

# Discrete Algorithmic Mathematics

## Third Edition

Stephen B Maurer  
Swarthmore College

Anthony Ralston  
State University of New York at Buffalo



A K Peters, Ltd.  
Wellesley, Massachusetts

CRC Press  
Taylor & Francis Group  
6000 Broken Sound Parkway NW, Suite 300  
Boca Raton, FL 33487-2742

© 2005 by Taylor & Francis Group, LLC  
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works  
Version Date: 20130204

International Standard Book Number-13: 978-1-4398-6375-6 (eBook - PDF)

Visit the Taylor & Francis Web site at  
<http://www.taylorandfrancis.com>  
and the CRC Press Web site at  
<http://www.crcpress.com>

# Contents

List of Algorithms	inside front cover
Instructor's Preface	ix
Student's Preface	xiv
Chapter Summaries	xvii
Pathways Through the Book	xx
Problem Difficulty Rating	xxii
Symbols, Notation, Abbreviations and Conventions	xxiii

<b>PROLOGUE</b>	<b>1</b>
-----------------	----------

<b>CHAPTER 0</b>	<b>9</b>
<b>Mathematical Preliminaries</b>	
0.1 Sets, Relations, and Functions	9
0.2 Some Important Functions	20
0.3 Growth Rates and Order Notation	30
0.4 Summation and Product Notation	40
0.5 Matrix Algebra	50
0.6 The Language and Methods of Reasoning	56
Supplementary Problems	69

<b>CHAPTER 1</b>	<b>73</b>
<b>Algorithms</b>	
1.1 Some Examples of Algorithms	73
1.2 Aspects of AL	88
1.3 Recursive Algorithms	94
1.4 Algorithmic Language: Procedures and Functions	105
1.5 The Analysis of Algorithms	117
Supplementary Problems	131

<b>CHAPTER 2</b>	<b>135</b>
<b>Mathematical Induction</b>	
2.1 Introduction	135
2.2 First Examples	137

<b>CHAPTER 3</b>	<b>Graphs and Trees</b>	<b>217</b>
2.3	Strong Induction and Other Variants	151
2.4	Induction and Recursive Algorithms	158
2.5	Induction and Iterative Algorithms	168
2.6	How to Conjecture What to Prove	180
2.7	Inductive Definitions	194
2.8	Faulty Inductions	201
	Supplementary Problems	210
<b>CHAPTER 4</b>	<b>Fundamental Counting Methods</b>	<b>321</b>
3.1	Examples and Terminology	217
3.2	Paths, Cycles and the Adjacency Matrix	236
3.3	Eulerian and Hamiltonian Paths and Cycles	250
3.4	Shortest Paths	266
3.5	Breadth First Search and Depth First Search	278
3.6	Coloring Problems	285
3.7	Trees	296
	Supplementary Problems	311
<b>CHAPTER 5</b>	<b>Difference Equations</b>	<b>411</b>
5.1	Introduction	411
5.2	Modeling with Difference Equations	413
5.3	Getting Information from Difference Equations	426
5.4	Terminology and a Fundamental Theorem	433
5.5	Constant Coefficient Homogeneous Linear Difference Equations	439
5.6	Qualitative Analysis	451
5.7	Nonhomogeneous Difference Equations	457
5.8	Applications to Algorithms	462

<b>CHAPTER 6</b>	<b>Probability</b>	<b>513</b>
5.9	Variable Coefficients, Sums, and Recent Advances in Computer Algebra	479
5.10	Nonlinear Difference Equations	484
5.11	Finite Differences	495
	Supplementary Problems	508
<b>CHAPTER 7</b>	<b>An Introduction to Mathematical Logic</b>	<b>611</b>
7.1	Introduction and Terminology	611
7.2	The Propositional Calculus	616
7.3	Validity and Tautology	630
7.4	Algorithm Verification	637
7.5	Boolean Algebra	643
7.6	The Predicate Calculus	663
7.7	Algorithm Verification Using the Predicate Calculus	676
7.8	Theorems and Proofs	682
	Supplementary Problems	689
<b>EPILOGUE</b>	<b>Coming Full Circle with Biology and Minimax Theorems</b>	<b>691</b>
E.1	DNA Matching	691
E.2	Algorithmic Mathematics and Minimax Theorems	707
	Final Problems	717
	References	723
	Appendix: Limits	729
	Hints and Answers	733
	Index	761

## List of Algorithms

This table lists all the numbered algorithms in the book. All algorithms displayed in boxes in the text are numbered, as well as the more important algorithms displayed in boxes in the problems. In addition, there are numerous algorithm fragments both in the text and in the problems. The fragments and unnumbered algorithms illustrate some point and are generally not important algorithms in themselves.

Number	Name	Output or Purpose	Page
1.1	PowerA	Computes integer powers of a number	74
1.2	PowerB	Speed-up of 1.1 for some powers	75
1.3	PowerC	Speed-up of 1.1	75
1.4	PrimeNum	First $K$ prime numbers	77
1.5	FiberNet	Minimum cost links in a network	80
1.6	Euclid	Greatest common divisor (gcd) of two integers	84, 123, 170
1.7	Euclid1	Possibly faster variant of 1.6	87
1.8	Euclid-RecFunc	Recursive function version of 1.6	96, 163
1.9	Euclid-RecPro	Recursive procedure version of 1.6	97, 165
1.10	Hanoi	Towers of Hanoi	101
1.11	StringProc	Process a string of characters	109
1.12	MaxNumber	Maximum of $n$ numbers	119, 587
1.13	SeqSearch	Sequential search of a list	125
1.14	BinSearch	Binary search of a list	127, 172
1.15	Euclid2	Possibly faster variant of 1.6	132
2.1	Hanoi	Slight rewrite of 1.10	160
2.2	TOH	Variant recursive procedure for Hanoi	166
2.3	EuclidRound-Func	Possibly faster variant of 1.8	167
2.4	EuclidRound-Pro	Possibly faster variant of 1.9	167
2.5	Euclid-RoundUp	Possibly faster variant of 1.6	167
2.6	Iterative-TOH	Non-recursive Towers of Hanoi	179
2.7	Sum-Search	Find if one number is the sum of some others	213
2.8	EuclidZ	Version of 1.6 that also works for negative integers	214
3.1	BuildTree	Binary tree of a list of words	223
3.2	Warshall	Path matrix of a graph	244
3.3	Pathgrow	Grows a path in a graph	255
3.4	Ecycle	An Eulerian cycle in a graph	256

Number	Name	Output or Purpose	Page
3.5	HamPath	A Hamiltonian path in a complete graph	261
3.6	Dijkstra	Shortest path in a weighted graph	272
3.7	BreadthFirstSearch	Breadth first search of a connected graph	279
3.8	PathLength	Lengths of all paths from a given vertex	281
3.9	DepthFirstSearch	Depth first search of a connected graph	282
3.10	Bipartite	Determine if a graph is 2-colorable	288
3.11	$K$ -Colorable	A proper coloring of a graph	291
3.13	Color-by-Vertices	Variant of 3.11	294
3.13	One-Color-at-a-Time	Different approach to coloring a graph	294
3.14	SpanTree	Spanning tree of a graph	297
3.15	MinSpanTree	Minimum weight spanning tree of a graph	299
3.16	BinaryTreeTraversal	Three ways of traversing a binary tree	306
3.17	Ford	Shortest path in a weighted graph	314
4.1	PermValue	The permutation number $P(n, r)$	343
4.2	PermValue2	The permutation number $P(n, r)$	347
4.3	Permute-1	Random permutation of $[n]$	388, 590
4.4	Permute-2	Random permutation of $[n]$	390, 592
4.5	Permute-3	Random permutation of $[n]$	391
4.6	AllPerm	All permutations of $[n]$	394
4.7	Monotonic	Longest monotonic subsequence	402
5.1	Straightforward	Polynomial evaluation	463
5.2	Horner	Faster polynomial evaluation	464
5.3	FibA	Fibonacci numbers	466
5.4	FibB	Fibonacci numbers by recursion	466
5.5	PowerPoly	Polynomial evaluation	475
6.1	BigNext	Two largest of $n$ numbers	593
7.1	Mult	Integer multiplication	615
7.2	Insertion	Insertion sort of a list	678
E.1	StringAlign-Rec	Optimal alignment of DNA strings	697
E.2	StringAlign	Iterative Optimal Alignment	699
E.3	StringAlign2	Variant of E.2	704
E.4	ProjectSched	Minimum completion time	709

# Instructor's Preface

The original version of this book, published in 1991, was intended to make concrete our idea that discrete mathematics could and should be a year course in undergraduate mathematics and be a coequal of calculus in the first two years of the undergraduate curriculum.

However, now as then, few students take a full year of discrete math during the first two undergraduate years. Therefore, this new edition of our book focuses on a one-semester discrete mathematics course.

Almost no one disputes any longer that discrete math is important and that it has a place in a well-rounded mathematics or computer science program. Moreover, more and more is showing up in the school curriculum. Also, physical and social scientists urge their students to learn some. Thus, many students from a variety of disciplines know about discrete math and now study it somewhere, or should.

Although this 3rd edition of DAM (as we like to call it, or in this case DAM3) is intended for a one-semester course, we think it contains enough material for about one-and-a-half semesters. It's our idea of what's most important to do, for students of various disciplines, given only a semester, with various options about what to emphasize. Most instructors will want to either skip some sections of each chapter, or skip some chapters. See Pathways Through the Book later in the front matter.

## Algorithms, Paradigms, and Discrete Mathematics

We think that discrete mathematics needs a central theme. In fact, our book has both a central *object* — algorithms — and two central *methods* — the inductive and recursive paradigms. We now say a bit about these.

There have always been algorithms in mathematics. Furthermore, with the great importance today of algorithms, it is now recognized that humans need a precise way to communicate to each other (not just to machines) about algorithms. Hence the increased use of “algorithmic language” in discrete (and other) mathematics books, including this book. (We don't call it “pseudocode” because from the point of view of communication between humans there is nothing “pseudo” about it.)

Our algorithmic notation is more formal than in most discrete mathematics books, but less formal than that in books which use an actual computer language. We think it is important that students learn early to treat algorithms as mathematical objects, with appropriate precision. Nevertheless, our language is sufficiently simple, and the meaning of its structures sufficiently transparent, that even students with little or no computer experience should be able to understand it, read it, and write simple algorithms with it. In particular, a programming course is

*not* a prerequisite for this book, although some prior experience with algorithmic constructs is helpful.

Many mathematicians still regard algorithms as afterthoughts — first you prove that something exists and later, as a lesser activity, you find some way to compute the thing. In our philosophy, algorithms are more central than that. Algorithms can be used to prove that objects exist and simultaneously provide a way to construct them — thus providing the elegance of a stone that kills two birds. This idea of “proof by algorithm” shows up repeatedly in our book. Algorithms are also an object of mathematical study as well as an engine for proofs. We devote much attention to the analysis of algorithms (how long they take to run) and to proofs of correctness.

Our attitude towards algorithms is summarized by our title. It is not “Discrete Mathematics”; it is not even “Discrete Mathematics with Algorithms”; it is “Discrete Algorithmic Mathematics”. “Algorithmic” is an essential modifier of “Mathematics”. Of course, we do not mean “algorithmic” in the sense of rote application of rules, but rather in the sense of *algorithmics*, which means thinking in terms of algorithms and their mathematical study.

But how do you devise an algorithm which solves the problem at hand, and how do you convince yourself you are right? This is where our central methods come in. The inductive paradigm is the method of building up cases, looking for patterns, and thus obtaining a (usually) iterative algorithm. The recursive paradigm is the method of supposing you already know how to handle previous cases, finding a connection to the current case, and thus obtaining a (usually) recursive algorithm. We emphasize repeatedly how much mileage one can get with these elaborations of the simple ideas “build from the previous case” and “reduce to the previous case”.

## What Hasn’t Changed from Earlier Editions

Our book remains notable among discrete mathematics books in a number of ways.

- It’s use of algorithmics as a central theme. While the coverage of algorithmics in other discrete books has increased since our book originally appeared, this theme remains more central in our book than in any other.

Put another way, DAM has a *point of view*. We believe that algorithms are a key *mathematical* concept, and issues about algorithmic implementation should not be far away when thinking about any mathematical idea. In addition, on several occasions (for instance, the Euler Graph Theorem or the monotone subsequence corollary of the Pigeonhole Principle), we illustrate both existential and algorithmic approaches to both the statement of the theorem and its proof. We favor the algorithmic approach, but we acknowledge the value of both.

- DAM provides a *broad* course, with discussion and examples on discrete topics from a wide variety of disciplines, not just mathematics and computer science. (See Chapter Summaries later in the front matter for a clearer idea of what we mean by this.) We think that DAM will therefore appeal to students with a wide variety of interests.

- DAM has worked especially well with better prepared students and is more sophisticated than most discrete texts. It's not that DAM is a theoretical book, concerned only with proofs and intended for juniors and seniors. On the contrary, for the most part our presentation is fairly informal and concentrates on problem solving and modeling using the standard techniques and concepts of the subject. (Chapter 2, Induction, is an exception.) Rather, the sophistication comes from how we talk about the methods (we show how we as a mathematician and a computer scientist think about them) and from the problem sets. Those students who actually read their math books will gain a lot. And while there are many routine problems, there are plenty of nonroutine ones too, often with lengthy discussions about why they are interesting. (See the discussion later of our ranking system for problems.)

In any event, DAM should help students develop mathematical maturity. They will gain some idea of what a good mathematical argument is and will take first steps toward learning how to devise and present such arguments themselves.

## What Has Changed

- Chapters 8 (linear algebra) and 9 (bridge to continuous math through infinite processes) have been deleted. Some material from Chapter 9 (generating functions, order of growth, finite differences) has been retained by moving it forward.
- Several parts of Chapter 0 have been shortened, for instance, the discussions of sets, relations, and functions. Chapter 1 has also been shortened, by reducing the number of examples.
- Because discussion of Ord and Big-Oh notation has been moved forward (to Chapter 0), we include more explicit discussion of the order of algorithms throughout the book.
- The syntax of our Algorithm Language has been changed modestly. Following a reader suggestion, we have made inputs and outputs to procedures explicit. This provides added clarity and makes it easier to explain the close tie between recursion in algorithms and proof by induction.
- Chapter 5 (difference equations) contains new sections on nonlinear recurrences (chaos) and finite differences.
- The old Epilogue on sorting has been replaced by a new Epilogue that highlights computational biology, an important current topic.
- Many problems have been added or revised.
- Many small changes to improve clarity have been made on virtually every page.

*Computer Algebra Systems.* CASs have been mentioned in DAM since DAM1, but when we first started writing, it was not clear whether such automation would revolutionize mathematics and its teaching or be a dud. It has worked out in-between. A CAS, like a pencil, is a wonderful tool, and it is important to know

what it can do and when to use it. So in DAM we recommend readers use a CAS whenever we ourselves would. It has also affected the relative importance of topics in DAM. For instance, in Chapter 5 we have reduced the amount of discussion of complicated procedures for solving difference equations that a good CAS now automates, replacing it with some discussion of recent advances in automated solutions.

## Additional Materials

First, we have devoted considerable care to the front and back matter. In the front we have a list of algorithms, a summary of the chapters, suggested pathways through the text, a brief discussion of our system for problem rating, and a careful description of our notation (mathematical and otherwise). In the back we have a bibliography of useful references, a brief treatment of limits, Hints and Answers to selected problems (those whose numbers are in color), and a full index.

Second, there will also be two solutions manuals — one for instructors, giving solutions for all problems, and another for sale to students, giving detailed solutions to those problems for which there are hints or answers in the back of the book. These manuals should prove valuable, as relatively few of our problems are completely routine.

An important change behind the scenes is that DAM is now a L<sup>A</sup>T<sub>E</sub>X document instead of a Plain T<sub>E</sub>X document, with electronic figures (done mostly with PSTricks) and hyperlinking code. This means it will be possible to turn DAM into a Web document, and we may do this later if we develop a website for the book. To find out, check with the A K Peters website from time to time ([www.akpeters.com](http://www.akpeters.com)). Such a website would include other material as well — a discussion board with users, updated errata, etc.

## Acknowledgments

We started writing DAM1 in 1980, and the book has been part of our daily lives ever since! If the way to learn a subject is to teach it, and the way to really learn it is to write a book about it, then writing a book *jointly* about it is tenfold as educational. From each other we have learned many things about the subject, about how to write, and about pedagogy. We have also learned that the obvious way to do things is not only not always obvious to someone else (who might that be?), but it is sometimes obviously *wrong* to someone else. Perhaps if we weren't both so stubborn, every edition would have come out much sooner (with less exhausted authors), but we have learned a lot from the process, and we think you have gotten a better book.

Many people have helped us over this period, and here we wish to thank those who have affected the third edition.

First we thank our publishers, Alice and Klaus Peters, for their belief in our book. For their prompt and careful work, we also thank the following A K Peters staff: Charlotte Henderson, Jonathan Peters, Michelle Peters, Susannah Sieper, and Darren Wotherspoon.

Next we thank the many other people who have helped us with comments, suggestions and corrections at various stages of preparation for the third edition: Stephen Altschul, Dane Camp, Garikai Campbell, Richard Duke, David Flesner, Joseph Halpern, Charles Kelemen, David Kincaid, Paul Klingsberg, Harris Kwong, David Levine, Udi Manber, Malcolm Mason, Jayadev Misra, Paul Rablen, Pyi Soe, Diane Spresser, and Paul Stockmeyer.

Next we thank the Swarthmore students who have, spread out over many years, helped us write the solution manuals: Laurel Evans, Hal Pomeranz, Gil Rosenberg, and Brian Taylor. Another Swarthmore student, Kam Woods, did most of the work converting the figures into electronic form.

We would also like to thank the students in our classes, also spread out over many years, whose questions have led us to clarify our thinking and to add many new problems to explore the issues they have raised.

Finally, we each have some personal acknowledgments.

One of us (SM) says: I thank my wife Fran and my kids Leon and Aaron for their patience and understanding. This book predates my kids, who are now teenagers, the older one leaving for college in the Fall; it pleases me that they both have now read some parts of the book, and found it interesting.

The other of us (AR) says: My wife, Jayne, has become used — too used, she would probably say — to being a book widow. Once again she has uncomplainingly played that role.

Stephen B Maurer  
Swarthmore PA  
[smaurer1@swarthmore.edu](mailto:smaurer1@swarthmore.edu)

Anthony Ralston  
London  
[ar9@doc.ic.ac.uk](mailto:ar9@doc.ic.ac.uk)

April, 2004

# Student's Preface

Welcome to DAM, *Discrete Algorithmic Mathematics*. Discrete math is relatively new as a distinct course in mathematics — courses by this name have only existed for about 30 years. For a quick idea what it is about, read the Prologue. To find out more, look at the Instructor’s Preface, especially the subsection “Algorithms, Paradigms, and Discrete Mathematics” and the chapter summaries that follow this preface.

This Student’s Preface contains our advice on how best to learn from our book.

To make the most effective use of this book, you will need to use all three of its components: most importantly, the prose; secondarily, the problems; and also the front and back matter.

*The Prose.* The text of each section is meant to be *read* as a whole, not just used for chunks to pull out as needed when doing homework.

Why? First, there aren’t a whole lot of routine methods in discrete math, so the best way to understand the full picture of what is going on is to read our explanations as a whole, at least the first time around.

Second, only by careful reading will you see that the subject matter of discrete mathematics is *highly interconnected*. As you read DAM, you will find many forward and backward references, to problems, examples, theorems, sections, etc. You never need to interrupt your reading to follow these links — each section in DAM is intended to be self-contained — but you may wish to follow them, at least on a second reading. These links are our way of emphasizing the connectedness of the material.

Third, DAM has a *point of view*. A mathematics textbook should be good for more than just learning the facts of the discipline. One of the other things it can be good for is a perspective on its subject, but you’ll never discover whether a book has a point of view unless you read it. The point of view of DAM, that *algorithmic* issues are central and interesting, is not shared by every mathematician, or even every computer scientist in the way that we mean it. So we hope you will find it interesting to consider our perspective.

How to read the book. Common traditional advice is to read a math book with pencil in hand, to fill in any steps that are unclear. This is good advice. Although we always try to explain our steps, we sometimes leave calculations to you. These days, you may also wish to have a CAS by your side — a Computer Algebra System, either handheld or on a computer. These are wonderful tools (though they take a while to learn to use well), and we recommend their use from time to time in the text. (But don’t use a CAS or a calculator for little bits of routine algebra

or arithmetic, most of which you should be able to do mentally or easily with pencil-and-paper.)

Throughout the text we have scattered parenthetical “Why?”s. These typically appear at the end of sentences where we claim that something follows from the previous text and it’s especially important to understand why. When you come across a “Why?” and don’t immediately understand why, take a minute — or a few — to back up to make sure you figure it out. That pencil or CAS may help, but most of the time all you need is thought or mental calculation.

On the other hand, don’t miss the forest for the trees. Learning the specific mathematical facts and techniques in this book is important, but even more important, we think, is that you become fully conversant with

- a) The two paradigms — inductive and recursive — that appear and reappear throughout this book. The ability to deploy these paradigms will serve you well in a variety of other math courses.
- b) The use of algorithms to describe mathematical processes formally and the use of algorithms to prove that claims about mathematics (i.e., theorems and their corollaries) are true.
- c) The proof methods that are used in this book, particularly proof by induction, since it is quintessentially the most important method of proof in discrete mathematics.

So read for broad understanding as well as specifics.

*The Problems.* Most mathematicians agree that the only way to *really* learn a mathematical subject is to do lots of problems. So we have spent as much time creating the problems in DAM as in writing the prose. In most sections there are only a few “turn the crank” problems, and there is quite a range of difficulty, with some problems being very hard. Many problems extend the content of the text in some way and will require considerable thought. Because much in the problems is not routine, we provide two aids in the text: 1) a 1-to-5 difficulty rating for each problem, as explained later in the front matter, and 2) a hints and answers section at the back of the book. Any problem (or problem part) whose number (or letter) is in color has a hint or answer in the back. But need we say that their purpose will be defeated if you look at them *before* you try (hard!) to do a problem?

*The Front and Back Matter.* Please take a quick look now at all the material from the front-inside cover up to the Prologue, and then at all the material after the Epilogue through the back inside cover, so that you know what’s there. These materials are meant to help you. For instance, the index is quite comprehensive, and many items are indexed in more than one way. Any item or discussion you might want to locate, that is smaller than a whole section, we hope is indexed. (Whole sections you will find in the Table of Contents.) Some other items in this “end matter” appear in many math or CS books; e.g., a bibliography and a table of notation. But some are less common and you might not be expecting them. For instance, in order to provide succinctly all the forward and backward references discussed early in this Preface, we need a systematic way to refer to problems,

sections, theorems, etc. The discussion of numbering and naming explains this. Similarly, please note the description of our rating system for problems.

*Feedback.* We welcome feedback from you. See our email addresses at the end of the Instructor's Preface.

Good luck!

Stephen Maurer & Anthony Ralston

# Chapter Summaries

**Prologue.** This gives an application which foreshadows both the subject matter and the point of view of the rest of the book.

**Chapter 0, Mathematical Preliminaries.** The goal is to make sure students are comfortable with standard concepts, notations and operations used throughout the book (e.g., set notation, the function concept, summation notation, order of growth, matrix multiplication, logical implication). This is not the real stuff of discrete math, and we believe that a discrete math course should not dwell on this chapter longer than necessary. Well-prepared students will have seen some of the material before, but some will probably be new to almost everyone (e.g., order calculus). Consider referring back to this material as needed instead of starting with it.

**Chapter 1, Algorithms.** This chapter introduces our algorithmic language through a number of examples, including the two we use repeatedly in later chapters – the Euclidean algorithm and the Towers of Hanoi. Also introduced through examples are key issues in the analysis of algorithms and an introduction to complexity ideas. The use of recursion in our language and its close connection to mathematical induction are essential parts of our recursive paradigm theme. If only modest algorithm writing fluency is expected, portions of this chapter can be covered fairly quickly. In any event, the concept of an algorithm has become better known in schools since we first wrote this book. Therefore, the examples and discussion of the nonrecursive parts of our language have been cut back.

**Chapter 2, Induction.** This is a very thorough presentation of mathematical induction. Induction is presented in all its aspects (not just a method of proof, but also a method for definitions, discovery, and algorithm construction), and students are asked to devise inductive definitions and proofs themselves. This may seem out of keeping with our emphasis on problem solving over theory, but we think not. First, we think induction proofs are in many ways the easiest type of proof to ask students to do. Induction has a pattern which varies relatively little from one proof to another (although sometimes it requires thought and ingenuity to apply the pattern). Second, we go to great lengths to explain to students how to do it – both how to discover the induction and how to write it up. Third, a reading knowledge of induction is absolutely necessary for the text, as induction is the foremost method of solution and proof in discrete mathematics.

While the purpose of this chapter has not changed since earlier editions, the material is now organized quite differently. For clarity, and for those who don't like algorithms as much as we do, the material on induction and algorithms is now in

separate sections. Also, material that some may wish to skip (how to conjecture, inductive definitions, faulty inductions) is now in the last sections.

**Chapter 3, Graphs and Trees.** Graphs are a primary construct for modeling discrete phenomena. After a discussion of standard terminology and matrix methods, we give a selection of topics which allow us to emphasize algorithms and the inductive/recursive paradigms. We use path growing to provide algorithmic proofs of the standard Eulerian graph theorems. We introduce various graph and tree searching paradigms and use these to determine shortest paths, connectivity, two-colorability, and minimum spanning trees. We also discuss graph coloring more generally as well as various aspects of binary trees. Perhaps the most notable change is that the presentation of Dijkstra's algorithm in Section 3.4 has been completely revamped.

**Chapter 4, Fundamental Counting Methods.** This chapter provides a fairly traditional development. However, there is significant emphasis on combinatorial arguments (conceptual approaches instead of computation), and the two final sections are very algorithmic (construction of combinatorial objects and algorithmic pigeonholes!). A major change is the replacement of the inclusion-exclusion section with a section on generating functions. A brief introduction to inclusion-exclusion now appears in an early section. As for generating functions, because they now appear much earlier (they used to be in the now omitted Chapter 9), they can be and are used occasionally in later chapters.

**Chapter 5, Difference Equations.** These are a fundamental modeling tool, on a par with differential equations. Because difference equations (a term we use synonymously with “recurrence relations” and “recurrences”) are so closely related to the inductive and recursive paradigms, we devote a whole chapter to them. However, this is also the part of discrete math that is most easily automated. Therefore our emphasis has changed somewhat from earlier editions, in light of the increased power and availability of CASs and new results in automated combinatorial theorem proving. The amount of time spent on more complicated cases has been reduced, with readers urged to use a CAS. On the other hand, setting up appropriate difference equations has not yet been much automated, and, in any event, students will always need to *understand* difference equations, that is, to understand how to formulate models, how to verify solutions (especially if they are not of some form where finding a solution has been automated) and why solution methods work. Therefore, we continue to devote considerable space to modeling problems with difference equations and to conjecturing and verifying solutions. We also introduce some new developments, notably the automated methods promulgated and in large part developed by Wilf and Zeilberger.

**Chapter 6, Probability.** Informed citizens need to know more about probability and statistics than they are currently taught. Consequently we include this chapter, which focuses on discrete probability and includes applications as far flung as screening tests for disease and what it means when poll results are announced within  $\pm 3\%$ . Probability also has specific connections to other parts of the book. We use it to justify some average case analyses of algorithms done informally earlier in the book. We also show how the recursive paradigm allows easy solutions to

certain types of probability questions. The inclusion of a section on statistical estimation is a change from earlier editions. While the average case algorithm analysis in this chapter will appeal to CS and math students, the many other applications (medical screening, statistics) are for the broad audience we have in mind.

**Chapter 7, Logic.** Usually a logic chapter comes much earlier, with the goal of helping to develop logical thinking. We have addressed this general goal through various means in our early chapters, particularly through the section on logic concepts in Chapter 0. Chapter 7 is at a more advanced level. There are two sections on formal algorithm verification, of particular interest in CS, and another on Boolean algebra, of interest to CS and math. However, there is lots of material for a broader audience as well, e.g., considerable discussion of validity and soundness of arguments. The first five sections treat the propositional calculus. The last three sections introduce the predicate calculus. The previous versions of Chapter 7 were heavy slogging, and we have worked hard to make it better — the material has been reorganized, with many more subsection headings, clearer wording and definitions, and fewer complicated examples.

**Epilogue.** Through a fairly detailed study of one application, string alignment for DNA sequences, this chapter recapitulates many of the themes introduced heretofore and shows how discrete math has become important in burgeoning fields where any sort of math was limited before. It turns out that the mathematics used takes us back to the Prologue, and this allows a final section in which we recapitulate the algorithmic point of view of our book and provide a bridge to later mathematics courses through the topic of minimax theorems.

**Final Problems.** These challenging problems lead the interested reader into several other areas. Most of the problems are long with several parts — they can serve as the start of papers and projects.

# Pathways Through the Book

## Dependencies among the Chapters

The chapters of this book are fairly independent after the first few. When we say that later chapters do depend on, say, Chapter 2, we mean they depend on the *essential* material in that Chapter, as discussed below in Dependencies *within* Chapters.

*The Prologue* is not required for anything else, but it is short and introduces the key themes, so we recommend it.

*Chapter 0* (Preliminaries). Almost everything in later chapters depends on elementary knowledge of this material. Therefore, as noted in the Chapter Summaries, cover as much or as little of it as your students' preparation requires, and cover it at the beginning of the course or just when it seems needed.

All subsequent chapters depend on *Chapter 1* (algorithms) and *Chapter 2* (induction). These two chapters can be taught in series or in parallel.

*Chapter 3* (graphs) and *Chapter 4* (counting) are very much the traditional guts of discrete mathematics. Although there is not a strong dependence of later chapters on Chapters 3–4, graphs are used as examples in many places and simple counting arguments show up repeatedly.

Difference equations, being recursive entities, appear throughout the book, so the systematic approaches to them discussed in *Chapter 5* are well worth covering.

*Chapter 6* (probability) and *Chapter 7* (logic) are independent of each other, and make less persistent use of our key themes than most earlier chapters. Still, they resolve and clarify many points that have come up earlier.

The *Epilogue* can be read after Chapters 1–3, but since the point is to recapitulate all of DAM's key themes, the end of the course is the best time for it.

## Dependencies within Chapters

While it would be counterproductive to jump back and forth within any chapter, not every section depends on previous ones in more than incidental ways, and a number of sections can be skipped. For each chapter we now indicate the dependencies and which sections are most easily skipped. (Any section not needed for a later section you cover can be skipped.)

Let  $x.y \rightarrow x.z$  mean that Section  $x.y$  is needed for Section  $x.z$ .

Chapter 0. All sections are independent, except that 0.2 depends on the simplest terminology about functions from 0.1.

Chapter 1. Each section depends on the previous one. For students who are familiar with iteration in programming, you can cover Sections 1.1 and 1.2 quickly, and for those who are also familiar with recursive programming, you can cover 1.3 and 1.4 quickly. However, make sure all students understand the versions of Euclid's algorithm and Towers of Hanoi in Sections 1.1 and 1.3, because these are used over and over later. Also, Section 1.5 (analysis of algorithms) introduces crucial ideas for the rest of the book.

Chapter 2. 2.2 → 2.3 → the rest, and 2.4 → 2.5. Sections 2.2 and 2.3 (weak and strong induction) are central to the book. As for 2.4 and 2.5 (applications to algorithms), they can be skipped if you don't wish to address proofs of algorithm correctness, but this would be out of keeping with what we think is important. Section 2.1, like the first section in several other chapters, is a brief introduction; it could be skipped, but to little purpose.

Most easily skipped (though we love them): 2.6–2.9.

Chapter 3. 3.1 → 3.2 → the rest, and 3.5 → 3.7. So after the first two sections, you can pick and choose. While 3.1–3.2 is all that is needed for later chapters, it would be silly not to show the power of graph theory by doing more.

Most easily skipped: 3.4 and 3.6.

Chapter 4. 4.2 → the rest and 4.4 → the rest.

Most easily skipped: 4.8–4.10.

Chapter 5. 5.4 → the rest, 5.5 → 5.6–5.7 and 5.7 → 5.8. While Sections 5.1–5.3 are not needed for the solution methods later in the chapter, the practice with modeling they provide is probably the most valuable thing students can learn about difference equations these days.

Most easily skipped: 5.9–5.11.

Chapter 6. 6.2 → 6.3 → 6.4 → 6.5 → the rest, and 6.6 → 6.7.

Most easily skipped: 6.6–6.7 and 6.9.

Chapter 7. 7.2 → 7.3 → the rest, 7.6 → the rest, and 7.4 → 7.7.

Most easily skipped: 7.5, or 7.4 and 7.7.

Epilogue. If you are specifically interested in biology, you can cover E.1 alone. But for more perspective on the course as a whole, cover E.2 as well.

## Two Possible Semester Courses

*Depth over Breadth.* Do the Prologue through Chapter 5, in entirety or almost so, going quickly over Chapter 0 if your students are mathematically well prepared, and quickly over Chapter 1 if they are very familiar with programming languages.

*Breadth over Depth.* Do all the chapters, skipping most of the sections listed above as easy to skip.

Of course, options between these two extremes are possible. We'd be interested to hear what works for you.

## Problem Difficulty Rating

It is often hard to judge the difficulty of a problem in discrete mathematics just by reading it. In part this is because it is not always easy to categorize discrete problems, and even when one can, they don't always have routine solution methods.

Therefore we rate problems, from 1 easiest to 5 hardest. The rating appears right after the problem number, in angular brackets. A higher rating can mean more computational difficulty, but it often means the problem involves a significant extension of the reading, or combines ideas from more than one section, or requires a flash of insight. In general, our rating scheme is this:

- ⟨1⟩ Straightforward. Can be done directly by a method illustrated in this section of the text, and without a lot of work.
- ⟨2⟩ Middling. Can be done by a method illustrated in this section of the text, but takes more work.
- ⟨3⟩ Difficult. Typically involves an extension of ideas in the text, or ideas from several sections, or if doable directly from the reading, is quite involved.
- ⟨4⟩ Very difficult.
- ⟨5⟩ A real honors challenge. This rating is used very sparingly, mostly in the Supplementary Problems at the end of chapters and in the Final Problems.

When a problem has several parts, only the problem as a whole is rated, and the rating refers to the more difficult parts or the total effort involved for all the parts.

The most common rating is ⟨2⟩ (slightly over half the problems), with ⟨3⟩ the next most common (30%), but over one-eighth of the almost 2000 problems are rated ⟨1⟩.

Any system of problem rating is subjective. If you think we are way off on a rating, please let us know.

# Symbols, Notation, Abbreviations and Conventions

Meaning	Symbol or Example	Page first defined or used
<b>A. Abbreviations (alphabetized on the abbreviation)</b>		
Arithemtic-Mean/Geometric-Mean Inequality	AGI	212 [14]
Breadth First Search	BFS	298
Computer Algebra System	CAS	xi
Constant coefficient homogeneous linear difference equation	CCHLDE	435
Constant coefficient linear difference equation	CCLDE	435
Conjunctive Normal Form	CNF	662 [29]
Divide and Conquer	D&C	462
Directed acyclic graph	DAG	716 [4]
Depth First Search	DFS	298
Disjunctive Normal Form	DNF	652
Fully Parenthesized Notation	FPN	630 [20]
Input Specification	IS	638
Left-hand side	LHS	138
Output Specification	OS	638
Polish Prefix Notation	PPN	319 [48]
Right-hand side	RHS	138
Reverse Polish Notation	RPN	318 [47]
Straightline Towers of Hanoi	STOH	166 [5]
Towers of Hanoi	TOH	99
Traveling Salesman Problem	TSP	323
Well-formed formula	wff	625

## B. Algorithms

Main features of notation:

Except for procedures and functions

Section 1.2, pp. 88–90

Procedures and functions

Section 1.4, pp. 105–112

Notation specific to algorithms:

Comments (flush right)

[ $k$  is a counter]

74

Assignment

$\leftarrow$

74

Interchange

$\leftrightarrow$

113 [7]

Meaning	Symbol or Example	Page first defined or used
References to algorithms		
“Algorithm” sometimes followed by a number and then by the algorithm name	Algorithm 1.1 POWERA	74
Just the name of the algorithm in large and small capitals	POWERA	74
<b>C. Notation related to problems</b>		
Problem Rating	$\langle 2 \rangle$	xxii
Reference, when problem is:		
At the end of the current section	[19]	
In the supplementary problems of the chapter	[7, Supplement]	
In another section or chapter	[3, Section 4.9]	
Included in the Hints and Answers in the back – problem number or part letter appears in color	4 or b)	
<b>D. Numbering Conventions</b>		
Algorithms: Those numbered are numbered consecutively in each chapter	Algorithm 2.1	
Definitions: Numbered consecutively in each section	Definition 1	
Equations: Numbered consecutively in each section	(1)	
Examples: Numbered consecutively in each section	Example 4	
Figures: Numbered consecutively in each chapter	Figure 4.8	
Tables: Numbered consecutively in each chapter	Table 3.5	
Theorems, Lemmas and Corollaries: Numbered consecutively in each section, as one group	Theorem 1, Lemma 2	
<b>E. Notation used in mathematical arguments</b>		
Checkmark (to denote “which is true”)	$\checkmark$	141
Comments in displays (flush right)	$[P(n)]$	140
End of proof of theorem	■	37
End of solution to example	■	14
Reason over equal sign	$\stackrel{P(n)}{=}$	139–140
<b>F. General mathematical notation (alphabetized on the <i>Meaning</i> column within each category)</b>		
<b>1. Miscellaneous notation</b>		
Approximately	$\approx$	184

Meaning	Symbol or Example	Page first defined or used
Congruence	$m \equiv n \pmod{k}$	29 [35]
Divisibility (of $t$ by $s$ )	$s t$	14
Ellipsis	$\dots$	10
Equal mod $m$	$=_m$	18 [24]
Indivisibility	$2\nmid n$	130 [6]
Infinity	$\infty$	11
Parallel	$\parallel$	68 [5]
Product	$\prod_{i=1}^n a_i$ or $\prod_{i=1}^n a_i$	49
Summation	$\sum_{i=0}^n a_i x^i$ or $\sum_{i=0}^n a_i x^i$	41
<b>2. Sets</b>		
Cardinality	$ S $	11
Cartesian product	$\times$	13
Complement	$\overline{S}$	12
Containment	$\subset$	11
First $n$ positive integers, $\{1, 2, \dots, n\}$	$[n]$	425 [30]
Integers ( $\dots, -2, -1, 0, 1, 2, \dots$ )	$\mathbb{Z}$ or $I$	10
Intersection	$\cap$	12
Nonnegative integers	$\mathbb{N}$	10
Nonnegative real numbers	$\mathbb{R}_{\geq 0}$	10
Null (or empty) set	$\emptyset$	10
Positive integers	$\mathbb{N}^+$	10
Positive real numbers	$\mathbb{R}^+$	10
Rational numbers	$\mathbb{Q}$	10
Real numbers	$\mathbb{R}$	10
Set difference	$S - T$	12
Set membership	$\in$	9
Set-builder notation	$\{\dots   \dots\}$	9
Union	$\cup$	11
<b>3. Functions</b>		
Absolute value	$ x $	20
Ceiling	$\lceil x \rceil$	21
Factorial	$n!$	22
Falling factorial	$x_{(k)}$	22

Meaning	Symbol or Example	Page first defined or used
Floor	$\lfloor x \rfloor$	21
Greatest common divisor	$\gcd(m, n)$	69 [1]
Inverse	$f^{-1}$	17
Least common multiple	$\text{lcm}(m, n)$	69 [2]
Limit (of a function)	$\lim_{x \rightarrow x_0} f(x)$	730
Logarithm base $b$	$\log_b x$	24
Max	$\max\{a_1, \dots, a_n\}$ or $\max\{a_i\}$	3
Mod (or remainder)	$m \bmod n$	22
Random integer	$\text{RAND}[m, n]$	387
Rising factorial	$x^{(k)}$	22
<b>4. Vectors and Matrices</b>		
Identity matrix	$I$	55
Inner (or dot) product	$\mathbf{a} \cdot \mathbf{b}$	53
Matrix	$A$ (upper case italic)	51
Vector	$\mathbf{a}$ (boldface letter)	53
Zero matrix	$\mathbf{0}$	55
<b>5. Graphs</b>		
Adjacency matrix	$A$	238
Adjacent set to a vertex	$A(v)$	226
Chromatic number	$\chi(G)$	286
Clique number	$c(G)$	289
Complete bipartite graph	$K_{mn}$	266 [39]
Degree (of vertex)	$\deg(v)$	227
Edge (directed)	$(v_i, v_j)$	223
Edge (undirected)	$\{v_i, v_j\}$	223
Edge set	$E$	223
Graph or digraph	$G$ or $G(V, E)$	223
Null graph	$\emptyset$	224
Vertex set	$V$	223
<b>6. Combinatorics</b>		
Bell numbers	$B(b, u)$	371
Binomial coefficient ( $n$ choose $k$ )	$C(n, k)$ or $\binom{n}{k}$	355
Combinations of $n$ things $k$ at a time	$C(n, k)$ or $\binom{n}{k}$	341
Combinations with repetitions	$C^*(n, k)$	346

Meaning	Symbol or Example	Page first defined or used
Multinomial coefficient	$C(n; k_1, k_2, \dots, k_m)$ or $\binom{n}{k_1, k_2, \dots, k_m}$	360
Partitions of $b$ into at most $u$ parts	$p(b, u)$	372
Partitions of $b$ into exactly $u$ parts	$p^*(b, u)$	374 [26]
Permutations of $n$ things $k$ at a time	$P(n, k)$	341
Permutations with repetitions	$P^*(n, k)$	346
Stirling numbers of the first kind	$s_{nk}$	409 [19]
Stirling numbers of the second kind	$S(b, u)$	371
Trinomial coefficient	$C(n; i, j, k)$ or $\binom{n}{i, j, k}$	360
<b>7. Sequences</b>		
Antidifference	$\Delta^{-1}$	498
$b_q - b_p$ (used in antidifferences)	$b_k _p^q$	500
Backward difference	$\nabla$	507 [30]
Big Oh	$O$	38
Difference	$\Delta$	496
DNA strings	$\mathcal{S}, \mathcal{T}$	692
Fibonacci numbers	$f_n, F_n$	196, 424 [15]
Finite sequence ( $n$ -tuple)	$(a_1, a_2, \dots, a_n)$ or $a_1, a_2, \dots, a_n$	13
$h$ forward difference	$\Delta_h$	505
Harmonic numbers	$H(n)$	213 [20]
Infinite sequence	$a_1, a_2, a_3, \dots$ or $\{a_n\}$	13
$k$ th-order difference operator	$\Delta^k$	501
Limit (of a sequence)	$\lim_{n \rightarrow \infty} a_n$ or $\lim_{n \rightarrow \infty} a_n$	31
Little Oh	$o$	33
Order	Ord or $\Theta$	31
Triangular number	$T(n)$	117 [22]
<b>8. Probability</b>		
Binomial distribution	$B_{n,p}(k)$	543
Conditional probability	$\Pr(B A)$	527
Event complement	$\sim E$	521
Expected value	$E(X)$ or $E(f)$	558
Poisson distribution	$f_\lambda(k)$	547
Probability measure	$\Pr$	519
Random variable probability	$\Pr(X=c)$ or $\Pr_X(c)$	540

Meaning	Symbol or Example	Page first defined or used
Standard deviation	$\sigma_X$	570
Variance	$\text{Var}(X)$	570
<b>9. Logic</b>		
And	$\wedge$	621
Boolean addition	$p + q$	645
Boolean multiplication	$p \cdot q$	645
Boolean negation	$\overline{p}$ (overbar)	645
Existential quantifier	$\exists$	669
If and only if	iff or $\iff$	60
Implication	$\implies$	57
Logically equivalent	$\iff$	59
Not	$\neg A$	59
Or	$\vee$	621
Peirce's arrow	$\downarrow$	623
Predicate	$R(i), P(x), P(x, y)$	43
Proposition	$P, Q, P(n)$	138
Sheffer stroke	$ $	623
Universal quantifier	$\forall$	666

# What Is Discrete Algorithmic Mathematics?

What is discrete algorithmic mathematics? If we could answer that in a section, it wouldn't be a subject worth a whole book.

Nonetheless, you deserve an inkling of what's in store. So this section is devoted to a problem illustrating several of our themes. As you read, concentrate on these themes. The problem itself, and the fine points of its solution, are not themselves so important right now.

The problem is: Find the minimum amount of time needed for an intermediate stop on an airline flight given that

- a) we have identified the various tasks which must be done during the stop;
- b) we know how long each task will take;
- c) we have people available to carry on as many tasks simultaneously as we want; but
- d) some tasks must be completed before others can begin.

To simplify things in this example, let's assume the only tasks are the following, where the numbers in the second column are the times needed in minutes. (We'll explain the third column in a moment.)

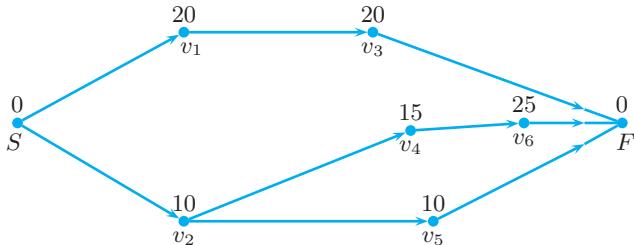
1. Unload luggage	20	
2. Unload deplaning passengers	10	
3. Load new luggage	20	after 1
4. Clean the cabin	15	after 2
5. Load on more food	10	after 2
6. Load new passengers	25	after 4

This example is typical of problems in which we need to determine a minimum amount of time to complete a complicated project. The result here would surely be of interest to passengers, who don't like to wait around, and to airlines, because airplanes don't make money while they are on the ground.

What makes such problems particularly interesting mathematically is d) above: Some of the tasks can be done simultaneously — unloading passengers and unloading luggage — but some must be done after others — new passengers cannot be loaded until the cabin is cleaned, which in turn can't be done until the deplaning passengers are off. This *precedence relation* is specified by the third column. For instance, “after 2” in line 4 means that cleaning the cabin can be started as soon as the deplaning passengers have left.

The key to solving many mathematical problems is to find an appropriate way to *represent* or *model* them. The best representation here turns out to be a very common one in discrete mathematics, a *directed graph*. *Caution:* This is not the graph of a function but an entirely different concept. It is a collection of vertices (dots) with edges connecting some of them. Directed means that the edges have arrows on them. We will discuss such graphs at length in Chapter 3.

Figure P.1 shows the graph representation. There is a vertex for each task. Vertex  $v_1$  stands for task 1, and so on. We have put the time needed to complete each task above its vertex. We have also added two more vertices,  $S$  for start and  $F$  for finish. These represent fictitious marker tasks, so we assign them both time 0. Finally, for every  $i$  and  $j$ , we put an edge from vertex  $i$  to vertex  $j$  if task  $i$  must directly precede task  $j$ . Naturally, we declare that task  $S$  must precede what would otherwise be the earliest tasks, and task  $F$  must follow what would otherwise be the last tasks.



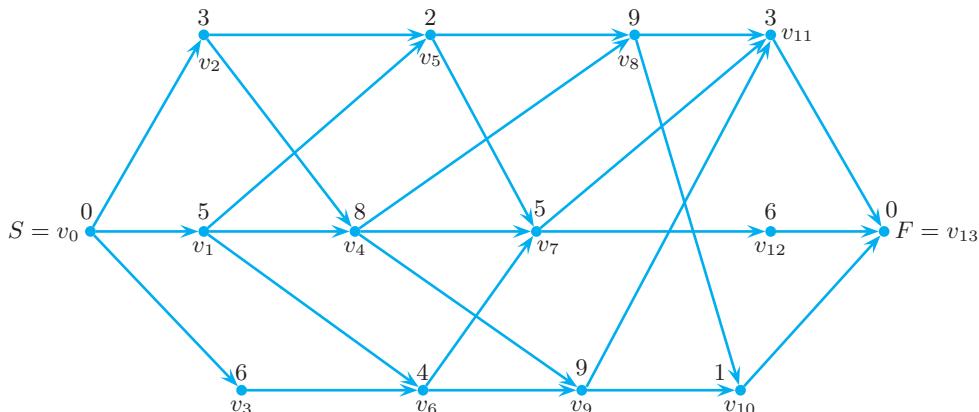
**FIGURE P.1**

Graph Representation of the Airline Problem

For a small example like this, you can probably compute the minimum time to complete all the tasks in your head from looking at the graph — which you probably couldn't do from the original tabular form of the data above. That already suggests the value of the graph model. However, we want a general method, because a task analysis for an actual project would be much more complicated. Imagine, for instance, the graph for the project declared by the United States in 1961, called “Put a man on the moon”. (That project took eight years and used the mathematical method we are developing here.) So what we really hope the graph model can do for us is suggest a general approach.

To help us discover a general method, we need a graph just complicated enough that we can't see the answer and wouldn't want to look for it by brute force. Figure P.2 provides such an example. Never mind what project it represents. We need only schedule the tasks so that the final task finishes as soon as possible.

