrel{—}u_p=v\quad\text{and}\quad v=v_0\mathrel{—}v_1\mathrel{—}\cdots\mathrel{—}v_q=w, \tag{19}$$

and we can find the least subscript r such that $u_r=v_s$ for some s. Then

$$u_0\mathrel{—}u_1\mathrel{—}\cdots\mathrel{—}u_{r-1}\mathrel{—}v_s\mathrel{—}v_{s+1}\mathrel{—}\cdots\mathrel{—}v_q \tag{20}$$

is a path of length $\le p+q$ from u to w.

The *diameter* of a graph is the maximum of $d(u,v)$, over all vertices u and v. The graph is *connected* if its diameter is finite. The vertices of a graph can always be partitioned into connected *components*, where two vertices u and v belong to the same component if and only if $d(u,v)<\infty$.

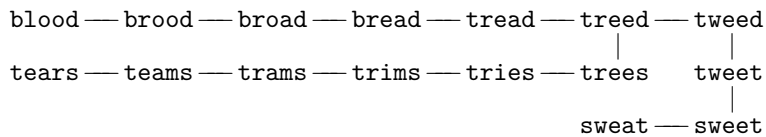
In the graph $words(5757,0,0,0)$, for example, we have $d(\texttt{tears},\texttt{smile})=6$, because (11) is a shortest path from `tears` to `smile`. Also $d(\texttt{tears},\texttt{happy})=6$, and $d(\texttt{smile},\texttt{happy})=10$, and $d(\texttt{world},\texttt{court})=6$. But $d(\texttt{world},\texttt{happy})=\infty$; the graph isn't connected. In fact, it contains 671 words like `aloof`, which have no neighbors and form connected components of order 1 all by themselves. Word pairs such as `alpha` — `aloha`, `droid` — `druid`, and `opium` — `odium` account for 103 further components of order 2. Some components of order 3, like `chain` — `chair` — `choir`, are paths; others, like {`getup`, `letup`, `setup`}, are cycles. A few more small components are also present, like the curious path

$$\texttt{login}\mathrel{—}\texttt{logic}\mathrel{—}\texttt{yogic}\mathrel{—}\texttt{yogis}\mathrel{—}\texttt{yogas}\mathrel{—}\texttt{togas}, \tag{21}$$

whose words have no other neighbors. But the vast majority of all five-letter words belong to a giant component of order 4493. If you can go two steps away from a given word, changing two different letters, the odds are better than 15 to 1 that your word is connected to everything in the giant component.

Similarly, the graph $words(n,0,0,0)$ has a giant component of order $(3825, 2986, 2056, 1186, 224)$ when $n=(5000,4000,3000,2000,1000)$, respectively. But if n is small, there aren't enough edges to provide much connectivity. For example, $words(500,0,0,0)$ has 327 different components, none of order 15 or more.

The concept of distance can be generalized to $d(v_1,v_2,\ldots,v_k)$ for any value of k, meaning the minimum number of edges in a connected subgraph that contains the vertices $\{v_1,v_2,\ldots,v_k\}$. For example, $d(\texttt{blood},\texttt{sweat},\texttt{tears})$ turns out to be 15, because the subgraph



(22)

has 15 edges, and there's no suitable 14-edge subgraph.