To begin, we expand the problem. Instead of asking how soon the final task can be finished, we ask how soon *each* task can be finished. This may seem to be



**FIGURE P.2**

A Graph for a More Complicated Scheduling Problem

multiplying our woes but, in fact, it is not. Indeed, such expansion or *generalization* of a problem often makes the solution easier to find.

To solve this expanded problem, we will use the **Recursive Paradigm**. This is one of the most powerful tools of discrete mathematics and of problem solving more generally. It is a major theme of our book.

The first step is: Jump into the middle! For instance, suppose you already knew the earliest completion times for all the tasks that must be completed before the task represented by  $v_7$  in Fig. P.2. Ask: How could I use this information to figure out the earliest completion time for  $v_7$  itself? Recursion means essentially reducing to previous cases, and that's just what we're doing here. (Note: The "middle" above doesn't mean the exact middle, but rather a typical, generic point.)

In Fig. P.3, we repeat enough of Fig. P.2 to show  $v_7$  and all the vertices immediately preceding it:  $v_4$ ,  $v_5$ , and  $v_6$ . (As we will see in a moment, it is important that they are lower numbered than the vertex they precede; fortunately such a numbering is always possible [6].) We imagine we have found the minimum completion times of these tasks. Since we haven't actually calculated them, let's just call them  $A$ ,  $B$ , and  $C$ . Now the *earliest* that task 7 can be completed depends upon which of tasks 4, 5, and 6 is completed *latest*. Thus the minimum time to finish task 7 is

$$5 + \max(A, B, C),$$

where  $\max(A, B, C)$  represents the largest (*maximum*) of  $A$ ,  $B$ , and  $C$ .

A better and more general notation is

$$T(k) = \text{minimum time to complete task } k,$$

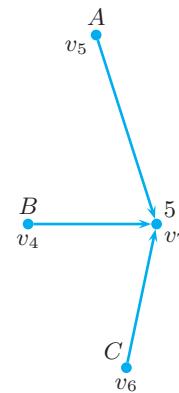
and

$$d(k) = \text{time duration of task } k \text{ itself.}$$

Then

$$T(7) = d(7) + \max\{T(4), T(5), T(6)\}.$$

We did this analysis for  $v_7$  to be concrete. But by now it should be clear that a similar analysis is correct for any vertex. Indeed, we can state a general relationship.



**FIGURE P.3**

A Portion of  
the Graph of  
Figure P.2

If we let

$$\max_{i \rightarrow j} \{T(i)\}$$

mean the maximum of the  $T(i)$ 's for all vertices  $v_i$  with edges directed to  $v_j$ , then

$$T(j) = d(j) + \max_{i \rightarrow j} \{T(i)\}. \quad (1)$$

Equation (1) is called a **recurrence relation** because it expresses the value we wish to calculate,  $T(j)$ , in terms of other values  $T(i)$ .

All right, we've jumped into the middle of our problem, but how does this help us figure out what happens at the end? After all, we want the minimum time to the Finish. That is, we want  $T(13)$ . Suppose we set  $j = 13$  in Eq. (1). We obtain

$$T(13) = 0 + \max\{T(10), T(11), T(12)\},$$

since  $d_{13} = 0$ . We don't know  $T(10)$ ,  $T(11)$ , and  $T(12)$ , but we can analyze them in the same way. For instance, setting  $j = 12$ , we have

$$T(12) = 6 + \max\{T(7)\} = 6 + T(7),$$

since  $v_{12}$  has only one predecessor,  $v_7$ .

In short, using Eq. (1) we can keep working backwards. This would be completely futile — just digging a deeper and deeper hole — except for one thing: *The process eventually stops*. We already know that  $T(0) = 0$ , and eventually we work back to that. Look at Table P.1. The first column gives us a complete list of the backward steps using Eq. (1). Once we get down to  $T(0) = 0$ , we can start plugging in numerical values and working back up.

Indeed, that's what we have started to do in the second column. Starting at the bottom, we fill in the second column by simplifying the equation in the same row of the first column. We can easily obtain the last three rows of the second column, since the corresponding expressions in the first column depend only on  $T(0)$ . The next row up,  $T(4)$ , requires knowledge of the lower rows in the second column, since  $T(4)$  depends on  $T(1)$  and  $T(2)$ . That's why we work *up* in the second column. It's easy. Finish it! [2]

**TABLE P.1**  
**Times needed to complete tasks of Figure P.2**

$T(13) = 0 + \max\{T(10), T(11), T(12)\}$	
$T(12) = 6 + T(7)$	
$T(11) = 3 + \max\{T(7), T(8), T(9)\}$	
$T(10) = 1 + \max\{T(8), T(9)\}$	
$T(9) = 9 + \max\{T(4), T(6)\}$	
$T(8) = 9 + \max\{T(4), T(5)\}$	
$T(7) = 5 + \max\{T(4), T(5), T(6)\}$	
$T(6) = 4 + \max\{T(1), T(3)\}$	$T(6) = 4 + 6 = 10$
$T(5) = 2 + \max\{T(1), T(2)\}$	$T(5) = 2 + 5 = 7$
$T(4) = 8 + \max\{T(1), T(2)\}$	$T(4) = 8 + 5 = 13$
$T(3) = 6 + T(0)$	$T(3) = 6$
$T(2) = 3 + T(0)$	$T(2) = 3$
$T(1) = 5 + T(0)$	$T(1) = 5$

We started in the middle to obtain the key recursive relationship Eq. (1). Then we applied this relationship to the top, i.e., the final value we wanted,  $T(13)$ . This led us, in order to actually compute anything, to work up from the bottom, i.e., from the initial values. So why didn't we just start computing with the initial values in the first place?

We could have. To do so would have been an example of the second major problem solving technique in our book, the **Inductive Paradigm**, in which we start computing at the beginning and continue until we either get to the end or see a pattern. If we see a pattern, then we make a conjecture about what the end result will be and prove this result without computing any more. We will consider this technique in depth in Chapter 2. In this example we could have proceeded inductively from the bottom of Table P.1 and worked our way to the top, as shown in the second column. Here no conjecture about the final result is possible, so we would just have to compute until the end. In using both the recursive and inductive paradigms we'll always be looking for ways to get the final result *analytically* rather than *computing* all the way to the end.

In this problem the two paradigms are not much different. Both quickly lead to computing from the bottom, and both get nowhere unless you hit on Eq. (1) pretty fast. But, in later examples, you'll see that the two paradigms often diverge. Applying the recursive paradigm does not always lead to marching up in order from the bottom. Applying the inductive paradigm does not always require understanding a recursive relationship at the beginning; sometimes you compute the early values to *discover* the relationships you need. Also, the inductive paradigm seems

to come naturally to most people — at least the try-some-small-cases aspect of it. The recursive paradigm, however, seems very unnatural at first to many people. Jumping into the middle seems like a crazy idea. Yet, in our experience, the recursive paradigm is as powerful as the inductive paradigm in discrete mathematics.

In any event, the two paradigms are related, and as a pair they are *very* powerful. In some sense, this whole book is about what you can do with them.

We set out to illustrate discrete algorithmic mathematics with a problem. Now it's time to stand back from our problem and see what it does illustrate:

- It involves distinct (“discrete”) objects, i.e., specific tasks and the precedence among them, as opposed to continuous objects, like the unbroken path of a satellite moving through space.
- Although it would have been nice to find a *formula* for the answer, we were satisfied to find a method for efficiently analyzing and computing the answer.
- We started with a real-world problem (admittedly simplified) and concerned ourselves with a mathematical *model* or *abstraction* (i.e., our representation as a directed graph) in which the essential details of the actual real-world problem were removed. This representation helped us analyze the problem.
- The problem concerned *optimization*, i.e., how to achieve the best possible result, which in this case is the *minimum* time needed to complete the task.

The last two points are not unique to discrete mathematics. But discrete mathematics is especially rich in the variety of applications it can treat, especially optimization applications. It is particularly powerful for treating applications in some areas where continuous mathematics has been less helpful (e.g., computer science, social science, and management science), but it is also increasingly useful in the biological and physical sciences. See the Epilogue for an important application to modern computational biology; in fact, in terms of its mathematical structure the application is surprisingly similar to the airline example in this Prologue!

The second point in the preceding list — about formulas — deserves more explanation. We don't deny that we used formulas in our analysis, notably Eq. (1). But this is not a formula from which you can immediately get an answer. It's not, say, like  $A = \pi r^2$ , into which you can immediately plug numbers and get the area of a circle. That's why we called Eq. (1) a recurrence relation. It contrasts to more familiar equations, which are said to be in *closed form*, or *explicit*.

Furthermore, you wouldn't expect a closed-form formula to exist for the sort of problem we have discussed: There's too much variability in the shape of the graph and the numbers at the vertices. True, if we made the problem much simpler and symmetrical, we might be able to derive a formula. For instance, suppose there were  $n$  vertices,  $d(k) = k$  for each vertex  $k$ , and each vertex  $k$  preceded vertex  $k + 1$  and no others. Then there is indeed a formula for  $T(n)$  [3]. But this is too restrictive to be of general interest. Sometimes formulas in discrete mathematics are of general interest and we'll be interested in some of them in this book. But more often than not there are no such formulas. That is why discrete mathematics might be called the study of *organized complexity*.

The fact is, we typically use formulas like Eq. (1) only as stepping-stones to organized procedures for computing answers. Such procedures are called **algorithms**. Whereas classical mathematics is about formulas, discrete mathematics is as much about algorithms as about formulas.

One reason mathematics has not, until the last half century, been much concerned about algorithms is this: You can't analyze something mathematically, or use it conveniently for calculation, unless you have a precise language for talking about it, and there wasn't such a precise but easily readable language for algorithms. But now, thanks to the computer age, we have *algorithmic language*, that is, simplified versions of high-level computer languages that put the ideas first and don't show any details of machine implementation. (Consequently, algorithmic language is often called "pseudocode" by computer scientists. But from our point of view, calling algorithmic language pseudocode is precisely backwards. For us, programming languages are the pseudo-objects, because they are not the real thing for precise communication among *people*.) In Chapter 1, we'll define carefully what we mean by an algorithm and introduce the algorithmic language we'll use throughout the book. The method we have developed for solving our minimum-time problem, the method illustrated by Table P.1, really is an algorithm. But until Chapter 1 we won't have the language to describe it as such.

Here we are in the Prologue, and we have already referred you to Chapters 1, 2, 3 and the Epilogue. Obviously, we have gotten far ahead of ourselves! As we said at the start, this section is simply an illustration, which you need not grasp in detail — yet. We just want to give you a feel for what discrete algorithmic mathematics is all about.

## Problems: Prologue

---

1.
  - a) <1> What is the minimum time for our airplane stopover?
  - b) <1> Finish filling in Table P.1. What is the minimum time to finish the project?
  - c) <3> Find a formula for  $T(n)$  when, for each vertex  $k$ ,  $d(k) = k$  and the only edge from vertex  $k$  is to vertex  $k + 1$ .
  - d) <2> Our solution procedure outputs, for each task, the optimal time at which that task *ends*. For example, if the procedure assigns 37 to task 4, that means task 4 can be completed at best 37 time units after the project begins. However, what your subcontractors probably want to know is when they can *start* their tasks.
- a) It's easy to figure this out if you have already carried out our solution procedure and you still have the initial data (or graph). How?
- b) But suppose you want to figure out the optimal starting times directly from the initial data or graph. That is, you want a different procedure that computes the start times directly. Devise such a procedure.

What key relationship replaces Eq. (1)?
5. <2> In a slightly different project scheduling model, the vertices merely represent checkpoints and they take essentially no time. The time-consuming tasks are represented by the edges between checkpoints. Thus we get a directed graph model with the weights on the edges. We can still ask: What's the minimum time to complete the final check? Solve this problem. What is the analog to Eq. (1)? How should the path lengths be defined?

6. ⟨3⟩ In our minimum-time algorithm, it was essential that the tasks be numbered so that preceding tasks always have lower numbers. This was essential because, without it, there would be no “up” in the table; there would be no organized way to know, for any task  $j$ , when you had already computed all the  $T(i)$ ’s needed before computing  $T(j)$ .

Yet, we never said anything about where the numbering came from. Indeed, in any real problem the tasks come with names, not numbers. It may not be at all obvious how to number them in order to be “coherent” with the precedence relations. Sometimes it won’t even be obvious that any such numbering exists!

There is a theorem that says such numberings

always exist, so long as there is no *directed cycle* in the graph, that is, a sequence of tasks such that each task on the cycle must precede the next one. Naturally, we prefer a proof of this theorem which actually shows how to construct a numbering. Such an algorithm could serve as a preprocessor for our minimum-time algorithm. Any ideas on how to construct the numbering? The recursive and inductive paradigms might help. (See [16, 17, Supplement, Chapter 3].)

7. ⟨2⟩ In solving the airline problem, where did we use assumption c) — that there are enough workers to carry out simultaneously as many tasks as we want?

# Mathematical Preliminaries

## 0.1 Sets, Relations, and Functions

---

The material in this chapter is not the real stuff of discrete mathematics, but rather material you need to have under your belt to do discrete mathematics. Some of it, like this section, is probably review. Some, like Section 0.3 (Growth Rates and Order Notation) is probably quite new. Some, like Section 0.6 (The Language and Methods of Reasoning) probably contains material you have mostly seen, but organized differently and with advice that may be fresh. In any event, you can read this chapter first, or read it later piecemeal as needed.

### Sets

Sets provide a language for discussing discrete mathematics, or any mathematics. Set language is not exciting, but it is very useful.

A **set** is a collection of objects. They can be any objects, but for now the objects will mostly be numbers. For instance, the set  $A = \{7, 3\}$  contains two things, the numbers 3 and 7. We write  $3 \in A$  and say that 3 is a **member** or **element** of  $A$ . Order doesn't matter, so  $\{7, 3\} = \{3, 7\}$ .

To say that 6 is not a member of  $A$  we write  $6 \notin A$ . In general, a slash through a symbol means “not”, as in  $\neq$  for not equal.

If we had to list all the elements of a set whenever we wished to enumerate them, things would get tedious fast. Luckily, there are other notations which enable us to avoid this problem. One is **set builder notation**, an example of which is

$$S = \{m \mid 2 \leq m < 100 \text{ and } m \text{ an integer}\}. \quad (1)$$

The letter before the vertical bar is a variable representing a typical element. Any letter could be used. The vertical bar should be read as “such that”. The conditions after the bar give the properties which each element must satisfy. Sometimes an

expression appears before the bar instead of a single variable — keep reading for examples. In any event, in Eq. (1)  $S$  is the set of all integers from 2 to 99. The use of “and” is optional; we may replace it with a comma and mean the same thing:

$$S = \{m \mid 2 \leq m < 100, m \text{ an integer}\}.$$

However, sometimes we wish to give a choice of conditions, in which case we must use “or”. Thus the set of positive integers less than 100 that are even or divisible by 3 is

$$T = \{k \mid 0 < k < 100, k \text{ even or } k \text{ divisible by 3}\}.$$

Finally, the  $|$  in set-builder notation may be replaced by a colon, especially if  $|$  is used with other meanings farther to the right, as in

$$U = \{x : |x| \leq 3\}.$$

*Special sets.* So far, our examples of sets have had a *finite* number of members. Discrete mathematics also deals with infinite sets. For instance, henceforth we let  $N$  stand for the **nonnegative integers**, that is,

$$N = \{0, 1, 2, 3, \dots\}.$$

In this definition the three dots are called an **ellipsis** and indicate that to get further elements you simply extend the rule used to form the elements already shown. Use ellipses only when you are confident that the pattern you intend is completely clear. Thus, the numbers after 0, 1, 2, 3 are 4, 5, 6 etc.

When we want just the **positive integers**, we will write

$$N^+ = \{1, 2, 3, \dots\}.$$

Many authors call  $N$  the “natural numbers”. Unfortunately, other authors call  $N^+$  the natural numbers.<sup>†</sup> Since one can no longer be sure what “natural numbers” means, we will avoid this lovely phrase.

Another set that we will use regularly is

$$R = \text{the real numbers.}$$

Variants worth noting are

$$R^+ = \text{the positive real numbers} = \{x \in R \mid x > 0\},$$

$$R_{\geq 0} = \text{the nonnegative real numbers} = \{x \in R \mid x \geq 0\}.$$

Other standard set names are  $Z$  or  $I$  for the full set of **integers**

$$\{\dots, -2, -1, 0, 1, 2, \dots\}$$

and  $Q$  for the rational numbers  $\{a/b \mid a, b \in Z, b \neq 0\}$ .

A surprisingly useful set is the **empty set**, which contains no members at all and is denoted by  $\emptyset$ . Thus

---

<sup>†</sup>Some authors, whom we are too embarrassed to name, have been known to switch back and forth in the same book.

$$\emptyset = \{\}.$$

On the other hand,  $\{\emptyset\}$  represents the set whose one member is the empty set.  $\emptyset$  plays a role in relation to sets very much like the role that 0 plays in relation to numbers (hence the similarity of notation).

May a set have repeated elements as in

$$\{2, 4, 4, 4, 7, 9, 9\} ? \quad (2)$$

No, we shall reserve “set” to mean a collection of *distinct* objects. We shall use **multiset** when we want to refer to a collection in which elements may be repeated. Therefore, expression (2) means a multiset with 7 elements (and we should say “multiset” unless the context makes that unnecessary).

Let  $S$  be a set. Then  $T$  is said to be a **subset** of  $S$  if every element of  $T$  is also in  $S$ . Thus if  $S = \{1, 4, 8, 9, 12\}$ , then  $T = \{4, 8, 12\}$  is a subset of  $S$  but  $V = \{1, 8, 10, 12\}$  isn’t, because  $10 \notin S$ . An immediate consequence of this definition is that  $S$  is a subset of itself. Another consequence is that the empty set  $\emptyset$  is a subset of every set. (See the discussion of “vacuously true” on page 66.)

If  $T$  is a subset of  $S$ , we write

$$T \subset S.$$

Don’t confuse the symbols  $\in$  and  $\subset$ . The symbol  $\in$  is used to denote a relationship between an element of a set and a set (the membership relation), whereas  $\subset$  denotes a relationship between two sets (the subset, or set inclusion, relation).

If  $T$  is a subset of  $S$  and  $T \neq S$ , then  $T$  is called a **proper** subset of  $S$ . *Caution:* In some books the notation  $T \subset S$  means that  $T$  is a proper subset of  $S$ . In the case when  $T$  might be a proper subset or could be equal to  $S$ , the symbol used in such books is  $\subseteq$ .

The elements of a set can themselves be sets. Thus, for example, the set of all subsets of a given set  $S$  is called the **powerset** of  $S$ . The powerset of  $\{a, b\}$  is

$$\{\emptyset, \{a\}, \{b\}, \{a, b\}\}.$$

Finally, we often wish to talk about the number of elements in a set, called its **cardinality**. We do this by placing vertical bars around the set. (Compare this notation with that of absolute value in Section 0.2.) Thus

$$|\{1, 3, 9, 15\}| = 4 \quad \text{and} \quad |N| = \infty.$$

## Set Operations

The four most common and useful operations on sets are:

1. Set **union**, denoted by  $\cup$ , is defined by

$$S \cup T = \{x \mid x \in S \text{ or } x \in T\}.$$

That is,  $S \cup T$  is the set of all elements which are in either  $S$  or  $T$  (but without repeating elements which are in both). For example,

$$\{1, 3, 5\} \cup \{5, 6\} = \{1, 3, 5, 6\}.$$

2. Set **intersection**, denoted by  $\cap$ , is defined by

$$S \cap T = \{x \mid x \in S \text{ and } x \in T\}.$$

Thus  $S \cap T$  is the set of all elements in *both*  $S$  and  $T$ . For example, with  $N$  and  $N^+$  as defined earlier,  $N \cap N^+ = N^+$ . Also, to indicate that sets  $A$  and  $B$  are **disjoint** (have no elements in common), we need merely write  $A \cap B = \emptyset$ .

3. Set **complement**, which we denote by an overbar, as in  $\overline{S}$ , is defined to be the set of all elements *not* in  $S$  but still in some larger **universe**  $U$  of interest at the moment. For instance, if  $U$  is all the integers, and  $S$  is the even integers, then  $\overline{S}$  is the odd integers. In other words,

$$\overline{S} = \{x \mid x \in U, x \notin S\}.$$

It's up to the writer to make clear what  $U$  is.

4. Set **difference**, denoted by  $-$ . If  $S$  and  $T$  are two sets, then by definition

$$S - T = \{x \mid x \in S, x \notin T\}.$$

For instance,  $N - N^+ = \{0\}$  and  $N^+ - N = \emptyset$ .

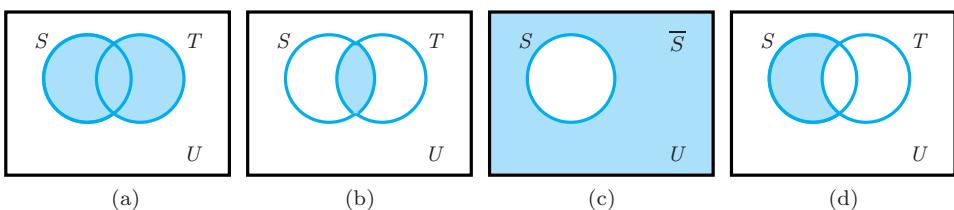
*Note:* Some books write set difference as  $A \setminus B$ , to avoid any possible confusion with subtraction of numbers.

In mathematics, as elsewhere, a picture *is* often worth a thousand words. The most useful visual way to display set operations is the **Venn diagram** (after John Venn, 1834–1923, a British logician), as shown in Fig. 0.1. The outer shape in a Venn diagram, usually a rectangle, represents the universe and the inner shapes, usually circles, represent sets. The shaded areas represent the result of the set operation being displayed. The four set operations we discussed are displayed in Fig. 0.1.

## FIGURE 0.1

Venn diagrams:

- (a) Union  $S \cup T$ ;
- (b) intersection  $S \cap T$ ;
- (c) complement  $\overline{S}$ ;
- (d) difference  $S - T$ .



As with arithmetic operations, set operations can be used to form more complex expressions than just  $S \cup T$  or  $S \cap T$ , etc. And just as in arithmetic, the order of operations matters. Just as  $1 + (3 \times 4) \neq (1 + 3) \times 4$ , so in general

$$A \cup (B \cap C) \neq (A \cup B) \cap C.$$

For instance, take  $A = \{1, 2\}$ ,  $B = \{3, 4\}$ ,  $C = \{5\}$ ; then  $A \cup (B \cap C) = \{1, 2\}$  and  $(A \cup B) \cap C = \emptyset$ .

Thus, as with arithmetic expressions, it is important to use parentheses when there might be any doubt about what you mean. Just as in arithmetic, there are precedence rules that determine what to do in the absence of parentheses, but it is

better to avoid any doubts by using ample parentheses. About the only case where parentheses may safely be avoided is for repeated unions like  $A \cup B \cup C$  or repeated intersections. In each of these cases the answer is the same no matter in what order you do the operations.

For set operations the usual precedence rules are: first do all the complements, then from left to right all the intersections, then from left to right all unions and differences. Thus  $A \cup B \cap C$  means  $A \cup (B \cap C)$  and  $A - B - C$  means  $(A - B) - C$ .

**Ordered Sets.** Sometimes order makes a difference. In this case we call our collection a **sequence**, or occasionally a **list**. To distinguish sequences from sets, one uses parentheses instead of braces, as in

$$(a, b) \quad \text{or} \quad (a_1, a_2, \dots, a_9),$$

or doesn't use any surrounding symbols at all, as in

$$a_1, a_2, \dots, a_9.$$

A sequence with  $n$  elements is often called an  **$n$ -tuple**, especially when written with parentheses. The items in a sequence are called **entries** or **terms**. Two sequences are equal if and only if they are equal entry by entry. Thus  $(3, 9, 5, 1) \neq (3, 1, 9, 5)$ , even though  $\{3, 9, 5, 1\} = \{3, 1, 9, 5\}$ . When we want to give a name to a sequence, we always use parentheses, as in "let  $S = (1, 7, 3)$ ".

For infinite sequences, one can again include or omit parentheses, but alas, by long tradition, the worst possible notation is usually used instead of parentheses — braces. That is, the alternative notation for the sequence  $a_1, a_2, a_3, \dots$  is  $\{a_n\}$ . So how do you tell when braces are being used for sets and when for sequences? If the braces contain a single entry with an index variable, say  $\{a_n\}$  or  $\{a_n + b_n\}$ , the expression refers to a sequence. (The latter sequence has entries  $a_1 + b_1, a_2 + b_2$ , etc.). Otherwise it is a set. Thus  $\{a_1, a_2\}$  is a set, and so is  $\{a_{13}\}$ , although in the latter case, why not avoid braces and just refer to the element  $a_{13}$ ?

Sometimes we will want to consider all **ordered pairs** (2-tuples) where the first element is from  $S$  and the second is from  $T$ . This set of ordered pairs is called the **Cartesian product** and is denoted by  $S \times T$ . That is,

$$S \times T = \{(s, t) \mid s \in S, t \in T\}.$$

For instance,

$$\{a, b\} \times \{1, 2\} = \{(a, 1), (a, 2), (b, 1), (b, 2)\}.$$

Note that *each* element of  $S$  is paired with each element of  $T$ .

Similarly, one may define the  $n$ -fold cartesian product  $S_1 \times S_2 \times \dots \times S_n$ .

## Relations

Many statements in mathematics relate two objects, as in  $4 > 2$  and  $x \in S$ . We will be introducing some new relations (e.g., rank order in Section 0.3), so we need a language for discussing general properties which relations may have, notably reflexivity, symmetry, and transitivity.

A relation like  $>$  or  $\in$  takes two objects and produces a *statement*. Contrast this with  $+$ , which produces a number. Intuitively, any property which can be true or false for pairs of objects is a relation.

We now give a formal definition of relation which may seem odd at first, but it provides the precision we need by placing relations within a language we've already introduced, the language of sets.

---

**Definition 1.** A **relation** on sets  $S$  and  $T$  is any subset of the Cartesian product  $S \times T$ .

---

### EXAMPLE 1

The “larger than” relation for real numbers is the set

$$L = \{(x, y) \mid x, y \in R, x > y\}.$$

Here  $S$  and  $T$  both equal  $R$ . To say that  $(x, y) \in L$  is just a set-theoretic way to say that  $x > y$ . Similarly,  $(x, y) \notin L$  uses sets to say  $x \not> y$ . ■

### EXAMPLE 2

Let  $S$  and  $T$  both be  $N^+$ . Then the relation of **divisibility** is defined as the set

$$D = \{(s, t) : s, t \in N^+, s|t\}, \quad (3)$$

where the vertical bar in  $s|t$  denotes that  $s$  divides  $t$  with a remainder of zero (i.e.,  $t/s$  is an integer). So  $(s, t) \in D$  is just a formal way to say that  $s$  divides  $t$ . Thus  $(2, 4)$ ,  $(4, 4)$ , and  $(47, 235)$  are in  $D$  but  $(17, 52)$  is not. (Note that the membership statement “ $s, t \in N^+$ ” does *not* require that  $s \neq t$ , so  $(4, 4)$  is indeed in  $D$ . Also note the use of colon for “such that” in Eq. (3), since the vertical bar has been used for another purpose.) ■

If  $S$  and  $T$  are finite, any relation on them can be depicted as a directed graph. This idea is taken up in [26] and again in Chapter 3.

We now introduce the three most important properties of relations. We refer to an arbitrary relation (that is, a set of ordered pairs) as  $\mathcal{R}$ , for Relation, but in script since we use  $R$  for the real numbers.

*Reflexivity.* A relation  $\mathcal{R}$  on  $S \times S$  is **reflexive** if  $(x, x) \in \mathcal{R}$  for every  $x \in S$ . Thus equality is reflexive (since  $x = x$  for every  $x$ ), as is  $\leq$  (since every  $x \leq x$ ). However,  $<$  is not reflexive (since  $x < x$  is not always true; in fact, it's never true;  $<$  is **antireflexive**). Set inclusion is also reflexive since  $A \subset A$ .

*Symmetry.* A relation  $\mathcal{R}$  on  $S \times S$  is **symmetric** if  $(x, y) \in \mathcal{R}$  implies that  $(y, x) \in \mathcal{R}$ . Of all the relations we have mentioned so far, only equality is symmetric. For instance  $x \leq y$  and  $y \leq x$  are sometimes both true (when  $x = y$ ) but more often not, so as a relation  $\leq$  is not symmetric.

*Transitivity.* A relation  $\mathcal{R}$  is **transitive** if, whenever  $(x, y) \in \mathcal{R}$  and  $(y, z) \in \mathcal{R}$ , then  $(x, z) \in \mathcal{R}$ . The relations

$$= \quad < \quad > \quad \leq \quad \geq \quad \subset$$

are all transitive. For instance,  $<$  is transitive because whenever  $x < y$  and  $y < z$ , then  $x < z$ .

When a relation is reflexive, symmetric, and transitive, it is called an **equivalence relation**. Equality is the only equivalence relation among the relations discussed so far, but others are introduced in the problems [22–24]. When  $\mathcal{R}$  is an equivalence relation on  $S \times S$ , then for each  $s \in S$ , the set  $\{s' \mid (s, s') \in \mathcal{R}\}$  is called its **equivalence class** [25].

Relations as we have defined them are more precisely **binary** relations, since they relate pairs. There are also ternary, quaternary, etc., relations. Most of the concepts of mathematics could be restated as relations in this broader sense, but we will have no need to do so. For instance,  $z = x + y$  is a statement relating three numbers; therefore addition could be defined as a ternary relation.

## Functions

A **function** is a rule or process that associates each element of one set (the **domain** or **input space**) with a unique element of another (the **codomain** or **output space**). If one emphasizes the dynamic process of getting from inputs to outputs, one often calls the function a **mapping**. If one is more interested in a static description, say through a formula, the word function is more common.

Take the familiar squaring function,  $y = x^2$ . One may think about how it maps 3 to 9, and  $-2$  to 4. Or one may merely think about the rule  $y = x^2$ . In any event, for each  $x$  there is *one and only one*  $y$  associated with it. If the function is named  $f$ , then  $f(a)$  is the name of the unique element of the codomain associated with domain element  $a$ . For example, if  $f(x) = x^2$ , then  $f(3) = 9$ .

Often functions can be defined by traditional one-line formulas, like  $f(x) = x^2$ , but in discrete math often they can't. For instance, consider the parity function, defined on the integers by

$$p(n) = \begin{cases} 0 & \text{if } n \text{ is even,} \\ 1 & \text{if } n \text{ is odd.} \end{cases} \tag{4}$$

Whether or not you consider this a formula, it shares an important feature with traditional formulas. The function is defined using an **input variable**, which could be any letter. (However, certain letters are traditional such as  $x$  for real numbers,  $n$  for integers.) Since the input variable is just a placeholder, allowing us to give a general rule for evaluating the function, it could just as well be called a “dummy variable”; see Sections 0.4 and 1.4 for situations where this phrase is indeed used.

If we write  $y = f(x)$  or  $k = p(n)$ , then  $y$  and  $k$  are **output variables**. Input and output variables are also called, respectively, **independent variables** and **dependent variables**. These names emphasize the fact that the input varies freely, but once you fix an input, the output is completely determined.

When a function has a formula, it actually has many. Consider the functions defined by  $f(x) = x^2$  and  $g(x) = (x+1)^2 - 2x - 1$ . These are the *same* function,

because for each input  $x$ ,  $f(x) = g(x)$ . In other words, it is the *correspondence*, not the formula, that counts. Functions  $f$  and  $g$  are the same because they pair the same inputs with the same outputs, e.g., 3 with 9.

When using functions, we substitute specific values for the input variables, to get, say,  $p(3) = 1$  and  $f(b+1) = (b+1)^2$ . We will call values substituted for input variables **arguments**. Notice that an argument need not be a specific value, like 3, but it does have to be something we want to compute with, rather than a placeholder. Many mathematicians use argument as a synonym for input variable, but the distinction between a placeholder used in a definition and a value used for computation is important, so we need different words. Furthermore, argument as defined here has become standard in computer science, and we will use it this way when discussing the grammar of our algorithmic language in Section 1.4.

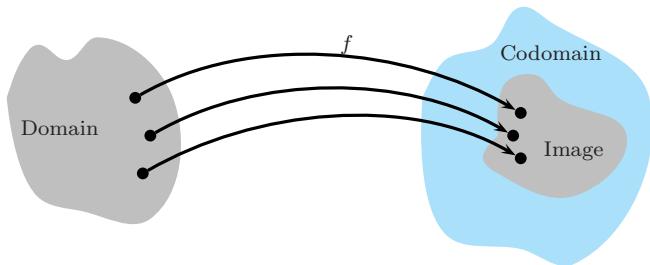
**Onto, One-to-one, and Invertible Functions.** Let us return to domain and codomain. Any function can be, technically speaking, many different functions, depending on the choice of domain and codomain. For instance, there is the squaring function with domain = codomain =  $R$ . (Recall that we use  $R$  for the real numbers,  $R_{\geq 0}$  for the nonnegative reals, and  $N$  for the nonnegative integers.) There is the squaring function with domain =  $R$  and codomain =  $R_{\geq 0}$ . But there is no squaring function with domain =  $R$  and codomain =  $N$ . Why not?

Let  $D$  be the domain of  $f$ . The set

$$f(D) = \{f(x) \mid x \in D\}$$

is called the **image** of  $f$ . The point  $f(a)$  is called the image of  $a$ . The image of  $f$  must be contained in the codomain, but it need not equal it.

A function is **onto** if its image equals its codomain. That is,  $f$  is onto if every element of the codomain is “hit” (or *mapped to*) by some element of the domain. See Fig. 0.2.



## FIGURE 0.2

Function terminology. The image is the set of points in the codomain mapped into by the function  $f$  operating on each point in the domain.

## **EXAMPLE 3**

Consider the squaring function. If its domain and codomain are  $R$ , it is not onto. If its domain is either  $R$  or  $R_{\geq 0}$  and its codomain is  $R_{\geq 0}$ , then it is onto. If its domain and codomain are  $N$ , then it is not onto. ■

A function is **one-to-one** if distinct domain points always map to distinct images.

## EXAMPLE 4

Consider the squaring function again. If its domain and codomain are  $R$ , it is not one-to-one. (In fact, with one exception, it is two-to-one; what's the exception?) If its domain is  $R_{\geq 0}$  and its codomain is  $R_{\geq 0}$ , then it is one-to-one. If its domain and codomain are  $N$ , then it is one-to-one. ■

A function is a **one-to-one correspondence** or **bijection** if it is one-to-one and onto. The squaring function from  $R_{\geq 0}$  to  $R_{\geq 0}$  is a bijection.

If  $f$  is a bijection, we say that it is **invertible** and we may define the **inverse function**  $f^{-1}$  on the codomain  $f$  as follows:

$$f^{-1}(y) = \text{the unique } x \text{ in the domain of } f \text{ such that } f(x) = y.$$

In short, if  $f$  is invertible, and we are told  $f(x) = c$ , we can solve uniquely for  $x$  by working backwards, that is, by computing  $f^{-1}(c)$ .

Don't be misled by the  $-1$ ;  $f^{-1}$  is not the arithmetic reciprocal of  $f$ :

$$f^{-1}(x) \neq \frac{1}{f(x)}. \quad (\text{See [38].})$$

You may have gotten the impression that the main thing one does with functions is fiddle with their domain and codomain. Nothing could be further from the truth. We fiddled to illustrate properties like one-to-one more clearly. In practice, the person who defines a function will have one domain in mind, and will then fix the codomain as either the image of that domain or some convenient superset of the image, and that will be that. In many cases, the domain and codomain won't even be stated explicitly, because the reader will know or it doesn't matter.

## Problems: Section 0.1

1.
  - 1) Display all subsets of  $S = \{x, y, z\}$ .
  - 2) Display the powerset of the powerset of  $\{a\}$ .
3.
  - 2) Describe in English each of the following sets:
    - a)  $\{m \mid m = 2n \text{ for some } n \in Z\}$ .
    - b)  $\{s \mid s = n^2 \text{ for some } n \in N^+\}$ .
    - c)  $\{q \mid q = mn \text{ for some } m, n \in N, m, n > 1\}$ .
  4.
    - 2) Use set builder notation to describe the set of
      - a) all odd numbers between 100 and 200;
      - b) all points on the graph of the function  $y = x^2$ .
    5.
      - 2) Let  $N$ ,  $N^+$ ,  $Z$ , and  $Q$  be as defined in the text. Let  $E$  be the set of even integers and  $F$  the set of integers divisible by 5. (Note that  $0, -5 \in F$ .) Use set operators to name the set of
        - a) positive integers divisible by 5;
        - b) even integers divisible by 5;
        - c) nonintegral rational numbers;
      6.
        - 1) Let  $S = \{1, 3, 5, 7, 9\}$ ,  $T = \{3, 4, 5, 6, 7, 8\}$ , and  $W = \{1, 2, 3, 4\}$ . Compute:
          - a)  $S \cup T$ .
          - b)  $S \cap T$ .
          - c)  $S - T$ .
          - d) the complement  $S$ , assuming the universe is the set of all integers from 1 to 10.
          - e)  $(W \cap S) \cup T$ .
          - f)  $W \cap (S \cup T)$ .

7. ⟨1⟩ (a)–(f) Draw Venn diagrams to illustrate each of the expressions in [6].
8. ⟨1⟩ List all the elements of  $\{b, c, d\} \times \{e, o\}$ .
9. ⟨2⟩ Is the Cartesian product commutative? That is, is  $S \times T$  the same as  $T \times S$ ? Why?
10. ⟨2⟩ If  $A \cap B = \emptyset$ , what does that tell you about  $|A|$ ,  $|B|$ , and  $|A \cup B|$ ?
11. ⟨2⟩ If  $A \subset B$ , what does that tell you about  $A \cap B$  and  $A \cup B$ ?
12. ⟨2⟩ If  $A \cap B = \emptyset$  and  $B \cap C = \emptyset$ , what does that tell you about  $A \cap C$ ?
13. ⟨2⟩ If  $A \cup B = D$ , must  $D - B = A$ ? If not, must  $D - B \subset A$ ?
14. ⟨2⟩ If  $A \subset S$  and  $B \subset S$ , what can you say about  $A \cup B$ ?
15. ⟨2⟩ If  $|A| = m$ ,  $|B| = n$ , and  $m \geq n$ , what is the least  $|A \cup B|$  can be? The most?
16. ⟨2⟩ If  $|A| = m$ ,  $|B| = n$ , and  $m \geq n$ , what is the least  $|A \cap B|$  can be? The most?
17. ⟨3⟩ Find the cardinality of the set

$$S = \{p/q \mid p, q \in N^+, p, q \leq 6\}.$$

*Note:* This set builder notation produces both  $2/4$  and  $1/2$ , which are the same number; remember that in a set each item is counted only once no matter how many times it is listed.

18. ⟨2⟩ Just as there are operations on sets, there are operations on lists. When we combine two lists, we need to have an ordering of the result, so union isn't good enough. For lists  $A = (a_1, \dots, a_m)$  and  $B = (b_1, \dots, b_n)$ , the **concatenation**  $A \& B$  is defined to be  $(a_1, \dots, a_m, b_1, \dots, b_n)$ . That is,  $B$  is put in order after  $A$ .
- a) Does  $A \& B = B \& A$ ? That is, is concatenation commutative?
- b) Does  $(A \& B) \& C = A \& (B \& C)$ ? That is, is concatenation associative?
19. ⟨2⟩ Express the following relations in set builder notation:
- a) One number is less than or equal to another.
- b) One integer is a factor of another.
- c) Two integers are unequal.
- d) One set is a subset of another.
20. ⟨2⟩ (a)–(d) For each of the relations in [19], determine whether it is reflexive; symmetric; transitive.

21. ⟨2⟩ We define the **successor relation** on  $N \times N$  to be the set
- $$\{(m, n) \mid m = n + 1\}.$$
- Give two pairs that are in the successor relation and two pairs that are not.
22. ⟨3⟩ Consider the relation between sets of having the same cardinality. Show that this is an equivalence relation. You may assume that all sets considered are finite.
23. ⟨3⟩ Let  $Z$  be the integers. Define the **parity relation** on  $Z \times Z$  by:  $p$  is related to  $q$  if and only if  $2 \mid (p - q)$  (i.e., 2 divides  $p - q$ ). Show that parity is an equivalence relation. What are its equivalence classes?
24. ⟨3⟩ We define the relation  $=_m$  (read “equal mod  $m$ ”) on  $Z \times Z$  to be the set
- $$\{(p, q) : m \mid (p - q)\}.$$
- a) Give two pairs which are in the relation  $=_4$  and two pairs which are not.
- b) Show that  $=_m$  is an equivalence relation.
25. ⟨3⟩ Let  $\mathcal{R}$  be an equivalence relation on  $S \times S$  and let  $[s]$  be the equivalence class of  $s$ . Prove:
- if  $[s] \cap [t] \neq \emptyset$ , then  $[s] = [t]$ . (5)
- This proves that the equivalences classes **partition**  $S$ , i.e., each  $u \in S$  is in exactly one equivalence class. *Hint:* The hypothesis of (5) is that there exists  $u$  such that  $(s, u), (t, u) \in \mathcal{R}$ . The conclusion is that  $(s, v) \in \mathcal{R}$  if and only if  $(t, v) \in \mathcal{R}$ .
26. ⟨3⟩ When  $S$  and  $T$  are finite, there is a useful visual representation of a relation on  $S \times T$ . Draw a dot for each element of  $S \cup T$ , and for each  $(s, t) \in R$  draw an arrow from  $s$  to  $t$ . This representation as a *directed graph* (see the Prologue) is especially useful if  $S = T$ .
- a) Let  $S = T = \{0, 1, 2, \dots, 8\}$ . Draw a graph on  $S$  of the relation  $=_4$  (see [24]). The striking nature of this graph is due to the fact that  $=_4$  is an equivalence relation.
- b) Let  $S = T = \{\text{your maternal grandfather and all his descendants}\}$ . Draw the graph of the parent relation on  $S$ , where  $(p, c)$  is in the relation if  $p$  is a parent of  $c$ . What you get is called a “family tree” by sociologists, a “rooted, directed tree” by mathematicians, and just a “tree” by computer scientists.

27. ⟨3⟩ The **reverse** of a relation  $\mathcal{R}$  on  $S \times T$  is a relation  $\mathcal{R}'$  on  $T \times S$  defined by

$$(t, s) \in \mathcal{R}' \iff (s, t) \in \mathcal{R},$$

where the double arrow should be read “if and only if”. Describe the reverses of

- a)  $>$
- b)  $\subset$
- c)  $\geq$
- d) the successor relation from [21].

28. ⟨3⟩ If  $\mathcal{R}$  is symmetric, is  $\mathcal{R}$ -reverse symmetric? Answer the same question for reflexive relations and transitive relations.

29. ⟨3⟩ The **transitive closure** of a relation  $\mathcal{R}$  on  $S \times S$  is another relation on  $S \times S$  called  $\text{Tr}(\mathcal{R})$  such that  $(s, t) \in \text{Tr}(\mathcal{R})$  if and only if there exists a sequence  $s_1, s_2, s_3, \dots, s_n$ , where  $s = s_1$ ,  $t = s_n$ , and  $(s_i, s_{i+1}) \in \mathcal{R}$  for each  $i$ . In other words, if there is a sequence of relations in  $\mathcal{R}$  connecting  $s$  and  $t$ , then they are directly related in  $\text{Tr}(\mathcal{R})$ .

- a) What is the transitive closure of the successor relation from [21]?

- b) What is the transitive closure of the  $>$  relation?

30. ⟨2⟩ Let  $S$  be the set of all people. The “parent” relation is just what you think:  $(a, b)$  is in the relation if and only if  $a$  is the parent of  $b$ .

- a) What is the reverse of the parent relation?

- b) What is the transitive closure (see [29]) of the parent relation?

31. ⟨2⟩ At a residential college, each student is assigned to an available dorm room. Which of the special properties of functions is appropriate here? Onto? One-to-one?

32. ⟨2⟩ The function  $\text{SSN}(p)$  gives the US Social Security number of person  $p$ . The domain is all people required to have one. The codomain is all 9-digit numbers of the right form, whatever that is. This function better be one-to-one and not onto. Why?

33. ⟨3⟩ Which of the following functions (each followed by its domain and codomain) are one-to-one, onto, and/or bijective?

- a)  $f(n) = 2n$ ,  $N$ ,  $N$
- b)  $g(n) = n + 1$ ,  $N$ ,  $N$
- c)  $h(n) = n + 1$ ,  $Z$ ,  $Z$
- d)  $p(m, n) = 2^m 5^n$ ,  $N \times N$ ,  $N$
- e)  $q(m, n) = 2^m 6^n$ ,  $N \times N$ ,  $N$

34. ⟨3⟩ Are the following statements true or false? Give reasons. Assume all sets are finite.

- a) If a function is bijective, its domain and codomain are the same set.
- b) If a function is bijective, its domain and codomain have the same cardinality.
- c) If a function is one-to-one, its domain and codomain have the same cardinality.
- d) If a function is one-to-one, its domain and image have the same cardinality.
- e) If a function is onto, its domain and codomain have the same cardinality.
- f) If a function is neither one-to-one nor onto, its domain and image do not have the same cardinality.
- g) If a function is neither one-to-one nor onto, its domain and codomain do not have the same cardinality.

35. ⟨3⟩ Let  $n \bmod k$  be the remainder when integer  $n$  is divided by integer  $k$ . (This mod function is discussed at length in Section 0.2.) Define

$$f(n) = (n \bmod 3, n \bmod 5).$$

Let the domain of  $f$  be the integers from 0 to 14. Show that  $f$  is one-to-one. Try to show this without using a brute force check.

36. ⟨2⟩ Explain why every infinite sequence  $a_1, a_2, \dots$  may be viewed as a function with domain  $N^+$ . If, as often happens, we wish to start our sequence with  $a_0$ , what is the natural domain now?

37. ⟨2⟩ Are the following functions (with domain = codomain =  $R$ ) the same or not? Explain.

- i)  $f$  defined by  $f(x) = 2x + 3$ .
- ii)  $g$  defined by  $g(z) = 2z + 3$ .
- iii)  $h$  defined by  $h(w) = 2(w+2) - 1$ .

38. ⟨3⟩ Let  $f(x) = 2x$  and  $g(x) = x^3 + 1$ , both with  $R$  for domain and codomain.

- a) Argue with pictures that both  $f$  and  $g$  have inverses.
- b) Find  $f^{-1}(x)$  and show that  $f^{-1}(x) \neq 1/f(x)$ .
- c) Find  $g^{-1}(5)$  and show that  $g^{-1}(x) \neq 1/g(x)$ .

39. ⟨3⟩ Explain why, if  $f$  is merely one-to-one, we may still define an inverse  $f^{-1}$  on the *image* of  $f$ . This is called a **partial inverse**, or, if one is willing to redefine the codomain to be the image, it is simply called the inverse.

40. **(3)** Let  $f$  have domain  $D$  and codomain  $C$ . For any  $D' \subset D$  and  $C' \subset C$  define the **image** of  $D'$ , denoted  $f(D')$ , and the **preimage** of  $C'$ , denoted  $f^{-1}(C')$  by

$$f(D') = \{f(x) \mid x \in D'\}$$
$$f^{-1}(C') = \{x \mid f(x) \in C'\}.$$

- a) Describe these concepts with pictures and words.  
b) Let  $f$  be the squaring function with  $D = C = \mathbb{R}$ , and let  $S = \{x \mid 2 \leq x \leq 4\}$ . Determine  $f(S)$  and  $f^{-1}(S)$ .

*Note:* For preimage we use the notation  $f^{-1}$  even though an inverse function need not exist. We say

that the *set* inverse exists even when the *point* inverse does not.

41. **(3)** To study functions as objects, it is often best to represent them as sets, just as we did for relations. That is, the function  $f$  with domain  $D$  and codomain  $C$  can be identified with the subset

$$\{(x, f(x)) \mid x \in D\}.$$

of  $D \times C$ .

- a) Use this notation to describe the squaring function with  $D = C = \mathbb{R}$ .  
b) Explain why all functions are relations. (However, the issues of interest when discussing functions are different from those when discussing general relations.)

## 0.2 Some Important Functions

---

In this section we discuss a variety of functions which we'll use — some of them over and over again — later in this book. Some are much more important in discrete than in continuous mathematics; others are important in both realms.

### Basic Functions

In defining various functions, we will use  $x$  to refer to an element of the domain of the function and  $y$  to an element of the codomain.

#### 1. Absolute value function.

Domain:  $\mathbb{R}$ , the set of real numbers.

Codomain:  $\mathbb{R}_{\geq 0}$ , the nonnegative reals.

Definition:

$$y = \begin{cases} x & \text{if } x \geq 0. \\ -x & \text{if } x < 0. \end{cases}$$

Notation:  $|x|$ .

In effect, the absolute value function takes a real number  $x$  and leaves it unchanged if it is nonnegative but strips off the sign if it is negative. This very simple function is valuable throughout mathematics whenever you wish to ensure that a quantity cannot be negative. Unfortunately, the notation for this function is the same as that for the cardinality function defined in Section 0.1. But there should be no confusion because the argument of the cardinality function must be a set and that of the absolute value function must be a real number.

## 2. Floor function.

Domain:  $R$ .

Codomain: The set of all integers  $Z$ .

Definition:

$y =$  the largest integer not greater than  $x$ .

Notation:  $\lfloor x \rfloor$ .

In discrete mathematics generally, the most common function domains are  $N$ ,  $N^+$  and  $Z$ . However, when performing operations on integers, we often end up with nonintegral quantities. For example,  $\sqrt{3}$  is not an integer. The floor function and its companion, the ceiling function, considered next, provide us with convenient ways to convert real numbers to integers when the fractional (nonintegral) part of the number isn't of interest to us. Thus  $\lfloor \sqrt{3} \rfloor = 1$  because 1 is the largest integer less than  $\sqrt{3} = 1.732\dots$ . Other examples of the floor function are  $\lfloor -4.67 \rfloor = -5$ ,  $\lfloor 86.739 \rfloor = 86$ ,  $\lfloor .302 \rfloor = 0$ , and  $\lfloor 77 \rfloor = 77$ .

## 3. Ceiling function.

Domain:  $R$ .

Codomain:  $Z$ .

Definition:

$y =$  the smallest integer not less than  $x$ .

Notation:  $\lceil x \rceil$ .

This function is the obvious analog of the floor function. Whereas the floor function finds the integral bottom or “floor” of its argument, the ceiling function finds the integral top or “ceiling” of its argument. Some examples of the ceiling function are  $\lceil -4.67 \rceil = -4$ ,  $\lceil 86.739 \rceil = 87$ ,  $\lceil .302 \rceil = 1$ , and  $\lceil 77 \rceil = 77$ .

From the definitions of the floor and ceiling functions and from our examples you should be able to verify the following [10]:

$$\lceil x \rceil = \begin{cases} \lfloor x \rfloor + 1 & \text{when } x \text{ is not an integer,} \\ \lfloor x \rfloor & \text{when } x \text{ is an integer.} \end{cases} \quad (1)$$

Floors and ceilings occur frequently in discrete mathematics when exact integer counts are needed, but their algebra is often messy. Therefore, we provide many problems about their often unfamiliar properties [10–21].

## 4. Factorial function.

Domain:  $N$ .

Codomain:  $N$ .

Definition: For a positive integer  $n$ , we define the factorial of  $n$ , usually just called  $n$  factorial, as

$$n(n-1)(n-2)(n-3)\cdots 3 \times 2 \times 1.$$

In other words, it is the product of all the integers from 1 to  $n$ . If  $n = 0$ , then by convention, its factorial is defined as 1.

Notation:  $n!$ .

One of the most important properties of a factorial is how fast it grows. Whereas  $0! = 1$ ,  $1! = 1$ ,  $2! = 2$ , and  $3! = 6$ , by the time we get to 10 we have  $10! = 3,628,800$ . And  $1000!$  is a number with more than 2500 digits in it.

Sometimes you start at a number but don't want to multiply all the way down to 1. The **falling factorial function** is

$$x_{(k)} = \underbrace{x(x-1)(x-2)\cdots}_{k \text{ factors}} = x(x-1)(x-2)\cdots(x-k+1). \quad (2)$$

For instance,  $n_{(n)} = n!$ .

Notice that the falling factorial is defined for any  $x \in R$ , since we no longer have to hit 1 on the way down. However,  $k$  must be a nonnegative integer. By convention,  $x_{(0)} = 1$ .

At this point, you can guess the definition of the **rising factorial function** [28], denoted  $x^{(k)}$ . Alas, there are several conflicting notations for factorial functions, so you must pay careful attention to the notation in any book which discusses this topic.

### 5. Mod (or remainder) function.

Domain:  $N \times N^+ = \{(m, n) \mid m \in N, n \in N^+\}$ .

Codomain:  $N$ .

Definition:

$$m - \lfloor m/n \rfloor n,$$

where  $m$  is a nonnegative integer and  $n$  is a positive integer.

Notation:  $m \bmod n$ .

The effect of this function is to compute the integer remainder when  $m$  is divided by  $n$ . Thus

$$8 \bmod 3 = 8 - \lfloor 8/3 \rfloor 3 = 8 - 2 \times 3 = 2;$$

$$57 \bmod 12 = 9;$$

$$7 \bmod 9 = 7.$$

The name of this function is an abbreviation of

$$m \text{ modulo } n,$$

where “modulo” means “with respect to a modulus”, that is, the value of  $m$  with respect to the size (or modulus) of  $n$ , which we define to be the remainder when  $m$  is divided by  $n$ .

Among other things, this example reminds us that the domain of a function need not be a set of numbers. The domain of a function need only be a set, *any* set. For the mod function the domain is a set of two element sequences (2-tuples), that is, ordered pairs of integers.

## Exponentials and Logarithms

Any function of  $x$  of the form  $f(x) = a^x$ , with  $a$  a positive constant, is called an **exponential function**.<sup>†</sup> The most important property of exponential functions is

$$a^{x+y} = a^x a^y,$$

or equivalently in function notation,

$$f(x+y) = f(x)f(y).$$

We say that an exponential function converts addition into multiplication, because a sum in the input gives a product in the output.

Another important property of exponential functions is that, if  $a > 1$ , then  $a^x$  grows faster than any power of  $x$ ,  $x^n$ , no matter how large the constant  $n$ . This means that for  $a > 1$  and for any  $n > 0$ ,

$$\frac{x^n}{a^x} \quad (3)$$

will be as close to zero as you wish if you make  $x$  large enough. You might find it useful to use your hand calculator or computer to compute some values of (3) for a variety of values of  $a$ ,  $n$ , and  $x$ . A useful notation to express the behavior of (3) as  $x$  becomes large is

$$\lim_{x \rightarrow \infty} \frac{x^n}{a^x} = 0,$$

where “lim” is shorthand for “limit”. More generally, we write

$$\lim_{x \rightarrow \infty} f(x) = L$$

to indicate that, as  $x$  becomes arbitrarily large,  $f(x)$  gets arbitrarily close to  $L$ .

Make sure you understand the difference between an exponential function, where the variable  $x$  is the exponent, and a **power function**, where the quantity raised to a power is the variable and the exponent  $n$  is a constant. If we were to write  $z^x$ , it would be ambiguous, but either

$$f(x) = z^x$$

or

$$f(z) = z^x$$

is unambiguous; in the first equation  $x$  is the independent variable and  $z$  is a constant, whereas their roles are reversed in the second equation.

Exponential functions are closely related to logarithms. For more than three centuries after Napier invented logarithms in about 1600, logarithms were an important computational tool. Now the hand calculator has eliminated entirely the computational use of logarithms. However, the **logarithm function** is still very important in mathematics, particularly in the study of algorithms (an important theme throughout this book and the subject of Chapter 1). Let’s review the basic properties of logarithms.

---

<sup>†</sup>Exactly what  $a^x$  means when  $x$  is not an integer, or at least rational, is quite subtle, but fortunately it’s been taught to your calculator.

The logarithm of a positive real number  $x$  with base  $b > 1$  is that power  $y$  to which  $b$  must be raised to get  $x$ . In other words,

$$y = \log_b x \text{ if and only if } b^y = x \quad (4)$$

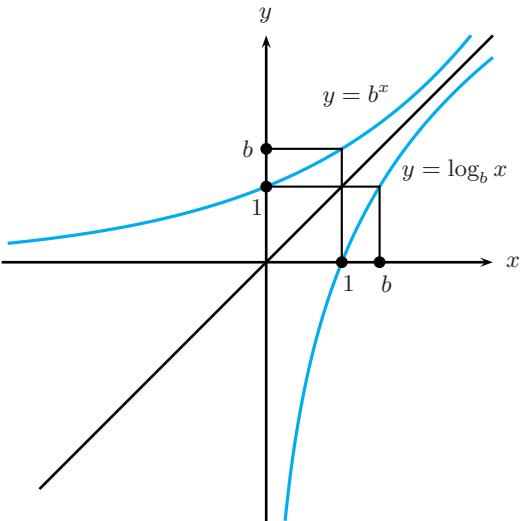
so that, using the terminology of Section 0.1, the logarithm is the inverse function of the exponential. The traditional base for computations is 10. Hence when we write  $\log x$  without a subscript, you may think of the base as 10 for now. However, as we'll soon show, the interesting properties of logarithms are independent of the base. Thus when we leave the base off, that really means that the base doesn't matter. Actually, logarithms base 2 have become quite useful in recent years because computers work in the binary system. Also, many algorithms involve repeated division of problems into two smaller problems. Indeed, on the few occasions when we will wish to use a specific base for a logarithm, it will always be 2. Some books use  $\lg x$  to denote  $\log_2 x$ .

### EXAMPLE 1

Evaluate  $\log_2 64$ ,  $\log_3(1/9)$ , and  $\log 0$ .

**Solution** Since  $2^6 = 64$ , it follows that  $\log_2 64 = 6$ . Since  $3^{-2} = 1/9$ , we have  $\log_3(1/9) = -2$ . Since  $b^y \neq 0$  no matter what base  $b > 1$  and what real number  $y$  you use,  $\log 0$  is *undefined* no matter what the base is. ■

In Fig. 0.3 we show the graph of  $f(x) = b^x$  for a typical  $b > 1$ . We also display the graph of  $f(x) = \log_b x$  for the same value of  $b$ . Note that  $\log_b x$  is positive if and only if  $x > 1$ .



**FIGURE 0.3**

Graphs of the functions  $y = b^x$  and  $y = \log_b x$ .

From display (4) you can see that a point  $(r, s)$  is on the graph of  $y = \log_b x$  if and only if the point  $(s, r)$  is on the graph of  $y = b^x$ , since  $r = \log_b s$  if and only if  $s = b^r$ . Geometrically, the two graphs in Fig. 0.3 are mirror images of each about the  $45^\circ$  line  $y = x$ ; rotating about this line flips the horizontal axis into the vertical

axis and vice versa. Using the terminology of Section 0.1, whenever the graphs of two functions have this property, the functions are inverses of each other (why?).

Thus it follows again that the exponential and logarithmic functions are inverses of each other. That is, either one undoes the other in the sense that

$$b^{\log_b x} = x \quad \text{and} \quad \log_b b^y = y.$$

These identities follow by substituting the left side of (4) into the right and vice versa.

Here are the properties of logarithms which we'll need:

1.  $\log xy = \log x + \log y$  (i.e., logarithms turn multiplication into addition).
2.  $\log x^r = r \log x$  (i.e., logarithms turn powers into multiples).
3.  $\log_c x = \log_b x / \log_b c$  (i.e., changing the base from  $b$  to  $c$  merely changes the logarithm by a constant multiplicative factor, namely,  $1 / \log_b c$ ).
4.  $\lim_{x \rightarrow \infty} \log x = \infty$ , but  $\lim_{x \rightarrow \infty} [(\log x)/x] = 0$ .

The first three properties are explained in traditional treatments of logarithms in high school texts; Property 4 isn't. But it is Property 4 which accounts for the special importance of logarithms in this book.

Property 4 is a statement about the *rate of growth* of the logarithm. Yes, the logarithm does grow; in fact, as indicated by both Fig. 0.4 and Property 4, it goes to infinity as  $x$  gets arbitrarily large. But — and this is the key point — *the logarithm gets large very slowly*. For instance, it grows so slowly in comparison to the growth of  $x$  that (see Property 4 again) the ratio of  $\log x$  to  $x$  decreases to zero. Even if we beef up the logarithm by raising it to the thousandth power, in the long run the result is negligible compared to  $x$ . That is,

$$\lim_{x \rightarrow \infty} \left[ \frac{(\log x)^m}{x} \right] = 0 \tag{5}$$

for any positive  $m$  whatsoever. This too is illustrated in Fig. 0.4 with  $m = 3$ . By the way, a common alternative notation for  $(\log x)^m$  is  $\log^m x$ .

Property 4 is also independent of the base. Why? If  $\log x$  goes to infinity, then so does  $k \log x$  for any constant  $k$ . Likewise, if  $(\log x)/x$  goes to 0, then  $(k \log x)/x$  does too. But in both cases replacing  $\log x$  by  $k \log x$  has the same effect as changing the base (why?).

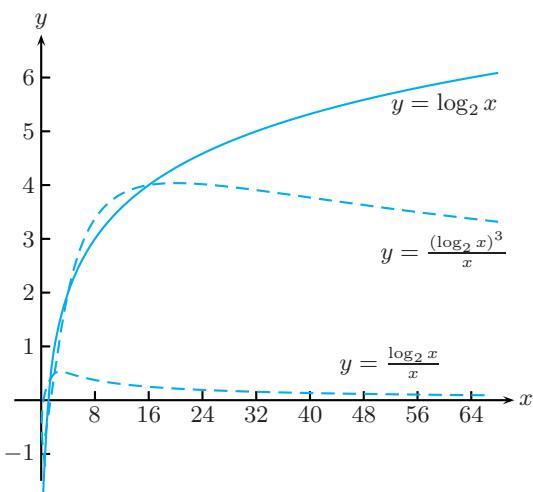
We won't prove Property 4, but we'll give it informal justification in the problems [44].

**EXAMPLE 2** Find  $\log_2 3$  using a calculator.

**Solution** Few calculators have a button for  $\log_2$ . So use Property 3, which tells us that

$$\log_2 3 = \frac{\log_{10} 3}{\log_{10} 2},$$

which, approximately, is  $.477/.301 = 1.585$ . Or use the  $\ln$  button, which gives logs base  $e = 2.71828\dots$ , a number useful throughout mathematics but especially in



**FIGURE 0.4**

The relative growth rates of three functions.

continuous mathematics. The answer is the same, but it is computed as

$$\log_2 3 = \frac{\ln 3}{\ln 2} \approx \frac{1.0986}{.6931} = 1.585. \blacksquare$$

## Polynomials

A **polynomial** is a function of the form

$$y = P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0, \quad [a_n \neq 0] \quad (6)$$

whose domain and codomain are usually  $R$ . We restrict  $a_n$  to be nonzero to ensure that the term in  $x^n$  actually appears. In this book we will sometimes restrict the domain to be  $N$  or  $N^+$ . The nonnegative integer  $n$  is called the **degree** of the polynomial and the members of the sequence

$$(a_n, a_{n-1}, a_{n-2}, \dots, a_1, a_0)$$

are the **coefficients** of the polynomial. Normally the coefficients are also elements of the domain of the function. Sometimes, however, the coefficients may be restricted to be integers even when  $x$  can be any real number.

From Eq. (6) the simplest case of a polynomial is when  $n = 0$ , in which case the polynomial is constant, with value  $a_0$ . For  $n = 1$  a polynomial has the familiar form

$$y = P_1(x) = a_1 x + a_0. \quad (7)$$

It is called a **linear function** and the graph is a straight line. Likewise, when the degree of the polynomial is  $n = 2$ , we have a **quadratic polynomial** (or quadratic function, or just quadratic) and the graph is a parabola.

Like all functions, polynomials can be combined, e.g., added, subtracted and multiplied. For polynomials, the result in these three cases is always another poly-

nomial. Addition and subtraction are accomplished by combining like terms. Multiplication is more complicated, and as we will have occasional use for it, we discuss it in more detail.

Suppose we have two polynomials  $P_n(x)$  and  $Q_m(x)$  whose degrees,  $n$  and  $m$ , could be different. Using the notation in Eq. (6), we write these two polynomials as

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

and

$$Q_m(x) = b_m x^m + b_{m-1} x^{m-1} + \cdots + b_1 x + b_0.$$

To find the product of  $P_n(x)$  and  $Q_m(x)$  we must compute

$$(a_n x^n + \cdots + a_1 x + a_0)(b_m x^m + \cdots + b_1 x + b_0) \quad (8)$$

term by term and combine terms which have like powers of  $x$  (as we'll show in Example 3). The notation we've introduced thus far is insufficient for writing (8) in a more convenient form. In Section 0.4 we'll introduce summation notation and then return to (8) to show specifically how to write it more conveniently.

### EXAMPLE 3

Given the two polynomials

$$P_3(x) = x^3 + 4x^2 - 3x - 2 \quad \text{and} \quad Q_2(x) = x^2 + 2x + 1,$$

find their product.

**Solution** We use (8) and multiply term by term to get

$$\begin{aligned} & (x^3 + 4x^2 - 3x - 2)(x^2 + 2x + 1) \\ &= x^3(x^2 + 2x + 1) + 4x^2(x^2 + 2x + 1) - 3x(x^2 + 2x + 1) - 2(x^2 + 2x + 1) \\ & \qquad \qquad \qquad [\text{By the distributive law}] \\ &= (x^5 + 2x^4 + x^3) + (4x^4 + 8x^3 + 4x^2) - (3x^3 + 6x^2 + 3x) - (2x^2 + 4x + 2) \\ &= x^5 + 6x^4 + 6x^3 - 4x^2 - 7x - 2. \quad \blacksquare \end{aligned}$$

You need not become very proficient at polynomial arithmetic — let the CAS (Computer Algebra System) on your calculator or computer do the harder problems. But it is important that you *understand* what it means to perform arithmetic on two polynomials.

## Problems: Section 0.2

1. **(1)** Give the values of

a)  $|4.73|$     b)  $|-2.531|$ .

2. **(2)** Graph

a)  $y = |x| + 4$     b)  $y = ||x| - 4|$ .

3. **(2)** An important relationship involving the absolute value function is the **triangle inequality**:

$$|a + b| \leq |a| + |b|,$$

where  $a, b \in R$ . Prove this by breaking it into cases using the signs of  $a$  and  $b$ . The first case is when

$a$  and  $b$  are both nonnegative.

Note: A generalization of this inequality, proved in [2, Section 2.3], is

$$|a_1 + a_2 + \dots + a_n| \leq |a_1| + |a_2| + \dots + |a_n|.$$

In another generalization,  $|a|$ ,  $|b|$  and  $|a+b|$  represent the sides of a triangle, hence the name. See [17, Supplement, Chapter 2].

4.  $\langle 1 \rangle$  Give the values of

a)  $\lfloor 3.99 \rfloor$       b)  $\lceil -6.21 \rceil$   
c)  $\lceil \lfloor -7.31 \rfloor - 6.43 \rceil$ .

5.  $\langle 1 \rangle$  Graph

a)  $f(x) = \lfloor x \rfloor$       b)  $f(x) = \lceil x \rceil$ .

6.  $\langle 2 \rangle$  Graph

a)  $f(x) = \lceil \lfloor x \rfloor \rceil$       b)  $g(x) = \lfloor \lceil x \rceil \rfloor$ .

7.  $\langle 1 \rangle$  Give the solution set for

a)  $\lfloor x \rfloor = 0$       b)  $\lceil 2x \rceil = 3$ .

8.  $\langle 2 \rangle$  How many school busses are necessary to transport  $c$  children on a field trip if each bus holds a maximum of  $m$  children? (This is a famous problem used on national assessments to see if people understand what makes a reasonable answer. Usually numerical values are used.)

9.  $\langle 1 \rangle$  Explain in words why  $|x||y| = |xy|$ .

10.  $\langle 2 \rangle$  Verify Eq. (1), which describes ceiling in terms of floor. (You just have to talk through both cases. For instance, when  $x$  is an integer, what are the values of the two expressions claimed to be equal? When  $x$  is nonintegral, we may write  $x = n + a$  where  $n$  is an integer and  $0 < a < 1$ . In terms of  $n$ , what are the values of the two expressions claimed to be equal?)

11.  $\langle 2 \rangle$  The parity function, Eq. (4) in Section 0.1, has a one-line formula after all, if we allow floors and ceilings. Come up with such a formula for it.

12.  $\langle 3 \rangle$  Suppose  $k \in N$  and  $x > 0$ .

- a) Show that if  $x \leq 2^k$ , then  $\lceil \log_2 x \rceil \leq k$ .  
b) Show further that  $2^{k-1} < x \leq 2^k$  if and only if  $k = \lceil \log_2 x \rceil$ .  
c) Find a formula for  $k$  in terms of  $x$ , similar to the formula in b), if  $2^{k-1} \leq x < 2^k$ .

13.  $\langle 3 \rangle$  If  $x > 2$ , there is a unique integer  $k$  such that  $3^{k-1} + 2 < x \leq 3^k + 2$ . Find a formula for  $k$  in terms of  $x$  (using floor, ceiling and/or logs to some base).

14.  $\langle 3 \rangle$  For each of the following equalities, state for which real numbers it is true (no proof needed).

- a)  $\lfloor x \rfloor = \lceil x \rceil$       b)  $\lceil x \rceil = \lfloor x \rfloor + 1$   
c)  $\lfloor x + 1 \rfloor = \lceil x \rceil + 1$       d)  $\lceil -x \rceil = -\lfloor x \rfloor$   
e)  $\lceil -x \rceil = -\lceil x \rceil$       f)  $\lceil 2x \rceil = 2\lfloor x \rfloor$   
g)  $\lceil 2x \rceil = 2\lfloor x \rfloor$       h)  $\lceil \lfloor x \rceil \rceil = \lceil x \rceil$   
i)  $\lfloor x/n \rfloor = x/n$ ,  $n$  an integer in parts i, j, k  
j)  $\lfloor x/n \rfloor = \lfloor \lfloor x \rceil/n \rfloor$       k)  $\lfloor x/n \rfloor = \lceil x \rceil/n$ .

15.  $\langle 3 \rangle$  In each of the following problems, two expressions are given. Name an inequality between the two which is always true, if there is one. If the inequality is not strict (e.g.,  $\leq$  not  $<$ ), give an example where the two expressions are equal and an example where they aren't equal. If no inequality is always true between the expressions, give an example where the first is greater, a second example where they are equal, and a third example where the second is greater.

- a)  $\lfloor x + y \rfloor, \lfloor x \rfloor + \lfloor y \rfloor$       b)  $\lceil x + y \rceil, \lfloor x \rfloor + \lceil y \rceil + 1$   
c)  $\lfloor x - y \rfloor, \lfloor x \rfloor - \lceil y \rceil$       d)  $\lceil x - y \rceil, \lfloor x \rfloor - \lfloor y \rfloor$   
e)  $n\lfloor x/n \rfloor, \lfloor x \rfloor$       f)  $\lceil xy \rceil, \lfloor x \rceil \lfloor y \rceil$   
g)  $\lceil x/y \rceil, \lceil x \rceil/\lceil y \rceil$ .

16.  $\langle 2 \rangle$  If  $x$ ,  $y$ , and  $z$  are real numbers, show that

- a)  $\lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x + y \rfloor \leq \lfloor x \rfloor + \lfloor y \rfloor + 1$ .  
b)  $\lfloor x + y + z \rfloor \leq \lfloor x \rfloor + \lfloor y \rfloor + \lfloor z \rfloor + 2$ .

17.  $\langle 2 \rangle$  The **rounding function**  $R(x)$  rounds any nonnegative number  $x$  with decimal part .5 or greater to the next larger integer. Show that  $R(x) = \lfloor x + .5 \rfloor$ .

18.  $\langle 3 \rangle$  Rounding .5 to the next larger integer is not the only rounding convention that might be used. Some people believe in rounding .5 down and only rounding up decimal parts greater than .5. Call this rounding function  $R'(x)$ . For nonnegative  $x$ ,

- a) express  $R'(x)$  in terms of ceiling or floor;  
b) express  $R'(x)$  in terms of  $R(x)$ . (Hint: Consider  $-x$ .)

19.  $\langle 3 \rangle$  Show that  $f(x) = \lfloor 10x \rfloor / 10$  rounds to the nearest tenth less than or equal to  $x$ .

20.  $\langle 3 \rangle$  Use floor or ceiling to define a function which rounds off to the nearest tenth. State what your function does in the case that  $x$  is exactly halfway between tenths (e.g.,  $x = 23.45$ ).

21.  $\langle 3 \rangle$  Generalize the previous problem to rounding to the nearest  $n$  decimal places.
22.  $\langle 1 \rangle$  Simplify  $(n+1)!/n!$ .
23.  $\langle 4 \rangle$
- a) Compute the values of  $n!$  for  $n = 1, 2, 3, \dots, 10$ .
  - b) Can you infer from these results how many zeros there are at the end of  $20!$ ? How did you get your answer?
  - c) Can you generalize the result of b) for  $n!$ ? (This is not easy and is not just a simple extension of the result for  $20!$ .)
24.  $\langle 2 \rangle$  Explain why  $\frac{(2n)!}{2^n n!} = 1 \times 3 \times 5 \times \dots \times (2n-1)$ .
25.  $\langle 2 \rangle$  Explain why the falling factorial  $n_{(m)}$  is 0 if  $n \in N$  and  $m > n$ .
26.  $\langle 3 \rangle$  If  $n$  is a positive integer, express  $(n/2)_{(k)}$  as a quotient of products of integers. Generalize.
27.  $\langle 3 \rangle$  Write  $\frac{(2n)!}{2^{2n} n!}$  as a single falling factorial — no additional terms or factors. See [24].
28.  $\langle 2 \rangle$  Give a definition of  $x^{(k)}$  for  $x \in R$  and  $k \in N^+$ . How do you think  $x^{(0)}$  is defined?
29.  $\langle 2 \rangle$  Express  $n!$  using a rising factorial.
30.  $\langle 3 \rangle$  Express  $x^{(k)}$  in terms of  $y_{(j)}$ , where  $y$  is some expression involving  $x$  and  $k$ , and  $j$  is some expression involving  $k$ .
31.  $\langle 2 \rangle$  Which grows faster,  $100^n$  or  $n!$ ? Answer by explaining intuitively what you think  $\lim_{n \rightarrow \infty} 100^n/n!$  is.
32.  $\langle 3 \rangle$  Define  $f$  on the domain  $N^+$  by  $f(n) =$  the remainder when  $n!$  is divided by 1000.
- a) What is the smallest  $n$  such that  $f(n) = 0$ ?
  - b) What is the preimage of 6, that is  $f^{-1}(\{6\})$ ? (Here we use the terminology and notation of [40, Section 0.1], and so we write  $\{6\}$  instead of 6 because we are taking the set inverse of a function which isn't one-to-one. In fact, for singleton sets authors often leave out the braces anyway.) The hard part is to show that you have all members of the set.
33.  $\langle 3 \rangle$  A function  $f$  with domain  $N^+$  is defined implicitly by
- $$f(1) = 4$$
- and  $f(n) = f(n-1) + 3$  for  $n > 1$ .
34.  $\langle 2 \rangle$  Compute
- a)  $10 \bmod 3$
  - b)  $127 \bmod 10$
  - c)  $127 \bmod (10 \bmod 4)$ .
35.  $\langle 3 \rangle$  If  $m \bmod k$  and  $n \bmod k$  are the same, then we say  $m$  is **congruent** to  $n \bmod k$  and we write
- $$m \equiv n \pmod{k}.$$
- (This is the same relation as  $=_k$  in [24, Section 0.1], but now we can give you the standard name and notation.)
- a) Express in set builder notation the solution set to  $4x \equiv 3 \pmod{5}$ .
  - b) If  $k$  is an integer, show that
- $$k^2 \equiv 0 \text{ or } 1 \pmod{4}.$$
- c) Show that  $x^2 \equiv 45 \pmod{100}$  has no integer solution. (*Hint:* Consider the form that the square of a number ending in 5 must have.)
36.  $\langle 2 \rangle$  Give the values of the following logarithms:
- a)  $\log_5 125$
  - b)  $\log_{12} 1$
  - c)  $\log_2 1024$
  - d)  $\log_8 1024$ .
37.  $\langle 2 \rangle$  If  $\log_a x$  is denoted by  $L$ , express the following logarithms in terms of  $L$ . Assume that  $a$  and  $b$  are positive and that  $k$  is any integer.
- a)  $\log_a x^k$
  - b)  $\log_b x^k$
  - c)  $\log_a(1/x^k)$
  - d)  $\log_b(1/x^k)$ .
38.  $\langle 2 \rangle$  How large does  $x$  have to be before  $\log_{10} x$  is
- a) at least 3?
  - b) at least 1000?
39.  $\langle 3 \rangle$  Without using a calculator,
- a) show that  $1 < \log_2 3 < 2$ ;
  - b) find  $\lfloor \log_3 24 \rfloor$ ;
  - c) find  $\lceil \log_4 100 \rceil$ .
40.  $\langle 2 \rangle$  Show that  $\log(x/y) = \log x - \log y$ . Does the base matter?
41.  $\langle 3 \rangle$
- a) Prove that  $\log_a b \log_b c = \log_a c$ . *Hint:* Show that  $a$  raised to both sides are equal. Since  $x \rightarrow a^x$  is a one-to-one function, if  $a^x = a^y$ , then  $x = y$ .
  - b) Use a) to prove Property 3 of logarithms in the text.

42. ⟨3⟩ Prove that  $a^{\log_b c} = c^{\log_b a}$ .

43. ⟨3⟩ Suppose a sequence  $\{a_n\}$  has the property that, for each  $k$ ,  $a_{2^k} = 7^k$ . (This situation actually arose in a famous paper introducing a fast way to do matrix multiplication; see [7, p. 719].) Show that for  $n = 2^k$  we therefore have

$$a_n = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807}.$$

44. ⟨3⟩ Argue that if

$$\lim_{z \rightarrow \infty} \frac{z^m}{b^z} = 0 \text{ for } b > 1$$

(that is, exponentials grow much faster than powers), then

$$\lim_{x \rightarrow \infty} \left[ \frac{(\log_b x)^m}{x} \right] = 0. \quad (9)$$

(that is,  $x$  grows much faster than any power of  $\log x$ ). Thus Property 4 of logarithms, and its generalization, are true if the better known dominance of exponentials over powers is true. Hint: Any  $x > 0$  may be written in the form  $x = b^z$ . Now substitute  $b^z$  for  $x$  in the LHS of (9) and simplify.

45. ⟨3⟩ Have you wondered why we limited the base for logarithms to  $b > 1$ ? The key property we

need in order to define  $\log_b x$  is that the exponential function  $b^x$  is defined for all real  $x$  and provides a one-to-one correspondence between  $R$  and  $R^+$ . Do other  $b$ 's have this property? Clearly,  $b = 1$  and  $b = 0$  do not. Neither does negative  $b$  (unless you're willing to allow complex numbers to be used). This leaves only those  $b$ 's satisfying  $0 < b < 1$  as additional candidates. Show that logarithms *can* be defined by Eq. (4) for  $b$ 's between 0 and 1. What parts of Properties 1–4 remain true for such logarithms?

46. ⟨1⟩ Let

$$P_5(x) = x^5 \quad \text{and} \quad Q_1(x) = x - 4.$$

Compute the sum, difference, and product of these polynomials.

47. ⟨1⟩ Multiply  $(x^3 + x^2 + x + 1)(x - 1)$ . This is worth doing by hand to see what happens.
48. ⟨1⟩ Evaluate  $7x^3 + 4x^2 + 9x + 3$  at  $x = 10$  in your head.
49. ⟨2⟩ If  $f(x)$  is quadratic, show that  $f(x+1) - f(x)$  is linear.
50. ⟨1⟩ Use a CAS to verify the ugly polynomial multiplication in Example 3.

## 0.3 Growth Rates and Order Notation

Consider two sequences with terms

$$a_n = n^2, \quad b_n = n(\log n)^2.$$

How do they grow as  $n \rightarrow \infty$ ? The answer is that they each grow without bound, but  $a_n$  grows much faster, because

$$\text{as } n \rightarrow \infty, \quad \frac{b_n}{a_n} = \frac{n(\log n)^2}{n^2} = \frac{(\log n)^2}{n} \rightarrow 0. \quad (1)$$

This is a special case of Eq. (5) from Section 0.2. Using language defined carefully below, we say  $a_n$  has higher order than  $b_n$  and we write  $\text{Ord}(b_n) < \text{Ord}(a_n)$ .

Why do we care? Because such sequences can represent the amount of work needed to execute (perform) an algorithm as the input grows. The variable  $n$  represents some measure of the data size, say the number of inputs;  $a_n$  could represent the cost or number of computations for one algorithm and  $b_n$  for another. As computers get faster, people use them on larger and larger problems, so if there are a number of competing algorithms to solve the same problem, it is important

to know which algorithm is most efficient as  $n \rightarrow \infty$ . Eq. (1) shows that the  $b_n$  algorithm is much more efficient than the  $a_n$  algorithm in this sense.

In anticipation of the use of sequences for the analysis of algorithms, we will refer to sequences in this section as “execution sequences”. And yes, logarithms do show up in execution sequences, as we shall see repeatedly, starting in Section 1.5.

Here is our plan of attack. We shall show how to associate each execution sequence with a simpler **standard sequence** that grows at essentially the same rate as our execution sequence. We will determine the ranking (relative rates of growth) of the standard sequences. Then given two execution sequences, to determine which is more efficient we just check the ranking of their associated standard sequences.

We start by defining the key ideas needed for this approach. As in Section 0.1 we use  $\{a_n\}$  to denote the sequence whose  $n$ th term is  $a_n$ .

---

**Definition 1.** Let  $\{a_n\}$  and  $\{b_n\}$  be two sequences. We say that

$$a_n = \text{Ord}(b_n),$$

if there is a constant  $c \neq 0$  such that

$$\lim_{n \rightarrow \infty} \frac{a_n}{b_n} = c, \tag{2}$$

or if there are positive constants  $L$  and  $U$  such that

$$L|b_n| \leq |a_n| \leq U|b_n| \tag{3}$$

for all but a finite number of  $n$ , that is, for all  $n >$  some  $n_0$ . Instead of  $a_n = \text{Ord}(b_n)$ , we sometimes write  $\text{Ord}(a_n) = \text{Ord}(b_n)$ .

---

If  $a_n = \text{Ord}(b_n)$ , we say that  $\{a_n\}$  has the **same order** as  $\{b_n\}$  or  $\{a_n\}$  is **order**  $\{b_n\}$ . Note that the “=” is to be read as “has” or “is”. (On the other hand, if we use the alternative notation  $\text{Ord}(a_n) = \text{Ord}(b_n)$ , then “=” can be read in the usual way: the orders are equal. See [23] for another take on this notation.) Also note that order is a property of *sequences*, not individual terms. Thus the notation ought to be  $\{a_n\} = \text{Ord}(\{b_n\})$ , but this is clumsy.

Condition (2) is a special case of (3); we include the former because it is simpler and is sufficient to analyze almost all sequences that arise from studying algorithms. Condition (2) says that, in the limit,  $a_n$  is a multiple of  $b_n$ . Condition (3) does not require  $a_n/b_n$  to have a limit, but only that  $a_n$  be bounded above and below by positive multiples of  $b_n$ , after perhaps at a few “bad” values of  $n$ . For instance, if  $a_3 = 1$  and  $b_3 = 0$ , there is no  $U$  so that  $|a_3| \leq U|b_3|$ ; the definition assures that a few cases like this do not invalidate the general pattern. The use of  $n_0$  is another way to assure this. If  $n = 3$  and  $n = 7$  are the only bad cases, we can say “for  $n \geq 8$ ”; that is, we set  $n_0 = 8$  to get beyond the bad cases. Another phrase that means the same as “for  $n > n_0$ ” is “for sufficiently large  $n$ ”.

Condition (3) has absolute value signs and avoids division so that it works as intended even when the sequences have negative and zero values [20, 23]. Of course, the execution sequences we will work with are always positive.

To see that (2) is a special case of (3), note that if (2) is true, then for  $n$  sufficiently large  $a_n/b_n$  is, say, within 10% of  $c$ , that is, we can pick  $L = .9|c|$  and  $U = 1.1|c|$ . For a pair of sequences that satisfy (3) but not (2), see [4].

The traditional notation for Ord is  $\Theta$ ; one writes  $a_n = \Theta(b_n)$  and says  $a_n$  is big Theta of  $b_n$ .

**EXAMPLE 1** Show that  $3n - 1 = \text{Ord}(n)$ .

**Solution** Since  $1/n$  approaches 0 as  $n$  gets large,

$$\lim_{n \rightarrow \infty} \frac{3n - 1}{n} = \lim_{n \rightarrow \infty} \left( 3 - \frac{1}{n} \right) = 3,$$

so by case (2) of Definition 1, we are done. To use (3) instead, note that

$$2 \cdot n \leq 3n - 1 \leq 3 \cdot n,$$

for  $n \geq 1$ , so we may let  $L = 2$ ,  $U = 3$ . The only exception is  $n = 0$ , that is,  $n = 1$  is sufficiently large. ■

Example 1 is a special case of the following:

**EXAMPLE 2** Let  $P_k(n)$  be any degree  $k$  polynomial. Show that  $P_k(n) = \text{Ord}(n^k)$ .

**Solution** Let the polynomial be

$$P_k(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0.$$

Then

$$\lim_{n \rightarrow \infty} \frac{P_k(n)}{n^k} = \lim_{n \rightarrow \infty} \left( a_k + \frac{a_{k-1}}{n} + \cdots + \frac{a_1}{n^{k-1}} + \frac{a_0}{n^k} \right) \quad (4)$$

$$= \lim_{n \rightarrow \infty} a_k + \lim_{n \rightarrow \infty} \frac{a_{k-1}}{n} + \cdots + \lim_{n \rightarrow \infty} \frac{a_1}{n^{k-1}} + \lim_{n \rightarrow \infty} \frac{a_0}{n^k} \quad (5)$$

$$= a_k,$$

since the limit of the constant  $a_k$  is just  $a_k$  and all the other limits are 0 (why?). Therefore  $P_k(n) = \text{Ord}(n^k)$ , with  $a_k$  playing the role of  $c$  in Eq. (2). ■

In going from Eq. (4) to (5) we have used the fact that the limit of a sum can be decomposed into the limits of its summands. We hope that this seems obvious, but in fact limits are subtle, and not everything that seems obvious about them is true! For more justification of this sum rule and some other limit theorems, see the Appendix. A full treatment of limits usually waits for a course in advanced calculus or real analysis.

Often an execution sequence is not a polynomial but rather a sum of different types of terms as in

$$\{n^3 + 2n^2 \log n\}.$$

How do we determine the order of such sequences? To answer this question, we need some additional definitions and theorems.

---

**Definition 2.** If  $\{a_n\}$  and  $\{b_n\}$  are two sequences such that

$$\lim_{n \rightarrow \infty} \frac{a_n}{b_n} = 0,$$

then we say that  $\{b_n\}$  has a **higher order** (or order ranking) than  $\{a_n\}$  and we write

$$\text{Ord}(a_n) < \text{Ord}(b_n).$$

---

An alternative, traditional notation for  $\text{Ord}(a_n) < \text{Ord}(b_n)$  is

$$a_n = o(b_n), \quad (6)$$

or in words, “ $a_n$  is **little oh** of  $b_n$ ”.

### EXAMPLE 3

Show that  $\text{Ord}(n^2 \log n) < \text{Ord}(n^3)$ , that is, that  $n^2 \log n$  is little oh of  $n^3$ .

**Solution** This is similar to the argument at the beginning of this section:

$$\frac{n^2 \log n}{n^3} = \frac{\log n}{n} \rightarrow 0 \quad \text{as } n \rightarrow \infty$$

from Property 4 of logarithms from Section 0.2. ■

**Standard Sequences.** We now introduce the standard sequences to which we will associate all our execution sequences. Theorem 1 below shows how those standard sequences compare, and Theorem 2 shows how to reduce all comparisons to comparisons of standard sequences.

Our standard sequences are all the sequences of the form

$$\{a^n\}, \{n^r\}, \text{ and } \{n^r \log n\}, \quad (7)$$

where  $a$  is any positive real constant greater than 1,  $r$  can be any nonnegative integer, and the base of the logarithm is unspecified because the order of the logarithm function is unaffected by the base [7]. Note that when  $r = 0$  in the last item of (7), the sequence is just  $\{\log n\}$ . Note also that each of the sequences displayed in (7) is itself a family because of the set of values which can be assumed by  $a$  or  $r$ .

Our justification for choosing the sequences in (7) is that they are the ones which usually arise in the analysis of algorithms. (Sometimes other sequences arise, most frequently  $\{n^r \log \log n\}$  and  $\{n!\}$ .) An algorithm whose execution sequence has an order given by the second or third sequence type of (7) is called a **polynomial time** algorithm because it can be executed in a time proportional to some power of  $n$  (or some power of  $n$  multiplied by  $\log n$ ). In contrast, an algorithm whose

execution sequence has the same order as  $\{a^n\}$  for some value of  $a$  is said to exhibit **exponential growth**.

In Section 1.3 we shall meet an algorithm whose execution sequence,  $\{2^n - 1\}$ , is an example of an exponential sequence with  $a = 2$ . Algorithms which exhibit exponential growth are very difficult or even impossible to execute for even moderate values of  $n$ , because no computer is fast enough. If no faster algorithm is known for a task, we can use the exponential algorithm for values of  $n$  for which it is practical. For larger values of  $n$ , the best we can do is to try to find a polynomial time algorithm which will give a good approximation to the true result.

In discussing the standard sequences, and contrasting exponential and polynomial sequences, we have already implied that all the standard sequences have distinct orders. The following theorem makes this implication concrete; it gives the ordering of all standard sequences.

---

**Theorem 1.** For the sequences in (7) we have:

- i)  $\text{Ord}(n^r) < \text{Ord}(n^s)$ , if  $r < s$ .
  - ii)  $\text{Ord}(n^r) < \text{Ord}(n^r \log n) < \text{Ord}(n^{r+1})$ .
  - iii)  $\text{Ord}(a^n) < \text{Ord}(b^n)$ , if  $a < b$ .
  - iv)  $\text{Ord}(n^r) < \text{Ord}(a^n)$ , for any  $r$  and any  $a > 1$ .
- 

We discuss the proof of Theorem 1 in a moment, but let us emphasize the consequences first. Part (ii) enables us to write a string of inequalities:

$$\begin{aligned} \text{Ord}(1) &< \text{Ord}(\log n) < \text{Ord}(n) < \text{Ord}(n \log n) \\ &< \text{Ord}(n^2) < \text{Ord}(n^2 \log n) < \dots \end{aligned} \tag{8}$$

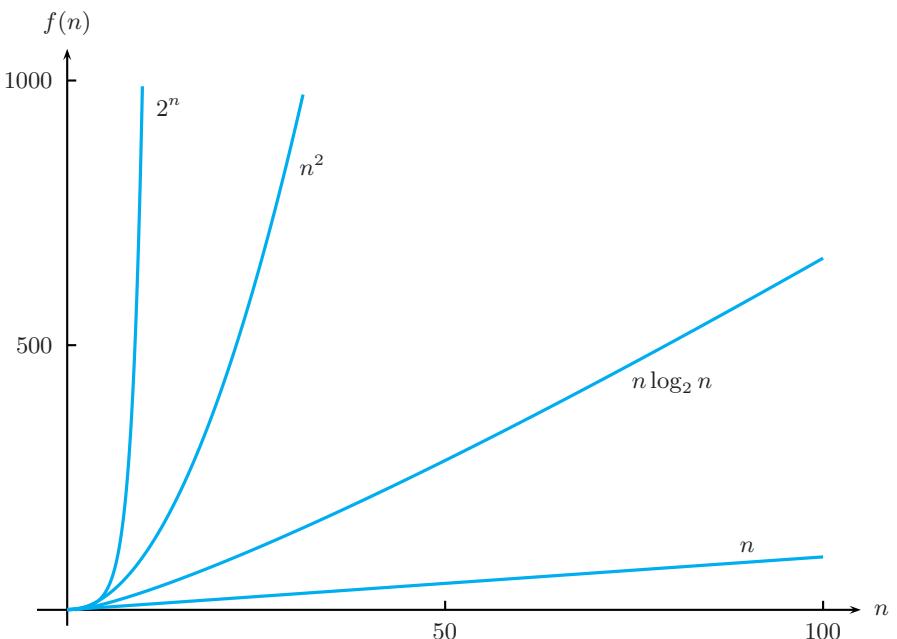
The first term in (8),  $\text{Ord}(1)$ , is the order of any sequence which converges to some nonzero value [6]. Note that the sequence  $1, 1, 1, \dots$  is, indeed, one of the sequences in (7). Just set  $r = 0$  in  $\{n^r\}$ . Note also that it follows from (iv) of Theorem 1 that any sequence in (8) is less than  $\text{Ord}(a^n)$  for any  $a > 1$ . Figure 0.5 illustrates the growth rates of various functions in (7).

Part (ii) of Theorem 1 is the most important part for the analysis of algorithms because most significant algorithms have execution sequences which boil down to one of the latter two sequences in (7) — powers and powers multiplied by logarithms.

The proof of Theorem 1 is addressed in [14, 15]. Parts (i) and (iii) depend only on facts you can easily argue about powers and exponents. The other parts depend on knowledge of relative growth of logs, powers, and exponents. A proof of

$$\text{Ord}(\log n) < \text{Ord}(n) \tag{9}$$

is the main step to proving the harder parts, but a proof of (9) would be quite lengthy unless you know some calculus, so we won't give it, nor will we ask you



**FIGURE 0.5**

Rates of growth of various standard functions.

to give a rigorous proof. But let's briefly try to convince you that (9) is correct. Actually, it is important for you to realize not just that  $\{\log n\}$  grows more slowly than  $\{n\}$  but that it grows *much* more slowly. The following tabulation which, for convenience, uses logarithms base 2, illustrates this point.

$n$	2	$2^{10} = 1024$	$2^{20} = 1048576$	$2^{30} = 1073741824$
$\log_2 n$	1	10	20	30

So, if you have two algorithms which have execution sequences of  $\text{Ord}(\log n)$  and  $\text{Ord}(n)$ , respectively, the first isn't just faster than the second but much faster for all but perhaps quite small values of  $n$ . The same conclusion is correct for two algorithms whose execution sequences have orders of  $\text{Ord}(n^r \log n)$  and  $\text{Ord}(n^{r+1})$  for any  $r$ .

*Using standard sequences.* The following theorem states all the facts you need to compare the order of two execution sequences by reducing them to standard sequences. The theorem has several parts, but they are all quite evident if you just remember that  $\text{Ord}(a_n) < \text{Ord}(b_n)$  means that  $a_n$  gets negligible relative to  $b_n$  as  $n$  gets large, and  $a_n = \text{Ord}(b_n)$  means that, for large  $n$ ,  $a_n$  is about the same size as  $b_n$  except for multiplicative constants.

Also, the details of applying the theorem can be quite involved (see Example 4), but as we will show, you never have to fuss with the details. You might want to see what we mean by looking now at the bulleted list after Example 4.

**Theorem 2.** For all sequences

- i) If  $\text{Ord}(a_n) < \text{Ord}(b_n)$ ,  $\text{Ord}(a'_n) < \text{Ord}(b_n)$ , and  $c, c'$  are constants, then  $\text{Ord}(c a_n + c' a'_n) < \text{Ord}(b_n)$  (and similarly for adding more than two sequences).
  - ii) If  $\text{Ord}(a_n) < \text{Ord}(b_n)$  and  $c, d$  are constants with  $d \neq 0$ , then  $c a_n + d b_n = \text{Ord}(b_n)$ .
  - iii) If  $a'_n = \text{Ord}(a_n)$ ,  $b'_n = \text{Ord}(b_n)$ , and  $\text{Ord}(a_n) < \text{Ord}(b_n)$ , then we have  $\text{Ord}(a'_n) < \text{Ord}(b'_n)$  (i.e., sequences of the same order may be substituted in order rankings).
- 

**PROOF** (i) Using Definition 2 the hypothesis in (i) may be written

$$\lim_{n \rightarrow \infty} \frac{a_n}{b_n} = \lim_{n \rightarrow \infty} \frac{a'_n}{b_n} = 0.$$

Thus

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{c a_n + c' a'_n}{b_n} &= \lim_{n \rightarrow \infty} \frac{c a_n}{b_n} + \lim_{n \rightarrow \infty} \frac{c' a'_n}{b_n} \\ &= c \lim_{n \rightarrow \infty} \frac{a_n}{b_n} + c' \lim_{n \rightarrow \infty} \frac{a'_n}{b_n} = c \cdot 0 + c' \cdot 0 = 0, \end{aligned}$$

which, again using Definition 2, is equivalent to the conclusion of (i). (Once again we have used the decomposability of limits into sums, and also the fact that constants can “pass through” limits.)

(ii) Again we are given that

$$\lim_{n \rightarrow \infty} \frac{a_n}{b_n} = 0,$$

and it suffices to show by (2) that  $(c a_n + d b_n)/b_n$  has a nonzero limit as  $n \rightarrow \infty$ . It has a nonzero limit because

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{c a_n + d b_n}{b_n} &= \lim_{n \rightarrow \infty} \left( c \frac{a_n}{b_n} + d \right) \\ &= c \lim_{n \rightarrow \infty} \frac{a_n}{b_n} + \lim_{n \rightarrow \infty} d \\ &= c \cdot 0 + d = d \neq 0. \end{aligned}$$

(iii) Now, for the first time, we need to use the general definition (3) of  $\text{Ord}$ . (Why?) Since  $a'_n = \text{Ord}(a_n)$ , there is a positive constant  $U$  such that

$$|a'_n| \leq U |a_n|, \tag{10}$$

and since  $b'_n = \text{Ord}(b_n)$ , there is a positive constant  $L$  such that

$$L|b_n| \leq |b'_n|;$$

(11)

There may be values of  $n$  that are exceptions to (10) or (11), but only finitely many. There may be further values of  $n$  for which  $b_n = 0$ , but only finitely many, because  $\text{Ord}(a_n) < \text{Ord}(b_n)$ , which means  $\lim_{n \rightarrow \infty} (a_n/b_n) = 0$ , so division by  $b_n$  must make sense for  $n$  large enough. Anyway, for  $n$  sufficiently large to avoid all three types of exceptions, we may divide (10) by (11) to obtain

$$\left| \frac{a'_n}{b'_n} \right| \leq \frac{U}{L} \left| \frac{a_n}{b_n} \right|. \quad (12)$$

The right-hand side of (12) goes to 0, so the left-hand side does too. That means  $a'_n/b'_n \rightarrow 0$  (even without absolute values). In short,  $\text{Ord}(a'_n) < \text{Ord}(b'_n)$ . ■

#### EXAMPLE 4

Compare the orders of

$$r_n = 2.4n^2 - 3n + 4$$

$$s_n = n \log n + 2n + 3 \log n$$

$$t_n = (1.3)^n + 14n^5.$$

**Solution** As for  $r_n$ , by Theorem 1,

$$\text{Ord}(1) < \text{Ord}(n) < \text{Ord}(n^2).$$

Thus  $\text{Ord}(-3n+4) < \text{Ord}(n^2)$  by Theorem 2(i). (We have used  $c = -3$  and  $c' = 4$ .) Then,  $r_n = 2.4n^2 - 3n + 4 = \text{Ord}(n^2)$  by Theorem 2(ii). (We have used  $a_n = -3n + 4$ ,  $b_n = n^2$ ,  $c = 1$  and  $d = 2.4$ .)

Similarly, since  $n \log n$  is the highest order standard sequence in  $s_n$ , then  $s_n = \text{Ord}(n \log n)$ . Likewise,  $t_n = \text{Ord}(1.3^n)$ .

Next, by Theorem 1,  $\text{Ord}(n \log n) < \text{Ord}(n^2) < \text{Ord}(1.3^n)$ . Thus, by Theorem 2(iii),  $\text{Ord}(s_n) < \text{Ord}(r_n) < \text{Ord}(t_n)$ . ■

We will never carry out this sort of analysis in so much detail again, because it is always easy to do in your head. Namely,

- For each sequence under consideration, break it into summands and throw out any multiplicative constants.
- Each remaining term should be a standard sequence; keep just the highest order one.
- Compare this standard sequence to the standard sequences obtained for the other sequences under consideration. The rank order of the original sequences is the same as the order of these standard sequences.

Indeed, as one determines the execution time of an algorithm, one rarely works with sequences as complicated as those in Example 4 because one throws out lower order terms as one goes along. For instance, suppose an algorithm has three parts and the first takes  $3n$  steps, the second takes  $2n^2$  steps and the third takes  $5 \log n$  steps. Then as soon as you have analyzed the second step, you forget about the  $3n$ , and as you go on to the third step, as soon as you see it's going to be a multiple of  $\log n$ , you stop worrying about it.

*Big Oh notation.* We conclude by mentioning the notation with which most books begin the discussion of growth rates:

---

**Definition 3.** Let  $\{a_n\}$  and  $\{b_n\}$  be two sequences. Then we say that

$$a_n = O(b_n),$$

if there is a constant  $U$  such that  $|a_n| \leq U|b_n|$  for sufficiently large  $n$ .

---

When “ $a_n = O(b_n)$ ” is spoken, one says “ $a_n$  is **big Oh** of  $b_n$ ”. Note that the “=” is once again read as “is”. Big Oh notation is traditional in mathematics and is widely used by writers about algorithms. In fact, some books translate it as “order” as well as Oh. They’ll prove that some algorithm takes  $O(n^2)$  time and call it an order  $n^2$  algorithm. But this isn’t quite right.

Yes, it is true that if  $b_n = \text{Ord}(a_n)$  then  $b_n = O(a_n)$ , but the converse can be false: Set  $a_n = n$  and  $b_n = n^2$ , for then  $n = O(n^2)$  but  $n \neq \text{Ord}(n^2)$ . It is to avoid the sloppiness that has crept into the usage of big Oh that we have introduced and will use Ord.

*Mystery sequences.* In this section we have learned how to find the order of a sequence if we know an exact formula for it. But often we don’t know a formula and still want to find the order, perhaps as a step towards finding a formula. There are methods for this too. See Section 2.6.

## Problems: Section 0.3

---

1.  $\langle 2 \rangle$  For each of the following sequences, whose  $n$ th term is given, state its order using one of the standard sequences (7), if possible.

- a)  $6n^3 + 4n + 1$       f)  $n^2 + n^3/\log n$   
b)  $3^{n+2}$       g)  $(3n^2 + 1)/(2n - 1)$   
c)  $n^2 + \log n$       h)  $n^3 \log n/(n^2 + 1)$   
d)  $n^2 + n \log n$       i)  $n^2 + (1.1)^n$   
e)  $n^2 + n^2/\log n$       j)  $n^{50} + (1.01)^{n/100}$

2.  $\langle 2 \rangle$  Do some experiments with a graphing calculator or graphing software to see how  $a_n/b_n$  changes as  $n \rightarrow \infty$  for various standard sequences. For example, try  $n^r/a^n$  for various  $r = 1, 2, 3$  and  $a = 2, 1.5, 1.25$ . Actually, you will probably need to use a continuous variable for your machine:  $x^2/2^x$  instead of  $n^2/2^n$ .

3.  $\langle 2 \rangle$  A certain algorithm has three parts. The first part takes  $n^2$  steps to run, where  $n$  is the size of

the input. The next part takes  $n \log n$  steps to run, but it must be looped through 5 times. The third part takes  $3n \log n$  steps to run, and it must be looped through  $n$  times. What standard sequence gives the order of this algorithm? How do you know?

4.  $\langle 2 \rangle$  Consider  $a_n = n(3 + \cos n)$  and  $b_n = n$ . Show that  $a_n = \text{Ord}(b_n)$ , because Eq. (3) holds, but that the simpler test (2) fails.
5.  $\langle 2 \rangle$  In the long run are the following positive or negative? Why?
- a)  $n^2 - \log n$   
b)  $n^2 - n^3/\log n$   
c)  $n^2 - (1.1)^n$
6.  $\langle 2 \rangle$  Suppose that  $\{a_n\}$  is such that  $\lim a_n = L \neq 0$ . Show that  $a_n = \text{Ord}(1) = \text{Ord}(c)$  for any

nonzero constant  $c$ . Also describe *all* sequences which are  $\text{Ord}(1)$ .

7. ⟨2⟩ We stated that the base of the logarithm in (7) is irrelevant — exactly the same order obtains whatever base we pick. Show that we were correct, using Property 3 of logarithms in Section 0.2.

8. ⟨2⟩ Is order also independent of the exponential base? That is, if  $a_n = \text{Ord}(a^n)$ , is it also  $\text{Ord}(b^n)$  if  $b \neq a$ ? Why?

9. ⟨2⟩ Suppose that you can rent time on a supercomputer that does 1 billion operations per second, and that the rental is \$1000/hr. Your company is willing to spend \$10,000 to solve the largest version of a certain problem it can.

a) If the best algorithm you know for solving this problem takes  $2^n$  operations, where  $n$  is the size of the problem, what is the largest size you can solve?

b) Suppose that someone discovers an algorithm for the same problem that takes  $n^5$  operations. What size problem can you solve now?

10. ⟨2⟩ By the “relative effectiveness” of two algorithms for a specific instance of a problem, we simply mean the ratio of the number of operations the two algorithms need. (Divide the number of operations for the *second* algorithm by the number for the first, so that a ratio greater than 1 means the first algorithm is more effective.) Suppose that you have two algorithms for alphabetizing  $n$  names; the first takes  $10n \log_2 n$  operations and the second takes  $\frac{1}{2}n^2$ .

a) What is the relative effectiveness of the first to the second if  $n = 100$ ?

b) If  $n = 10,000$ ?

11. ⟨2⟩ Obtain the result of Example 2 again using the theorems later in the section.

12. ⟨3⟩ Is the order of  $2^{\log n}$  also independent of the base of the logarithm? Why?

13. ⟨2⟩ (requires calculus — L’Hopital’s Rule)

a) Argue that  $\lim_{n \rightarrow \infty} (\log n)/n = 0$ .

b) Argue that  $\lim_{n \rightarrow \infty} (\log n)^j/n = 0$ , for any positive integer  $j$ .

14. ⟨3⟩

a) Prove (i) of Theorem 1. Just look at the limit of  $n^r/n^s$ .

b) Prove (iii) of Theorem 1.

c) Use Property 4 of logarithms from Section 0.2 to prove (ii) of Theorem 1.

d) Justify the string of inequalities (8), using Theorem 1.

15. ⟨3⟩

a) Use Eq. (5), Section 0.2, to prove (iv) of Theorem 1. *Hint:* Since no base is mentioned for the log in Eq. (5), it doesn’t matter and you may take the base to be  $a$ . Then let  $x = a^n$ .

b) Thus argue that the order of any term in (8) is less than the order of  $a^n$ .

16. ⟨3⟩ If  $r \notin N$ , then  $f(n) = n^r$  is not a polynomial, but it is still a function.

a) Determine the order ranking for  $\{n^r\}$  for all  $r \in R_{\geq 0}$ .

b) Fix  $r \in R_{\geq 0}$ . For what  $s$  is  $\text{Ord}(n^r \log n) < \text{Ord}(n^s)$ ?

17. ⟨2⟩ If  $a_n = n!$  were added to our set of standard sequences, where would it fit in their order ranking?

18. ⟨3⟩ Show that  $\log(n!) = \text{Ord}(n \log n)$ . *Hint:* First show that  $(n/2)^{n/2} < n! < n^n$  for  $n > 1$ .

19. ⟨3⟩ Show that, so long as  $a_n$  and  $b_n$  are never 0, the phrase “for all but a finite number of  $n$ ” may be omitted from Definition 1. *Hint:* If  $a_n$  was shown to be  $\text{Ord}(b_n)$  with some specific  $L$  and  $U$ , and  $n_1, n_2, \dots, n_k$  are the exceptional values, how do  $L$  and  $U$  have to be changed so that  $n_1, \dots, n_k$  are no longer exceptional?

20. ⟨3⟩ One might think one could avoid absolute value signs in (3) of Definition 1 by restating it as follows:

$a_n = \text{Ord}'(b_n)$  if there exist constants  $L, U$  (not necessarily positive) such that  $L b_n \leq a_n \leq U b_n$  except for a finite number of  $n$ .

However, this doesn’t work. Find a pair of sequences  $\{a_n\}, \{b_n\}$  such that  $a_n = \text{Ord}'(b_n)$  but  $a_n \neq \text{Ord}(b_n)$ .

21. ⟨2⟩ Restate Theorem 2 with little oh notation.

22. ⟨3⟩ Since we express  $a_n = \text{Ord}(b_n)$  by saying  $\{a_n\}$  and  $\{b_n\}$  have the *same* order (and since we also write  $\text{Ord}(a_n) = \text{Ord}(b_n)$ ), it better be true that the relationship is symmetric, that is, it better be that

if  $[a_n = \text{Ord}(b_n)]$ , then  $[b_n = \text{Ord}(a_n)]$ .

Prove this.

**23.** *(3)* Prove that  $a_n = \text{Ord}(b_n)$  is an equivalence relation as defined on p. 15. If we now reinterpret  $\text{Ord}(a_n)$  to denote the equivalence class of  $\{a_n\}$ , it follows that the statement  $\text{Ord}(a_n) = \text{Ord}(b_n)$  now makes sense as an equality of sets. See [25, Section 0.1]. *Note:* Using Condition (3), this equivalence relation is defined on the set of all sequences. Using (2) it must be restricted to sequences with only finitely many zero values, so that  $a_n/b_n$  is eventually defined.

**24.** *(2)* Since we used  $<$  as the symbol for order ranking, it better be true that order ranking behaves the way  $<$  should. Specifically, show that if  $\text{Ord}(a_n) < \text{Ord}(b_n)$  and  $\text{Ord}(b_n) < \text{Ord}(c_n)$  then  $\text{Ord}(a_n) < \text{Ord}(c_n)$ . That is, prove that order ranking is transitive.

**25.** *(3)* Is big Oh a transitive relation? Is it an equivalence relation?

**26.** *(3)* Define sequence  $\{a_n\}$  to be **asymptotic** to  $\{b_n\}$ , written  $a_n \sim b_n$ , if

$$\lim_{n \rightarrow \infty} \frac{a_n}{b_n} = 1.$$

If  $\{a_n\}$  is not asymptotic to  $\{b_n\}$ , we write  $a_n \not\sim b_n$ .

- a)** Show that  $3n - 1 \sim 3n$  but  $3n - 1 \not\sim n$ .
- b)** If  $a_n \sim b_n$ , then  $a_n = \text{Ord}(b_n)$ . Why?
- c)** Show that  $\sim$  is an equivalence relation.
- d)** Prove: If  $a_n \sim b_n$  and  $b_n = o(c_n)$ , then  $a_n = o(c_n)$ .
- e)** Prove: If  $a_n \sim b_n$  and  $b_n = \text{Ord}(c_n)$ , then  $a_n = \text{Ord}(c_n)$ .

**27.** *(3)*

- a)** Prove that  $\text{Ord}(\log n) = \text{Ord}(\lceil \log n \rceil)$ .
- b)** More generally, for any function  $f(n)$  show that  $\text{Ord}(f(n)\lfloor \log n \rfloor) = \text{Ord}(f(n)\log n) = \text{Ord}(f(n)\lceil \log n \rceil)$ . For instance, it follows that  $\text{Ord}(n^k \log n) = \text{Ord}(n^k \lceil \log n \rceil)$  for every power  $k$ .

This problem is relevant because algorithms must take an integer number of steps. Thus, when we say an algorithm takes  $5n \log n$  steps, we probably mean something like  $5n\lceil \log n \rceil$  steps. This problem suggests that we won't get the Order wrong by ignoring floors and ceilings.

**28.** *(4)*  $\text{Ord}$ ,  $O$ , and  $o$  need not be restricted to sequences, or to limits at  $\infty$ . Consider the following version: Let  $f$  and  $g$  be functions defined around 0, and write  $f(x) = \text{Ord}(g(x))$  (as  $x \rightarrow 0$ ) if there are positive constants  $L$  and  $U$  such that  $L \leq |f(x)/g(x)| \leq U$  for all  $x$  sufficiently close to 0. While this cognate definition is not very useful for the analysis of algorithms, it can be quite useful in continuous mathematics.

- a)** Show that, as  $x \rightarrow 0$ ,  $\text{Ord}(x+1) = \text{Ord}(x+2)$  but that  $\text{Ord}(x+1) \neq \text{Ord}(x)$ .
- b)** Come up with the associated definition for  $\text{Ord}(f(x)) < \text{Ord}(g(x))$ .
- c)** Consider all functions of either the form  $f(x) = x^n$ ,  $n > 0$ , or the form  $f(x) = a^x$ ,  $a > 0$ . Determine the order ranking (as  $x \rightarrow 0$ ) for all these functions.

## 0.4 Summation and Product Notation

In Section 0.1 we wrote a polynomial  $P_n(x)$  as

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0. \quad (1)$$

Perhaps you find reading this as tedious as we found typing it on our word processor. In any case, some shorthand notation other than the use of ellipses for writing long sums like this would clearly be useful and would save a lot of space, if nothing else. **Summation notation** is such a shorthand. It is one of the most powerful bits of mathematical notation there is, and we'll use it often. Although summation notation is simple in concept, students often have trouble using and understanding it. Since it is so important and since its use involves considerable subtleties, we devote an entire section to it and its close relative, product notation.

Summation notation allows us, for example, to write the polynomial in Eq. (1) as

$$P_n(x) = \sum_{i=0}^n a_i x^i. \quad (2)$$

Instead of writing a sequence of terms, we have written a single general term ( $a_i x^i$ ) using a new subscript, the letter  $i$ . We could just as easily have used  $j$  or  $k$  or  $l$  or even  $n$ , if we weren't already using it for another purpose in Eq. (2). The letter  $i$  is called a **dummy variable** in that, were we to write out the entire expression, this variable wouldn't appear at all. (Section 1.4 discusses a related use of this terminology in algorithms.) Using the dummy variable, we have, in effect, captured the explicit essence of the *pattern* in Eq. (1), which is only implicit using an ellipsis. If you substitute  $i = 0$  into the pattern term, you get  $a_0 x^0 = a_0$ , since a nonzero number raised to the zero power is 1. If you substitute 1 for  $i$ , you get  $a_1 x^1 = a_1 x$ , the linear term. And so on, until substituting  $n$  for  $i$  you get  $a_n x^n$ .

It is probably already clear that the  $\sum$  in front of the pattern term indicates that you are to add all the terms you have gotten by substituting into the pattern. ( $\sum$  is the Greek capital letter Sigma, which corresponds to a Roman "S"; hence Sigma for Sum.) The line below the  $\sum$  gives the lowest value of the dummy variable, whereas the line above indicates its highest value. (Sometimes, if we want to write a summation in a paragraph rather than setting it off on a line by itself, we write the upper and lower limits to the right of the  $\sum$ , as in  $\sum_{i=1}^8$ .) What we have called so far the dummy variable is more commonly called the **index of summation** (or just the index). The index takes on all integral values between the lowest and highest values.

If you understand our explanation of Eq. (2), you'll immediately see that it had two advantages over use of ellipsis:

- It is more compact.
- Whereas the pattern in the use of an ellipsis may be obvious to the writer, it will sometimes be less than obvious to the reader; summation notation gets rid of any possible ambiguity.

Note also the power of using subscripts in Eq. (2) where  $a_i$  stands for any of the values  $a_1, a_2, \dots, a_n$  and  $x^i$  stands for the associated power of  $x$ . Without such use of subscripts, summation notation would not be possible, for how could you possibly represent, say,  $a, b, \dots, z$  by a single symbol?

In general, suppose we wish to express

$$c_j + c_{j+1} + \cdots + c_{k-1} + c_k$$

with summation notation. We do this as

$$\sum_{i=j}^k c_i,$$

where each  $c_i$  is called a **term** or **summand** of the summation. This general example includes our polynomial example: just set  $j = 0$ ,  $k = n$ , and  $c_i = a_i x^i$ .

Eq. (2) indicates that the index of summation can appear in the quantity being summed as a subscript, an exponent, or indeed, in any way you wish. Or it need not appear at all, as in

$$\sum_{i=1}^n 1,$$

which represents the sum of  $n$  1's since, for each value of  $i$ , the quantity being summed is the constant 1.

### EXAMPLE 1

Evaluate:

a)  $\sum_{i=3}^6 i^2$     b)  $\sum_{p=0}^5 (2p + 3)$     c)  $\sum_{i=1}^4 i^{2i}$ .

**Solution** For a) we get

$$3^2 + 4^2 + 5^2 + 6^2 = 9 + 16 + 25 + 36 = 86.$$

For b) we have

$$(0+3) + (2+3) + (4+3) + (6+3) + (8+3) + (10+3) = 3 + 5 + 7 + 9 + 11 + 13 = 48.$$

Note that the index of summation was  $p$ . Note also that, if we had left the index of summation as  $p$  but had written  $2i + 3$  instead of  $2p + 3$ , i.e.,

$$\sum_{p=0}^5 (2i + 3),$$

then this sum would have called for the addition six times of  $2i + 3$  (with the result  $12i + 18$ ). Why? Because anything after (i.e., to the right of) the summation sign which does not include the index is a constant with respect to the summation.

For c), where the index of summation appears more than once under the summation sign, we get

$$\begin{aligned} 1^2 + 2^{2 \cdot 2} + 3^{2 \cdot 3} + 4^{2 \cdot 4} &= 1^2 + 2^4 + 3^6 + 4^8 \\ &= 1 + 16 + 729 + 65,536 \\ &= 66,282. \blacksquare \end{aligned}$$

The set of values taken on by the index of summation is called the **index set**. There is no requirement that this set consist of a sequence of consecutive integral values, although this is by far the most common case. For example, we may write

$$\sum_{\substack{i=2 \\ i \text{ even}}}^8 a_i, \tag{3}$$

which represents  $a_2 + a_4 + a_6 + a_8$ . Or we may write

$$\sum_{\substack{i=1 \\ i \text{ not divisible} \\ \text{by 2 or 3}}}^{11} 5^i, \tag{4}$$

which represents

$$5 + 5^5 + 5^7 + 5^{11} = 5 + 3125 + 78,125 + 48,828,125 = 48,909,380.$$

In full generality we may write

$$\sum_{R(i)} a_i, \tag{5}$$

where  $R(i)$  is a function, often called a **predicate**, whose domain is usually  $N$  and which has the value *true* for each  $i$  included in the index set and the value *false* for all other  $i$  in the domain. (Are you disturbed at all by a function whose values are not numbers but rather *true* and *false*? No need to be. Remember that, like the domain, the codomain of a function may be any set whatsoever. In Chapter 7, when we discuss logic, we'll be dealing almost entirely with functions whose values are *true* or *false*.) For example, in part a) of Example 1,  $R(i)$  would be true when  $i = 3, 4, 5$ , and  $6$  and false otherwise. In (3),  $R(i)$  would be true only when  $i = 2, 4, 6$ , and  $8$ . In (4),  $R(i)$  would be true only when  $i = 1, 5, 7$ , and  $11$ . Indeed, we could have written all these summations using predicate notation. For example, we could have written part a) of Example 1 as

$$\sum_{3 \leq i \leq 6} i^2,$$

although we wouldn't usually write it that way when the lower limit–upper limit notation is natural. By the way, we write the predicates *below* the summation sign merely because of convention.

We may, in fact, generalize (5) even further by not even requiring that the predicate refer to an integer index. For example, we may write

$$\sum_{x \in T} f(x),$$

where  $T$  is a set and  $f$  is a function defined on elements of  $T$ . Or we may write

$$\sum_{S \subset T} f(S),$$

where  $T$  is a set and  $f$  is a function defined on subsets of  $T$ .

One implication of (5) is that, if  $R(i)$  is true for all nonnegative numbers, then that summation contains an *infinite* number of terms. We'll meet a few such summations in later chapters. The usual notation when summing over all of  $N$  is to put the infinity symbol  $\infty$  over the Sigma, as in

$$\sum_{i=0}^{\infty} a_i.$$

Suppose  $R(i)$  is not true for *any* value of  $i$ , as in

$$\sum_{8 \leq i \leq 6} i^3.$$

Then the sum is empty and, by convention, its value is zero. Or suppose we write

$$\sum_{i=8}^6 a_i.$$

Since, again by convention, we *always* step up from the lower limit to the upper one, this sum is also empty and has the value zero.

The compactness of summation notation allows us to manipulate it easily in a variety of ways. For example, if  $c$  is a constant, then

$$\sum_{R(i)} ca_i = c \sum_{R(i)} a_i, \quad (6)$$

because the constant  $c$  multiplies every term and may therefore be factored out independently of  $R(i)$ . More generally, if  $c$  and  $d$  are constants, then

$$\sum_{R(i)} (ca_i + db_i) = c \sum_{R(i)} a_i + d \sum_{R(i)} b_i. \quad (7)$$

*Rewriting summations.* Throughout mathematics one must know how to **rewrite** expressions, that is, find alternative forms with the same value, for another form is often just what is needed to make progress. We now give three important examples of how to change the form of a summation.

First, we consider how to change the limits of the index of summation or, to put it another way, how to change the index of summation from one variable to another. A common case occurs when you have a sum of the form

$$\sum_{i=1}^n a_i \quad (8)$$

and you would like to change it so that the lower limit is 0. You do this by defining

$$j = i - 1 \quad \text{that is,} \quad i = j + 1. \quad (9)$$

When  $i = 1$ ,  $j = 0$  and when  $i = n$ ,  $j = n - 1$ . Using these limits and replacing the subscript  $i$  by the equivalent  $j + 1$ , you may rewrite (8) as

$$\sum_{j=0}^{n-1} a_{j+1}. \quad (10)$$

If you write out (8) and (10), you will see that both have precisely the same terms, namely,

$$a_1 + a_2 + \cdots + a_n,$$

and thus they are equal. In actual practice, after we've made this change of variable, we often then go back to the original index and write

$$\sum_{i=0}^{n-1} a_{i+1}.$$

Remember that the index is a dummy variable and its “name”, therefore, makes no difference.

With a change of variable like that in Eq. (9) you'll be able to change the limits of summation in almost any way you wish. You must remember, though, to change all places where the index of summation appears in the term under the summation sign.

Our second example concerns the case where you wish to reverse the order of summation. That is, when the summation is expanded, instead of having the terms appear in the natural order from lower limit to upper limit,

$$a_1 + a_2 + \cdots + a_n,$$

we wish to have them appear in the order

$$a_n + a_{n-1} + \cdots + a_2 + a_1.$$

We do this by making the change of variable

$$j = n - i,$$

which changes (8) to

$$\sum_{i=1}^n a_i = \sum_{j=0}^{n-1} a_{n-j}.$$

Our third example concerns a common occurrence where we have two separate summations that we would like to combine into one, e.g.,

$$\sum_{i=0}^n a_i + \sum_{i=1}^n b_i.$$

How can we combine these expressions, which have different limits, into a single summation? Remove the first term of the first summation to obtain

$$a_0 + \sum_{i=1}^n a_i + \sum_{i=1}^n b_i.$$

Now, since the limits on both summations are the same, we can write this as

$$a_0 + \sum_{i=1}^n (a_i + b_i).$$

## Double Summation

Suppose we have a set of doubly subscripted quantities  $a_{ij}$ , where the range of values of  $i$  is  $m, m+1, \dots, n-1, n$  and the range of  $j$  is  $p, p+1, \dots, q-1, q$ . Now suppose we wish to add the  $a_{ij}$ 's for all possible pairs of values of  $i$  and  $j$ . We can do this with a double summation, which we write as

$$\sum_{i=m}^n \sum_{j=p}^q a_{ij} \tag{11}$$

and interpret as follows:

For each value of the index in the outer (i.e., left-most) summation let the index in the inner summation range over all its values and sum the terms generated; then increase the outer index by 1 and repeat, always adding the result to the previous sum until  $i$  reaches  $n$ .

In effect, this means that (11) contains implied parentheses, as in

$$\sum_{i=m}^n \left( \sum_{j=p}^q a_{ij} \right).$$

Thus (11) represents the sum

$$\begin{aligned} \sum_{j=p}^q a_{mj} + \sum_{j=p}^q a_{m+1,j} + \cdots + \sum_{j=p}^q a_{nj} &= a_{mp} + a_{m,p+1} + a_{m,p+2} + \cdots \\ &\quad + a_{mq} + a_{m+1,p} + \cdots + a_{m+1,q} + \cdots + a_{np} + a_{n,p+1} + \cdots + a_{nq}, \end{aligned} \tag{12}$$

which is what we wanted.

## EXAMPLE 2

Evaluate:

$$\sum_{i=1}^3 \sum_{j=2}^4 i^j.$$

**Solution** We obtain

$$\begin{aligned} (1^2 + 1^3 + 1^4) + (2^2 + 2^3 + 2^4) + (3^2 + 3^3 + 3^4) \\ = 1 + 1 + 1 + 4 + 8 + 16 + 9 + 27 + 81 = 148. \blacksquare \end{aligned}$$

When the index values for the two sums are defined, again from left to right, by two general predicates  $R(i)$  and  $S(j)$ , the principle is the same: For each value of  $i$  for which  $R(i)$  is true, evaluate all the terms for which  $S(j)$  is true and add them.

When a double sum is particularly simple, it can be written using a single sum. For instance,

$$\sum_{i,j=1}^n a_{ij} \text{ means the same as } \sum_{i=1}^n \sum_{j=1}^n a_{ij}.$$

Now suppose that the term under the summation sign in (11) is  $a_i b_j$ , that is, the product of two terms, one with subscript  $i$  and one with subscript  $j$ . Since Eq. (12) indicates that the double summation includes all possible combinations of the subscripts  $i$  and  $j$ , it follows that in this case we may write (11) as

$$\sum_{i=m}^n \sum_{j=p}^q a_i b_j = \left( \sum_{i=m}^n a_i \right) \left( \sum_{j=p}^q b_j \right), \tag{13}$$

since the product of the two sums includes all possible combinations of  $i$  and  $j$ . Another way to see this is to note that on the left of Eq. (13)  $a_i$  is a constant with

respect to the summation with index  $j$  and therefore can be brought outside that summation as a constant, as in Eq. (6). For example, with  $n = 2$ ,  $q = 3$ , and  $m = p = 1$ ,

$$\begin{aligned}
 \sum_{i=1}^2 \sum_{j=1}^3 a_i b_j &= \sum_{j=1}^3 a_1 b_j + \sum_{j=1}^3 a_2 b_j \\
 &= a_1 \sum_{j=1}^3 b_j + a_2 \sum_{j=1}^3 b_j \quad [\text{Constant brought out}] \\
 &= (a_1 + a_2) \sum_{j=1}^3 b_j \\
 &= \left( \sum_{i=1}^2 a_i \right) \left( \sum_{j=1}^3 b_j \right).
 \end{aligned}$$

If you understand this example, you should be able to justify Eq. (13) in general [16].

So far, then, double summation hasn't been much more difficult than single summation. And you should be able to extend the preceding discussion without any trouble to triple summation or to any number of sums in succession. However, suppose we write

$$\sum_{i=1}^4 \sum_{j=1}^{5-i} \frac{j^2}{(2i-1)}. \tag{14}$$

Here one of the limits for the inner sum is not a constant but contains a variable, namely, the index of the outer sum. Still, it should be fairly clear that we should evaluate (14) as follows:

- i) Set  $i = 1$ .
- ii) Set  $j = 1, 2, 3, 4$  (since  $5 - i = 4$ ), evaluate  $j^2/(2i-1) = j^2$ , and sum.
- iii) Set  $i = 2$  and  $j = 1, 2, 3$  (since  $5 - i = 3$ ), evaluate, and add to the result of (ii).
- iv) Repeat for  $i = 3$  and  $j = 1, 2$ .
- v) Finally, repeat for  $i = 4$ ; since here the upper and lower limits are the same, i.e., 1, we have only  $j = 1$ .

The result is

$$1 + 4 + 9 + 16 + \frac{1}{3} + \frac{4}{3} + \frac{9}{3} + \frac{1}{5} + \frac{4}{5} + \frac{1}{7} = 35\frac{17}{21}.$$

We could give other, even trickier examples, but all we want to do here is introduce you to summations with variable limits. From Eq. (12) it should be clear that, when all the limits are constants, (11) represents the same sum regardless of whether the  $i$  summation or the  $j$  summation comes first. Thus

$$\sum_{i=m}^n \sum_{j=p}^q a_{ij} = \sum_{j=p}^q \sum_{i=m}^n a_{ij}.$$

But what about summations like (14)? Clearly, we can't just interchange the order of summation, because then the outer summation would have a limit depending on a summation to its right, which doesn't make any sense because by definition double sums are expanded from left to right. We won't discuss just how you go about interchanging the order of summation when one or more of the limits is variable. But the need to do so does occur occasionally. Be on your guard if it ever does occur, because care is required to handle such cases.

*Multiplying polynomials.* We mentioned in Section 0.2, just before Example 3, that we didn't have a very good notation for expressing the product of two polynomials. Now with double summation notation, we do. As in Section 0.2, let the two polynomials be

$$P_n(x) = \sum_{i=0}^n a_i x^i \quad \text{and} \quad Q_m(x) = \sum_{i=0}^m b_i x^i.$$

Our aim is to express the product of these polynomials in a simple, compact form. Here it is:

$$P_n(x)Q_m(x) = \sum_{i=0}^{m+n} \left( \sum_{j=0}^i a_j b_{i-j} \right) x^i, \quad \begin{cases} a_j = 0, & j > n \\ b_j = 0, & j > m. \end{cases} \quad (15)$$

For example,

$$(a_1 x + a_0)(b_2 x^2 + b_1 x + b_0) = a_1 b_2 x^3 + (a_1 b_1 + a_0 b_2) x^2 + (a_1 b_0 + a_0 b_1) x + a_0 b_0. \quad (16)$$

Note that each term in parentheses on the right-hand side of (16) corresponds to one term of the summation in parentheses in Eq. (15). For each  $i$ , the inner sum in Eq. (15) consists of a sum of products of two coefficients of powers, one from  $P_n(x)$  and one from  $Q_m(x)$ , where the sum of the powers is  $i$ . This sum of products is then the coefficient of  $x^i$ .

## Product Notation

We have much less to say about product notation than summation notation for two reasons:

- Everything we've said about summation notation carries over pretty directly to product notation.
- Product notation isn't nearly as common as summation notation.

Suppose we wish to compute the product of many terms, as in

$$a_1 a_2 \cdots a_n.$$

The direct analogy with summation notation is to write this as

$$\prod_{i=1}^n a_i,$$

where we use  $\prod$  because Pi in Greek corresponds to “P” (for product) in the Roman alphabet. Thus, for example,

$$\prod_{k=1}^n k = 1 \times 2 \times 3 \cdots (n-1)n = n!,$$

$$\prod_{i=2}^7 \frac{(i-1)}{i} = \left(\frac{1}{2}\right) \left(\frac{2}{3}\right) \cdots \left(\frac{6}{7}\right) = \frac{1}{7},$$

$$\prod_{n=1}^4 2^n = 2^1 2^2 2^3 2^4 = 2^{10}.$$

If the product is empty, that is, if the predicate which defines the values of the index is true for no value of the index, then by convention, the product is 1. Why do you suppose we use this convention for products when the corresponding convention for sums is to replace an empty sum by 0?

In [17–18] we consider how to extend the idea of summation and product notation to other operators, such as set union and intersection.

## Problems: Section 0.4

---

1.  $\langle 1 \rangle$  Write the following using  $\sum$  and  $\prod$  notation.

- a)  $1 + 2 + 3 + \cdots + 100$
- b)  $1 \cdot 2 \cdot 3 \cdots 100$
- c)  $2 + 4 + 6 + 8 + \cdots + 100$
- d)  $1 \cdot 3 \cdot 5 \cdot 7 \cdots 99$

2.  $\langle 2 \rangle$  Write the following polynomials using  $\sum$  notation.

- a)  $x + 2x^2 + 3x^3 + \cdots + 10x^{10}$
- b)  $1 - x + x^2 - x^3 + \cdots + x^{10}$
- c)  $x + x^2 + x^3 + \cdots + x^{14}$
- d)  $1 + 2x^2 + 3x^4 + 4x^6 + \cdots + 8x^{14}$

3.  $\langle 1 \rangle$  Evaluate the following sums.

- a)  $\sum_{i=2}^4 (2i)^{-i}$
- b)  $\sum_{i=3}^5 (3i + 2p)$

4.  $\langle 2 \rangle$  Evaluate the following sums.

- a)  $\sum_{\substack{i=3 \\ i \text{ even}}}^8 1/i^2$
- b)  $\sum_{\substack{i=3 \\ 3|i \text{ or } 5|i}}^{20} (2i^2 + 6)$

5.  $\langle 2 \rangle$  Evaluate each of the following sums for the values of  $n$  indicated. If the answer is particularly simple, see if you can explain why.

- a)  $\sum_{j=1}^n \left(\frac{1}{j} - \frac{1}{j+1}\right)$   $n = 4, 5.$
- b)  $\sum_{S \subset [n]} (-1)^{|S|}$   $n = 2, 3$  and  
 $[n] = \{1, 2, \dots, n\}.$
- c)  $\prod_{j=1}^n \frac{j}{j+1}$   $n = 5, 6.$
- d)  $\prod_{j=2}^n \frac{j^2-1}{j^2}$   $n = 5, 6.$
- e)  $\sum_{d|n} \left(\frac{d-n}{d}\right)$   $n = 6, 30.$
- f)  $\sum_{S \subset [n]} 2^{|S|}$   $n = 2, 3$  and  
 $[n] = \{1, 2, \dots, n\}.$

6.  $\langle 1 \rangle$

- a) Express the following inequality in summation notation:

$$(a_1^2 + a_2^2 + \cdots + a_n^2)(b_1^2 + \cdots + b_n^2) \geq (a_1 b_1 + a_2 b_2 + \cdots + a_n b_n)^2.$$

This is called the Cauchy-Schwarz Inequality, and it is always true.

- b) Express the following inequality using sum and product notation.

$$\frac{a_1 + a_2 + \cdots + a_n}{n} \geq (a_1 a_2 \cdots a_n)^{1/n}.$$

This is called the arithmetic-geometric mean inequality and is true for all nonnegative  $a_i$ 's.

7. ⟨2⟩ Explain why  $\sum_{j=1}^n j2^j = \sum_{k=0}^n k2^k$ .
8. ⟨2⟩ Rewrite  $\sum_{k=1}^{10} 2^k$  so that  $k$  goes from 0 to 9 instead. Remember, a rewrite must have the same value.
9. ⟨2⟩ Rewrite  $\sum_{i=0}^n (2i+1)$ , putting it in the form  $\sum_{i=?}^? (2i-1)$ . You have to figure out what the question marks should be.
10. ⟨2⟩ Rewrite  $\sum_{k=0}^n 3k + \sum_{j=1}^{n+1} 4j$  so that it involves just one  $\sum$  sign. There may be some extra terms outside the  $\sum$  sign.
11. ⟨2⟩ Rewrite  $\sum_{\substack{i=1 \\ i \text{ odd}}}^9 i^2$  in the form  $\sum_{j=a}^b f(j)$ , where you have to figure out the constants  $a$  and  $b$  and the function  $f$ .
12. ⟨2⟩ Evaluate the sums:
- a)  $\sum_{i=1}^3 \sum_{\substack{j=1 \\ ij \text{ even}}}^3 [(i/j) + (j/i)]$ .
- b)  $\sum_{i=1}^9 \sum_{j=1}^{\lfloor 9/i \rfloor} ij^2$ .
13. ⟨2⟩ Evaluate:
- a)  $\sum_{j=1}^n \sum_{i=j}^n \frac{1}{i}$        $n = 3, 4$ .
- b)  $\sum_{k=1}^n \sum_{j=1}^n \frac{(-1)^k}{j}$        $n = 3, 4$ .
- c)  $\sum_{j=1}^n \sum_{i=1}^n \cos\left(\frac{\pi i}{2j}\right)$        $n = 3$ .
14. ⟨3⟩ If  $\sum_{i=0}^{100} x^i$  is squared and rewritten as  $\sum_{i=0}^n a_i x^i$ , what is  $a_{50}$ ? What is  $n$ ?

15. ⟨3⟩ If  $(\sum_{i=0}^{100} x^i)(\sum_{i=0}^{25} x^i)$  is multiplied out, what is  $a_{50}$ ?

16. ⟨2⟩ Mimic the computations in the special case displayed after Eq. (13) to prove the correctness of Eq. (13) in general.

17. ⟨3⟩ For this problem we use the following **interval notation**:

$$(a, b) = \{x \mid a < x < b\}$$

and

$$[a, b] = \{x \mid a \leq x \leq b\}.$$

Also,  $\bigcup_{i=1}^n S_i$  means  $S_1 \cup S_2 \cup \cdots \cup S_n$  and  $\bigcap_{i=1}^n S_i$  means  $S_1 \cap S_2 \cap \cdots \cap S_n$ . Evaluate the following expressions.

- a)  $\bigcup_{i=1}^{10} (0, i)$       b)  $\bigcup_{j=1}^5 [2j-2, 2j]$   
 c)  $\bigcup_{i=1}^{\infty} (-i, i)$       d)  $\bigcup_{k=1}^{10} [0, 1/k]$   
 e)  $\bigcap_{k=1}^{10} [0, 1/k]$       f)  $\bigcap_{k=1}^{10} (0, 1/k)$   
 g)  $\bigcap_{i=1}^{\infty} [0, 1/i]$       h)  $\bigcap_{i=1}^{\infty} (0, 1/i)$

18. ⟨2⟩ Let  $P_k$  be the statement that the integer  $k$  is a prime. Let  $\vee$  stand for “or” and let

$$\bigvee_{i=1}^n P_i$$

mean  $P_1 \vee P_2 \vee \cdots \vee P_n$ . Similarly let “ $\wedge$ ” stand for “and” and let

$$\bigwedge_{i=1}^n P_i$$

mean  $P_1 \wedge P_2 \wedge \cdots \wedge P_n$ . What do the following expressions assert and are they true?

- a)  $\bigvee_{k=102}^{106} P_k$       b)  $\bigwedge_{k=1}^3 P_{2k+1}$   
 c)  $\bigwedge_{k=1}^4 P_{2k+1}$

## 0.5 Matrix Algebra

In this section we introduce you briefly to matrices and some related concepts and notation. Matrices are very useful computational tools that organize a lot of information compactly.

A **matrix** is a two-dimensional, rectangular array of numbers which we write as follows:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}. \quad (1)$$

We write the elements of the matrix,  $a_{ij}$ , in doubly subscripted form, with the first subscript representing the *row* (horizontal line) in which the element occurs and the second subscript representing the *column* (vertical line). Thus the matrix in (1) contains  $m$  rows and  $n$  columns. We say it is an  $m \times n$  matrix. As with sets, we use uppercase letters to denote matrices and often write a matrix as  $A = [a_{ij}]$ . For example, if

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 4 & -2 & 6 \end{bmatrix},$$

then  $a_{13} = 3$ ,  $a_{22} = -2$ , and so on.

When  $m$  in (1) is 1, the matrix consists of a single row. Therefore the row subscript conveys no useful information, and we may write a single row with a single subscript,

$$[a_1 \ a_2 \ \cdots \ a_n],$$

which is called a **row vector**, or just a **vector**, or sometimes, a **one-dimensional array**. Similarly, a  $m \times 1$  matrix is a **column vector**.

We are going to define how two matrices of form (1) may be added, subtracted, and multiplied. In order to perform any operation on two matrices,  $A$  and  $B$ , they must be **conformable**. In the case of addition and subtraction, this means that both matrices must have the same number of rows and the same number of columns. If this is the case, then we can find their sum by adding corresponding elements. That is, the (1, 1) entries are added, the (1, 2) entries, and so on. Formally, if  $A = [a_{ij}]$  and  $B = [b_{ij}]$ , then  $C = A + B$  has elements  $c_{ij}$  such that

$$c_{ij} = a_{ij} + b_{ij}. \quad (2)$$

In a display like (2), it is understood that there is one equation for each choice of  $i$  and  $j$ , but we can also make this explicit by writing

$$c_{ij} = a_{ij} + b_{ij}, \quad i = 1, \dots, m, \quad j = 1, \dots, n.$$

This format is especially helpful when there are other variables in the equation too; see, for instance, Eq. (3).

## EXAMPLE 1

Let the rows of the following two matrices refer to the first-year, sophomore, junior, and senior classes at Podunk College, and let the columns refer to its humanities, social science, and science divisions. Each entry is the number of students in a particular class (first-year, etc.) studying in a particular division. The matrix on the left represents the fall semester and the matrix on the right the spring semester.

$$\begin{bmatrix} 212 & 475 & 624 \\ 273 & 419 & 602 \\ 263 & 410 & 560 \\ 290 & 408 & 553 \end{bmatrix} \quad \begin{bmatrix} 253 & 423 & 587 \\ 295 & 386 & 549 \\ 280 & 400 & 535 \\ 301 & 376 & 540 \end{bmatrix}$$

Since both matrices have four rows and three columns, they are conformable. Using Eq. (2), we find that their sum is

$$\begin{bmatrix} 465 & 898 & 1211 \\ 568 & 805 & 1151 \\ 543 & 810 & 1095 \\ 591 & 784 & 1093 \end{bmatrix}.$$

What does this matrix represent [4]? This example illustrates the usefulness of matrices for tabulating and manipulating data. ■

To subtract two conformable matrices we just use Eq. (2) with the plus sign replaced by a minus sign.

Another simple operation which can be performed on any matrix is to multiply the matrix by a constant. If we are given  $A = [a_{ij}]$ , then we define  $rA$ , where  $r$  is a constant, to be the matrix whose elements are  $ra_{ij}$ . That is, we multiply each element of  $A$  by  $r$ .

Next we define and give some examples of matrix multiplication. This definition may not be what you'd first expect, and it makes good use of summation notation. For two matrices,  $A$  and  $B$ , to be conformable for multiplication, the number of columns in the first matrix must equal the number of rows in the second. So let  $A = [a_{ij}]$  by  $m \times k$  and let  $B = [b_{ij}]$  be  $k \times m$ . We define their product  $AB = C = [c_{ij}]$  by

$$c_{ij} = \sum_{r=1}^k a_{ir} b_{rj}, \quad i = 1, \dots, m, \quad j = 1, \dots, n. \quad (3)$$

In Fig. 0.6 we illustrate how to interpret Eq. (3). In matrix multiplication the  $(i, j)$  element in the product matrix  $C$  is formed by taking the row vector formed by the  $i$ th row of  $A$  and multiplying it element by element times the column vector formed by the  $j$ th column of  $B$  and then adding all the products.

## EXAMPLE 2

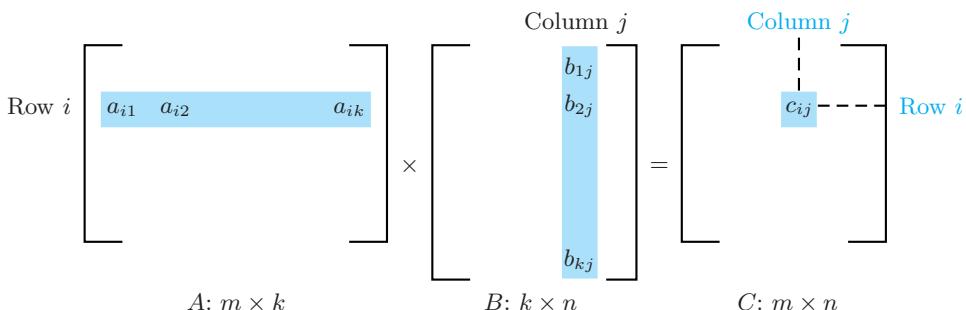
Calculate the product of

$$A = \begin{bmatrix} 6 & 7 & 8 \\ -5 & -3 & 17 \\ 2 & 0 & 4 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} -5 & 10 \\ 12 & 6 \\ 2 & -1 \end{bmatrix}.$$

**Solution** To calculate the first row of the product matrix  $C$ , begin by taking the first row of  $A$  and multiplying it term by term by the first column of  $B$ , obtaining

$$6 \times (-5) + 7 \times 12 + 8 \times 2 = -30 + 84 + 16 = 70.$$

Then multiply the first row of  $A$  times the second column of  $B$ , obtaining 94. Proceeding similarly with the second and third rows of  $A$  — make sure you understand



Matrix multiplication. The multiplication of  $A = [a_{mk}]$  by  $B = [b_{kn}]$  gives matrix  $C$ , where

$$c_{ij} = \sum_{r=1}^k a_{ir} b_{rj}.$$

**FIGURE 0.6**

which calculations need to be done — you obtain finally

$$C = \begin{bmatrix} 70 & 94 \\ 23 & -85 \\ -2 & 16 \end{bmatrix}. \blacksquare$$

The vector times vector product expressed by the sum in Eq. (3) is called an **inner product** or **dot product**. A common notation for inner products is  $\mathbf{a} \cdot \mathbf{b}$ , where boldface letters mean vectors. Using the notation of inner products, we may express matrix multiplication as  $AB = [\mathbf{a}_i \cdot \mathbf{b}_j]$ , where  $\mathbf{a}_i$  is the  $i$ th row of  $A$  and  $\mathbf{b}_j$  is the  $j$ th column of  $B$ .

You may be somewhat curious about our explanation of matrix multiplication. Why use this strange way of defining matrix multiplication when we could have multiplied matrices term by term just as we added them? The answer, as we hope you've guessed, is that our definition is more useful than this alternative. One common use is to express systems of linear equations succinctly. For simplicity, consider the system of two equations in two unknowns

$$\begin{aligned} 2x_1 + 3x_2 &= 4 \\ 5x_1 + 6x_2 &= 7. \end{aligned} \tag{4}$$

If we set

$$A = \begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 4 \\ 7 \end{bmatrix},$$

then system (4) becomes the single matrix equation

$$A\mathbf{x} = \mathbf{b}. \tag{5}$$

Indeed, if your hand calculator has matrix capabilities, (5) is the form in which you can solve such systems most easily. Input  $A$  and  $\mathbf{b}$ , push one button, and out comes  $\mathbf{x}$ .

Take a course in linear algebra, and you'll see that thinking in terms of (5) allows for powerful theory, not just convenience.

Here's another example of the use of matrix multiplication. Let

$$\begin{aligned} z_1 &= 2y_1 + 3y_2 + 4y_3 & y_1 &= x_1 + 2x_2 \\ z_2 &= 3y_1 + 2y_2 - y_3 & y_2 &= 3x_1 + 4x_2 \\ & & y_3 &= 5x_1 + 6x_2. \end{aligned}$$

The matrices of the coefficients of the  $z$ 's in terms of the  $y$ 's and the  $y$ 's in terms of the  $x$ 's are, respectively,

$$A = \begin{bmatrix} 2 & 3 & 4 \\ 3 & 2 & -1 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}. \quad (6)$$

To express the  $z$ 's in terms of the  $x$ 's algebraically, we substitute:

$$\begin{aligned} z_1 &= 2(x_1 + 2x_2) + 3(3x_1 + 4x_2) + 4(5x_1 + 6x_2) \\ &= (2 \times 1 + 3 \times 3 + 4 \times 5)x_1 + (2 \times 2 + 3 \times 4 + 4 \times 6)x_2 \\ &= 31x_1 + 40x_2; \\ z_2 &= 3(x_1 + 2x_2) + 2(3x_1 + 4x_2) - 1(5x_1 + 6x_2) \\ &= (3 \times 1 + 2 \times 3 + (-1) \times 5)x_1 + (3 \times 2 + 2 \times 4 + (-1) \times 6)x_2 \\ &= 4x_1 + 8x_2. \end{aligned}$$

Thus the coefficient matrix of the  $z$ 's in terms of the  $x$ 's is

$$C = \begin{bmatrix} 31 & 40 \\ 4 & 8 \end{bmatrix}.$$

But you can see by referring to display (6) that the upper left entry in  $C$  is just the first row of  $A$  multiplied term by term by the first column of  $B$  (i.e., it's the inner product of this row and column), and similarly for the other entries in  $C$ . Therefore, using our definition of matrix multiplication,  $C = AB$ . Indeed, matrix multiplication was originally defined by the mathematician Arthur Cayley (1821–1895, a leading British mathematician) just for the purpose of succinctly expressing substitutions such as the preceding.

Ordinary multiplication is commutative. That is,  $ab = ba$ . Is matrix multiplication commutative? Even when  $A$  and  $B$  are both  $n \times n$  (so that both  $AB$  and  $BA$  are defined and are the same size) the answer is No. See the following example.

### EXAMPLE 3

Let

$$A = \begin{bmatrix} 2 & 3 \\ 4 & -1 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 5 & -2 \\ 1 & 0 \end{bmatrix}.$$

Then

$$AB = \begin{bmatrix} 13 & -4 \\ 19 & -8 \end{bmatrix} \quad \text{and} \quad BA = \begin{bmatrix} 2 & 17 \\ 2 & 3 \end{bmatrix},$$

so  $AB \neq BA$ . ■

We have not shown that  $AB$  can never equal  $BA$  but just that it need not. Keep reading for some commuting matrices.

We close this section by introducing two special matrices.

1. An **identity matrix**  $I$  is a square matrix with 1's on the main diagonal and 0's elsewhere. For instance, in the  $3 \times 3$  case,

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

For each  $m$  there is an  $m \times m$  identity matrix, which can be called  $I_m$  as well as  $I$ . If you've understood our definition of matrix multiplication, you'll see that if  $M$  is  $m \times k$ , then  $I_m M = M I_k = M$ . In particular,  $I_k$  commutes with every  $k \times k$  matrix  $M$ .

2. A **zero matrix**, denoted by  $\mathbf{0}$ , is a matrix all of whose entries are 0. Clearly,  $\mathbf{0}M$  and  $M\mathbf{0}$  are zero matrices whenever they are defined.

## Problems: Section 0.5

1. (1) Evaluate

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}.$$

2. (1) Evaluate

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} - \begin{bmatrix} 6 & 4 & 2 \\ 5 & 3 & 1 \end{bmatrix}.$$

3. (1) Write down

$$4 \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}.$$

4. (1) In Example 1 what is the interpretation of the sum matrix?

5. (1) Let  $\mathbf{a} = [1 \ 2 \ 3]$  and  $\mathbf{b} = [4 \ -1 \ 6]$ . Evaluate the following.

a)  $\mathbf{a} + \mathbf{b}$     b)  $4\mathbf{a}$     c)  $\mathbf{a} \cdot \mathbf{b}$

6. (2) Let the matrix  $A$  have elements  $a_{ij} = i + j$ , and let  $B$  have elements  $b_{ij} = i \times j$ .

- a) Compute  $AB$ , if  $A$  has 2 rows and 3 columns and  $B$  has 3 rows and 2 columns.  
b) Compute  $AB$  if both  $A$  and  $B$  have 3 rows and 3 columns.

7. (1) Evaluate the following products:

a)  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

b)  $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

8. (2) On the side of a box of Yummy-Ohs, there is a chart that shows what percentage of minimum daily food requirements are provided by 1 ounce of the cereal:

protein	5%
fat	10%
carbohydrates	20%

If you eat 2 ounces, what percents do you get? What matrix and what matrix operation were involved in your calculation?

9. (2) Assume the following matrix shows the number of grams of nutrients per ounce of food indicated. (Actually, the numbers are made up.)

	meat	potato	cabbage
protein	20	5	1
fat	30	3	1
carbohydrates	15	20	5

If you eat a meal consisting of 9 ounces of meat, 20 ounces of potatoes, and 5 ounces of cabbage, how many grams of each nutrient do you get? Use matrix multiplication to obtain your answer.

10.  $\langle 1 \rangle$  If  $A$  is a column vector of  $n$  elements and  $B$  is a row vector of  $n$  elements, describe the form of  $AB$ .

11.  $\langle 1 \rangle$  Consider the four matrices

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$
$$C = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} \sin \theta & \cos \theta \\ \cos \theta & -\sin \theta \end{bmatrix}.$$

Show that the square of each one of them is  $I$ . This shows that the identity matrix, unlike the identity number, can have many square roots.

12.  $\langle 2 \rangle$  Let

$$M = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Compute  $M^2$  and  $M^3$ . What do you conjecture about the form of  $M^n$ ?

13.  $\langle 1 \rangle$  Express the following system of equations as one matrix equation.

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 4 \\ 2x_1 - 3x_2 + 4x_3 &= 5. \end{aligned}$$

14.  $\langle 2 \rangle$  Let

$$\begin{aligned} c &= b_1 + 2b_2 + 3b_3; \\ b_1 &= a_1 + 2a_2; \\ b_2 &= 3a_1 + 4a_2; \\ b_3 &= 5a_1 + 6a_2. \end{aligned}$$

Express  $c$  in terms of  $a_1$  and  $a_2$  two ways:

- a) By direct algebraic substitution.  
b) By matrix multiplication.

Show how various calculations in the two methods correspond (say, by drawing arrows between corresponding parts).

15.  $\langle 2 \rangle$  What sizes must  $A$ ,  $B$ , and  $C$  be for  $ABC$  to exist? Assuming it exists, write a formula using summation notation for the typical entry of  $ABC$ . Start by letting  $A = [a_{ij}]$ .

## 0.6 The Language and Methods of Reasoning

Our purposes in this section are to

1. Make sure you understand the key words used in mathematical reasoning, like “necessary”, “hypothesis”, and “any”;
2. Introduce some logic notation, like  $\Rightarrow$ ;
3. Discuss some basic methods of proof.

Our approach is informal. In Chapter 7 we'll introduce a formal algebra of logic and use it to (among other things) review and further clarify all the points about logic and proof we will have made.

### Implication

The most common type of sentence in a mathematical argument is an **implication**. This is anything of the form  $A$  implies  $B$ , that is, *if A then B*. For instance,

$$\text{if } x = 2, \text{ then } x^2 = 4.$$

The if-part of the sentence,  $x = 2$ , is called the **hypothesis**. Synonyms are **assumption** and **premise**. The then-part,  $x^2 = 4$ , is called the **conclusion**.

Sometimes an implication is broken into two or more sentences. Indeed, this is quite common when the implication is asserted as a theorem. For instance, you might find

Theorem. Let  $n$  be an odd integer. Then  $n^2$  is also odd.

This theorem is really the assertion of the implication

If  $n$  is an odd integer, then  $n^2$  is odd.

Also, neither the hypothesis nor the conclusion of an implication needs to be a simple statement. For instance, consider the theorem

Let  $ABC$  be a triangle. Suppose angle  $A =$  angle  $B$ . Suppose further that angle  $C = 60^\circ$ . Then  $ABC$  is equilateral. Furthermore, all three angles are  $60^\circ$ .

In this theorem the first three sentences together form the hypothesis and the last two are the conclusion.

Sometimes an implication, rather than expanding from two clauses (if and then) to several sentences, instead shrinks down to a single clause. Consider the statement

All squares are rectangles.

This is really an implication:

If a quadrilateral is a square, then it is a rectangle.

Instead of “quadrilateral”, we could have said “figure” or “something”. It turns out it doesn’t make any difference.

There is a nice notation for  $A$  implies  $B$ :

$$A \implies B.$$

Some authors write  $A \rightarrow B$  instead, but single-shaft arrows are also used for limits of sequences and functions, e.g.,  $x_n \rightarrow 0$  as  $n \rightarrow \infty$  (see our use of this notation in Sections 0.2 and 0.3). We prefer to keep the different uses distinct. In this book,  $\implies$  is a **logical operator** making a single statement from two separate statements, whereas  $\rightarrow$  is an abbreviation for the verb “approaches”.

(In fact, one can distinguish between several strengths of implication, and in some books  $\implies$  is only used for one of them. For instance,  $(x = 2) \implies (x^2 = 4)$  is a stronger implication than “if this is the Sears tower, then this is the tallest building in the world.” Although the former claim is always true, the latter switched from true to false between the first and second editions of this book. At any rate, we will not distinguish between levels of implication and we use  $\implies$  for all of them.)

One reason implications are so useful in mathematics is because they chain together so nicely. That is, if  $A \implies B$  and  $B \implies C$  are both true, then clearly so is  $A \implies C$ , and so on with longer chains. This transitivity property (see Relations in Section 0.1) means that, using implications, a mathematician can try to show that  $A$  implies  $Z$  by reducing it to many smaller steps.

Many different English phrases boil down to  $\Rightarrow$ . We have already mentioned “implies” and “if–then”. Here are some others. Each sentence below means  $A \Rightarrow B$ .

- Whenever  $A$  is true, so is  $B$ .
- $B$  follows from  $A$ .
- $A$  is sufficient for  $B$ .
- $B$  is necessary for  $A$ .
- $A$  only if  $B$ .

It’s not supposed to be obvious that all of these mean  $A \Rightarrow B$ , especially when stated so abstractly. So, first, let’s restate them with  $A$  = “It is raining” and  $B$  = “It is cloudy”.

- Whenever it’s raining, it’s cloudy.
- It follows that it’s cloudy, if it’s raining.
- It’s sufficient that it rain in order that it be cloudy.
- It is necessary that it be cloudy for it to rain.
- It’s raining only if it’s cloudy.

We hope you see that they mean essentially the same thing as “if it’s raining, then it’s cloudy”. There is a danger in using ordinary English sentences as examples. First, some of these constructions make awkward English. Second, different constructions in English (or any natural language) have different nuances, so you may not feel that these sentences mean exactly the same thing. But we hope you agree that they have the same “logical” content.

Did it surprise you that, in some of these equivalent statements, the order of appearance of  $A$  and  $B$  in the sentence changed? It may seem particularly odd that “ $A$  only if  $B$ ” should mean the same as “if  $A$ , then  $B$ ” — the “if” has moved from  $A$  to  $B$ ! But that’s the way it is — natural languages are full of oddities.

The use of “necessary” and “sufficient” is particularly common in mathematics. We’ll return to these words in a moment.

Closely related to the implication  $A \Rightarrow B$  is its **converse**,  $B \Rightarrow A$ . (This is sometimes written  $A \Leftarrow B$ .) The converse has a separate name because it does *not* mean the same thing as the original. For instance,

If it’s raining, then it’s cloudy

is true, but

If it’s cloudy, then it’s raining

is false because it can be cloudy without raining.

We hope you are already familiar with the difference between a statement and its converse. If you are told to assume  $A$  and prove  $B$ , you must demonstrate  $A \Rightarrow B$ ; if you demonstrate  $B \Rightarrow A$  instead, you have wasted your time, because  $A \Rightarrow B$  could still be false. If you start with  $A$  and end up with  $B$ , you have done the job. If instead you start with  $B$  and end up with  $A$ , you have shown  $B \Rightarrow A$  and goofed. This is called *assuming what you are supposed to show*.

There is another implication closely related to  $A \Rightarrow B$  which *does* mean the same thing: its **contrapositive**, i.e.,

$$(\text{not } B) \Rightarrow (\text{not } A),$$

or using the notation for “not” that we’ll use in Chapter 7,

$$\neg B \Rightarrow \neg A.$$

In the contrapositive, the order of statements is reversed and they are both negated. Thus the contrapositive of

If it’s raining, then it’s cloudy

is

If it’s not cloudy, then it’s not raining.

When we say an implication and its contrapositive mean the same thing, we mean that either they are both true or they are both false.

Let’s return to the words “necessary” and “sufficient”. Remember,

$A \Rightarrow B$  means the same as  $A$  is *sufficient* for  $B$  and also the same as  $B$  is *necessary* for  $A$ .

For instance, since being a square ( $A$ ) implies being a rectangle ( $B$ ), we can say that being a square is sufficient for being a rectangle and being a rectangle is necessary for being a square. “Sufficient” is a useful word because it makes clear that  $A$  need not be the only thing which guarantees  $B$  — squares are not the only rectangles — but being a square suffices. “Necessary” is a useful word because it makes clear that  $B$  may not be enough to get  $A$  — being rectangular is only part of being square — but you can’t do without it.

Since the usual goal in mathematics is to prove one thing from another, the value of knowing that  $A$  is sufficient for  $B$  is clear: Once you get  $A$  you are home free with  $B$ . But suppose instead you know that  $A$  is necessary for  $B$ . The implication is going the wrong way as far as deducing  $B$  is concerned, so what good is this knowledge of necessity? The answer is: The concept of necessity is natural and useful when you are after negative information. That is, suppose you know that  $A$  is necessary for  $B$  and you also know that  $A$  is *false*. Then you can conclude that  $B$  is false, too. In effect, you are using the contrapositive  $\neg A \Rightarrow \neg B$ .

## Definitions and Equivalence

If both  $A \Rightarrow B$  and  $B \Rightarrow A$  are true, we abbreviate by writing

$$A \Leftrightarrow B.$$

Such a statement is called a **biconditional** or **bi-implication**. We also say that statements  $A$  and  $B$  are (logically) **equivalent**. Two other ways to say  $A \Leftrightarrow B$  without symbols are

$A$  is necessary and sufficient for  $B$ ,

and

$A$  if and only if  $B$ .

There is yet another accepted shorthand for “if and only if”: iff. Should you need to pronounce iff, really hang on to the “ff” so that people hear the difference from “if”.

Equivalence is a very useful concept. Two statements are logically equivalent if they are always both true or always both false. In other words, wherever we have said that two statements mean the same thing, we could instead have said they are equivalent. For instance, an implication and its contrapositive are equivalent. Symbolically,

$$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A).$$

A **definition** is a declaration that henceforth a certain term will mean the same as the phrase that follows the term. For instance,

Definition. A square is a four-sided figure with all sides equal and all angles  $90^\circ$ .

That was a **displayed** definition. You could also write an **in-line** definition: A square is defined to be a four-sided figure with all sides equal and all angles  $90^\circ$ .

Either way, a definition is always an equivalence and could be rephrased as such: A figure is a square iff it is four-sided with all sides equal and all angles  $90^\circ$ . Logically speaking, it would be incomplete to write “if” instead of “iff” here, but we regret to inform you that, by tradition, this is exactly what is usually done. This substitution of “if” when “iff” is meant is allowed *only* in definitions. You can always tell in this book when a sentence is a definition: either the word “define” is present, or else the term being defined is printed in boldface.

Once a term is defined, it becomes a **reserved word**. It can no longer be used in the mathematical parts of your document with any other or broader meanings which it has in ordinary English. It is easy to avoid such conflict with most defined math terms; they are words that rarely occur in ordinary English.

There are some exceptions. “Equivalent” is one. In ordinary English it often means “equal”. However, in mathematics, “equal” and “equivalent” are *not* the same. For instance, the statements  $A \Rightarrow B$  and  $\neg B \Rightarrow \neg A$  are equivalent but not equal, since they are not identical. There are several other uses of equivalent in mathematics in addition to logically equivalent, but none of them mean equal.

In ordinary language, words are rarely defined but are understood through usage. In mathematics they are defined because added precision is important. A slight difference in what a word means can make the difference between whether a statement using that word is a theorem or not. Still, usage can help you understand defined terms. Whenever you read a definition, try to think up examples of things that meet the definition, and other things that don’t but almost do. For instance, upon reading the definition of “square”, first try to picture several different things that are squares – you’ll see you can only make them differ in size. Then for contrast try to picture something that has four equal sides but not four right angles; then something with four sides and four right angles, but not all sides equal.

## Quantifiers

Mathematicians are rarely interested in making statements about individual objects (2 has a square root); they prefer to make statements about large sets of objects (all nonnegative numbers have square roots). Words which indicate the extent of sets are **quantifiers**. Here are some true statements illustrating the use of the key quantifiers in mathematics. The quantifiers are in boldface.

**Every** integer is a real number.

**All** integers are real numbers.

**Any** integer is a real number.

**Some** real numbers are not integers.

**There exists** a real number which is not an integer.

**No** irrational number is an integer.

“Every”, “all”, and “any” are called **universal** quantifiers. “Some” and “there exists” are **existential** quantifiers.

Sometimes a quantifier is not stated explicitly but is understood, i.e., *implicit*. For instance, the statement

Primes are integers

or even

A prime is an integer

mean “All primes are integers.” Implicit quantifiers are especially common for algebraic statements. If a text displays

$$x + y = y + x,$$

it is asserting that the order of addition is immaterial *for all*  $x$  and  $y$ . Implicit quantifiers are almost always universal quantifiers.

“Some” has a somewhat special meaning in mathematics. Whereas in ordinary discourse it usually means “at least a few but not too many”, in mathematics it means “at least one, possibly many, or possibly all”. “There exists” has exactly this same meaning. Thus the following peculiar mathematical statements are technically correct, as explained just below:

Some even integers are primes.

Some positive numbers have square roots.

There exists a positive number with a square root.

The first statement is correct because there is exactly one even prime: 2. The other two are correct because every positive number has a square root.

Why “some” is used this way will be explained in the subsection on negation. For now let’s admit that mathematicians would rarely write the three statements displayed above. It takes no more space to tell the whole story, so we say,

There is exactly one even prime: 2.

and

All positive numbers have square roots,

or even better,

A real number has a real square root  $\iff$  it is nonnegative.

However, if all we know is that at least one “smidgit” exists, and we have no idea how many there are, then we do not hesitate to write “There exists a smidgit” (even though there might be many) or “There are some smidgits” (even though there might be only one).

Another way to state that there is just one of something is to use “the”, with emphasis. Suppose  $x = 3$  is the only solution to some problem. Then say “ $x = 3$  is *the* solution” (pronounced *thee*). If there is more than one solution, say “ $x = 3$  is *a* solution” (pronounced like capital A) or “*an* answer”. Thus, 2 is the even prime and 5 in an odd prime.

*Differences among universal quantifiers.* We have not drawn any distinctions between “every”, “any”, and “all”. However, in English usage there are sometimes differences, especially between “every” and “any”. For instance, you will surely agree that the following sentences mean the same:

Everybody knows that fact.

Anybody knows that fact.

But just as surely the following sentences don’t mean the same thing:

She’ll be happy if she wins every election.

She’ll be happy if she wins any election.

In general, “any” is a very tricky word, meaning “all” in most instances but “one” in some [12].

Why then don’t mathematicians simply avoid “any” and always use “every” or “all”? Perhaps the reason is that “any” highlights a key aspect of how to prove statements about an infinite number of objects in a finite number of words. Suppose we wish to prove that “Any square is a rectangle.” The word “any” suggests, correctly, that it suffices to pick *any one* square and show that *it* is a rectangle — so long as we don’t use any special knowledge about that square but only its general squareness. See Example 1 later. The words “every” and “all” don’t suggest this approach quite as well.

Proving for all by proving for any one is sometimes called arguing from the **generic particular**. In such a proof, it is important to check that the example really is generic — it cannot have special features, or if it does, they cannot be used in the proof.<sup>†</sup> As you read through our proofs in this text, check to see that we have

---

<sup>†</sup>Having completely general features is what “generic” meant long before it referred to discount drugs.

paid attention to this requirement. The discussion of buildup errors in Section 2.8 is particularly relevant to this concern.

Another frequently used word for generic is *arbitrary*, as in “consider an arbitrary square”.

In Section 7.6, we’ll discuss the subject of quantifiers in a more formal mathematical setting and show that many aspects of using them correctly can be handled mechanically.

## Negation

The **negation** of statement  $A$  is a statement which is true precisely when  $A$  is false. For simple statements, it is very easy to create the negation: just put in “not”. For instance,

The negation of “ $x$  is 2” is “ $x$  is not 2.”

Thus we have been using negations all along whenever we wrote (not  $A$ ) or  $\neg A$ .

You should be warned though that negating can get much trickier if the original statement contains quantifiers or certain other words. For instance, the negation of

Some of John’s answers are correct

is not

Some of John’s answers are not correct,

because it is not the case that the latter statement is true if and only if the former is false. They are both true if John gets half the questions right and half wrong. The correct negation of the first sentence is

None of John’s answers is correct.

We said earlier we would explain why “some” is used in mathematics to mean “one up to all”. The reason is so that it will mean precisely the negation of “none”. If “some” meant “a few”, it would only cover part of the situations included in the negation of “none”. Similarly, if “there exists” was limited to meaning “exactly one exists”, then it, too, would not negate “none”.

## Theorems and Proofs

Three names are commonly used for proven mathematical statements. A **theorem** is an important proven claim. A **lemma** is a proven claim which, while not so important in itself, makes the proof of a later theorem much easier. A **corollary** is a result which is quite easy to prove given that some theorem has already been proved. In other words, lemmas help to prove theorems, and theorems imply corollaries.

Strictly speaking, lemmas are unnecessary. The statement of a lemma, and its proof, could be incorporated into the proof of the theorem it is intended to help. However, including the lemma may make the proof of the theorem so long that the

key idea is obscured. Also, if a lemma is helpful for several different theorems, it clearly is efficient to state it separately.

A **conjecture** is a statement for which there is evidence or belief, but which has not been proved. Interestingly, in other sciences such a statement is instead called an hypothesis. (It is currently an hypothesis that the primary cause of global warming is human-produced hydrocarbons.) However, with rare exceptions mathematicians reserve the word “hypothesis” for the if-clause of implications, especially implications that are theorems. (The hypothesis of the Pythagorean Theorem is that  $a$  and  $b$  are the lengths of the legs of a right triangle and  $c$  is the length of the hypotenuse.)

But what is a proof? An air-tight logical demonstration that a claim is true. Of course, saying this doesn’t help much, because there many varieties of logical arguments. For now, we want you to recognize three key formats of proofs: direct, indirect, and proof by contradiction.

Suppose, as usual, that we are trying to prove  $A \implies B$ . A **direct proof** starts with the hypothesis  $A$  and ends (after carefully using implications!) with the conclusion  $B$ . An **indirect proof** starts instead with the assumption that  $\neg B$  is true (that is, not- $B$ ). How can such a start prove  $B$ ? One way is by ending up with  $\neg A$ . That is, you prove the theorem in the contrapositive form. A **proof by contradiction** is a special sort of indirect proof in which you end up with an obvious impossibility (say,  $1 = 0$ ), so the assumption  $\neg B$  must be false.

## EXAMPLE 1

Give two proofs, one direct and one indirect, for the theorem that every square is both a rhombus and a rectangle.

**Solution** First we must state this theorem as an implication, so that we know what we may assume and where we have to go. We may take the implication to be  $A \implies B$ , where  $A$  is the statement “A quadrilateral is a square” and  $B$  is “A quadrilateral is a rhombus and a rectangle.” Recall that, by definition, a rhombus is a 4-sided figure with all sides equal in length, and a rectangle is a 4-sided figure with four  $90^\circ$  angles.

Direct proof:

Let  $Q$  be a square. By definition,  $Q$  has four equal sides and four  $90^\circ$  angles. Because it has four equal sides, it is a rhombus. Because it has four  $90^\circ$  angles, it is a rectangle. Thus  $Q$  is a rhombus and a rectangle.

Indirect Proof:

Suppose  $Q$  is not both a rhombus and a rectangle. If it is not a rhombus, then some pair of sides are not equal. If it is not a rectangle, then some angle is not a right angle. Either way  $Q$  is not a square.

In the indirect proof, we have indeed proved the contrapositive,  $\neg B \implies \neg A$ .

In both proofs,  $Q$  is a generic particular. This is particularly clear in the direct proof, because of the phrase “a square”.

Also, both proofs in Example 1 were *from the definitions*, meaning you merely had to substitute the definitions of the key terms, be orderly, and you were done. Not all proofs work this way, but the first few after a new concept is introduced usually do.

## EXAMPLE 2

Use proof by contradiction to prove that there are no even primes greater than 2.

**Solution** Here is such a proof.

Suppose not (meaning, suppose the conclusion is false and thus there exists an even prime  $n > 2$ ). By definition of even,  $n = 2m$  for some integer  $m > 1$  since  $n > 2$ . Thus  $n$  is not a prime since 2 is a factor. Contradiction! ■

Indirect proofs are particularly well suited to **nonexistence** theorems, as in Example 2. If you are trying to show that an object with certain properties does not exist, assume that it does and strive for a contradiction. By assuming it does, at least you have an object to play around with.

Any indirect proof by contrapositive can be turned into a proof by contradiction by adding one more step. To prove  $A \implies B$ , you supposed  $\neg B$  and reached  $\neg A$ . Then add that, by hypothesis,  $A$  is also true. Therefore  $A$  and  $\neg A$  are both true — a contradiction. However, you need not add that additional step. A proof by contrapositive stands on its own as a type of indirect proof.

*Proofs of equivalences.* In Example 1 you probably noticed that more is true: a quadrilateral is a square *if and only if* it is a rhombus and a rectangle. The simplest way to prove  $A \iff B$  is to have two parts, a proof of  $A \implies B$  and a proof of  $B \implies A$ . Each part can be direct or indirect [15]. At the start of each part, be sure to tell the reader which part you are doing.

*Other types of proofs.* There are many ways to classify proofs. An important distinction in this book is between existential proofs and constructive proofs. Constructive proofs have an important subcase, proofs by algorithm. These three categories are discussed by examples beginning on p. 252. Another important type of proof is by mathematical induction, the subject of Chapter 2. All these types can involve either direct or indirect arguments; see [13, Section 2.2].

*Justification below the proof level.* While proof is the standard of justification in mathematics, lesser types of evidence have their place. What do we mean if we ask you to “verify” or “show” something instead of prove it?

It is a fact (shown in Chapter 4) that each  $n$ -set has exactly  $2^n$  subsets. Suppose we ask you to verify this in the case of  $n = 2$  or show it for the set  $S = \{a, b\}$ . For the latter request you would just write down all subsets of  $S$  and count that there are 4. For the former request you might say that it suffices to consider this same set  $S$  and again list its subsets. In short, for an example “verify” and “show” mean check it out in that case by any method, including brute force.

If instead we simply say “show that an  $n$ -set has exactly  $2^n$  subsets”, we are asking you to try to come up with a general argument, but we won’t be too picky.

Since we haven't discussed yet the key counting principles (to come in Chapter 4), we can't expect you to write up your reasoning very formally. Moreover, if you don't see any general argument, it's OK to show what evidence you can. You could write out all the subsets for  $n = 1, 2, 3$  and observe that the number is indeed doubling each time.

Sometimes we leave steps out of a proof in the text. If we then say "verify the details", that means fill them in.

## Trivial, Obvious, Vacuously True

Many people think "trivial" and "obvious" mean the same thing. However, writers of mathematics use them differently. An argument is **trivial** if every step is a completely routine calculation (but there may be many such steps). A claim is **obvious** if its truth is intuitively clear (even though providing a proof may not be easy). For instance, it's trivial that

$$(x + 2)(3x + 4)(5x^2 + 6x + 7) = 15x^4 + 68x^3 + 121x^2 + 118x + 56,$$

but it's hardly obvious. On the other hand, it's obvious that space is three-dimensional but the proof is hardly trivial.

Students sometimes think that writers or lecturers who use these words are trying to put them down, and perhaps sometimes they are. But, used appropriately, these words are useful in mathematical discourse and need not be threatening.

Sometimes a claim is made about something that doesn't exist. Then mathematicians consider that claim to be **vacuously true**. How could such claims be of any interest and why declare them to be true?

Here is the archetype example. Let  $S$  be any set. Is the empty set  $\emptyset$  a subset? Well, by definition of subset, we have to check whether

$$\text{Every element of } \emptyset \text{ is an element of } S.$$

Since  $\emptyset$  has no elements and thus provides no examples to contradict the claim, mathematicians declare the claim to be vacuously true.<sup>†</sup>

Now, one doesn't naturally think about the empty set when looking at subsets, so why it is useful to insist that it is a subset? Well, if  $A \subset S$  and  $B \subset S$ , should it be true that  $A \cap B \subset S$ ? Of course; this is obvious from a Venn diagram [20]. But what if  $A \cap B = \emptyset$ ? If the empty set wasn't a subset of every set, then this little theorem about the intersection of subsets would have an exception. Or rather, it would have to be stated more carefully: *nonempty* intersections of subsets are subsets. It turns out that by declaring vacuous statements to be true, all sorts of annoying little exceptions are avoided.

Here is another example. Suppose we have an algorithm for taking a list of numbers and putting them in increasing order. In order means: for any two distinct elements  $x, y$  of the list, if  $x < y$ , then  $x$  precedes  $y$ . Notice that this definition

---

<sup>†</sup>Technically speaking, we are considering the implication  $x \in \emptyset \implies x \in S$ , in which the hypothesis is false for every  $x$ . An implication  $A \implies B$  is considered to be true whenever  $A$  is false, as well as when  $A$  and  $B$  are true. All this is discussed further in Chapter 7.

is vacuous if the list has only 0 or 1 elements — there are no distinct pairs  $x, y$ . Suppose we want to state a theorem that our algorithm works correctly. If vacuous statements were considered to be false or meaningless, our theorem would have to say that the algorithm works *when given a list of at least two numbers*. However, with vacuous statements declared to be vacuously true, we don't have to include the italicized caveat.

Finally, sometimes a claim is true not because it is about nothing, but because nothing has to be done to prove it. Such claims are also, sometimes, called vacuously true. Here is a common situation. Suppose an algorithm takes some positive number and does some operation repeatedly, say squaring, and we want to argue that the result is still positive. A standard approach is to restate the claim as “After squaring  $k$  times, the result is still positive.” Next we would proceed to prove the claim for all  $k$  by the method of induction (Chapter 2). We often choose to start with the case  $k = 0$ , because it is easy. The original number is “still” positive after squaring 0 times because nothing has been done to it yet. The number exists, but one sometimes says that it is vacuously true that the number is still positive.

## And and Or

In mathematical expressions “and” may be implicit. For instance, both

$$A = \{(x, y) \mid x > 0, y > 0\} \quad \text{and} \quad A = \{(x, y) \mid x, y > 0\}$$

mean the set of all points  $(x, y)$  with  $x$  *and*  $y$  greater than 0, i.e., the first quadrant. If you want  $x$  *or*  $y$  greater than 0, you have to say so explicitly:

$$C = \{(x, y) \mid x > 0 \text{ or } y > 0\}. \tag{1}$$

Be careful about “or”. In English it is often used to mean that exactly one of two possibilities is true:

Either the Democrats or the Republicans will win the election. (2)

On other occasions “or” allows both possibilities to hold:

You can get there via the superhighway *or* by local roads.

The first usage is called the **exclusive or**, the second the **inclusive or**. In mathematics, “or” is always inclusive unless there is an explicit statement to the contrary. Thus  $C$  in Eq. (1) is all points except those in the third quadrant (more precisely, except those with  $x, y \leq 0$ ). If for some strange reason statement (2) was the object of mathematical analysis, it would have to be prefaced by “exactly one of the following holds” or followed by “but not both”.

# Problems: Section 0.6

1. (1) Restate each of these in if-then form:
  - a) If it's Tuesday, this must be Belgium.
  - b) When it's hot, flowers wilt.
  - c) To snow it must be cold.
2. (1) Explain the meaning of the words
  - a) implication
  - b) hypothesis
  - c) equivalence
  - d) negation.
3. (2) Find an implication which is
  - a) true but its converse is false;
  - b) false but its converse is true;
  - c) true and its converse is true;
  - d) false and its converse is false.
4. (2) Which of the following statements are true? Why?
  - a) To show that  $ABCD$  is a square, it is sufficient to show that it is a rhombus.
  - b) To show that  $ABCD$  is a rhombus, it is sufficient to show that it is a square.
  - c) To be a square it is necessary to be a rhombus.
  - d) A figure is a square only if it is a rhombus.
  - e) A figure is a square if and only if it is a rhombus.
  - f) For a figure to be a square it is necessary and sufficient that it be an equilateral parallelogram.
5. (2) State the converse for the following.
  - a) If  $AB \parallel CD$ , then  $ABCD$  is a parallelogram.
  - b) If a computer program is correct, it will terminate.
6. (2) State the contrapositive for the sentences in [5].
7. (3) The *inverse* of  $A \implies B$  is the statement  $\neg A \implies \neg B$ .
  - a) Give an example to show that a statement can be true while its inverse is false.
  - b) Give an example to show that a statement can be false while its inverse is true.
  - c) What statement related to  $A \implies B$  is its inverse logically equivalent to? Explain why.
8. (2) Which of the following statements are logically equivalent? Why?
  - (i)  $x = 3$ .
  - (ii)  $x^2 = 9$ .
  - (iii)  $x^3 = 27$ .
  - (iv)  $x$  is the smallest odd prime.
  - (v)  $|x| = 3$ .
9. (2) Find a sentence in this section where we used the convention that in a definition “if” may be used where “iff” is meant.
10. (2) Which of the following are true? Interpret the statements using the conventions of mathematical writing.
  - a) Some rational numbers are integers.
  - b) Some equilateral triangles are isosceles.
  - c) Some nonnegative numbers are nonpositive.
  - d) Some negative numbers have real square roots.
  - e) There exists an odd prime.
  - f) There exists a number which equals its square.
  - g) Either  $\pi < 2$  or  $\pi > 3$ .
  - h) Not all primes are odd.
11. (2) Which pairs from the following statements are negations of each other?
  - (i) All mammals are animals.
  - (ii) No mammals are animals.
  - (iii) Some mammals are animals.
  - (iv) Some mammals are nonanimals.
  - (v) All mammals are nonanimals.
  - (vi) No mammals are nonanimals.
12. (2) In some of the following sentences, “every” can be replaced by “any” without changing the meaning. In others, the meaning changes. In still others the replacement can’t be made at all — the resulting sentence isn’t good English. For each sentence, decide which case applies. (When it comes to what is good English, there may be disagreements!)
  - a) He knows everything.
  - b) Everybody who knows everything is to be admired.
  - c) Everybody agreed with each other.
  - d) Not everybody was there.
  - e) Everybody talked at the same time.
  - f) Every square is a parallelogram.

- 13.** ⟨2⟩ There is always a way to negate correctly a sentence using *not*. What is it? (*Hint:* Use “not” as part of a phrase at the beginning of the sentence.)
- 14.** ⟨2⟩ Is the following a valid proof that “an integer is divisible by 9 if the sum of its digits is divisible by nine”?
- Proof: Consider such a number  $n$  in the form  $10A + B$ . Since  $10A + B = 9A + (A + B)$  and since  $9|(A + B)$  by hypothesis, then  $9|n$ .
- The next several problems ask you to prove things. Many of the things are obvious, and we wouldn’t normally ask for written proofs. The point is to practice proof styles on toy examples.
- 15.** ⟨2⟩ Example 1 proved half of the theorem “A quadrilateral is a square  $\iff$  it is a rhombus and a rectangle. Prove the other half
- by a direct argument,
  - by an indirect argument.
- 16.** ⟨2⟩ Prove that there is no biggest integer by proving “For every integer  $n$  there is a bigger integer.” Start by picking *any* integer  $n$  and naming a bigger integer using the letter  $n$ . (By using  $n$ , rather than an example, your argument is generic.)
- 17.** ⟨3⟩ Let  $O$  be the set of odd numbers, and let  $S$  be the set of all integers that may be expressed as the sum of an odd number and an even number.
- Show that  $S \subset O$ . *Hint:* By definition of subset, we must show that every element of  $S$  is an element of  $O$ . So pick a generic particular number in  $S$  and show that it is odd.
  - Show that  $S = O$ . One standard method for doing this is to show both  $S \subset O$  and  $O \subset S$ . Thus, you need a second generic particular argument beyond that in a).
- 18.** ⟨2⟩ Prove that there does not exist a number that solves  $x = x + 1$ . Remember, contradiction is usually the best way to prove nonexistence.
- 19.** ⟨3⟩ Prove in several styles that  $3x+4 = 10$  if and only if  $x = 2$ . Use the labels “If” and “Only if”; use arrows; do direct proofs; do indirect proofs.
- 20.** ⟨2⟩ Show with a Venn diagram that if  $S, T \subset U$  then  $S \cap T \subset U$ .
- 21.** ⟨2⟩ Give a written proof that
- $$[S, T \subset U] \implies [S \cap T \subset U].$$
- Start by picking a generic element in  $S \cap T$ ; show it is in  $U$ .
- 22.** ⟨2⟩ Can the theorem in [21] be strengthened by weakening the hypothesis? Suppose only  $S$  is known to be a subset of  $U$ ;  $T$  can be any set. Must  $S \cap T$  still be subset of  $U$ ? Give a proof or **counterexample** — an example that shows the claim to be wrong.
- 23.** ⟨2⟩ Which of the following are vacuously true; which not? Explain. When a statement is not vacuous, is it trivial? Obvious? Explain briefly.
- For all integers  $k$  such that  $1 \leq k \leq 0$ ,
$$2k^3 + 2k^2 - 3k = 1.$$
  - For all integers  $k$  such that  $1 < k \leq 2$ ,
$$k^9 - 3k^7 + 4k^5 - 30k^3 = 16.$$
  - Every prime which is a perfect square is greater than 100.
  - The case  $n = 2$  of “The sum of the degrees of the interior angles of every planar  $n$ -gon is  $180(n - 2)$ .”
  - The same proposition as part d) when  $n = 3$ .
  - The case  $n = 1$  of: If all  $a_i > 0$ , then

$$\left( \sum_{i=1}^n a_i \right)^2 \geq \sum_{i=1}^n a_i^2.$$

# Supplementary Problems: Chapter 0

1. ⟨2⟩ The **greatest common divisor**,  $\gcd(m, n)$ , of two integers is the largest positive integer which can be divided into both without leaving a remainder. In Chapter 1 we'll derive a method for computing the greatest common divisor of two integers. For this problem, however, just use any (brute force) method you wish. Find the gcd of the following pairs of integers.

- a) 315 and 91    b) 440 and 924

2. ⟨2⟩ The **least common multiple**,  $\text{lcm}(m, n)$ , of two nonzero integers is the smallest positive integer into which both  $m$  and  $n$  can be divided without leaving a remainder. Find the lcm of the following pairs of integers.

- a) 6 and 70    b) 33 and 195

3. ⟨3⟩

a) Find the greatest common divisor of each pair of integers in [2].

b) Using the results of part a) and those of [2], conjecture what the product of the gcd and lcm of two integers always is.

c) Try to prove your part b) conjecture.

4. ⟨2⟩ The function  $\log\log x$  is defined by  $\log\log x = \log(\log x)$ .

a) What is the domain of  $\log\log$ ? The codomain?

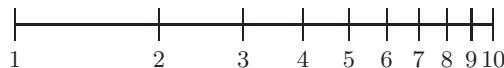
b) Show that  $\log\log(c^k) = \log k + \log\log c$ .

5. ⟨3⟩ For some time after logarithms were invented, scientists did computations with logarithms by looking them up from books of tables. Long ago the slide rule eliminated the need to use logarithms for computation except where accuracy to more than 2 or 3 decimal places was needed. Only in the last quarter of the 20th century did hand calculators eliminate the slide rule.

This problem is for people who can find an old slide rule in the attic and want to figure out how it worked.

a) Numbers were multiplied on slide rules by moving two “log scales” relative to each other. On a log scale, each number  $x$  is marked at the distance  $k \log_{10} x$  from the left of the scale, where  $k$  is the length of the entire scale (see the scale

in the figure). Explain how Property 1 allows us to multiply numbers by using such scales.



Since  $\log_{10} 3 = .477$ , the point labeled 3 is .477 of the way from 1 to 10.

- b) Explain how you could find  $c^k$  by moving a “loglog scale” relative to a log scale. Hint: Use the result from [4].
6. ⟨3⟩ A value  $c$  is called a **fixed point** of the function  $f$  if  $f(c) = c$ .

a) Find all the fixed points of the function  $f(x) = 3x + 8$ .

b) Find all the fixed points of the function  $f(x) = |x|$ .

c) Let  $U = \{1, 2, 3, 4\}$  and let  $f$  be defined on subsets of  $U$  by  $f(A) = U - A$ . Find all the fixed points of  $f$ . (Note: In this case “points” means “set”.)

d) Let  $f$  have as its domain the set of strings of lowercase roman letters, and let  $f$  map each string into its reverse. For instance,  $f(xyz) = zyx$  and  $f(\text{love}) = \text{evol}$ . Name a fixed “point” of length 3. Name a fixed point of length 4.

7. ⟨3⟩ Let

$$H_n = \sum_{k=1}^n \frac{1}{k}.$$

Let

$$O_n = \sum_{\substack{k=1 \\ k \text{ odd}}}^n \frac{1}{k}.$$

For instance,

$$O_{10} = \frac{1}{1} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \frac{1}{9}.$$

Show that  $O_n = \text{Ord}(H_n)$ .

8. ⟨4⟩ Find a general formula for

$$\sum_{S \subset [n]} |S|,$$

with  $[n] = \{1, 2, \dots, n\}$ . Hint: You can write this as the double sum  $\sum_{S \subset [n]} \sum_{i \in S} 1$ ; then try changing the order of summation.

9. **(3)** Suppose that you have two algorithms for the same task. Let the execution time (in seconds) for the first algorithm be  $n^2/10 + n/5 + 1/3$  and let that for the second be  $50n \log_2 n$ .
- Calculate the execution time for each algorithm for  $n = 2^k$ ,  $k = 0, 1, 2, \dots, 15$ .
  - Suppose that you knew that you had to do the task for which these two algorithms were written for 10 different but unknown (because they might depend on some other, not yet performed calculation) values of  $n$  which, with equal probability, might be any value from 1 to 50,000. Which algorithm would you choose?
10. **(3)** Let  $f(x) = x^2 - 3x + 2$ .
- Show that  $f(-x) = x^2 + 3x + 2$ .
  - Show that if  $c$  is a root of  $x^2 - 3x + 2 = 0$ , then  $-c$  is a root of  $x^2 + 3x + 2 = 0$ .
  - Generalize part a). If  $f(x) = \sum_{k=0}^n a_k x^k$ , what is  $f(-x)$ ? (If you can't figure out how to write this with summation notation, write it with dots.)
  - Generalize part b). If  $f(c) = 0$ , what equation does  $-c$  solve?
11. **(3)** Let  $f(x) = \sum_{k=0}^n a_k x^k$ , where all the  $a_k$  are integers.
- Show: If 0 is a root of  $f(x) = 0$ , then  $a_0 = 0$ .
  - Show: If  $f(c) = 0$ , where  $c$  is an integer, then  $c|a_0$ . (*Hint:* Subtract  $a_0$  from both sides of the identity  $f(c) = 0$ .)
12. **(3)**
- Verify that the zeros of  $2x^2 - x - 6$  are the reciprocals of the zeros of  $-6x^2 - x + 2$ .
  - Show: If  $x \neq 0$  is a root of  $\sum_{k=0}^n a_k x^k = 0$ , then  $1/x$  is a root of  $\sum_{k=0}^n a_{n-k} x^k = 0$ .
13. **(3)** Descartes's **rule of signs** says that the number of positive zeros of the polynomial  $P_n(x)$ , as given by Eq. (6) of Section 0.2, is the number of variations in sign (from plus to minus and vice versa) in the sequence  $a_0, a_1, \dots, a_n$  (with zeros ignored) or less than this by an even number.
- State an analogous rule for the number of negative zeros.
  - Deduce a rule for the number of real zeros.
14. **(2)** Apply Descartes's rule and your results in [13] to find an upper bound on the number of positive, negative, and real zeros of
- $P_5(x) = x^5 + 3x^4 - x^3 - 7x^2 - 16x - 12$ ;
  - $P_{10}(x) = x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$ ;
  - $P_6(x) = x^6 - 3x^4 + 2x^3 - x + 1$ .

# Algorithms

## 1.1 Some Examples of Algorithms

---

Mathematics requires both precision of thought and precision of language. To do mathematics by yourself, precise thought may, perhaps, be sufficient. But to communicate mathematics to others, precise language is essential. In this chapter we'll focus on one of the most important entities used to communicate mathematical thoughts: the *algorithm*. Algorithms are a central theme in this book. They are crucial for solving many sorts of problems and interesting in themselves. Because algorithms provide a precise way to communicate mathematical thoughts, we can use mathematics to analyze them and to determine if they are correct.

In this section we introduce the basics of our **algorithmic language** (often hereafter AL for short) through examples, along the way defining just what an algorithm is and suggesting the sorts of issues one wants to study about them. We think you will easily understand the AL used in these examples. In Section 1.2 we'll discuss some features of AL not introduced in this section. In Section 1.3 we'll introduce the subject of recursive algorithms and show how our algorithmic language is used to express recursive algorithms. Recursive algorithms can be subtle and you may find them difficult at first. But they are needed with the inductive and recursive paradigms that were discussed in the Prologue and that will appear and reappear throughout this book. Then in Section 1.4 we shall formalize the AL features introduced in Section 1.3. Finally, in Section 1.5 we view algorithms as mathematical objects when we introduce the analysis of algorithms.

We assume that you have used a computer or calculator language. Our algorithmic language is simpler than computer or calculator languages in that we allow full mathematical symbolism and occasional English, and we avoid issues such as declaring variable types. Remember that AL is meant for precise communication among *humans*.

Algorithms have a long history. The *word* algorithm itself derives from the name of a ninth-century Baghdad mathematician al-Khorezmi.<sup>†</sup> However, the *notion* of an algorithm goes back much further, at least to Euclid around 300 B.C. Euclid's algorithm for finding the greatest common divisor of two integers is presented in Example 4 below.<sup>‡</sup> In any event, it was only with the advent of high-speed computation in the second half of the 20th century that algorithms became so central that precise ways for describing them became essential.

We begin with an example that, although simple, will serve to illustrate a number of features of algorithms and AL.

## EXAMPLE 1

Given  $x$ , a real number, and  $m$ , a positive integer, compute  $x^m$ .

**Solution** First in Algorithm 1.1 POWERA we use brute force.

### Algorithm 1.1

#### PowerA

<b>Input</b> $x, m$ <b>Output</b> <i>power</i> <b>Algorithm</b> POWERA $power \leftarrow x$ <b>for</b> $i = 2$ <b>to</b> $m$ $    power \leftarrow power \times x$ <b>endfor</b>	$[m \text{ an integer } \geq 1, x \text{ real}]$ $[= x^m]$ $[\text{If } m = 1, \text{ skip for-endfor}]$
--	--

What could be simpler? To compute the  $m$ th power of  $x$  just multiply  $x$  by itself  $m - 1$  times, which is what happens in the body of the for-loop in POWERA. A **loop** is an **iteration structure** in which the statements between the **delimiters** (**for** and **endfor** in POWERA) are executed repeatedly. In this for-loop (we'll introduce other ways of using for-loops later), termination is controlled by a **counter**, in this case  $i$ , which goes up by 1 each time you finish the loop statements and return to the top. The left arrow ( $\leftarrow$ ) denotes assignment: The variable on the left gets a new value, namely the current value of the quantity on the right. Specifically, each time the line  $power \leftarrow power \times x$  is invoked, *power* gets a new value, which is  $x$  times its old value. Since the original value of *power* was  $x$ , at the end of each iteration  $power = x^i$ .

Is POWERA efficient? That is, can the computation be done with fewer than the  $m - 1$  multiplications required by POWERA? Our next algorithm, POWERB, answers this question for the case when  $m$  is a power of 2.

This time we start with  $x$ , compute  $x^2$  (during the first time through the repeat-loop),  $x^4$  (the second time through the repeat-loop) ... until we reach  $x^m$ .

---

<sup>†</sup>One of his books gives algorithmic procedures for solving familiar linear and quadratic equations; today's symbols and formulas for them came centuries later. He is also responsible for the word *algebra*.

<sup>‡</sup>This algorithm has proved so efficient that it plays a role today in some of the coding algorithms for secure e-commerce.

## Algorithm 1.2

### PowerB

<b>Input</b> $x, m$	$[m = 2^j \text{ for some nonnegative integer } j]$
<b>Output</b> $power$	$[= x^m = x^{2^j}]$
<b>Algorithm</b> POWERB	
$power \leftarrow x; k \leftarrow 1$	$[k \text{ is a counter}]$
<b>repeat until</b> $k = m$	
$power \leftarrow power \times power$	
$k \leftarrow 2k$	
<b>endrepeat</b>	

The repeat-loop in POWERB is another iteration structure, in which the statements between **repeat** and **endrepeat** are executed repeatedly (!) until the first time that the condition after **until** is true upon return to the top of the loop. Note, in particular, that this algorithm works even when  $m = 1$  (i.e.,  $j = 0$ ) since the condition  $k = m$  is satisfied immediately and the two statements in the *body* of the loop are never executed. In Section 1.2 we shall give some other forms for **repeat–endrepeat** loops.

How many multiplications are required this time? The answer is  $\log_2 m$  (why?), which is less than  $m - 1$  for all  $m > 2$ .

But suppose  $m$  is not a power of 2. Can we still improve on Algorithm POWERA? Algorithm 1.3 POWERC answers this question.

## Algorithm 1.3

### PowerC

<b>Input</b> $x, m$	$[m \text{ an integer and } \geq 1]$
<b>Output</b> $power$	$[= x^m]$
<b>Algorithm</b> POWERC	
$power \leftarrow x; k \leftarrow 1$	$[m \geq k \text{ since } m \geq 1]$
<b>repeat until</b> $2k > m$	$[\text{Until } k \text{ is biggest possible power of } 2 \leq m]$
$power \leftarrow power \times power$	
$k \leftarrow 2k$	
<b>endrepeat</b>	
<b>repeat until</b> $k = m$	
$power \leftarrow power \times x$	
$k \leftarrow k + 1$	
<b>endrepeat</b>	

Algorithm POWERC uses successive doublings as in POWERB as long as it can and then multiplies successively by  $x$  until  $power = x^m$ . We leave the number of multiplications required to a problem [2a].

Perhaps you see how to improve Algorithm POWERC further; see [2c]. ■

We hope you found the algorithms in Example 1 easy to read. In any case, we discuss AL constructs further in Section 1.2.

Even if you haven't encountered the term "algorithm" before, we hope that Example 1 gave you an intuitive feeling for what this term means. But for a term that will play such an important role in this book we should be more specific. Any algorithm must have the following properties:

1. Finiteness. Application of the algorithm to a particular set of data must result in a *finite* sequence of actions.
2. Uniquely determined initial step. The action which starts off the algorithm must be unambiguously determined. This is true in AL because we always begin with the line following the line with the name of the algorithm.
3. Uniquely determined succession. Each action in the algorithm must be followed by only one valid successor action.
4. Solution. The algorithm must terminate with a solution to the problem it purports to solve, or it must indicate that, for the given data, the problem is insoluble by this algorithm.

If it's not clear already, convince yourself now that the algorithms in Example 1 satisfy these four properties. For instance, what does the *input criterion* that  $m$  is a integer have to do with meeting the first of these properties [1]? Usually, it is the last of the four properties which is the most difficult to verify.

Thus, an algorithm is nothing more — or less — than a precise recipe for carrying out a sequence of operations to solve a problem. The algorithmic language used in this book is intended to enable us — and you — to present algorithms precisely and unambiguously.

In the remainder of this section we shall present three further examples of algorithms to give you additional experience with algorithmic language and to illustrate the variety of problems amenable to an algorithmic solution.

## EXAMPLE 2

### Prime Numbers

A prime number is a positive integer greater than 1 which is divisible only by 1 and itself. Thus 2, 3, 5, 7, 11, ... are prime, but 4, 6, 8, 9, 10, 12, ... are not. Our objective in this example is to generate the first  $K$  prime numbers, where  $K$  is some given positive integer. We could generate them by taking each integer  $i$  and trying to divide it by

$$2, 3, 4, \dots, i-1.$$

But this approach is clearly wasteful; since all primes after 2 are odd, it's enough to test only odd integers and to try to divide them by

$$3, 5, 7, 9, 11, \dots, i-2. \tag{1}$$

For all odd integers  $> 2$ , if  $i$  is not divisible by any integer in (1), then  $i$  is prime. These observations form the basis of Algorithm 1.4, PRIMENUM.

## Algorithm 1.4

### PrimeNum

<b>Input</b> $K$	[Number of primes]
<b>Output</b> $p_k, k = 1, \dots, K$	[First $K$ primes]
<b>Algorithm</b> PRIMENUM	
$p_1 \leftarrow 2$	[First prime]
$k \leftarrow 2$	[Counter for primes]
$i \leftarrow 3$	[First integer to test]
<b>repeat until</b> $k > K$	
$j \leftarrow 3$	[First test divisor]
<b>repeat until</b> $j = i$	
<b>if</b> $\lfloor i/j \rfloor \times j = i$ <b>then exit</b>	[Exit if $i$ has divisor]
$j \leftarrow j + 2$	[Next test divisor]
<b>endrepeat</b>	
<b>if</b> $j = i$ <b>then</b> $p_k \leftarrow i; k \leftarrow k + 1$	[Prime found]
$i \leftarrow i + 2$	[Next integer to be tested]
<b>endrepeat</b>	

A key expression in this algorithm is

$$\lfloor i/j \rfloor \times j, \quad (2)$$

where  $j$  is an integer,  $\times$  represents multiplication, and the floor function is as defined in Section 0.2. This expression is equal to  $i$  if and only if  $i$  is divisible by  $j$  (i.e., there is no remainder when  $i$  is divided by  $j$ ). If you don't see this, go back and review the meaning of the floor function.

One new aspect of AL introduced in this example is the **selection** statement **if–then**. If the condition before **then** is true, then the statement after **then** (i.e., **exit**) is executed; otherwise not. In Section 1.2 we shall discuss a more general form of the selection statement.

Another new aspect of AL in this example is the **exit** statement, which has the effect of passing control to a point directly *after* the innermost loop in which the **exit** is contained (i.e., to the line **if**  $j = i \dots$  in Algorithm PRIMENUM).

Table 1.1 contains the results of using PRIMENUM when  $K = 7$ . Such a table is called a **trace** of an algorithm.

A few remarks are in order:

1. When, for example,  $k = 5$ ,  $i$  is initially 9 since  $p_4 = 7$ , but since  $j = 3$  divides 9 evenly, the **exit** is taken and  $i$  is increased to 11 ( $k$  isn't increased because  $j$  isn't equal to  $i$ ). Then successive values of  $j$  from  $j = 3$  to  $j = 9$  do not divide  $i$ . Finally,  $j$  is set equal to 11 and the **exit** is taken, this time with  $i = j$ .

**TABLE 1.1**  
**Results of Algorithm PRIMENUM when**  
 **$K = 7$ .**

$k$	$i$	$j$	$p_k$	$k$	$i$	$j$	$p_k$
2	3	3	3				7
3	5	3					9
		5	5				11
4	7	3					13
		5		7	15	3	
		7	7		17	3	
5	9	3					5
	11	3					7
		5					9
		7					11
		9					13
		11	11				15
6	13	3					17
		5					17

*Note:* The values of  $i$  and  $j$  are those taken on for each value of  $k$  inside the outer loop.

2. We could have written the first line of the inner loop as

**repeat until**  $j = i$  **or**  $\lfloor i/j \rfloor \times j = i$

and thereby avoided the next statement with the **exit** entirely. We didn't do it this way because the essentially different nature of the two criteria (i.e., whether  $j$  equals  $i$  and whether  $j$  divides  $i$ ) argues, for reasons of readability, against including them in the same test. (The **or** operator is an *inclusive or* as discussed on p. 67: If either the expression preceding it or following it (or both of them) is true, then the entire expression containing **or** is true.)

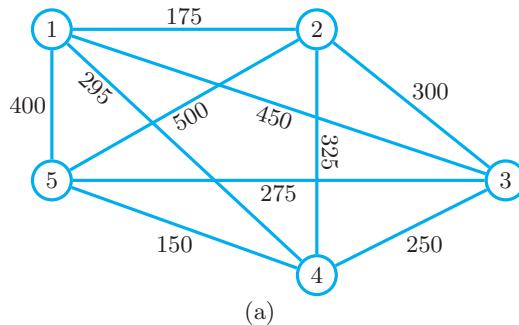
3. This algorithm is correct; that is, it does, indeed, do the job of finding the first  $K$  primes. However, it does the task very inefficiently because it performs many more computations than are necessary. Can you see ways to improve it? ([34, Section 1.2]) ■

### EXAMPLE 3

#### Building a Fiber Optics Network

Let Thirdworld be an underdeveloped country that wants to build a fiber optics network to facilitate Internet access. Suppose it decides to start by connecting all its major cities at the minimum possible cost. To build such a network, you don't have to have direct cabling (a *link*) between every pair of cities. All you need is a *path* between any two cities (which may pass through some intermediate city or

cities). We'll assume that the only costs we need to consider are the cost of the optical fibers and the construction cost between cities. (Each city will need a local fiber optics network independently of how we connect the cities with optical fiber cables.) Suppose also that the costs for each possible link between two cities has been estimated as shown in Fig. 1.1(a). Then what is the most economical way to build the network? That is, what pairs of cities should be linked directly?



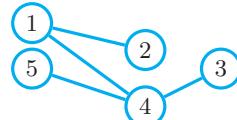
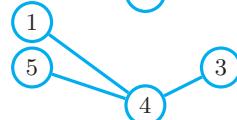
Edge	Cost (= Distance)
$(4, 5)$	150

$(3, 4)$	250
----------	-----

$(1, 4)$	295
----------	-----

$(1, 2)$	175
----------	-----

Tree



(b)

**FIGURE 1.1**

A network algorithm, showing a) a map of the cities in Thirdworld and the costs to build a fiber optics connection between them, and b) how the algorithm works for this map.

Our algorithm is based on the almost simple-minded idea that we should start by joining those two cities whose link is the least costly of all. Then at each subsequent stage we build the link which is least costly among all links from previously joined cities to cities not yet part of the network. Algorithms of this type, which are, so to speak, based on immediate gratification, are called **greedy algorithms**. In this case we may state our idea in algorithmic form as in Algorithm 1.5 FIBERNET.

Note our use of the set notation; the links, represented as lines joining two cities in Fig. 1.1b, are best represented in the algorithm as unordered pairs of cities.

Let's see how Algorithm FIBERNET works for the "map" of Fig. 1.1a. The

## Algorithm 1.5

### FiberNet

<b>Input</b> $n$	[Number of cities]
$c_{ij}, i, j = 1, 2, \dots, n$	$[c_{ij}$ is cost of the link from city $i$ to city $j]$
<b>Output</b> $T$	[The set of pairs of cities to be joined in the network]
<b>Algorithm</b> FIBERNET	
Find $c_{lm}$ , the minimum cost for a link between any two cities; if there is a tie, choose arbitrarily any link of minimum cost	
$S \leftarrow \{l, m\}$	[Initialize $S$ , the set of cities in the network]
$T \leftarrow \{\{l, m\}\}$	[Initialize $T$ , the set of <i>pairs</i> of cities joined together in the network]
<b>repeat until</b> $ S  = n$	[That is, until all cities are in the network]
Find $k \notin S$ and $q \in S$ such that link $\{k, q\}$ has minimum cost among all links joining a city in $S$ to one not in $S$	
$S \leftarrow S \cup \{k\}$	[Add $k$ to $S$ ]
$T \leftarrow T \cup \{\{k, q\}\}$	[Add $\{k, q\}$ to $T$ ]
<b>endrepeat</b>	

linkages are illustrated in Fig. 1.1b, which shows how each city is successively added to the network. It is worth emphasizing that, at each step, we do not add the cheapest link not already in the network but rather the cheapest link from a city not in the network to a city which is in the network. For instance, after we picked link  $\{4, 5\}$ , we did not pick  $\{1, 2\}$ , even though it is the cheapest link not yet used, because it does not include a city already in the network. Also, after we picked  $\{3, 4\}$ , we picked  $\{1, 4\}$  instead of the cheaper  $\{3, 5\}$  because both 3 and 5 are already in the network. The final network shown at the bottom of Fig. 1.1b is called, technically, *a minimum spanning tree* for the *graph* in Fig. 1.1a. Don't worry about this terminology now. We'll come back to it in Section 3.7, where we'll discuss this algorithm again. ■

Algorithm FIBERNET does not contain any new AL constructs but it does, for the first time, contain some narrative English. When we haven't developed more formal tools or when it's just plain convenient to do so, we will not hesitate to use English in our algorithms. Our main objective is to make algorithms readable and understandable.

How convincing do you find FIBERNET? We hope you're convinced that it does result in *a* network which enables all cities to communicate with all others. But can such a simple-minded idea result in the *best* (i.e., minimum cost) network? We'll return to this question in Section 3.7. We hope that you wouldn't expect greedy choices to be best all the time. Indeed, we all know cases where taking the path of least resistance gets us into trouble later on (e.g., see [9]). A vital concern in the study of algorithms is analyzing them to see how close they come to being the best possible. We did this a bit in Example 1 and will return to this topic in Section 1.5.

## EXAMPLE 4

### The Euclidean Algorithm

The **greatest common divisor** of two integers  $m$  and  $n$ , not both zero, written  $\gcd(m, n)$ , is the largest positive integer that divides them both without remainder. For example,  $\gcd(6, 9) = 3$ ,  $\gcd(4, -6) = 2$ ,  $\gcd(5, 5) = 5$ , and  $\gcd(10, 0) = 10$ . (Remember that any nonzero integer divides 0 without remainder.)

The case  $m = n = 0$  is excluded above because the definition doesn't work for that pair: every positive integer divides into 0 without remainder, so there is no largest common divisor. Therefore, we make a special definition in this case:  $\gcd(0, 0) = 0$ . Thus, it follows that  $\gcd(m, 0) = \gcd(0, m) = m$  for all nonnegative integers. We will use this fact repeatedly as it will serve as a "basis" for several proofs.

While we are mostly interested in the gcd when  $m, n$  are positive and distinct, we will shortly see a use for allowing them to be 0, and later a use for allowing them to be negative.

The Euclidean algorithm is a method for calculating the gcd. It is an approach based on the recursive paradigm introduced in the Prologue. Suppose  $m \geq n$ . We lose no generality with this assumption because  $\gcd(m, n) = \gcd(n, m)$ . We will calculate  $\gcd(m, n)$  by reducing it to a "previous" case, that is, to a calculation of  $\gcd(s, t)$ , where  $(s, t)$  is "smaller" than  $(m, n)$  in the sense that  $t < n$ .

To discover how to reduce the calculation of  $\gcd(m, n)$  to the calculation of the gcd of a smaller pair, we first note that any common factor of  $m$  and  $n$  is also a factor of  $m - n$ . For example, 45 divides both 945 and 225, so 45 divides  $945 - 225 = 720$ . Also, any common factor of  $n$  and  $m - n$  is a factor of  $m$ , since  $m = n + (m - n)$ . Thus  $(m, n)$  and  $(m - n, n)$  have the same common factors. In particular, they have the same largest factor, that is, the same gcd. Therefore,  $\gcd(m, n) = \gcd(m - n, n)$ . Thus,  $\gcd(945, 225) = \gcd(945 - 225, 225) = \gcd(720, 225)$ .

But, as long as we are going to subtract  $n$  from  $m$  once, why not subtract as many multiples,  $k$ , of  $n$  as we can, so long as the difference remains nonnegative? By the same argument as in the previous paragraph,

$$\gcd(m, n) = \gcd(m - kn, n) = \gcd(n, m - kn), \quad (3)$$

with  $n > m - kn$  because otherwise we could subtract another  $n$  from  $m - kn$ . To calculate the value of  $k$ , we divide  $m$  by  $n$ , to get a quotient  $q_1 = k = \lfloor m/n \rfloor$  and a remainder  $r_1 = m - kn = m - nq_1$  with

$$q_1 \geq 0 \quad \text{and} \quad 0 \leq r_1 < n.$$

(Is it clear why the bounds on the remainder are as given?) As an example, if  $m = 36$  and  $n = 14$ , then  $q_1 = 2$  and  $r_1 = 8$ . Omitting the second term in Eq. (3) and replacing  $m - kn$  by  $r_1$ , we obtain the equation which is the essence of the Euclidean algorithm:

$$\gcd(m, n) = \gcd(n, r_1). \quad (4)$$

Since  $r_1 < n$  in Eq. (4), we have indeed reduced finding  $\gcd(m, n)$  to a smaller case.

Now we come to the key step of the Euclidean algorithm. Just as we reduced  $\gcd(m, n)$  to  $\gcd(n, r_1)$  by subtracting  $n$  repeatedly from  $m$ , so can we reduce  $\gcd(n, r_1)$  by subtracting  $r_1$  repeatedly from  $n$ . We get

$$r_2 = n - r_1 \lfloor n/r_1 \rfloor = n - r_1 q_2, \quad q_2 \geq 0; \quad 0 \leq r_2 < r_1,$$

and so

$$\gcd(n, r_1) = \gcd(r_1, r_2). \quad (5)$$

What we have done twice we can continue to do. At each stage we let the divisor at one stage (e.g.,  $n$ ) become the dividend at the next stage, while the remainder in the present stage (e.g.,  $r_1$ ) becomes the divisor at the subsequent stage. Doing this we obtain

$$\begin{aligned} r_1 &= m - nq_1, & 0 \leq r_1 < n; \\ r_2 &= n - r_1 q_2, & 0 \leq r_2 < r_1; \\ r_3 &= r_1 - r_2 q_3, & 0 \leq r_3 < r_2; \\ r_4 &= r_2 - r_3 q_4, & 0 \leq r_4 < r_3; \\ &\vdots \\ r_{k+1} &= r_{k-1} - r_k q_{k+1}, & 0 \leq r_{k+1} < r_k; \\ 0 &= r_{k+2} = r_k - r_{k+1} q_{k+2}, \end{aligned} \quad (6)$$

with the final remainder  $r_{k+2} = 0$ . Can we be certain that sooner or later some remainder will be zero? Yes, because as display (6) indicates,

- i) the remainders form a *strictly decreasing* sequence (i.e., each remainder is strictly less than its predecessor), but
- ii) all are nonnegative integers.

Thus, if you keep going, at some point the last  $r_j$  is 0 (why?). The reasoning that led to Eqs. (4) and (5), when applied to display (6) gives us

$$\begin{aligned} \gcd(m, n) &= \gcd(n, r_1) = \gcd(r_1, r_2) = \gcd(r_2, r_3) = \cdots \\ &= \gcd(r_{k-1}, r_k) = \gcd(r_k, r_{k+1}). \end{aligned} \quad (7)$$

Now observe that the last line in (6) implies that  $r_k$  is a multiple of  $r_{k+1}$  (why?), so that

$$\gcd(r_k, r_{k+1}) = r_{k+1}.$$

(Why?) Thus from Eq. (7),

$$\gcd(m, n) = r_{k+1} \quad (8)$$

and we are done.

As an alternative derivation of Eq. 8, go one more step in Eq. (7) to obtain

$$\gcd(m, n) = \gcd(r_{k+1}, r_{k+2}),$$

and since  $r_{k+2} = 0$ , we get

$$\gcd(m, n) = \gcd(r_{k+1}, 0) = r_{k+1}.$$

If doing the Euclidean algorithm by hand you wouldn't go this extra step, but it leads to a simpler termination test (getting 0), so all our formulations of the algorithm to follow take this step.

To illustrate our algorithm, we let  $m = 315$  and  $n = 91$ . Then, as in display (6), we compute

$$42 = 315 - 91 \times 3$$

$$7 = 91 - 42 \times 2$$

$$0 = 42 - 7 \times 6$$

and therefore  $\gcd(315, 91) = 7$ . In summary, we compute new values of  $q$  and  $r$  until we get  $r = 0$ . Then the previous  $r$  is the gcd of the original numbers.

Using display (6) we can now state the Euclidean algorithm as Algorithm 1.6 EUCLID. Algorithm EUCLID contains no new features of AL but it nevertheless has some noteworthy aspects:

1. Instead of using subscripts as in (6), we have used *num* and *denom* to represent the dividend and divisor (i.e., the *numerator* and *denominator*) at every step. Subscript notation is often useful in the discussion of algorithms. However, when quantities play the same role each time through a loop and, most importantly, when the values at each stage do not have to be saved for the next stage, it's convenient and efficient in the algorithm to call the quantities by the same names each time through the loop.
2. Nevertheless, we do have to do some saving from one pass through the loop to the next. This is accomplished using the *temporary* variable *rem*. If, instead of  $\text{rem} \leftarrow (\text{num} - \text{quot} \times \text{denom})$ , we had written

$$\text{denom} \leftarrow \text{num} - (\text{quot} \times \text{denom}),$$

then the old *denom* (the one on the right-hand side, which becomes *num* in the next line) would have been replaced in this line by the new *denom* (the one on the left-hand side). The result would be that *num* in the next line would be set equal to the new *denom* rather than the old *denom*. Note that we couldn't do  $\text{num} \leftarrow \text{denom}$  before computing the new *denom* because then the old *num* would not be available for  $\text{num} - \text{quot} \times \text{denom}$ .

3. The first thing we did in this algorithm was to take the inputs  $m$  and  $n$  and redefine them as *num* and *denom*. But why do this since we could just have stuck with  $m$  and  $n$  throughout the algorithm? The reason is that it is always bad practice to change the values of the **Input** variables. This is because, in real life, the same input values may be needed in different parts of an algorithm or for more than one algorithm, so you shouldn't tamper with them. When you do calculations which need to change input values, immediately assign the inputs ( $m$  and  $n$  in Algorithm EUCLID) to other variables (*num* and *denom*) and then let the latter be changed in subsequent computation.

Another reason for presenting Algorithm EUCLID is that it provides a convenient way to introduce the important concept of a **loop invariant**. A loop invariant is a condition that is true when the loop is first entered, on each subsequent entry into the loop, and at the final exit from the loop. Loop invariants play an important role in analyzing the behavior of many algorithms. We shall define loop invariants more rigorously and discuss them at length in Section 2.5.

## Algorithm 1.6

### Euclid

<b>Input</b> $m, n$	[Integers $\geq 0$ ]
<b>Output</b> $num$	[ $= \gcd(m, n)$ ]
<b>Algorithm</b> EUCLID	
$num \leftarrow m; denom \leftarrow n$	[Initialize $num$ and $denom$ ]
<b>repeat until</b> $denom = 0$	
$quot \leftarrow \lfloor num / denom \rfloor$	[Quotient]
$rem \leftarrow num - (quot \times denom)$	[Remainder]
$num \leftarrow denom; denom \leftarrow rem$	[New $num$ and $denom$ ]
<b>endrepeat</b>	

A loop invariant is not just *any* condition which has the properties just listed but rather a condition that enables you to prove the algorithm correct. In Algorithm EUCLID there is thus really only one possible loop invariant. If you have understood the derivation of this algorithm, you may see that the condition

$$\gcd(m, n) = \gcd(num, denom), \quad (9)$$

is true when the loop is first entered, since at this point  $num = m$  and  $denom = n$ ; and it remains true each time the loop is reentered since the new values given to  $num$  and  $denom$  are, respectively, the old denominator and the remainder when  $num$  is divided by  $denom$ . Thus, it follows from Eq. (4) that Eq. (9) is satisfied each time the loop is entered. Since  $denom = 0$  at the final exit from the loop and since  $\gcd(x, 0) = x$ , we have  $num = \gcd(num, denom) = \gcd(m, n)$  at the end just as we wish. So using the loop invariant, we have been able in effect to prove the correctness of Algorithm EUCLID. We'll present such proofs more formally in Ch. 2 using induction. See especially Section 2.5. We return to proofs of algorithms with loop invariants with even more apparatus in Section 7.4.

Loop invariants are not just useful in proving algorithms correct. As we shall see at various places later in this book, when we use the recursive paradigm, the loop invariant is often just the relation used to motivate the algorithm in the first place.

Although Algorithm EUCLID isn't complicated, its *analysis* (i.e., determining how efficiently it finds the gcd) isn't simple at all; we'll come back to it in Section 1.5. ■

As you read our descriptions of Algorithms 1.1–1.6, did you ask yourself: "Is each really an algorithm? Does each satisfy the four properties listed earlier in this section (p. 76)?" The answers to those questions are as follows:

- All satisfy the second and third properties — uniquely determined initialization and uniquely determined succession — because our notation enforces both.
- All the algorithms are certainly finite because the sequence of steps inevitably leads to the end of the algorithm.

- The most difficult property of an algorithm to verify, as we noted earlier, is whether it really leads to a solution of the problem. Later in this book we will return to some of these algorithms and verify in a fairly formal sense that they do in fact lead to solutions. For the moment, we hope that our informal discussions of these algorithms have convinced you that they do indeed do the jobs for which they are intended.

You may have noticed that all the algorithms we have presented in this section, have a similar structure, consisting of

1. an *initialization* in which some variables are set to their initial values (e.g., *power* and *k* in POWERB and *num* and *denom* in EUCLID); and
2. one or more *loops*, in which a sequence of steps is applied repetitively, each time to *different data*, until some condition is satisfied.

This paradigm pervades all *iterative algorithms*. It is one you will see again and again in this book, and one you will construct yourself in the iterative algorithms you write.

Algorithms play a very important role in problem solving. But you shouldn't get the impression that using an algorithm is the right approach to solving *all* problems. Let's consider briefly two questions:

1. Is there an algorithm that will solve any problem if only we can state the problem in some unambiguous fashion?
2. If we know that an algorithm exists to solve a particular problem, can we necessarily find it?

Perhaps it will surprise you to learn that the answer to the first question is No. Although it is beyond our scope to discuss this further in this book, there are, in fact, problems for which it can be proved mathematically that no possible algorithm could solve them. The answer to the second question is also No. For example, we know that an optimal strategy (i.e., an algorithm) exists in chess for both players. That is, no other strategy followed by one player could lead to a better result if the other player follows an optimal strategy. (Interestingly, however, we don't know whether the result of using these optimal strategies would be a win for white, a win for black, or a draw.) But to find this algorithm, even using a computer which could evaluate  $10^6$  (= one million) board positions per second, would take on the order of  $10^{120}$  years — far longer than the life of the universe. (This is not to say that a program that cannot be *guaranteed* to win could not nevertheless beat even the best human player, as the program Deep Blue did when it beat Garry Kasparov in 1997.)

Instead of an algorithmic approach, chess programs and good chess players use what is usually called a **heuristic** approach, that is, one which points in what seems to be a useful direction even if you can't *prove* that it leads to a solution. An obvious heuristic in chess is to have more pieces than your opponent does. A somewhat more sophisticated heuristic is to strive to control the center of the board with your pieces.

Knowledge about algorithms is crucial to understanding how to solve problems on computers. Almost all computer programs are *realizations* of algorithms. That is, almost everything that computers do consists of implementing algorithms in computer programs.

## Problems: Section 1.1

1. ⟨1⟩ In Algorithms 1.1–1.3 we said that  $m$  had to be an integer (sometimes positive, sometimes non-negative) but put no restrictions on  $x$ . For both variables, why?

2. ⟨3⟩

- a) Use the statement of Algorithm POWERC to count the number of multiplications required by POWERC for the following values of  $m$  (i) 13, (ii) 21, (iii) 31. In each case also state the number of multiplications required by POWERA.

- b) Let  $m = 2^i + j$ ,  $0 \leq j < 2^i$ . In terms of  $i$  and  $j$  derive a formula for the number of multiplications required by POWERC. Make sure your formula gives the same results as in a) for the three values of  $m$  there.

- c) So POWERC is better than POWERA but not as much better as you might wish. Can you find a simple way to improve POWERC. For now don't try to use AL but just sketch your answer in English. We'll ask you to put your answer into AL later. (*Hint:* Consider the case when  $m = 24 = 16 + 8$ .)

3. ⟨2⟩ Let  $x$  and  $y$  be two integers with  $x$  nonnegative.

- a) Describe in English how you could calculate  $x \times y$  using only addition. Why did we restrict  $x$  to be nonnegative but put no restrictions on  $y$ ?

- b) Suppose there were no restrictions on either integer as to their signs and that either could be zero. Describe in English how you would compute their product using only addition.

4. ⟨2⟩ Apply the algorithm you described in [3a] to the following data:

- a)  $x = 12, y = 4$       b)  $x = 9, y = -5$   
c)  $x = 9, y = -3$ .

5. ⟨2⟩ Suppose that PRIMENUM has been executed up to the point where  $p_7 = 17$  and  $i$  has been set

- to 19. Show how  $p_8$  is calculated by indicating each value of  $i$  and  $j$  and each calculation in the outer loop.

6. ⟨2⟩ Apply FIBERNET to the following data:

- a)  $n = 4$ :

$i \backslash j$	1	2	3	4
1	210	590	620	
2		330	450	
3				520

- b)  $n = 6$ :

$i \backslash j$	1	2	3	4	5	6
1	150	175	125	200	150	
2		225	250	150	125	
3			200	125	200	
4				175	200	
5						175

7. ⟨2⟩ Here is another greedy approach to the fiber optic network problem of Example 3. Instead of picking the cheapest edge between so-far visited and so-far unvisited vertices, as in FIBERNET, pick the cheapest edge *anywhere*, so long as it doesn't connect two vertices between which there is already a path of chosen edges. Call it Algorithm FIBERNET'.

- a) Apply FIBERNET' to the graph in Example 3.

- b) Apply FIBERNET' to the data in [6b].

Any conjectures?

8. ⟨3⟩ Sometimes a person wants to find the network having the *maximum* value. For instance, suppose that you own a construction company in Thirdworld (see Example 3). Suppose further that you have computed for each possible link between the two cities the difference between the price the

government will pay and the price the construction will cost you. Then you would propose building those links for which the sum of the profits would be greatest. This naturally suggests two algorithms, FIBERNET-MAX and FIBERNET'-MAX, which are just like FIBERNET and FIBERNET' (see [7]), except you go for the biggest available edge each time.

- a) Apply FIBERNET-MAX to the data of [6b].
- b) Apply FIBERNET'-MAX to this data.
- c) We will show in Chapter 3 that FIBERNET always finds the minimum weight spanning tree. Assuming this, can you argue that FIBERNET-MAX always finds the maximum weight spanning tree?

9. {2} We have a knapsack with volume 20 cubic units, and we have items of volume 15, 10 and 9 that we want to put in our knapsack. Our goal is to stuff it with as much as we can — the greatest volume of our items. A natural approach is “largest first”, a greedy algorithm approach where you insert items in order of decreasing size until no more go in.

- a) What items get into the knapsack using this method?
- b) What is the optimal packing of this knapsack given the items we have?
- c) In this case, the approach “smallest first” (also greedy, or at least reverse greedy) would have led to the optimal packing. But give another example (same knapsack, different volume for the items) where smallest first is not optimal.

For a fuller discussion of knapsack problems, see [6, Supplement].

10. {1} Apply the Euclidean algorithm to find the greatest common divisors of

- a) 315 and 91      b) 440 and 924
- c) 322 and 3795

(Compare with [1, Supplement, Chapter 0].)

11. {2} What happens in the first iteration of Algorithm EUCLID when  $m < n$ ? What happens in the whole algorithm when either  $m$  or  $n$  is zero?

12. {3} Below is EUCLID1, a modification of EUCLID in which the remainder is immediately reduced in the repeat-loop if it is large relative to the denominator.

### Algorithm 1.7 EUCLID1

<b>Input</b> $m, n$	[Integers $\geq 0$ ]
<b>Output</b> $num$	$[= \gcd(m, n)]$
<b>Algorithm</b> EUCLID1	
$num \leftarrow m$ ; $denom \leftarrow n$	
<b>repeat until</b> $denom = 0$	
$quot \leftarrow \lfloor num/denom \rfloor$	
$rem \leftarrow num - (quot \times denom)$	
<b>if</b> $rem > denom/2$ <b>then</b> $rem \leftarrow denom - rem$	
$num \leftarrow denom$ ; $denom \leftarrow rem$	
<b>endrepeat</b>	

- a) Trace EUCLID1 and EUCLID on the input pairs (45,26) and (21,13). What do you notice?

- b) Explain why EUCLID1 is valid. Why does the extra step, **if**  $rem > denom/2$  **then**  $rem \leftarrow denom - rem$ , never mess up the loop invariant that  $\gcd(num, denom) = \gcd(m, n)$ ? Why is EUCLID1 still guaranteed to terminate?

13. {3} This problem illustrates that Euclid’s method for finding gcd’s is much faster than the factoring methods many of you learned in school. You may use a calculator to help you with basic arithmetic, but don’t use a CAS or any software that has factoring and gcd’s built in. The reason to force you to do all the arithmetic in this problem is to really drive home the point.

Find the gcd of 133307 and 268249 two ways:

- a) By the factoring method. (The gcd is the product of all the common prime factors.) Of course, you have to find the factors.

- b) By Euclid’s method.

For both methods, double check your arithmetic as you go — otherwise you may waste a lot of time.

14. {2} Use a calculator to figure out what the following algorithm is doing.

```

 $a \leftarrow 1$ 
repeat
     $a' \leftarrow \frac{1}{2} \left( a + \frac{4}{a} \right)$ 
    if  $|a' - a| < .0001$  then exit
     $a \leftarrow a'$ 
endrepeat

```

## 1.2 Aspects of AL

In this section we consider briefly some aspects of AL that were not discussed in Section 1.1 but which will often be used subsequently.

Throughout this book, algorithms have the following three parts.

**Input:** A listing of all quantities, values of which are assumed to be “available” (i.e., able to be used) at the start of the algorithm itself. Therefore the input is the *data* provided to the algorithm.

**Output:** A listing of the results of the algorithm.

**Algorithm:** The sequence of steps which transforms the input into the desired output.

Although the output quantities are always listed in the **Output** portion of the algorithm, in order to increase readability we allow output to be specified in the algorithm itself by

```
print list of expressions
```

where each expression itself is either a variable, a mathematical expression, or a string of characters (called a **literal**) inside quotes. In the first two cases, the value of the variable or the value of the expression (using current values of the variables in it) is output. In the last case, the output is the string of characters itself. Examples are:

```
print m, n + 1  
print 'No solution'
```

Two general precepts about AL that we used in Section 1.1 are:

1. Any mathematical notation in normal usage may be used in an algorithm. Obvious examples are subscripts, exponents, and the summation notation introduced in Section 0.4.
2. When we wish to emphasize conceptual understanding rather than mathematical precision, we will not hesitate to use English phrases and sentences in our algorithms.

Algorithms, like computer programs written in a procedural language, are constructed from **statements**, each of which conveys a complete algorithmic thought. The three main algorithmic statements are those for **assignment**, **selection** and **iteration**. All of these featured prominently in the algorithms of Section 1.1. Assignment always has the form *variable*  $\leftarrow$  *expression* but the forms for selection and iteration can be more general or varied than in the examples of Section 1.1.

The general form of the selection or **decision** statement is

```

if  $B$  then  $S_1$ 
    [else  $S_2$ ]
endif

```

where the [...] notation means that what is contained within it may be but need not be present,  $B$  is a predicate (i.e., an expression that is either true or false) and the  $S_1$  and  $S_2$  are any sequence of one or more statements in AL. Often  $S_i$  consists of one or more assignment statements but selection statements are allowed (in which case we have **nested** decisions) and iteration statements are also allowed. When no ambiguity can result, we allow **endif** to be omitted as in Algorithm PRIMENUM in Section 1.1.

As in the examples in Section 1.1, if  $B$  is true, the statement  $S_1$  is executed. But if  $B$  is false *and* the **else** portion is present, then  $S_2$  is executed. If  $B$  is false and **else** is absent, then nothing happens.

*Iteration structures.* In addition to the **repeat until** loop used in the examples of Section 1.1, other types of repeat-loops are also allowed in AL:

1. The **until** condition may be omitted in which case the iteration continues until an **exit** statement (such as in **if**  $B$  **then exit**) is encountered. This **exit** takes you just below the innermost loop in which it occurs.
2. **until** may be replaced by **while** in which case, instead of executing the statements between **repeat** and **endrepeat until** the condition  $B$  is true, the statements are executed *while*  $B$  is true. Condition  $B$  is tested at the top of the loop and then again only after the entire loop has been executed.
3. Instead of testing a condition at the beginning of the loop, we can test at the end using the form **repeat ... endrepeat when**  $B$ , in which case the statements of the loop are always executed at least once and the loop is terminated the first time  $B$  is true at the bottom.

In Algorithm 1.1 of Section 1.1 we used another form of loop, based on counting the number of times the loop is executed, whose form was:

```

for  $C = Cstart$  to  $Cstop$ 
     $S$ 
endfor

```

where  $S$  is any sequence of AL statements and implicitly  $Cstart \leq Cstop$  with the values of  $C$  being  $Cstart, Cstart + 1, \dots, Cstop - 1, Cstop$ . We also allow counting down instead of up where **to** above is replaced by **downto**,  $Cstart \geq Cstop$  and the values of  $C$  now are  $Cstart, Cstart - 1, \dots, Cstop + 1, Cstop$ . Still another alternative is based on set membership instead of a counter:

```

for  $x \in T$ 
     $S$ 
endfor

```

where  $x$  is a variable and  $T$  is a set. In this case the statements between **for** and **endfor** are executed once for each member  $x$  of the set  $T$  with no specification of the order in which the members are to be considered. We shall use this construct in various places in Chapter 3.

Just as selection statements can be nested, loops can also be nested. That is, one or more of the statements between **repeat** and **endrepeat** or between **for** and **endfor** can be any loop statement.

One other statement that we shall use occasionally consists of the single word **stop** whose effect, when it is reached, is to halt execution of an algorithm.

This completes for now our discussion of the structures of AL and their meaning. It remains to discuss in this section the *literary* aspects of writing algorithms so that our algorithms in this book and the ones you will write will be as readable as possible. The main techniques we use to achieve readability are:

- printing key command words like **repeat** in bold;
- indenting under each main construct; thus inside a **repeat ... endrepeat** loop we indent all the statements in the loop; if there is another loop inside the outer one, we indent the statements in this loop farther, and so on;
- printing each word which closes a construct directly under the word which opens it and similarly making sure that corresponding **then**'s and **else**'s are printed under each other;
- adding annotations or *comments* in ordinary English in square brackets and, to the extent that typography allows, flush right on the same line as the algorithm step being annotated.

When you write algorithms, you should generally follow these conventions. But the rules for an algorithmic language need not be as rigidly obeyed as those for a programming language. So long as clarity is maintained, the rules may be bent. But not too far. These rules are a vital part of good algorithmic presentation as well as an aid in analyzing algorithms and proving them correct. For instance, much as we might like to believe that algorithms stated using just AL without comments are transparently clear to all readers, we know better. Sometimes they're not even clear to us! Good comments are an invaluable aid to all readers of algorithms. Remember that an algorithm, like other forms of written communication, typically has one writer but many readers.

## Problems: Section 1.2

---

1. {2} What value does the algorithm fragment below give to  $S$  when  $m = 20$ ? When  $m$  is arbitrary?

```
 $S \leftarrow 0; n \leftarrow 1$ 
repeat until  $S \geq m$ 
   $S \leftarrow S + n; n \leftarrow n + 2$ 
endrepeat
```

2. {2} What does this algorithm do for various  $n$ ?

```
if  $n > 20$  then print "high"
  else if  $n < 10$  then print "low"
    else print "middle"
  endif
endif
```

3. ⟨2⟩ Do a trace of the values of the variables ( $j$  and  $k$ ) in the following (somewhat arbitrary) algorithm fragment. The trace is begun below.

```

 $k \leftarrow 5$ 
for  $j = 1$  to 3
   $k \leftarrow k - j^2$ 
  repeat while  $k < 10j$ 
     $k \leftarrow 2k$ 
  endrepeat
endfor
```

Here's the start of a trace:

$j$	$k$
	5
1	
	4

4. ⟨2⟩ Do a trace of the values of ( $j$  and  $k$ ) in the following algorithm.

```

 $k \leftarrow 5$ 
for  $j = 2$  to 4
   $k \leftarrow k + j^2$ 
  repeat while  $k > 5j$ 
     $k \leftarrow k - 4$ 
  endrepeat
endfor
```

5. ⟨2⟩ Run a trace of Algorithm MYSTERY below when  $n = 4$ . In general (meaning, for any positive integer  $n$ , not just 4), what is the output? *Note:* The semicolon on the first line is a convenient way to put multiple commands on the same line.

## Algorithm

**Input** positive integer  $n$

**Output**  $p$

**Algorithm** MYSTERY

$p \leftarrow 1; k \leftarrow 1$	[Multiple commands]
<b>repeat while</b> $k < n$	
$k \leftarrow k + 1$	
$p \leftarrow pk$	
<b>endrepeat</b>	

6. ⟨2⟩

- a) The following instructions switch the values of  $a$  and  $b$ . Show this by doing a trace. For clarity, use  $a_0$  and  $b_0$  for the initial values of  $a$  and  $b$ .

```

 $c \leftarrow a$ 
 $a \leftarrow b$ 
 $b \leftarrow c$ 
```

- b) In part a), why can't we eliminate  $c$  and still succeed in switching  $a$  and  $b$  as follows?

```

 $a \leftarrow b$ 
 $b \leftarrow a$ 
```

7. ⟨2⟩ What is the value of  $j$  when the loop in the following algorithm fragment is exited?

- a) if  $i = 37$   
 b) in general

```

 $j \leftarrow 1$ 
repeat
  if  $j > i$  then exit
   $j \leftarrow 2j$ 
endrepeat
```

8. ⟨2⟩

- a) What is computed by the loop in the following algorithm fragment?

```

 $P \leftarrow a_n$ 
for  $k = 1$  to  $n$ 
   $P \leftarrow Px + a_{n-k}$ 
endfor
```

- b) Rewrite this loop using repeat–endrepeat.

9. ⟨2⟩ For this problem, define the set of numbers  $S_{a,b}$ , by

$$S_{a,b} = \{x \mid a \leq x \leq b\}.$$

Using this notation, what are the outputs  $S, T$  of the following algorithm?

```

 $S \leftarrow S_{0,10}; T \leftarrow S$ 
for  $k = 1$  to 4
   $S \leftarrow S \cup S_{k,k+10}$ 
   $T \leftarrow T \cap S_{k,k+10}$ 
endfor
```

10. ⟨2⟩ (For-loop with step size) Here is another variant of a loop which is sometimes useful:

```
for b = c to d step s
```

do something (usually involving b)

```
endfor
```

Its meaning is that “do something” would first be done with  $b = c$ , then again with  $b = c+s$ , then with  $b = c+2s$ , and so on, so long as  $b \leq d$ . For instance, if  $c = 1.5$ ,  $d = 4.1$ , and  $s = .6$ , then  $b$  would take on the values 1.5, 2.1, 2.7, 3.3, and 3.9.

- a) You are welcome to use this construct. Still, it is unnecessary. Figure out how to do the same thing using a repeat loop.
- b) There is also a downwards version of this construct:

```
for b = c downto d step s
```

do something (usually involving b)

```
endfor
```

where now  $b$  decreases each time by  $s$  and  $d \leq c$ . Show how this too may be accomplished with previous commands.

11. (2) While it is legitimate in AL to write a line like  $S \leftarrow \sum_{k=1}^n k$ , this sum can also be computed by repeated addition, as it must be in many computer languages:

```
S ← 0  
for k = 1 to n  
    S ← S + k  
endfor
```

Rewrite this fragment two others ways: using a repeat-while loop, and using a repeat-until loop.

12. (2) Write if-endif structures and inequalities to do the following tasks:

- a) Set a variable  $L$  equal to the larger of two numbers  $a$  and  $b$ .
- b) Set a variable  $L$  equal to the largest of three numbers  $a$ ,  $b$ , and  $c$ .

13. (2) Write an algorithm which, given an input integer  $N > 1$ , prints the largest power of  $N$  which divides one million. For example,  $2^6$  divides one million and  $2^7$  does not, so if  $N = 2$ , the algorithm should output 64. You may use the notation  $p \nmid q$  for the statement that  $p$  is not a factor of  $q$ .

14. (2) Write an algorithm to output all the divisors of a positive integer  $N$ . For instance, if  $N = 12$ , the algorithm should output 1, 2, 3, 4, 6, and 12.

15. (2) Write an algorithm to output the odd positive integers less than 30.

- a) Use the for-endfor construct.
- b) Do not use the for-endfor construct.

16. (2) Given a list of numbers  $x_1, x_2, \dots, x_n$ , write an algorithm which, for each number in the list in turn, prints that number if it is even and also prints it (thus sometimes for the second time) if it is positive.

17. (2) Sum all the integers from 1 to 10 that are not divisible by 3. You may use the notation  $3 \nmid n$ , which is the statement “3 is not a divisor of  $n$ ”.

18. (2) If we want to print  $n$  if it is an integer greater than 10, we can write

```
if n > 10 and n ∈ Z then print n
```

Instead, for practice, write this using two or more if-statements, perhaps nested, that begin either “if  $n > 10$  then” or “if  $n \in Z$  then”.

19. (2) Write an algorithm which inputs  $n$ , and outputs  $answer$ , where  $answer$  is 1 if  $n$  is an integer greater than 10, and 0 otherwise.

20. (2) If we want to print  $n$  if it is positive or an integer, we can write

```
if n > 0 or n ∈ Z then print n
```

For practice, write this using two or more if-statements, perhaps nested, that don’t use or. If  $n$  is both positive and an integer, it should be printed only once.

21. (2) Write an algorithm that for  $n$  from 1 to 10 assigns the value of  $n!$  to  $F_n$ . Compute each  $n!$  by looping through some multiplications (even though  $n!$  is legitimate notation in AL).

22. (2) Write an algorithm that prints the pair  $n, n!$  for each  $n$  from 1 to 10 that is not divisible by 3. See [17].

23. (2) Write an algorithm to compute  $\sum_{k=1}^{10} k!$ . Use just multiplications and additions with nested loops. Don’t use summation or factorial notation in your algorithm even though these are legitimate notation in AL.

24. (3) Print the positive integers from 1 to  $\lfloor \sqrt{1000} \rfloor$  without using floors or square roots, and without using the fact that  $31 < \sqrt{1000} < 32$ .

25. ⟨2⟩ Show that a for-endfor loop can always be replaced by a repeat-endrepeat loop.
26. ⟨3⟩ In [2c, Section 1.1] you were asked to sketch an algorithm to improve upon POWERC. Now write this algorithm in AL. (*Hint:* The best way to do this is with nested iterations. Be careful how you initialize variables before each loop.)
27. ⟨3⟩
- In [3a, Section 1.1] you were asked how, using only addition, to multiply two integers,  $x$  and  $y$ , with  $x$  restricted to be nonnegative. Now write this algorithm in AL. Call it MULT.
  - Describe in your own words why MULT is correct. That is, give an argument using ordinary English, but with mathematical notation as appropriate, which proves that the value output by MULT is, indeed,  $xy$ .
  - How does your argument show that  $x$  must be nonnegative for the algorithm to work, but that there need be no such restriction on  $y$ ?
  - As in [4, Section 1.1] apply MULT to the data given there.
28. ⟨3⟩ Just as it is possible to do multiplication by repeated addition as in [3, Section 1.1] it is possible to do division by repeated subtraction. Write such an algorithm. It should take as input positive integers  $m$  and  $n$  and output the integer quotient and remainder of  $m/n$ .
29. ⟨2⟩ Write an algorithm to output  $h_9$ , where  $h_1 = 1$  and for  $n > 1$ ,  $h_n = 2h_{n-1} + 1$ . As with Algorithm EUCLID, you should be able to write this algorithm with no subscripts.
30. ⟨2⟩ Define  $f_1 = f_2 = 1$  and for  $n \geq 3$ ,  $f_n = f_{n-1} + f_{n-2}$ . Write an algorithm that outputs the first number in this sequence that is greater than 100. You should be able to write it without subscripts (but you will need more than one variable). See also [21a, Section 1.4].
31. ⟨3⟩ This problem concerns methods for exiting from nested loops, when sometimes you want to exit just one level but other times you want to exit further levels. (Recall that our **exit** command only exits one level, to the first line below the innermost loop in which it occurs.) In the algorithm fragment below a special variable is set up to indicate when an inner loop has been exited early. We use a traditional name, *flag*. We could give it values 0 and 1, but we will instead give it values *on* and *off*.
- ```

for  $k = 0$  to 2
  for  $j = 1$  to 4
     $flag \leftarrow off$ 
     $n \leftarrow 4k + j$ 
    print  $n$ 
    if  $5|n$  then  $flag \leftarrow on$ ; exit
  endfor
  if  $flag = on$  and  $2|n$  then exit
endfor
```
- What integers does this algorithm print? What condition makes the algorithm terminate?
  - Suppose we eliminated the flag (see revision below); now what does the algorithm print and when does it terminate?
- ```

for  $k = 0$  to 2
  for  $j = 1$  to 4
     $n \leftarrow 4k + j$ 
    print  $n$ 
    if  $5|n$  then exit
  endfor
  if  $2|n$  then exit
endfor
```
32. ⟨3⟩ Counting is easy but explaining how to count to someone who doesn't know how is not so easy. Write an algorithm that takes as input an integer  $n \geq 0$  and the digits  $D_n, D_{n-1} \dots D_1, D_0$  of an  $n+1$  digit nonnegative integer  $I$  and produces as output the digits of  $I+1$ . (This is not quite as simple as it may at first appear.)
33. ⟨2⟩ Going a step further than [32], write an algorithm that adds two arbitrary positive integers by repeatedly adding one.
34. ⟨3⟩ Algorithm PRIMENUM (see Example 2, Section 1.1) is not very efficient. Can you think of any ways to improve it so that the primes would be calculated with less computation? Display your answer as an algorithm.

## 1.3 Recursive Algorithms

The recursive paradigm which we introduced in the Prologue is going to play a major role throughout this book. Many of our algorithms will be direct translations into our algorithmic language of recursive paradigm approaches to problems. In this section we'll discuss two such algorithms and a variation on one of these, all written in our algorithmic language. The algorithmic expression of the essence of the recursive paradigm — reducing to a previous case — requires a feature of the language not needed for the algorithms of Section 1.1. We will discuss this feature — **procedures** — and their simpler special case — **functions** — informally here and in more detail in Section 1.4. While procedures and functions do not have to involve recursion, their use to implement recursion will be their main purpose in this book.

*Warning.* The material in this section and the next is more difficult than that in Section 1.1, particularly if you have not used recursion in a programming language. What is important, however, is not that you grasp all the details now but rather that you get a feel for the idea of recursion and how it is expressed in AL.

Our first example illustrates a function. We return to the Euclidean algorithm of Section 1.1, but now we discuss it in recursive terms in contrast to the iterative approach previously taken.

### EXAMPLE 1

Derive a recursive formulation of the Euclidean algorithm and express it in algorithmic language.

**Solution** The essential formula in our previous derivation of the Euclidean algorithm was Eq. (4), Section 1.1:

$$\gcd(m, n) = \gcd(n, r_1), \quad (1)$$

where  $r_1$  is the remainder when  $m$  is divided by  $n$ . That is,

$$r_1 = m - n\lfloor m/n \rfloor.$$

Substituting into Eq. (1), we get

$$\gcd(m, n) = \gcd(n, m - n\lfloor m/n \rfloor). \quad (2)$$

Equation (2) expresses the gcd of two integers  $m$  and  $n$  in terms of two others,  $n$  and  $m - n\lfloor m/n \rfloor$ . In particular, comparing the two second inputs,

$$n > m - n\lfloor m/n \rfloor,$$

because the right-hand side is the remainder when  $m$  is divided by  $n$  and is therefore less than  $n$ . So the second argument on the right-hand side of Eq. (2) is less than the corresponding argument on the left-hand side. (How about the two first inputs? [1]) Thus Eq. (2) embodies the essence of the recursive paradigm: reducing to a previous (i.e., smaller) case. Using the recursive paradigm we continue to reduce to the previous case, thereby getting successively smaller — but always nonnegative — second arguments, until finally we arrive at a case which is obvious from the definition. For the Euclidean algorithm this case is that  $\gcd(m, 0) = m$  for any  $m$ .

Here's an example of how Eq. (2) works, for  $m = 63$ ,  $n = 27$ :

$$\begin{aligned}\gcd(63, 27) &= \gcd(27, 63 - 27\lfloor 63/27 \rfloor) && [\text{Using Eq. (2)}] \\ &= \gcd(27, 9) && [\text{Evaluating } 63 - 27\lfloor 63/27 \rfloor] \\ &= \gcd(9, 27 - 9\lfloor 27/9 \rfloor) && [\text{Using Eq. (2) again}] \\ &= \gcd(9, 0) && [\text{Evaluating } 27 - 9\lfloor 27/9 \rfloor] \\ &= 9. && [\text{Since } \gcd(m, 0) = m]\end{aligned}$$

Now, how do we implement the calculation of the gcd via Eq. (2) in our algorithmic language? Algorithm 1.8, EUCLID-RECFUNC, supplies the answer but needs some explanation.

1. In this algorithm and the discussion of it we distinguish between the greatest common divisor as a mathematical function ( $\gcd$ ) and the function used here to compute it ( $\text{gadf}$ ). We give them different names because they really are different conceptually: The mathematical function, as defined at the start of Example 4 in Section 1.1, specifies what  $\gcd(m, n)$  means without giving any method for computing it, whereas  $\text{gadf}(m, n)$  specifies a computation without giving any idea what it means. Moreover, as a practical matter, if will avoid confusion to have different names when we give a formal proof in Chapter 2 that the two functions really do have the same values.
2. Focus first on the lines from **function** to **endfunc**. Any such sequence of lines are a **function definition** in our language. In this case the function amounts to an algorithmic representation of Eq. (2) with the input values  $(m, n)$  replaced by the variables  $(i, j)$ , different letters to highlight the different roles, as will be discussed at several points below. To compute  $\text{gadf}(i, j)$  you first check to see if  $j = 0$ , in which case the gcd is  $i$  so  $\text{gadf} \leftarrow i$ . Otherwise, you **call** (invoke) the function again, with two new arguments given by the right-hand side of Eq. (2). This second call may make further calls. Somehow all these calls to the same function get unraveled and the correct value is assigned to the original  $\text{gadf}(i, j)$ .
3. The portion of the algorithm consisting of the line commented "Main algorithm" activates the whole process by calling the function with the given values  $m$  and  $n$  and assigning the result to *answer* (so it can be an Output).

Figure 1.2 illustrates how Algorithm EUCLID-RECFUNC works for the same data as above,  $m = 63$  and  $n = 27$ . Make sure you understand that the function is called three times (once from the main algorithm and twice more by itself) and that each of these calls is completed (i.e., we get to **endfunc**) in the reverse order in which the calls were made. Think of it this way: Each time function  $\text{gadf}$  is invoked, it is as if a new copy of its definition is made, but one in which the new values for  $i$  and  $j$  are substituted. The previous copy is interrupted but is still there to be returned to later at the place where it was interrupted. Each such copy is enclosed in a box in Fig. 1.2.

## Algorithm 1.8

### Euclid- RecFunc

<b>Input</b> $m, n$	[Integers $\geq 0$ ]
<b>Output</b> <i>answer</i>	[ $= \text{gcd}(m, n)$ ]
<b>Algorithm</b> EUCLID-RECFUNC	
<b>function</b> $\text{gcdf}(i, j)$	
<b>if</b> $j = 0$ <b>then</b> $\text{gcdf} \leftarrow i$	
<b>else</b> $\text{gcdf} \leftarrow \text{gcdf}(j, i - j \lfloor i/j \rfloor)$	[Recursive call]
<b>endif</b>	
<b>endfunc</b>	
<i>answer</i> $\leftarrow \text{gcdf}(m, n)$	[Main algorithm]

*answer*  $\leftarrow \text{gcdf}(63, 27) \leftarrow [Main algorithm call of function]$

**function**  $\text{gcdf}(i \leftarrow 63, j \leftarrow 27)$

**if**  $27 = 0$  **then** ...

**else**  $\text{gcdf} \leftarrow \text{gcdf}(27, 9) \leftarrow$

**endif**

**endfunc**

[Returns  $\text{gcdf}$ ,  
still = 9,  
to function in  
main call]

**function**  $\text{gcdf}(i \leftarrow 27, j \leftarrow 9)$

**if**  $9 = 0$  **then** ...

**else**  $\text{gcdf} \leftarrow \text{gcdf}(9, 0) \leftarrow$

**endif**

**endfunc**

[Returns  $\text{gcdf}$ ,  
still = 9,  
to function in  
previous  
call]

**function**  $\text{gcdf}(i \leftarrow 9, j \leftarrow 0)$

**if**  $0 = 0$  **then**  $\text{gcdf} \leftarrow 9$

**else** ...

**endif**

**endfunc**

[Assigns 9 to  
 $\text{gcdf}$  and  
returns this  
value to function  
in previous call]

**FIGURE 1.2**

Operation of  
Algorithm EUCLID-  
RECFUNC for  
 $m = 63, n = 27$ .

Each time **endfunc** is reached in a box, the value of the variable  $\text{gcdf}$  at that time is assigned to the expression  $\text{gcdf}(i, j)$  [or  $\text{gcdf}(m, n)$ ] in the higher level which called that copy. For instance, in the middle box in Figure 1.2, at the end  $\text{gcdf} = 9$ , and so 9 is assigned to  $\text{gcdf}(27, 9)$  in the box above it. (Notice that our multisymbol function (and procedure) names are roman, not italic, just as they are roman in traditional mathematics writing, e.g.,  $\log x$ ). ■

As this example shows, a function is just a portion of an algorithm which computes a value when called upon by another portion of the algorithm. A **procedure** is also just a portion of an algorithm that is called upon from some other portion, but whereas a function has one specific task (to compute and return a value), a procedure can do any sort of thing: compute several values, do operations that don't involve numerical values, etc. In the next example, just to give you the flavor of how a procedure is structured, we give a recursive procedure to compute the greatest common divisor using Eq. (2). In this example, similarly to what we did in Example 1, we use `gcdp` as the name of the procedure so as not to confuse the procedure with the mathematical function (`gcd`) that it computes.

## EXAMPLE 2

Express Euclid's algorithm as a recursive procedure instead of a recursive function.

**Solution** Since procedures can do any sort of thing, they can return a single value, and thus any function can be rewritten as a procedure. Algorithm 1.9, EUCLID-RECPRO, shows how to do it for Euclid's algorithm. Note that, unlike in our discussion of functions, the expressions `gcdp(m, n; answer)` and `gcdp(j, i - j[i/j]; result)` in this algorithm do not denote numbers. Rather they represent the *process* of doing the calculation.

### Algorithm 1.9

#### Euclid-RecPro

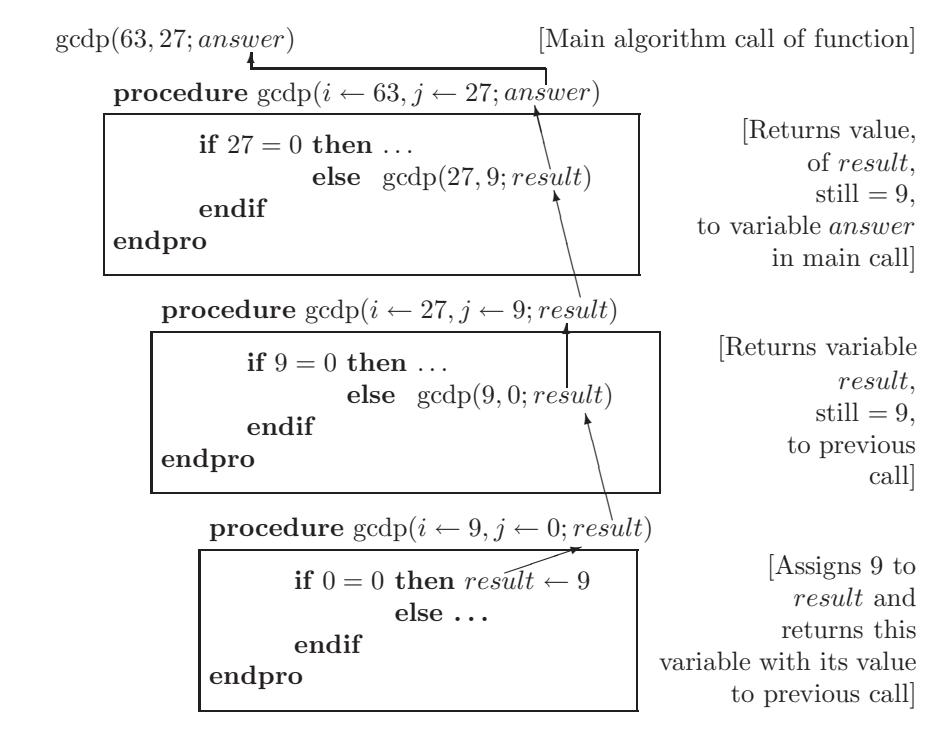
<b>Input</b>	$m, n$	[Integers $\geq 0$ ]
<b>Output</b>	$answer$	[ $= \text{gcd}(m, n)$ ]
<b>Algorithm</b> EUCLID-RECPro		
<b>procedure</b> gcdp( <b>in</b> $i, j$ ; <b>out</b> $result$ )		
<b>if</b> $j = 0$ <b>then</b> $result \leftarrow i$		
<b>else</b> gcdp( $j, i - j[i/j]$ ; $result$ ) [Recursive call]		
<b>endif</b>		
<b>endpro</b>		
gcdp( $m, n$ ; $answer$ ) [Main algorithm]		

The trace of calls and subcalls is shown in Fig. 1.3. This figure is very similar to Figure 1.2, except that now the value determined by each subcall is returned to the previous call as the value of the variable *result* instead of under the name `gcdp`.

These returns take place because, in the header line of the procedure definition, the third variable, *result*, is specified as an **out** variable so that its value is what is returned (i.e., output) to the previous call of procedure `gcdp`. Analogously, variables like *i*, *j*, which input values to the procedure, are called **in** (i.e., input) variables. In the main call of the procedure we use *answer* instead of *result* because, just as it helps to distinguish the procedure variables *i*, *j* from the input values *m*, *n*, so also it helps to distinguish the *result* of each call of the procedure from the final *answer*. ■

For instance, look at the middle box of Fig. 1.3. It is the expansion of the internal procedure call `gcdn(27, 9; result)` from the top box. In the expansion, `gcdn(9, 0; result)` is called, which results (via the bottom box) in the variable `result` being created in the middle box with the value 9. (This act of creation is represented by the arrow coming into the middle box from below.) Since there is no instruction inside the middle box that changes `result`, it is still 9 when the box's work is done. Now, the call that created the middle box was made in the top box, and it says to return the computed value to the top box, again with the name `result`. This is represented by the arrow going into the top box from below.

The rules for `in`, `out`, and a third possibility, `inout`, will all be detailed in Section 1.4.



Example 2 shows that functions are not *necessary* in our algorithmic language. We introduce them because they make a common case a little simpler, the case when the result is a single number; no additional variable name needs to be assigned to return the value. (See also [6, Section 1.4] for another advantage of functions.) Indeed, the functions and procedures of our language are both functions in the broadest mathematical sense; they both take input and produce something. When a function or a procedure calls itself, it is said to be **recursive** and the entire process is called **recursion**.

In the next example, we use a procedure when a function could not be used, namely to move pieces in a game. This example will recur many times in later chapters, in a variety of forms.

## EXAMPLE 3

### The Towers of Hanoi

The Towers of Hanoi puzzle (TOH) was first posed by a French professor, Édouard Lucas, in 1883. Although commonly sold today as a children's toy, it is often discussed in discrete mathematics or computer science books because it provides a simple example of recursion. In addition, its analysis is straightforward and it has many variations of varying difficulty.

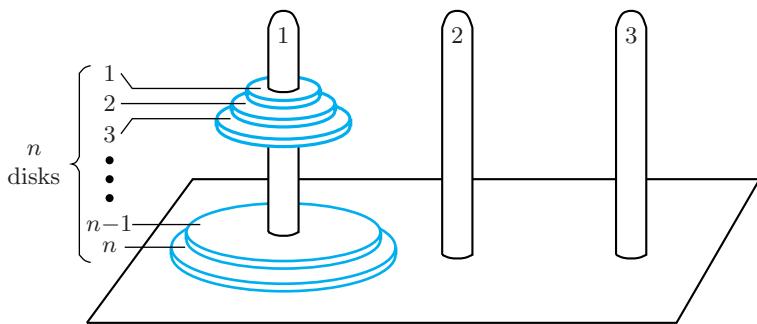
The object of the Towers of Hanoi problem (see Fig. 1.4) is to specify the steps required to move the disks (or, as we will sometimes call them, rings) from pole  $r$  ( $r = 1, 2$ , or  $3$ ) to pole  $s$  ( $s = 1, 2$ , or  $3$ ;  $s \neq r$ ), observing the following rules:

- i) Only one disk at a time may be moved.
- ii) At no time may a larger disk be on top of a smaller one.

The most common form of the problem has  $r = 1$  and  $s = 3$  (Fig. 1.4).

**FIGURE 1.4**

The Towers of Hanoi problem. The problem is shown with the initial pole  $r = 1$ ; typically the final pole is  $s = 3$ .

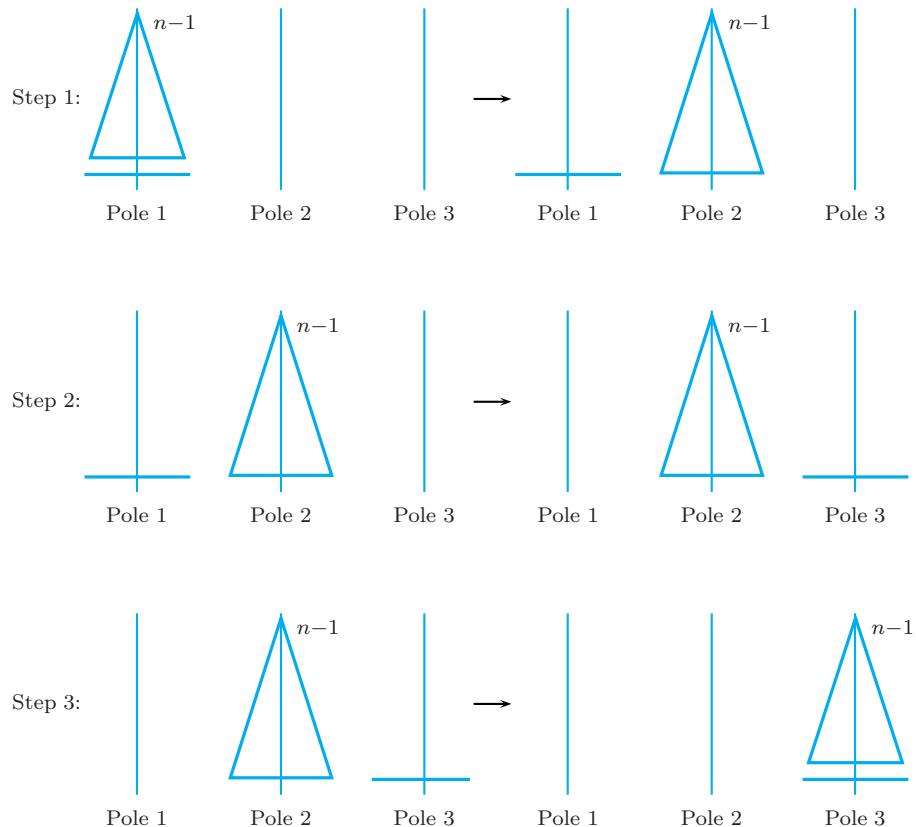


**Solution** Our algorithm to solve this problem exemplifies the recursive paradigm. We imagine that we know a solution for  $n - 1$  disks ("reduce to a previous case"), and then we use this solution to solve the problem for  $n$  disks. Thus to move  $n$  disks from pole 1 to pole 3, we would:

1. Move  $n - 1$  disks (the imagined known solution) from pole 1 to pole 2. However we do this, the  $n$ th disk on pole 1 will never be in our way because any valid sequence of moves with only  $n - 1$  disks will still be valid if there is an  $n$ th (larger) disk always sitting at the bottom of pole 1 (why?).
2. Move disk  $n$  from pole 1 to pole 3.
3. Use the same method as in Step 1 to move the  $n - 1$  disks now on pole 2 to pole 3.

Figure 1.5 illustrates these three steps. Even so, this solution may seem a little like sleight of hand. It's one thing to imagine we can do the previous case. It's another to do it! But now we'll show that it works.

Starting with  $n$  disks:



**FIGURE 1.5**

The Towers of Hanoi — a recursive solution.

Let's suppose that, being lazy, we don't want to move the disks ourselves but have contrived a robot arm to do this for us. All we have to do is tell the robot what to do. We assume that our robot understands instructions of the form

$$\text{robot}(r \rightarrow s), \quad (3)$$

which means to move the top disk on pole  $r$  to pole  $s$ . Using this instruction we can express the solution embodied in Steps 1–3 as Algorithm 1.10 HANOI.

Algorithm HANOI looks short and sweet, but you may not find it all that easy to understand. There is a procedure at the beginning named  $H$  and its arguments are  $n$ ,  $r$ , and  $s$ . Also like function gcdf in EUCLID-RECFUNC and procedure gcdp in EUCLID-RECPRO, procedure  $H$  is called from within itself. Once again, the main algorithm starts the execution. It consists of only one line,  $H(\text{num}, \text{Pinit}, \text{Pfin})$ , which invokes the procedure  $H$ . In effect, the line  $H(\text{num}, \text{Pinit}, \text{Pfin})$  tells the procedure to solve the Towers of Hanoi problem with  $n = \text{num}$  disks, which start on pole  $r = \text{Pinit}$  and are to be moved to pole  $s = \text{Pfin}$ .

**Algorithm 1.10****Hanoi**

<b>Input</b>	<i>num</i>	[Number of disks]
	<i>Pinit</i>	[Initial pole; $1 \leq Pinit \leq 3$ ]
	<i>Pfin</i>	[Final pole; $1 \leq Pfin \leq 3$ , $Pinit \neq Pfin$ ]
<b>Output</b>	The sequence of commands to the robot to move the disks from pole <i>Pinit</i> to pole <i>Pfin</i>	
<b>Algorithm HANOI</b>		
<b>procedure</b> <i>H</i> ( <b>in</b> <i>n, r, s</i> )		[Move <i>n</i> disks from pole <i>r</i> to pole <i>s</i> ]
<b>if</b> <i>n</i> = 1 <b>then</b> <i>robot</i> ( <i>r</i> → <i>s</i> )		
<b>else</b> <i>H</i> ( <i>n</i> − 1, <i>r</i> , 6 − <i>r</i> − <i>s</i> )		
<i>robot</i> ( <i>r</i> → <i>s</i> )		
<i>H</i> ( <i>n</i> − 1, 6 − <i>r</i> − <i>s</i> , <i>s</i> )		
<b>endif</b>		
<b>endpro</b>		
<i>H</i> ( <i>num, Pinit, Pfin</i> )		[Main algorithm]

One other remark before we give an example of HANOI: The reason for the use of  $6 - r - s$  is that, if *r* and *s* are any two of the three poles 1, 2, and 3, then  $6 - r - s$  is the third pole (e.g., if *r* = 1 and *s* = 3, then  $6 - r - s = 2$ ). Without this algebraic trick, HANOI would need all three pole numbers as inputs [8].

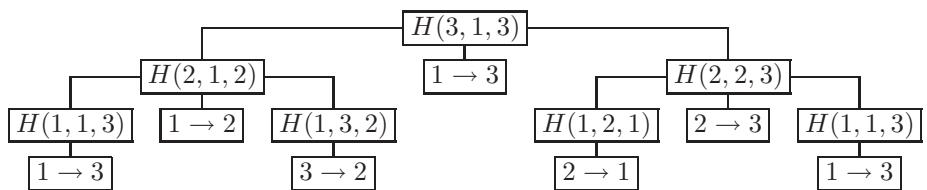
To show how HANOI works, let's consider what happens when *num* = 3, *Pinit* = 1, and *Pfin* = 3 (i.e., we wish to move three disks on pole 1 to pole 3). Figure 1.6 contains two ways of viewing the process that we're about to describe. Execution of the algorithm begins in the main algorithm with the call *H*(3, 1, 3) to the procedure, in which 3, 1, and 3, are assigned, respectively, to the variables *n*, *r*, and *s* in the procedure definition. Since *n* = 3 ≠ 1, we go initially to the **else** portion of the if-structure. With the current values of *n, r, s* this portion becomes

$$\begin{aligned} &H(2, 1, 2) \\ &\text{robot}(1 \rightarrow 3) \\ &H(2, 2, 3) \end{aligned} \tag{4}$$

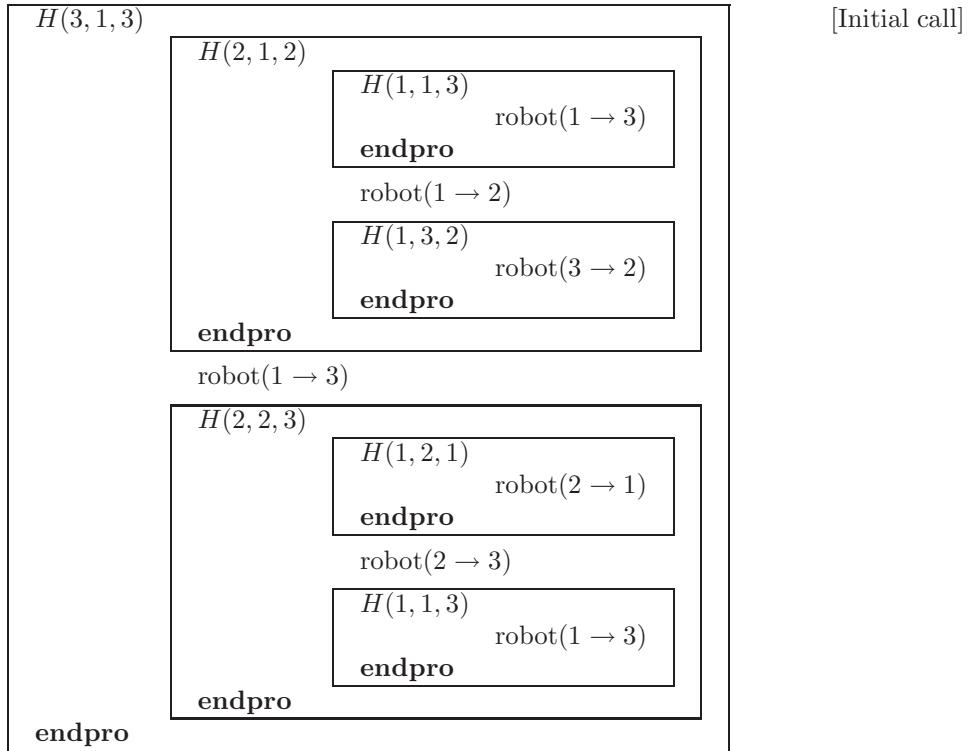
The first invocation of *H* within *H*(3, 1, 3), namely, *H*(2, 1, 2), causes us to *interrupt the normal sequential execution* of steps of the algorithm and *go back to the beginning* of a new copy of the procedure (indicated by indenting and by a new level of boxing in Fig. 1.6b), now with *n* = 2, *r* = 1, and *s* = 2. Again, within this copy *n* ≠ 1, so we go to the **else** portion, which now becomes

$$\begin{aligned} &H(1, 1, 3) \\ &\text{robot}(1 \rightarrow 2) \\ &H(1, 3, 2) \end{aligned} \tag{5}$$

a) Algorithm HANOI for  $num = 3$ ,  $Pinit = 1$ , and  $Pfin = 3$ .



b) Trace of  $H(3, 1, 3)$ . Each call of  $H(n, r, s)$  takes the trace one column to the right and each **endpro** takes it one column to the left.



**FIGURE 1.6**

Operation of  
Algorithm HANOI.

Once more the invocation of the procedure,  $H(1, 1, 3)$ , causes us to interrupt the sequence and return to the beginning of a new copy, with  $n = 1$ ,  $r = 1$ , and  $s = 3$ . Since now  $n = 1$ , the **then** portion applies:

$$H(1, 1, 3): \quad \text{robot}(1 \rightarrow 3).$$

We then reach **endpro** for  $H(1, 1, 3)$ , which means that we have completed  $H(1, 1, 3)$ , and so we return to the interrupted execution of (5). The second line of (5) adds  $\text{robot}(1 \rightarrow 2)$  to our list of instructions to the robot. Then we execute  $H(1, 3, 2)$ , whose total activity, since  $n = 1$  again, consists of a single instruction,  $\text{robot}(3 \rightarrow 2)$ . This then completes the execution of  $H(2, 1, 2)$ , the result of which is

robot( $1 \rightarrow 3$ )  
 robot( $1 \rightarrow 2$ )  
 robot( $3 \rightarrow 2$ )

You should verify that this sequence of moves solves the problem when  $n = 2$ ,  $r = 1$ , and  $s = 2$ . But, for our problem, we must now return to the interrupted execution of (4). Having completed  $H(2, 1, 2)$  we add robot( $1 \rightarrow 3$ ) to a list of commands to the robot and then execute the call  $H(2, 2, 3)$ . Since  $n \neq 1$ , we proceed as above and compute

$H(1, 2, 1)$   
 robot( $2 \rightarrow 3$ )  
 $H(1, 1, 3)$  (6)

Since  $n = 1$  in both invocations  $H$  in (6), we get for  $H(2, 2, 3)$ :

robot( $2 \rightarrow 1$ )  
 robot( $2 \rightarrow 3$ )  
 robot( $1 \rightarrow 3$ )

Finally, putting all these steps together, we get for  $H(3, 1, 3)$ :

$H(2, 1, 2)$	$\left\{ \begin{array}{ll} \text{robot}(1 \rightarrow 3) & [\text{Small disk to } 3] \\ \text{robot}(1 \rightarrow 2) & [\text{Middle disk to } 2] \\ \text{robot}(3 \rightarrow 2) & [\text{Small disk to } 2] \\ \text{robot}(1 \rightarrow 3) & [\text{Large disk to } 3] \end{array} \right.$	(7)
$H(2, 2, 3)$	$\left\{ \begin{array}{ll} \text{robot}(2 \rightarrow 1) & [\text{Small disk to } 1] \\ \text{robot}(2 \rightarrow 3) & [\text{Middle disk to } 3] \\ \text{robot}(1 \rightarrow 3) & [\text{Small disk to } 3] \end{array} \right.$	

You should verify that the list of instructions in (7) really is a solution. You might take a quarter, a nickel, and a dime and use them as the disks on imaginary poles. Figure 1.6a graphically illustrates the steps in solving this problem. Figure 1.6b is a trace of this algorithm, in which the “depth” of successive calls of  $H(n, r, s)$  is illustrated by the indentations from left to right across the page and by the boxes surrounding the successive calls of  $H$ . ■

Might self-invocation (recursion) get out of hand? That is, might you dig a hole so deep — that is, invoke the computation so many times without ever completing an execution of it — that you could never climb out with a solution to the original problem? A necessary condition for avoiding this situation is that there be at least one value of a defining variable for the recursive procedure or function which does not invoke another call. These are  $j = 0$  in EUCLID-RECFUNC and EUCLID-RECPRO, and  $n = 1$  in HANOI. But this alone is not enough to ensure that you will be able to “climb out of the hole”, that is, get back to the initial case at the top level of the recursion. For both recursive versions of EUCLID it’s rather easy to see that you do, indeed, climb out, but it’s not nearly so easy to see for HANOI. In

Section 2.4 we'll prove all these algorithms correct, and thus that they terminate. The proofs will use mathematical induction, which is the usual proof method for recursive algorithms.

The use of the recursive paradigm need not lead to a recursive algorithm. Sometimes the implementation of a recursive idea as an iterative algorithm allows a more felicitous presentation of the algorithm than would a recursive algorithm. Additionally, an iterative implementation will often lead to a more efficient algorithm (see Section 1.5 for what we mean by “efficient”) than the recursive implementation. Algorithm EUCLID (as given in Section 1.1) is an example of this, and we'll exhibit another in Section 1.5. When a straightforward iterative approach is available, you will usually understand it more easily. Often, however, there is no straightforward iterative approach. This is the case with the Towers of Hanoi problem, which is naturally recursive. Although it can be implemented iteratively (see [28, Section 2.5]), the iterative form is trickier and not so easy to understand.

Understanding Examples 1–3 is worth some effort. Try working through some cases. For the Towers of Hanoi, you'll find that solving the problem when  $n > 6$  is very tedious. But solving the problem for  $n \leq 6$  should be fun. Success will surely give you some insight into the Towers of Hanoi problem in particular and recursion more generally.

## Problems: Section 1.3

---

*Note:* There are more problems about recursive algorithms at the end of Section 1.4.

1. ⟨1⟩ We argued that Eq. (2) fits the recursive paradigm because the *second* argument is reduced at each stage. But how about the *first* argument? Isn't it also reduced at each stage? Explain.

2. ⟨2⟩

- a) In Example 4, Section 1.1, we derived another relation besides Eq. (2) which equates the gcd of two different pairs of arguments:

$$\gcd(m, n) = \gcd(m - n, n)$$

when  $m \geq n$ . Write a recursive algorithm analogous to EUCLID-RECFUNC which uses this equation. Show how your algorithm works when  $m = 63$  and  $n = 27$  by displaying the calculation and by drawing a figure analogous to Fig. 1.2. *Note:* The algorithm will have to switch the variables when  $m - n < n$ .

- b) Repeat a) for EUCLID-RECPRO.

3. ⟨2⟩ Rewrite Algorithm 1.7 from [12, Section 1.1] in the form of

- a) a recursive function,
  - b) a recursive procedure.
4. ⟨3⟩ The least common multiple (lcm) of two nonzero integers  $m$  and  $n$  is the smallest positive integer which is a multiple of both  $m$  and  $n$ . For example,  $\text{lcm}(15, 24) = 120$ . Although there is no simple relationship like Eq. (2) or that in [2] for the lcm, we can derive efficient methods for computing the lcm using its relationship to the gcd.
  - a) Show that  $\gcd(m, n) \text{lcm}(m, n) = mn$  if  $m, n > 0$ . (This is the conjecture you should have made if you attempted [3b, Supplement, Chapter 0].) *Hint:* Express both  $m$  and  $n$  as the products of their prime factors. Thus,  $15 = 3 \times 5$  and  $24 = 2^3 \times 3$ . Then you can find both the gcd and the lcm by comparing the exponents of the same prime in the two factorizations and choosing the maximum for the lcm and the minimum for the gcd. Thus  $\text{lcm}(15, 24) = 2^3 \times 3 \times 5 = 120$  and  $\gcd(15, 24) = 3$ .
  - b) Use the relationship in a) and the recursive function in Algorithm EUCLID-RECFUNC to

write an algorithm to compute the lcm of two nonnegative integers  $m$  and  $n$ .

- c) Repeat b) but this time using EUCLID-REC<sub>PRO</sub> and writing a recursive procedure.
5. ⟨2⟩ Use the relationship in [4a] to write an algorithm to compute the lcm, using Algorithm EUCLID in Section 1.1 for the computation of the gcd.
6. ⟨3⟩
- a) Write a recursive procedure whose input is two integers as in EUCLID and whose output is both the gcd and the lcm of these two integers.
  - b) If only recursive functions (but not recursive procedures) were available to you, could you write an algorithm which calculated both the gcd and lcm but used only a single recursive function? If not, why not? If so, how?
7. ⟨1⟩ List the set of moves in the Towers of Hanoi problem for
- a)  $n = 2$
  - b)  $n = 4$ .
8. ⟨1⟩ Avoid the use of  $6 - r - s$  by rewriting the procedure  $H$  in HANOI as  $H(n, P_i, P_f, P_t)$ , where  $P_i$  is the initial pole,  $P_f$  is the final pole, and  $P_t$  is the third pole.
9. ⟨2⟩ In the Towers of Hanoi problem suppose that there are four pegs instead of three. Then the
- problem of finding a solution with *the minimum number of moves* is quite difficult. But try doing it for
- a)  $n = 3$
  - b)  $n = 4$
10. ⟨2⟩ You are given  $n$  different rings as in TOH, but now there are only two poles. Your goal is to move them from the smallest-down-to-largest configuration on the start pole to a largest-down-to-smallest on the other pole. (Clearly, larger rings may now go on smaller ones.) There is an obvious way to do this: Just transfer the rings one at a time.
- a) Describe this solution recursively in ordinary English. That is, use a phrase like “suppose that we already knew how to do this for  $n - 1$  rings.”
  - b) Write your solution as an algorithm using a recursive procedure. You may use the command robot( $r \rightarrow s$ ).
11. ⟨2⟩ There are actually two different recursive representations of the solution of [10]. In one version, you move a single ring and then do the recursive step; in the other, you move a single ring after the recursive step. Whichever you did in [10], now do the other. (The former solution is an instance of **tail-end** recursion; the latter, an instance of **front-end** recursion.)

## 1.4 Algorithmic Language: Procedures and Functions

In Section 1.3 we introduced procedures and functions informally and used them in algorithms. In this section we'll give you the details of their syntax and semantics (i.e., the *meaning* of the syntax). We pay special attention to the rules for passing values to and from calls of procedures and functions. Some things that appear to be picky aspects of these rules are crucial to why recursive algorithms work, and why they can be proved correct by induction (as we'll discuss in Ch. 2).

### Procedures

Both recursive algorithms with procedures in Section 1.3 had the form

Procedure definition  
Main algorithm with  
call to procedure

The procedure definition specified the task to be performed. Then the call (i.e., invocation) in the main algorithm activated the procedure for particular initial values of the arguments. This model is also characteristic of nonrecursive algorithms in which procedures are useful.

The structure of a procedure is

```
procedure procname(variable list)
    procedure body
endpro
```

where the procname is, as with gcdp and  $H$  in Section 1.3, just something used to refer to the procedure.

The variable list names variables which are either:

1. *input* to the procedure; that is, these are the data provided to the procedure;
2. *output* from the procedure; that is, these are results produced by the procedure; or
3. *input* to and *output* from the procedure; that is, these are variables which provide data to the procedure but whose values may be changed by the procedure so that these changed values are also results of the procedure.

The syntax of the variable list is:

**in** list of input variables; **out** list of output variables; **inout** list of variables which are both input and output variables,

where each of the lists is a sequence of variables separated by commas, and the lists are separated by semicolons. Each of these lists may be omitted; for example, if there are no variables that are just output variables, then **out**, its list and the semi-colon will be omitted.

The call of a procedure, as in the case of EUCLID-RECPRO and HANOI, consists of the name of the procedure followed by a list of expressions in parentheses, separated by commas within the **in**, **out** and **inout** lists and by semi-colons between the lists. These **calling arguments** are then associated one-to-one with the **defining (or dummy) variables** in the definition of the procedure by considering the **in**, **out** and **inout** lists to be a single list. Thus when we call procedure gcdp(**in**  $i, j$ ; **out**  $result$ ) in EUCLID-RECPRO with

$$\text{gcdp}(m, n; answer)$$

$m$  is associated with  $i$ ,  $n$  with  $j$ , and  $answer$  with  $result$ .

However, what “associated with” means depends on whether the defining variable is **in**, **out** or **inout**. For each argument in the **in** list, what is transmitted to the procedure, when the procedure is called, is merely the *value* of the argument. For instance, in the call gcdp( $m, n; answer$ ) of EUCLID-RECPRO the value of  $m$  is assigned to the variable  $i$ . If  $i$  is then changed when the procedure is carried out,

$m$  does not change. (In the parlance of computer science, this means that **in** arguments are *called by value*.) Since an **in** argument is not changed by a procedure, it does not, in fact, have to be a variable; it could be a constant, or any mathematical expression. If it is an expression, what is passed to the procedure is the value of that expression (i.e., its value using the current values of its variables at the time the procedure is called).

For both the **out** and **inout** lists the arguments must be variables. When these arguments are associated with variables in the procedure, that means each argument is made *synonymous* with its corresponding defining variable. For instance, the call `gcdn(m, n; answer)` of EUCLID-RECPRO makes the argument *answer* in the main algorithm a synonym for the **out** defining variable *result* in the procedure.

When a variable changes value in the procedure, its synonym changes value in the main algorithm. In flesh and blood computers, the way this is accomplished is by transmitting to the procedure the *address* of the calling argument (e.g. *answer*). This address is the location in the computer's memory where the value of the procedure variable (e.g., *result*) is stored, thus, in effect, making the calling argument and the defining variable synonyms. If the calling argument has not already been assigned a value by the calling algorithm — which is exactly the case for the argument *answer* in EUCLID-RECPRO when `gcdn(m, n; answer)` is called from the one-line main algorithm — then this argument is immediately created in the main algorithm. In any case, using the terminology of computer science, we say that the **out** and **inout** variables are *called by reference*. These transmittal rules are illustrated for EUCLID-RECPRO in Fig. 1.7.

The variables following the procedure name are called *defining* variables because they serve only to define the procedure. They are also called *dummy* variables because such a variable only reserves a place for the arguments actually used when the procedure is called.

The names of the calling arguments and defining variables have always been different in our examples thus far to emphasize their distinct roles (e.g., in EUCLID-RECFUNC the calling arguments were  $m, n$  and the defining variables were  $i, j$ ). But these names may be — and sometimes in later sections will be — the same. For instance, in the analysis of algorithms we will often use the same names for calling arguments and defining variables to avoid unpleasant switches between the two in the analysis.

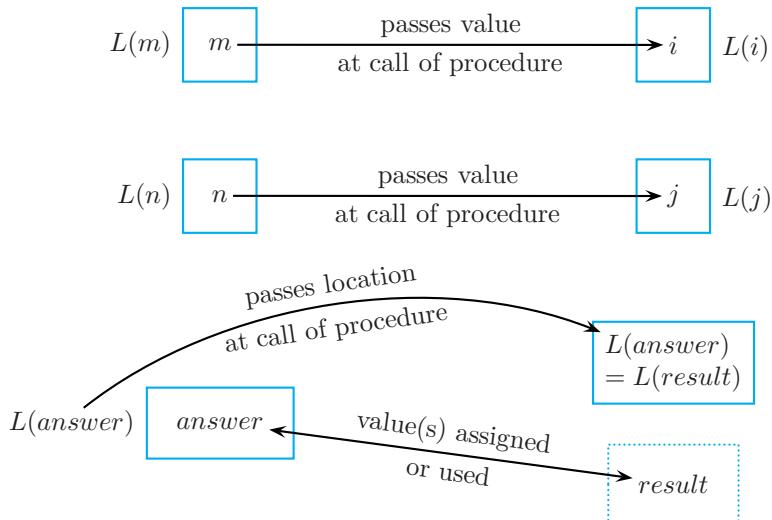
In general, if care is taken, variable names from, for example, the main algorithm, can be reused in a procedure without mishap, because of local variable rules to be explained shortly. Nevertheless, our advice is: Only reuse variable names when there is a distinct advantage to doing so, as in the analysis of algorithms.

We've mentioned that each of the three lists of defining variables can be empty. The **in** list may be empty because we allow all variables named in the **Input** section of an algorithm to be accessible from any place in the algorithm. (Such Input variables are sometimes said to be **global**.)

At first glance it might seem strange that both the **out** and **inout** sections could be empty for, if so, how could the procedure compute any results? But the purpose of the procedure might be something other than computing values of

Main Algorithm  
 $\text{gcdp}(m, n; \text{answer})$

Procedure  
 $\text{gcdp}(\text{in } i, j; \text{out } result)$



Parameter Passing for EUCLID-RECPRO.  $L(m)$  is the name of the location (box) where the value of  $m$  is stored, and likewise for other variables. When a procedure is called, for each **in** variable, the value in the calling argument location is transferred at the start of the procedure to the defining variable location (e.g., the value of  $m$  is transferred to  $i$ ). However, for each **out** variable a special box is set up which contains the name of the location of the calling argument, not the argument's value. For instance, when  $\text{gcdp}(m, n; \text{answer})$  is called (making  $\text{answer}$  the calling argument in the main algorithm for the defining variable  $result$  in the procedure), a box is set up in the procedure in which  $L(\text{answer})$  is stored. Any time  $result$  is assigned a value in the procedure, the effect is to change the value of  $\text{answer}$  in the main algorithm. Similarly, any time a value of  $result$  is used in the procedure, the value used is taken from  $\text{answer}$  in the main algorithm. The box with a dotted outline (a virtual box) represents the fact that a value of  $result$  never actually exists in the procedure itself.

**FIGURE 1.7**

variables. This is the situation in HANOI where the purpose is to produce a list of physical commands to the robot, which are directly output by the procedure, so no output variables are needed.

The following procedure illustrates many of the points above as well as two further points not illustrated by the previous algorithms in this chapter: the use of an **inout** variable, and the use of procedures without recursion. Real-life non-recursive algorithms are often quite large, and using procedures to break them into chunks is often quite helpful.

## EXAMPLE 1

### String Processing

Suppose that you have a **string**  $S$ , that is, a sequence of characters  $s_1, s_2, \dots, s_n$ , where each character is either a letter, a digit, or a blank, except that the final character, indicating the end of the string, is some special character, say,  $\|=$ . Suppose also that, while processing  $S$  (i.e., performing some operations on  $S$ ), you wish to ignore all the blanks, which you know may come in groups of more than one. This situation is common to many computer applications. How do you process  $S$ ?

**Solution** Algorithm 1.11 STRINGPROC is the outline of a solution. Pay special attention to the procedure Skipblanks. Variable  $j$  ranges over positions in the string. Note that  $S$  need not be a defining argument of Skipblanks because, as an Input, it is automatically accessible in Skipblanks. The procedure Skipblanks does nothing if the character  $s_j$  is not a blank; otherwise, it moves  $j$  along the string  $S$  until a nonblank character is found. The output of Skipblanks is  $j$ , which is the index of the next nonblank character. This value replaces the previous value of  $i$  in the main algorithm. Hence  $j$  is an **inout** variable.

We won't display a body for the procedure Processcharacter because its form depends on what processing you wish to do. However, it better be processing that doesn't change the input  $S$ , or else we should start by assigning  $S$  to another variable, as discussed in Remark 3 on p. 83, Section 1.1. In [26] we'll ask you to write some procedure bodies for Processcharacter to accomplish various useful tasks that fit this requirement.

#### Algorithm 1.11

##### StringProc

<p><b>Input</b> <math>S</math></p> <p><b>Output</b> Results of processing <math>S</math></p> <p><b>Algorithm</b> STRINGPROC</p> <p><b>procedure</b> Skipblanks(<b>inout</b> <math>j</math>)</p> <p style="padding-left: 2em;"><b>repeat while</b> <math>s_j = \emptyset</math> <span style="float: right;">[<math>\emptyset</math> stands for blank]</span></p> <p style="padding-left: 2em;"><math>j \leftarrow j + 1</math></p> <p style="padding-left: 2em;"><b>endrepeat</b></p> <p><b>endpro</b></p> <p><b>procedure</b> Processcharacter(<b>in</b> <math>k</math>) <span style="float: right;">[Uses <math>s_k</math>]</span></p> <p style="padding-left: 2em;">body of Processcharacter</p> <p><b>endpro</b></p> <p style="padding-left: 2em;"><math>i \leftarrow 1</math> <span style="float: right;">[Main algorithm]</span></p> <p style="padding-left: 2em;"><b>repeat while</b> <math>s_i \neq \ =</math> <span style="float: right;">[<math>\ =</math> is end of string marker]</span></p> <p style="padding-left: 2em;">Skipblanks(<math>i</math>)</p> <p style="padding-left: 2em;">Processcharacter(<math>i</math>)</p> <p style="padding-left: 2em;"><math>i \leftarrow i + 1</math></p> <p><b>endrepeat</b></p>
---

Note that in this algorithm, as well as the recursive algorithms in Section 1.1, we have first defined procedures at the beginning of the algorithm and then later used them in the main algorithm. When writing an algorithm, it is natural to write the main algorithm first and then write the necessary procedures. But, in presenting an algorithm, it's often better to do it the other way around because, for the *reader* of an algorithm, it's usually nice to know what the procedures are supposed to accomplish before the reader reaches the calls to them in the main algorithm. Still, we set no firm rule about this. ■

Even when a procedure is as short as Skipblanks, it may make an algorithm quite a bit more readable to structure it as in Example 1 instead of — as we could have done — replacing the call to Skipblanks in the main algorithm with the three statements:

```
repeat while  $s_i = \emptyset$ 
     $i \leftarrow i + 1$ 
endrepeat
```

Note that we used  $i$  here instead of  $j$  as in Algorithm STRINGPROC. Why is this OK, indeed necessary?

## Recursion and Passing Variables

We still need to discuss one aspect — and a quite tricky one — of how procedure variables are transmitted between a calling program and a recursive procedure: What happens with recursive procedures when, in effect, the same variables are transmitted again and again before the original call of the recursive procedure is completed? For example, consider Algorithm HANOI. The main algorithm consists of one call of  $H(\text{in } n, r, s)$  with initial assignments:

$$n \leftarrow num, \quad r \leftarrow Pinit, \quad \text{and} \quad s \leftarrow Pfin. \quad (1)$$

Inside the procedure  $H$  we have

$$\begin{aligned} &H(n-1, r, 6-r-s) \\ &\text{robot}(r \rightarrow s) \\ &H(n-1, 6-r-s, s) \end{aligned} \quad (2)$$

The subcall  $H(n-1, r, 6-r-s)$  then initiates a new assignment of values to the dummy variables  $n$ ,  $r$ , and  $s$  in the definition of  $H$ :

$$n \leftarrow n-1 = num - 1, \quad r \leftarrow r = Pinit, \quad s \leftarrow 6-r-s = 6 - Pinit - Pfin.$$

The new calls of  $H$  then spawned will produce still other assignments to these dummy variables. So, when the execution of the first line of (2) is finished, what reason do you have to trust that  $n$ ,  $r$ , and  $s$  have the original values (1), which are needed for execution of the second and third lines of (2)?

The answer is found in the following convention for all procedures (and functions), recursive or not: Each call of a procedure creates a template (collection) of

entirely new **local** variables. Each local variable in each call is entirely separate from all variables in all other calls, or in the main algorithm, even though the variables in the different calls may have the same names. (Indeed, since a recursive procedure is defined just once but normally called many times, the variables in it *will* automatically have the same names in each call.)

The local variables in each call are:

1. all variables in the **in** list;
2. all other variables used in the procedure that are not **Input** (global), **inout**, or **out** variables.

An example of a type 2 local variable would be a counter, say  $i$  in **for**  $i = a$  **to**  $b$ , where  $a$  and  $b$  might be **in** variables.

Notice that **inout** and **out** variables are not local; they cannot be since their purpose is to affect a higher level. Variables from the **Input** section of an algorithm also cannot be local because they are intended to be available as data everywhere. However, as noted earlier, you should take care not to change the value of an **Input** variable. If you need to change such a value within a procedure, then make the **Input** variable the calling argument of a defining **in** variable, which then will be local to the procedure.

Let's see how the local variable rules work for Hanoi, by returning to the example on p. 101 where  $num = 3$ ,  $Pinit = 1$  and  $Pfin = 3$ . The original call  $H(num, Pinit, Pfin)$  causes a template of the values of the local variables (only  $n$ ,  $r$  and  $s$  in this example) to be generated:

$$3 (= n), \quad 1 (= r), \quad 3 (= s). \tag{3}$$

Then the subcall  $H(n-1, r, 6-r-s)$  generates another template with another set of local variables with values  $n = 2 (= 3-1)$ ,  $r = 1$  and  $s = 2 (= 6-1-3)$ . Similarly, all subsequent calls of  $H$  within  $H(n-1, r, 6-r-s)$  create new templates of local variables with new values. When each call is made, the local variables at the level from which the call is made (the **calling level**) are put aside in favor of those in the template created at the new level, and when each level is completed and control is returned to the calling level, the template in the calling level is restored. Thus, when  $H(n-1, r, 6-r-s)$  completes execution (i.e., we reach the second line of (2)), the template (3) is restored, as it must be if the second and third lines of (2) are to do what they are supposed to.

Now look again at Figure 1.2 for Euclid's algorithm. The templates at each level are, in effect, the assignments to  $i$  and  $j$  just above the rectangles.

Any recursive procedure may thus create many variables with the same name (one in the template for each call) but our convention ensures that these variables do not modify each other in unintended ways. In particular, these conventions ensure that the variables have the right meanings at the right times in order to prove the algorithms correct by induction in Chapter 2. Our convention also allows a defining **in** variable to have the same name as a calling argument; changes to the former won't change the latter because **in** variables are local.

## Functions

Functions treat the case where a procedure has exactly one **out** variable and no **inout** variables. Furthermore, since the **out** quantity will be associated with the function name itself, there is no need to give it another variable name, or list it on the first line. Consequently, the syntax for functions can be simpler. Here is what we use:

```
function funcname(variable list)
    function body with at least one line of the form:
        funcname ← value of expression
endfunc
```

Since all variables on the first line must be **in** variables, we don't bother to write **in**. The rules for what variables are local are exactly the same as for procedures, and thus in a function *every* non-global variable except funcname is local; only funcname affects higher levels. If, say, function funcname is called by funcname(arguments), then when **endfunc** is reached, the (last) value assigned to variable funcname in the function body (using  $\leftarrow$ ) is assigned to the expression funcname(arguments) at the calling level.

For example, consider again EUCLID-RECFUNC from Example 1, Section 1.3, with the input values given in Figure 1.2:

<b>Input</b> $m, n$	[Integers $\geq 0$ ]
<b>Output</b> $answer$	$[= \gcd(m, n)]$
<b>Algorithm</b> EUCLID-RECFUNC	
<b>function</b> gcdf( $i, j$ )	
<b>if</b> $j = 0$ <b>then</b> gcdf $\leftarrow i$	
<b>else</b> gcdf $\leftarrow \text{gadf}(j, i - j \lfloor i/j \rfloor)$	[Recursive call]
<b>endif</b>	
<b>endfunc</b>	
$answer \leftarrow \text{gadf}(63, 27)$	[Main algorithm]

The main call of gcdf comes from the line  $answer \leftarrow \text{gadf}(63, 27)$ , so funcname is gcdf and the arguments are 63 and 27. In the function body of this call, the **then** line is skipped, so the only active line is **else** gcdf  $\leftarrow \text{gadf}(27, 9)$ . This line assigns to the variable gcdf the value 9 (obtained by returns from subcalls). The processing then proceeds to **endfunc** in the main call, which ends the call and assigns the value of gcdf (namely 9) to the expression gcdf(63, 27) in the main algorithm, which is then assigned to answer and we are done.

Remember, funcname(arguments) always produces a single value, whereas procname(arguments) is a name for the whole process of doing the procedure. The fact that funcname is a value is especially useful in algebraic computations [6].

# Problems: Section 1.4

1. ⟨1⟩ Suppose that AL did not include the absolute value notation (but did include all the other mathematical notation we have used). Write a function ABS( $x$ ) which returns  $|x|$ .
2. ⟨2⟩ In some computer languages, INT( $x$ ) is the integer part of  $x$  closer to 0. That is, INT( $-3.4$ ) =  $-3$  and INT( $4.9$ ) =  $4$ . (In other languages, INT( $x$ ) =  $\lfloor x \rfloor$ , so that INT( $-3.4$ ) =  $-4$ .) In a) and b), use the “closer to 0” meaning.
  - a) Define INT( $x$ ) as a function in AL. You may use  $\lfloor \cdot \rfloor$  and  $\lceil \cdot \rceil$ , since mathematical notation is allowed in AL.
  - b) Now suppose we assume that INT( $x$ ) is part of our language (i.e., could be used without writing the function in a)) and, further, suppose that  $\lfloor \cdot \rfloor$  and  $\lceil \cdot \rceil$  are not allowable notation. Write a definition of Floor( $x$ ) in our language.
3. ⟨2⟩ According to one convention, rounding off should be done as follows. Any number not exactly halfway between two integers should be rounded to the closest integer. Those numbers which end in .5 should be rounded to the even integer which is  $\frac{1}{2}$  away. Call this rounding function  $R(x)$ . Use AL to define  $R(x)$  in terms of floor and/or ceiling (See [17–21, Section 0.2]).
4. ⟨2⟩ Consider this algorithm:

```
Algorithm SUMMYSTERY
procedure SqSum(in N; out k, S)
    k ← 0; S ← 0
    repeat until S > N
        k ← k+1
        S ← S + k2
    endrepeat
endpro
SqSum(10; k, S)
```
5. ⟨1⟩
  - a) Write a function min( $x, y$ ) whose value is the smaller of  $x$  and  $y$ .
  - b) Use this function in an algorithm which prints the minimum of three inputs,  $a$ ,  $b$ , and  $c$ . Compare your results with [12, Section 1.2], where you were asked to accomplish a similar task without functions (or procedures).
6. ⟨2⟩ Functions are much simpler to use than procedures when you need to compute an expression involving two or more values of the same function. For instance, suppose you need to compute  $f(x) + f(3)$  for some function  $f$ . You can define your function in terms of some dummy variable, and then have just one line in your main algorithm:  $answer \leftarrow f(x) + f(3)$ . However, suppose you instead defined a procedure fpro(in  $z$ ; out result). Write an appropriate main algorithm.
7. ⟨2⟩ Later we shall use the notation  $x \leftrightarrow y$  to denote the interchange of the values of  $x$  and  $y$ . Write a procedure Switch(inout  $x, y$ ) which exchanges the values of the inputs. Use this procedure within an algorithm, REVERSE, which takes a sequence and reverses it. (Let the input be  $n, a_1, a_2, \dots, a_n$ .)
8. ⟨2⟩
  - a) What does this algorithm do? Explain.

```
Algorithm COUNTDOWN
procedure Shout(in n)
    if n > 0 then Yell n
        Shout(n-1)
    else Yell "Blastoff!"
    endif
endpro
Shout(10) [Main algorithm]
```
  - b) Anything wrong with this slight revision?

```
Algorithm COUNTDOWN1
procedure Shout'(in n)
    Yell n
    if n > 0 then Shout'(n-1)
    else Yell "Blastoff!"
    endif
endpro
Shout'(10) [Main algorithm]
```
  9. ⟨2⟩ In Algorithm COUNTDOWN2 below we reverse the positions of Yell and Shout from their order

in Algorithm COUNTDOWN in [8a]. Does this little change make a difference?

```
Algorithm COUNTDOWN2
procedure Shout"(in n)
  if n > 0 then Shout"(n-1)
    Yell n
  else Yell "Blastoff!"
  endif
endpro
Shout"(10)
```

[Main algorithm]

10. ⟨2⟩ Algorithm MULT2 below multiplies two numbers, one an integer, by repeated addition, and does so by using a recursive procedure.

### Algorithm

```
Input x, y                                [y ∈ N]
Output prod

Algorithm MULT2
procedure MR(in p, q; out prod)
  if q = 0 then prod ← 0
  else MR(p, q-1; prod)
    prod ← p + prod
  endif
endpro
MR(x, y; prod)
```

- a) Draw a tree diagram (see Fig. 1.6a) for the calls of MR with inputs (i)  $x = 2$ ,  $y = 3$ , and (ii)  $x = 3$ ,  $y = 5$ .
- b) This problem is slightly easier if you use a recursive function instead of a procedure, so redo it that way.

### Algorithm

```
Input x, y                                [y ∈ N]
Output prod

Algorithm MULT3
procedure MR'(in p, q; inout prod)
  if q > 0
    then MR'(p, q-1; prod)
    prod ← p + prod
  endif
endpro
prod ← 0
MR' (x, y; prod)
```

11. ⟨2⟩ Algorithm MULT3 is a variant of the algorithm presented in [10], which also correctly multiplies. Why isn't an **else** clause needed for the **if** structure?

12. ⟨2⟩ Here is a procedure to do something very simple with the TOH rings.

```
procedure TOH?(in n, r, s)
  robot(r → s)
  if n > 1 then TOH?(n-1, r, s) endif
endpro
```

What does this procedure accomplish? Write an iterative procedure to do the same thing.

13. ⟨2⟩ Here is another TOH procedure.

```
procedure TOH??(in n, r, s)
  robot(r → 6-r-s)
  if n > 1 then TOH??(n-1, r, s) endif
  robot(6-r-s → s)
endpro
```

What does this in fact accomplish? Write an iterative procedure to do the same thing.

14. ⟨2⟩ Look at Algorithms MIN1 and MIN2 below, each of which employs a recursive function to find the smallest of  $n$  inputted numbers. Note: Both algorithms make use of the fact that Input variables are global. If they weren't, functions Min and Min' would have to list all the inputs as **in** variables. This is a variable number  $n$  of inputs and ugly to write at the least.

a) Min' takes an additional line and introduces an additional variable, *temp*. Yet it is more efficient than Min. Why?

b) Write an Algorithm MIN3 which does the same thing as MIN1 and MIN2 but uses a recursive procedure instead of a recursive function. (Using a recursive procedure is often better if you have to make a lot of conditional decisions before you can use the results of the previous case to decide the current case.)

15. ⟨3⟩ Algorithm MAX2 below is supposed to find and print the largest of  $n$  numbers. What's wrong with it? Fix procedure Max( $m$ ) without turning it into a function.

## Algorithm

**Input**  $n, a_1, a_2, \dots, a_n$

**Output**  $small1$

### Algorithm MIN1

```

function Min( $k$ )
  if  $k = 1$  then Min  $\leftarrow a_1$ 
  else if  $a_k < \text{Min}(k-1)$ 
    then Min  $\leftarrow a_k$ 
    else Min  $\leftarrow \text{Min}(k-1)$ 
  endif
endif
endfunc
small1  $\leftarrow \text{Min}(n)$ 

```

## Algorithm

**Input** Same as MIN1

**Output**  $small2$

### Algorithm MIN2

```

function Min'( $k$ )
  if  $k = 1$  then Min'  $\leftarrow a_1$ 
  else temp  $\leftarrow \text{Min}'(k-1)$ 
    if  $a_k < \text{temp}$  then Min'  $\leftarrow a_k$ 
    else Min'  $\leftarrow \text{temp}$ 
  endif
endif
endfunc
small2  $\leftarrow \text{Min}'(n)$ 

```

## Algorithm

**Input**  $n, a_1, a_2, \dots, a_n$

[ $n \geq 1$ ]

**Output** largest number

### Algorithm MAX2

```

procedure Max(in  $m$ )
  if  $m = 1$  then print  $a_1$ 
  else
    if  $a_m > \text{Max}(m-1)$ 
      then print  $a_m$ 
      else Max( $m-1$ )
    endif
  endif
endpro
Max( $n$ )

```

16. (3)

- a) Suppose we wish to find the greatest integer

that divides  $k$  integers,  $r_1, r_2, \dots, r_k$ . We claim that

$$\gcd(r_1, r_2, \dots, r_k) = \gcd(\gcd(r_1, r_2, \dots, r_{k-1}), r_k).$$

Give an argument to justify this equation — using prime factorization is easiest.

- b) Write an iterative algorithm to find the gcd of  $k$  integers using the formula in a).  
 c) Write a recursive algorithm kgcd which does the same calculation as the algorithm you wrote for b).

17. (3) What values does Algorithm SAMPLE1 below print? (The sole purpose of the algorithm is to illustrate the nature of local variables, and how they are isolated from the main algorithm unless they are explicitly passed back.) The notation

$$(a_1, \dots, a_n) \leftarrow (b_1, \dots, b_n)$$

is shorthand for the  $n$  commands  $a_1 \leftarrow b_1, a_2 \leftarrow b_2, \dots, a_n \leftarrow b_n$ .

### Algorithm SAMPLE1

```

procedure Sum(in  $n$ ; out  $S$ )
   $S \leftarrow 0$ 
  for  $k = 1$  to  $n$ 
     $S \leftarrow S + k$ 
  endfor
   $n \leftarrow n^2$ 
  print  $k, n, S$ 
endpro

```

$$(S, k, m, n) \leftarrow (1, 2, 3, 4)$$

[Main alg.]

$$\text{Sum}(n; S)$$

$$\text{print } S, k, m, n$$

$$\text{Sum}(m; T)$$

$$\text{print } S, k, m, n$$

18. (2) What values does Algorithm SAMPLE2 below print? Do a trace. Show the value of variables at the main level and inside the procedure at the start and end of each procedure call.

### Algorithm SAMPLE2

```

procedure Foo(inout  $p$ )
   $p \leftarrow p + 7$ 
endpro

```

$$p \leftarrow 2; q \leftarrow 100$$

[Main algorithm]

$$\text{Foo}(p)$$

$$\text{print } p, q$$

$$\text{Foo}(q)$$

$$\text{print } p, q$$

[Which one is changed?]

19. **(4)** For all parts of this problem, assume the following procedure has already been defined.

```
procedure Powers(in  $x, y$ ; out  $p, q$ )
   $p \leftarrow x^y$ 
   $q \leftarrow y^x$ 
endpro
```

You will be asked to run several traces. In many cases there will be variables of the same name at both the main level and within the procedure. (There are only two levels since the procedure is not recursive.) It is best to distinguish them. For instance, you might write  $p_m$  and  $p_i$  for  $p$  at the main level and  $p$  internal to the procedure.

- a) What does  $\text{Powers}(2,3; p, q)$  do? Run a trace to confirm. (Assume the whole main algorithm consists of this one line.)
- b) What does the following algorithm fragment do? Run a trace to confirm. This is a bit tricky, since more than one defining variable gets associated with each calling variable. Such multiple associations are *bad* algorithm-writing practice, but it allows this example to highlight the different roles of **in** and **out** variables.

```
 $p \leftarrow 2; q \leftarrow 3$ 
Powers( $p, q; p, q$ )
```

- c) What is the output of the following algorithm fragment? Again, it involves the bad practice of associating one calling variable with more than one defining variable.

```
 $p \leftarrow 2; q \leftarrow 3; x \leftarrow 4$ 
Powers( $p, q; p, q$ )
Powers( $p, q; p, q$ )
Output  $p, q, x$ 
```

20. **(3)** Here is a slightly different version of the Powers procedure

```
procedure Powers2(out  $p, q$ ; inout  $x, y$ )
   $p \leftarrow x^y$ 
   $q \leftarrow y^x$ 
endpro
```

- a) The analog of [19a] would be to call  $\text{Powers2}(p, q; 2,3)$ . Why is that not possible?
- b) The analog of [19b] would be

```
 $p \leftarrow 2; q \leftarrow 3$ 
Powers2( $p, q; p, q$ )
```

Does this have the same results as [19b]? Run a trace to find out.

21. **(3)** Draw a tree diagram of procedure calls, similar to Figure 1.6a for TOH, for the following algorithms.

- a) Algorithm FIB-RECUR, when  $n = 5$ . This algorithm computes the *Fibonacci numbers*, which are defined in Section 2.7 (see also Section 5.2).

### Algorithm

**Input**  $n$   
**Output** *answer*

**Algorithm** FIB-RECUR

```
function Fib( $n$ )
  if  $n = 0$  or  $1$ 
    then Fib  $\leftarrow 1$ 
    else Fib  $\leftarrow \text{Fib}(n-1) + \text{Fib}(n-2)$ 
    endif
  endfunc
answer  $\leftarrow \text{Fib}(n)$ 
```

*Note:* This is a very inefficient algorithm for generating the Fibonacci numbers; your tree diagram should show you why. For this problem, and indeed for most recursively defined sequences of numbers, an iterative algorithm is better.

- b) Algorithm SUM-RECUR, for  $n = 4$ .

### Algorithm

**Input**  $n, a_1, a_2, \dots, a_n$   
**Output** *sigma*

**Algorithm** SUM-RECUR

```
function Sum1( $k$ )
  if  $k = 1$  then Sum1  $\leftarrow a_1$ 
  else Sum1  $\leftarrow a_k + \text{Sum1}(k-1)$ 
  endif
  endfunc
sigma  $\leftarrow \text{Sum1}(n)$ 
```

- c) Algorithm SUM-RECUR2 for  $n = 4$ . (Your diagram should help clarify the difference between this algorithm and the one in b.).

## Algorithm

**Input**  $n, a_1, a_2, \dots, a_n$

**Output**  $\sigma$

**Algorithm** SUM-RECUR2

```
function Sum2( $k$ )
    if  $k = n$  then Sum2  $\leftarrow a_n$ 
    else Sum2  $\leftarrow a_k + \text{Sum2}(k+1)$ 
    endif
endfunc
 $\sigma \leftarrow \text{Sum2}(1)$ 
```

22. (3) Write two functions in AL for each of the following expressions. One function should be iterative and one recursive.

- a)  $n!$
- b)  $H(n) = \sum_{k=1}^n 1/k.$
- c)  $T(n) = \sum_{k=1}^n k.$  ( $T(n)$  is called the  **$n$ th triangular number**. Why?)

d)  $s_n$ , where  $s_0 = 0$  and  $s_n = 3s_{n-1} + 2$ .

23. (3) For each of the following quantities, write two algorithms to compute it, one iterative and one recursive.

- a) The product of  $n$  inputted numbers.
- b) The union of  $n$  inputted sets,  $S_1, S_2, \dots, S_n$ .

24. (3) Write a recursive procedure BN( $m$ ) which looks at the first  $m$  available inputs and assigns to the variable  $B$  the value of the biggest of those inputs and assigns to  $N$  the value of the next biggest. Put this procedure in an algorithm which, given specific inputs  $a_1, a_2, \dots, a_n$  and  $n$ , prints the biggest and next biggest of the  $a_i$ 's.

25. (3) Return to the minimum completion time problem of the Prologue. We didn't write up the

solution in algorithmic language because we hadn't introduced the language yet.

- a) Write the solution method iteratively. (This should produce just the "work out of the hole" part of the solution.)
- b) Write the solution method recursively. (This should produce both the "dig a hole" and the "work out of the hole" aspects of the solution.)

26. (3) Write procedures for Processcharacter in Algorithm STRINGPROC which do each of the following. You will probably need to give Processcharacter at least one **out** or **inout** variable.

- a) Count the number of each of the digits 0, 1, ..., 9 while ignoring all other characters.
- b) Count the number of words, assuming that a word contains no blanks and is followed by a space.
- c) Replaces each letter by the sixth letter alphabetically following it ( $c$  by  $i$ ,  $w$  by  $b$ , etc.) as you might in simple encoding; leave blanks alone but also replace each digit by the sixth following digit (2 by 8, 7 by 3, etc.).

27. (3) You are hired to raise a million dollars for a political candidate. This candidate runs a grass-roots campaign, appealing to people who can't be expected to contribute more than \$10 each. You develop the following recursive strategy. You corral 10 friends and delegate to each of them the job of raising \$100,000. They in turn each corral 10 of their friends and give them the task of collecting \$10,000. And so on, until 100,000 people are contacted, each of whom can give \$10. Write this up as a recursive algorithm.

## 1.5 The Analysis of Algorithms

Designing algorithms for solving a variety of problems is one of the central themes of this book. It isn't enough, however, merely to develop a correct algorithm. We must also reckon the *cost* associated with actually using the algorithm. Even though computers are becoming faster and cheaper, if an algorithm is to be used again and again on a computer, the accumulated cost may still be considerable. Therefore, we must be able to *analyze* the performance of an algorithm.

There are two general classes of questions that may be asked about algorithms:

i) *Analysis of Algorithms Questions.*

We are always interested in how well algorithms perform. Usually our measure of performance will be how fast the algorithm works. This means that we are interested in the number of steps required to execute the algorithm to completion. Since the number of steps almost always depends on the *amount* or *size* of the data, we shall be trying to find the speed of an algorithm as a function of some quantity that represents the amount or size of the data. And, given a value of this quantity, our interest will usually be not in a particular set of the input data, but in *any* data that might be presented to the algorithm. We shall be interested in how fast the number of steps increases as the quantity of data increases. The ideas introduced in Section 0.3 on the rates of growth of functions will be important here. For example, we will wish to say that a particular algorithm runs in  $\text{Ord}(n^2)$  time, where  $n$  is our measure of the size of the data. Occasionally, instead of determining how fast an algorithm executes, we'll be interested in how much (computer) storage it requires.

The branch of mathematics and computer science concerned with the performance characteristics of algorithms is called the **analysis of algorithms**.

ii) *Computational Complexity Questions.*

Instead of asking how well a particular algorithm does its job, suppose we ask instead how good the *best* algorithm for a particular problem is, given a suitable definition of “best”. Your intuition probably will tell you that this is usually a much more difficult question than asking about the performance characteristics of a particular algorithm. After all, if you don't even know what all the possible algorithms for a problem are (and you almost never will), how could you determine how good the best one would be? And even if you could determine that, would you necessarily be able to find an algorithm which realizes the best performance? As with the analysis of specific algorithms, the most common definition of “best” is fastest, but sometimes the least-storage criterion is used here, too.

Discovering the properties of best algorithms and finding algorithms which have these properties is the concern of a branch of mathematics and computer science called **computational complexity**. When the concern is with fast algorithms, we speak of **time complexity**. If minimum storage is the object, we refer to **space complexity**. The study of computational complexity generally is a very difficult subject and requires mathematics beyond the scope of this book, although once in this section we shall determine the computational complexity of a problem.

When analyzing the performance of an individual algorithm, some authors use the word complexity, as in “the time complexity of this algorithm is  $\text{Ord}(n^2)$ ” where  $n$  is some measure of size. However, this is quite a different meaning of complexity than in the theory of computational complexity, where complexity refers to the intrinsic difficulty of a problem. Therefore, we prefer to use phrases like “this algorithm runs

in  $\text{Ord}(n^2)$  time”, or “it takes  $\text{Ord}(n^2)$  steps”, or “at worst this algorithm makes  $3n^2 + n$  assignments” (if, as sometimes happens, we decide to count just one type of operation).

Unless we explicitly say otherwise, we shall be interested only in speed considerations when analyzing algorithms. In this context we use three different kinds of analysis:

- *Best case analysis.* How fast is the algorithm under the most favorable circumstances (i.e., for all data with the same measure of size, which data allows the algorithm to run to completion with the least total computation)?
- *Worst case analysis.* How slow is the algorithm under the least favorable circumstances?
- *Average case analysis.* What is the algorithm’s average performance over all possible sets of data with the same measure of size, usually under the assumption that all these data sets are equally probable?

Your intuition will probably tell you — correctly — that average case analysis is usually much more difficult than best case or worst case analysis. The reason, obviously, is the necessity of somehow considering exhaustively all possible sets of data.

To illustrate these concepts, we shall begin with a quite simple example which, if nothing else, may help to convince you that the analysis of algorithms is a nontrivial subject.

## EXAMPLE 1

### Maximum of a Set of Numbers

We analyze the performance of Algorithm 1.12, MAXNUMBER, which finds the maximum of a set of numbers

$$x_1, x_2, \dots, x_n.$$

The quantity on which our analysis will focus is  $n$ , the size of the set of data. Our interest then is on how particular sets of data influence the running time.

#### Algorithm 1.12

#### MaxNumber

<b>Input</b> $n$	$[n > 1; \text{number of items}]$
$x_i, i = 1, \dots, n$	$[\text{Items in set}]$
<b>Output</b> $m$	$[\text{Maximum value}]$
<b>Algorithm</b> MAXNUMBER	
$m \leftarrow x_1$	$[\text{Initialize maximum value } m]$
<b>for</b> $k = 2$ <b>to</b> $n$	
<b>if</b> $x_k > m$ <b>then</b> $m \leftarrow x_k$	
<b>endfor</b>	

For this first example, we separate the analysis into fixed and variable costs. Some computations happen for every data set ( $m$  is initialized, and each time

through the loop the counter is increased and  $x_k$  is compared to  $m$ ). These are fixed costs. But how many times  $x_k$  is assigned to  $m$  depends on the data.

The cost to run through the loop each time, excepting the possible execution of  $m \leftarrow x_k$ , is some fixed amount  $c$ . Since the loop is iterated  $n-1$  times, the fixed cost for the algorithm body is  $c(n-1) + d$ , where  $d$  is the time for the initialization. By the analysis in Section 0.3, this cost is  $\text{Ord}(n)$  (the  $d$  and the 1 don't matter because  $\text{Ord}(1) < \text{Ord}(n)$ ). Thinking in terms of Orders allows us not to concern ourselves with unknown constants and lower-order terms.

Since the cost of the replacement  $m \leftarrow x_k$  cannot affect the order of the cost of the algorithm itself (why?), the algorithm is  $\text{Ord}(n)$  no matter how many replacements there are. Still, it is of interest to try to count the number of replacements.

**Best case:** 0 replacements, when  $x_1 \geq x_i$  for all  $i$ .

**Worst case:**  $n-1$  replacements, when  $x_1 < x_2 < x_3 < \dots < x_n$ .

**Average case:** We assume that all the  $x_i$ 's are distinct, because equal values considerably complicate the analysis. Then without loss of generality, we may assume that the  $n$  values of the  $x_i$ 's are  $1, 2, 3, \dots, n$ . And we naturally assume that all possible permutations (i.e., orderings; see Section 4.4) of the integers  $1, 2, 3, \dots, n$  are equally probable sequences  $x_1, x_2, \dots, x_n$ . What now? Somehow we must consider all  $n!$  permutations of  $1, 2, \dots, n$  and count the number of replacements for each. Brute force — that is, considering each permutation separately — is out of the question, except for very small values of  $n$ .

But we can make some progress by reasoning not about each permutation but rather about the *position* in a permutation of the item that replaces  $m$ . The algorithm starts by making the first item the current largest one. What, then, is the probability that the second item will replace the first one as largest when the second is compared to the first? The answer is  $\frac{1}{2}$ , because it is 50–50 whether the first item or the second is larger. Thus on the average, there is  $\frac{1}{2}$  of a replacement when  $k = 2$ . Now how about the third item? The probability that the third item is larger than either of the two that precede it is  $\frac{1}{3}$ , because 2 of the 6 possible orderings of 3 numbers have the largest number last. Therefore on average there is  $\frac{1}{3}$  of a replacement when  $k = 3$ . Working our way toward the last element, we find that — all told — the average number of replacements is

$$\sum_{k=2}^n \left( \frac{1}{k} \right) \tag{1}$$

It happens (but we won't show it here) that  $\sum_{k=2}^n (1/k)$  is  $\text{Ord}(\log n)$ ; in fact, it is asymptotic to  $\ln n = \log_e n$  (see [26], p. 40, for the definition of asymptotic). So whereas the algorithm itself is  $\text{Ord}(n)$ , on the average the number of times the replacement is made is  $\text{Ord}(\log n)$ .

Note that, even for this very simple algorithm, average case analysis was not easy. It led to (1), which is not in closed form and whose derivation was not very satisfactory.

Why, for example, is it OK to sum these average numbers of replacements by position as in (1)? You'll have to wait for Section 6.8 for a good answer.

Since from Section 0.3 we know that  $\text{Ord}(\log n) < \text{Ord}(n)$ , even in the worst case, the variable cost does not change the Order. However, if instead of a simple replacement  $m \leftarrow x_k$ , there was a substantial computation before  $m$  was given a new value, the variable cost might not be ignorable. ■

## EXAMPLE 2

### Analysis of Towers of Hanoi

Consider again Algorithm HANOI from Section 1.3. Whereas algorithm MAX-NUMBER required us to consider best, worst and average cases for the many possible data sets, with TOH, once you know  $n$ , the number of disks in procedure  $H(n, r, s)$ , then you know all there is to know that affects the running time. (The values  $r$  and  $s$  of the initial and final poles have no effect whatever on the number of disk moves that need to be made.) Thus, there are no best, worst and average cases. There is only one appropriate analysis question:

For a given  $n$ , how many moves are required to move the disks from pole  $r$  to pole  $s$ ?

Let's denote the number of moves we are trying to calculate by  $h_n$ . When  $n = 1$ , it is clear that  $h_1 = 1$ , since the solution consists of the single move robot( $r \rightarrow s$ ). When  $n > 1$ ,

$$h_n = 2h_{n-1} + 1, \tag{2}$$

because the calculation of  $H(n, r, s)$  consists of two calls to the algorithm with first argument  $n - 1$  (this gives the  $2h_{n-1}$  term) plus one move between these two calls. In Chapter 5 we shall discuss general methods for solving *difference equations*, of which Eq. (2) is an example. But here we can resort to the discovery (i.e., the “compute”) portion of our inductive paradigm.

Suppose we use Eq. (2) to compute some values of  $h_n$ , using the known value  $h_1 = 1$  to get started. Then using  $n = 2$  in Eq. (2), we calculate  $h_2$ . We use this result with  $n = 3$  to calculate  $h_3$  and so on. We obtain

$n$	$h_n$
1	1
2	3
3	7
4	15
5	31
6	63
7	127

What formula can you conjecture for  $h_n$  as a function of  $n$  from this brief set of values? How about

$$h_n = 2^n - 1? \tag{3}$$

Is this correct? For  $n = 1$  it gives 1, which we know is the correct answer. For  $n > 1$ , if Eq. (3) is correct, then the difference equation Eq. (2) must be satisfied by

the  $h_n$  in (3). So we substitute this  $h_n$  into the left-hand side of Eq. (2) and, with  $n$  replaced by  $n - 1$  in Eq. (3), we substitute into the right-hand side of Eq. (2). Doing this, the right-hand side of Eq. (2) becomes

$$2(2^{n-1} - 1) + 1 = 2^n - 2 + 1 = 2^n - 1.$$

This equals the left-hand side with  $h_n$  replaced by Eq. (3). Thus, for any  $n$ , Eq. (3) satisfies Eq. (2) and, therefore, Eq. (3) gives the correct number of moves for any  $n$ . This result should convince you that, for all but quite small  $n$ , moving the disks from one pole to another takes a prodigious amount of work. (Do you see why? See [11].)

This demonstration that our conjecture is correct is an informal example of a proof by mathematical induction (the subject of Chapter 2). Recall, by the way, that in Example 3, Section 1.3, there were three rings requiring seven moves, as predicted by Eq. (3). Note also that, since  $2^n - 1 = \text{Ord}(2^n)$  (do you see why? [12]), TOH is  $\text{Ord}(2^n)$  with  $2^n$  one of our standard sequences in Section 0.3.

We will now determine the complexity of the TOH problem itself. What we shall show is that no algorithm for TOH could require fewer moves than Algorithm HANOI. To do this, let  $h_n^*$  be the minimum number of moves for the best possible algorithm. Consider now the relation of  $h_{n+1}^*$  to  $h_n^*$ . For the  $n + 1$  disk case, the bottom disk must move at least once. Also any time the bottom disk does move, the other  $n$  disks must all be in order on one other pole. For them to have gotten to this other pole the first time must have taken at least  $h_n^*$  moves. Similarly, after the bottom disk finally moves onto the final pole, the other  $n - 1$  disks must all be on a different pole (why?) and so there must be at least  $h_n^*$  more moves to put the other  $n$  disks on top of the bottom disk. Thus

$$h_{n+1}^* \geq 2h_n^* + 1, \tag{4}$$

where the 1 is there because the bottom disk must move at least once. Since we know that  $h_1^* = 1$  (why?), it follows from Eq. (2) and Eq. (4) that  $h_2^* \geq h_2$ , then that  $h_3^* \geq h_3$ , etc., and thus that  $h_n^* \geq h_n$  for all  $n$ . But since  $h_n^*$  is the number of moves for the best algorithm,  $h_n^* \leq h_n$ . One number can be both  $\geq$  and  $\leq$  another only if they are equal. Thus  $h_n^* = h_n$ , so that our algorithm is optimal. Moreover, our algorithm prescribes the *unique* set of best moves, given the start pole and the finish pole. Do you see why [13]? ■

## EXAMPLE 3

### Analysis of EUCLID (reproduced in Fig. 1.8)

Since algorithm analysis is always in terms of the size of the data, the first issue here is how to measure size, since there are two input variables,  $m, n$ . Fortunately, just as it turned out in Section 1.1 that we could argue the correctness of EUCLID in terms of  $n$  alone, so can we analyze the run time in terms of  $n$  alone, at least to the extent of getting a bound on the worst case.

The running time of EUCLID is determined by the number of times the statements between `repeat` and `endrepeat` are executed. This is equal to the number of times the step

$$quot \leftarrow \lfloor num/denom \rfloor$$

is executed before  $denom$  becomes 0. So what we wish to count is the number of divisions performed in computing the  $\gcd(m, n)$ . The running time, except for the initialization step, will then be some constant multiple of this number.

The best case is trivial. If  $n = 0$ , then  $denom$  is set immediately to zero and the loop is never executed at all.

<b>Input</b>	$m, n$	[Integers $\geq 0$ ]
<b>Output</b>	$num$	[ $= \gcd(m/n)$ ]
<b>Algorithm</b> EUCLID		
$num \leftarrow m; denom \leftarrow n$		[Initialize $num$ and $denom$ ]
<b>repeat until</b> $denom = 0$		
$quot \leftarrow \lfloor num/denom \rfloor$		[Quotient]
$rem \leftarrow num - quot * denom$		[Remainder]
$num \leftarrow denom; denom \leftarrow rem$		[New $num$ and $denom$ ]
<b>endrepeat</b>		

**FIGURE 1.8**

Algorithm EUCLID.

For the worst case we reason as follows:

1. At the  $j$ th stage the divisor is  $r_{j-1}$  — see Eq. (6), Section 1.1. There are two possibilities for the remainder  $r_j$ . Either  $r_j \leq r_{j-1}/2$ , in which case the remainder will have been reduced by at least a factor of  $1/2$ , or  $r_j > r_{j-1}/2$ .
2. If  $r_j > r_{j-1}/2$ , at the next stage the quotient will be 1 (why?), and the remainder  $r_{j+1}$  will be the difference between  $r_{j-1}$  and  $r_j$ , which is  $\leq r_{j-1}/2$  (why?).
3. Thus after each two steps, the remainder will be reduced by at least a factor of  $\frac{1}{2}$ . That is, either  $r_j \leq r_{j-1}/2$  or  $r_{j+1} \leq r_{j-1}/2$ . For example, after the first two steps (with  $n$  playing the role of  $r_0$ )  $r_2 \leq n/2$ .

What, then, is an upper bound on the number of divisions to reduce the remainder to 0? Suppose  $n$  is a power of 2. If the remainder were exactly halved every two steps, it would take  $2 \log_2 n$  steps to obtain a remainder of 1 and one more step to reduce the remainder to zero. Thus, the worst case number of divisions is not more than [15]

$$(2 \log_2 n) + 1.$$

It turns out this is an upper bound even when  $n$  is not a power of 2 [15]. Note that this upper bound is quite conservative. That is, it is usually far larger than the actual number of divisions, since the remainder will often be halved — or more than halved — in one step rather than two. To use the Big Oh notation of Section 0.3, we have only shown that the worst case of EUCLID is  $O(\log n)$ , not  $\text{Ord}(\log n)$  (why? [1b]). ■

An important function of the analysis of algorithms is to allow us to compare the efficiency of two different algorithms intended for the same task. As our final examples in this section, we will illustrate this with two algorithms, both of which have the purpose of finding an entry in an ordered list of, say, words.

## EXAMPLE 4

### Sequential Search

Suppose that we have the following alphabetized list of words, where  $w_i$  is the  $i$ th word in order on the list.

$i$	$w_i$
1	Are
2	Body
3	Career
4	Computer
5	DAM [i.e., Discrete Algorithmic Mathematics!]
6	Mathematics
7	Science
8	Ugh
9	Why

Suppose further that we are given a new word,  $w$ , and that our problem is to determine whether the new word is on the list and, if it is, where. We denote by  $w = w_i$ ,  $w > w_i$ , and  $w < w_i$ , respectively, the situations where  $w$  is identical to  $w_i$ ;  $w$  follows  $w_i$  alphabetically; and  $w$  precedes  $w_i$  alphabetically. (If  $w$  is not on the list, we might wish to insert it; see [18].)

Algorithm 1.13, SEQSEARCH provides a solution for this problem for a list of  $n$  words.

This algorithm contains an AL construct we haven't used before, namely a selection statement in which the **if** is followed by a sequence of more than one clause of the form  $B_i$  **then**  $S_i$ . The interpretation of this multiline construct (sometimes called a *case-if*) is that the  $B_i$ 's are evaluated one after another until one of them is true, in which case the corresponding  $S_i$  is executed and the entire **if-endif** statement is terminated. If none of the  $B_i$  are true, nothing at all is done.

*Caution 1.* This is not quite the same as several one-line **if-then** statements in a row [17].

*Caution 2.* You may wonder why we put **exit** at the end of the first two internal lines of the case-if structure in SEQSEARCH when these lines cause termination anyway if true. The answer lies in the precise definition of **exit**. As stated in Section 1.2, **exit** takes you out of your innermost *repeat* structure. Therefore, in SEQSEARCH, these **exits** take you just after the **endrepeat** whereas without the **exits** you would only jump to just below the **endif**, which is not where you wish to go — why?

To analyze the speed of this algorithm as a function of  $n$ , we focus on the step in it which most nearly determines how much computation must be performed. In this algorithm the number of times the loop is executed is the same as the number

**Algorithm 1.13****SeqSearch**

<b>Input</b> $n$	$[n > 0; \text{number of items in list}]$
$w_i, i = 1, 2, \dots, n$	$[\text{Alphabetized list of items}]$
$w$	$[\text{Search word}]$
<b>Output</b> $i$	$[\text{Index of word if in list}]$
or	
‘failure’	$[\text{If word not in list}]$

**Algorithm SEQSEARCH**

```

 $i \leftarrow 1$  [Initialize index of position in list]
repeat
    if  $w = w_i$  then print  $i$ ; exit [Success]
     $w < w_i$  then print ‘failure’; exit
     $w > w_i$  then  $i \leftarrow i + 1$  [Try next word]
    endif
    if  $i > n$  then print ‘failure’; exit
endrepeat

```

of times we must *compare*  $w$  and  $w_i$ . We will count this 3-way comparison ( $=, >, <$ ) as one comparison. Thus we reason as follows:

**Best case:** 1 comparison;  $w \leq w_1$ . So the best case is  $\text{Ord}(1)$ .

**Worst case:**  $n$  comparisons;  $w > w_{n-1}$ . So the worst case is  $\text{Ord}(n)$ .

**Average case:** For simplicity we focus on the case where  $w$  is on the list.

Suppose that  $w$  is equally likely to be equal to any  $w_i$  (a crucial assumption — why?) [20]. Then it is equally likely that there will be  $1, 2, 3, \dots, n$  comparisons. Thus the *average* number of comparisons is

$$\left( \frac{1}{n} \right) (1 + 2 + 3 + \cdots + n). \quad (5)$$

To find a closed form expression for this, let’s consider just the sum of the first  $n$  integers in (5). Writing this sum twice as

$$1 + 2 + 3 + \cdots + n - 1 + n$$

and

$$n + n - 1 + n - 2 + \cdots + 2 + 1,$$

we see that each of the  $n$  columns sums to  $n + 1$ . Therefore

$$2(1 + 2 + \cdots + n) = n(n + 1),$$

so that

$$1 + 2 + \cdots + n = \frac{n(n + 1)}{2}.$$

Substituting this expression into (5), we obtain for the average number of comparisons

$$\left(\frac{1}{n}\right) \left[\frac{n(n+1)}{2}\right] = \frac{(n+1)}{2}.$$

So the average case is also  $\text{Ord}(n)$  since  $n/2$  is  $\text{Ord}(n)$  and  $\text{Ord}(1) < \text{Ord}(n)$ . Are you surprised that the worst and average cases have the same order even though the number of comparisons in the average case ( $(n+1)/2$ ) is less than the number in the worst case ( $n$ )? There is no contradiction here. Two unequal sequences may have the same order so long as their growth rates are proportional to each other.

Why might you have guessed that the average case is  $(n+1)/2$  without any algebra at all [21]?

Note, by the way, that the method we used to derive the sum of the first  $n$  integers is a trick of limited applicability. You should be skeptical of such tricks, which have little use for solving other problems. We'll derive this formula several times later in this book by methods which have much wider applicability.

This completes our analysis of sequential search. The algorithm is very simple and the analysis was fairly easy, too, with the average case analysis being the most difficult. As with many analyses of algorithms, it focused on counting the number of times a loop is executed. We didn't concern ourselves with the amount of work done during each loop execution because, when each iteration does essentially a fixed amount of work, that amount does not affect the order of the execution sequence but only the constants in Eqs. (2–3) in Section 0.3. However, the value of this constant is crucial when we compare the efficiency of one algorithm with another of the same order. ■

The sequential search algorithm does its job; willy nilly it finds  $w$  if  $w$  is on the list. But we hope you aren't very happy with this algorithm. If the list had 100,000 words in it, finding  $w$  would be rather tedious (even for a computer) if  $w$  were near the end of the list. And, for example, if the “list” were a dictionary, you wouldn't want to look a word up by starting at the beginning and examining one word at a time until you found the word or determined that it wasn't in the dictionary. In Example 5 we consider a better procedure, one that is somewhat closer to what we actually do when using a dictionary.

## EXAMPLE 5

### Binary Search

Remember the game where someone picks a number in the range 1 to 100 and you have to guess it in the fewest number of tries? Every time you guess, you are told whether you are correct, high, or low. A good strategy is to guess about in the middle of the range (say, 50) and, if this isn't correct, guess about 25 if you were high and about 75 if you were low. You then continue to narrow the range in which the number must lie by always guessing in the middle of the range where you know the number must be. **Binary search** implements this idea for searching an ordered list.

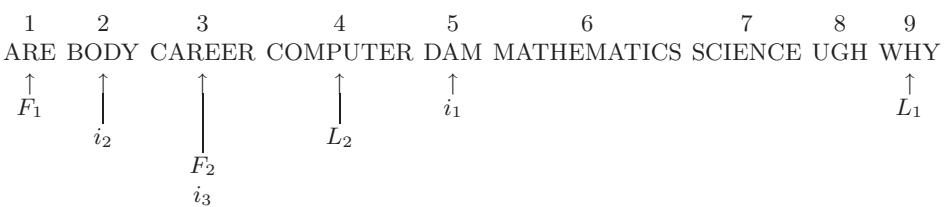
As in Example 4, suppose we are given an alphabetized list and a search word  $w$ . We will apply the recursive approach outlined in the preceding paragraph by first looking in (approximately) the middle of the list. If we don't find the right word (as we normally won't for a list of any significant length), we then look in that half (approximately) of the list where  $w$  must be, if it is there at all, based on whether  $w$  is alphabetically before or after the word with which it was compared. Thus by jumping into the middle of the list, we either succeed at once or we immediately reduce our problem to half its original size. Then if we don't succeed, we repeat this process by comparing  $w$  with a word in about the middle of the half of the original list where it must be if it is there at all. And so on until we find  $w$  or determine that it isn't in the list. The name of this algorithm comes from the fact that the list is divided into *two* parts at each step. Although we could present this naturally recursive idea as a recursive algorithm, its iterative form, as given by Algorithm 1.14 BINSEARCH, is so simple that it is probably the easier one to follow. You are asked to write this algorithm recursively in [22]. An example of how this algorithm works is shown in Fig. 1.9.

### Algorithm 1.14

#### BinSearch

<b>Input</b> $n$	$[n > 0; \text{number of items in list}]$
$w_i, i = 1, \dots, n$	$[\text{Alphabetized list of items}]$
$w$	$[\text{Search word}]$
<b>Output</b> $i$	$[\text{Index of word if in list}]$
or	
'failure'	$[\text{If word not in list}]$
<b>Algorithm</b> BINSEARCH	
$F \leftarrow 1; L \leftarrow n$	$[\text{Initialize pointers to first } (F) \text{ and last } (L) \text{ items on list}]$
<b>repeat</b>	
$i \leftarrow \lfloor (F + L)/2 \rfloor$	$[\text{Index of approximate midpoint}]$
<b>if</b> $w = w_i$ <b>then print</b> $i$ ; <b>exit</b>	$[\text{Success}]$
$w < w_i$ <b>then</b> $L \leftarrow i - 1$	$[\text{Leave if-endif; search first half}]$
$w > w_i$ <b>then</b> $F \leftarrow i + 1$	$[\text{Leave if-endif; search second half}]$
<b>endif</b>	
<b>if</b> $L - F + 1 = 0$ <b>then print</b> 'failure'; <b>exit</b>	
<b>endrepeat</b>	

Before going on to the analysis of the algorithm, you should understand that binary search works by 1) each time through the loop setting  $i$  to the approximate midpoint of the sublist defined by the two (moving) pointers  $F$  and  $L$ ; and 2) if  $w_i \neq w$ , then changing either  $F$  or  $L$  to define a new sublist. You may have trouble understanding why  $L - F + 1 = 0$  is the correct test for terminating execution of the algorithm with an indication of failure. The short answer is:  $L - F + 1$  is the



An Example of Binary Search. Let the search word be CAREER. Subscripts distinguish the successive values of  $F$ ,  $L$ , and  $i$  in Algorithm BINSEARCH.

length of a list that starts at position  $F$  and ends at position  $L$  (why?), and you should quit when the list is empty. We explain in more detail in Section 2.5.

Now here is our analysis.

**Best case:** 1 comparison;  $w = w_j$ , where  $j = \lfloor (n+1)/2 \rfloor$ , so the best case is  $\text{Ord}(1)$ .

**Worst case:** This is not so easy. Since each pass through the loop reduces the size of the list to be searched by approximately half and since the last list to be searched in case of failure has length 1 (make sure you understand why), we expect that the worst case should require a number of comparisons related to how many halvings of  $n$  are required to get to 1. For  $n = 16$  this is four  $[16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1]$ . (We go from 16 to 8 because, if a list has 16 items, after the first test the *longer* sublist has 8 items.) For  $n = 21$  it is also four  $[21 \rightarrow 10 \rightarrow 5 \rightarrow 2 \rightarrow 1]$ . (We go from 21 to 10 because, if a list has 21 items, after the first test each sublist will have 10 items.) From the discussion of logarithms in Section 0.2, you'll realize that this number of halvings is, in general, related to the logarithm of  $n$  and, specifically, should be something quite close to  $\log_2 n$ . This is so because, as we argued when discussing Algorithm EUCLID, if  $n = 2^m$ ,  $\log_2 n = m$  is the number of times  $2^m$  must be successively halved until its value reaches 1. If  $2^m < n < 2^{m+1}$ , then  $\log_2 n$  approximates the number of times  $n$  must be halved to get a value nearly equal to 1. In [25a] we ask you to conjecture an exact formula for the worst case for all  $n$ , and in [22, Section 2.5] you prove it.

**Average case:** This is quite difficult, and we will not treat it, but see [25b]. ■

## Problems: Section 1.5

1. (3) At the end of Section 0.3 we chided some other writers for using Big Oh notation instead of  $\text{Ord}$  to describe the order of algorithms. But there are two circumstances in which this usage can be justified.

- a) Suppose that the best case complexity of an algorithm is lower order than its worst case. For

instance, the best case of SEQSEARCH is  $O(1)$  while the worst case is  $O(n)$ . Explain why it is therefore appropriate to say that SEQSEARCH is  $O(n)$ , and not appropriate to say that it is  $\text{Ord}(n)$ . (To do this, you have to generalize the meaning of order notation a bit. As defined in

Section 0.3, notation like Ord applies to single sequences, e.g., the sequence of worst case run times for a specific algorithm as a function of an input  $n$ . But to talk about the order of an algorithm, we have to talk about Ord applied simultaneously to the set of all possible sequences of run times for the algorithm. The key step to answering this part of the problem is to give an appropriate definition.)

- b) Continuing with SEQSEARCH, we would still frown on saying the the worst case of SEQ-SEARCH is  $O(n)$ , because we know that actually it's Ord( $n$ ). But if we don't know more, a Big Oh statement is appropriate even when we limit ourselves to, say, worst case. For instance, for EUCLID we claimed (and you will show in [15]) that  $2\log_2 n + 1$  is an upper bound on the number of divisions required to compute the gcd. Why does this justify the claim that the worst case of EUCLID is  $O(\log n)$  but not the claim that it is Ord( $\log n$ )?

2. (2) Here is an algorithm for evaluating a polynomial:

### Algorithm

**Input**  $n, a_0, a_1, \dots, a_n, x$  [Degree, coefficients, and variable value]

**Output**  $V$

**Algorithm** POLY-EVAL

```

 $V \leftarrow 0$  [Value of polynomial]
for  $k = 0$  to  $n$ 
     $T \leftarrow a_k$  [Current term]
    for  $j = 1$  to  $k$ 
         $T \leftarrow T * x$ 
    endfor
     $V \leftarrow V + T$ 
endfor

```

- a) How many additions are needed to run this algorithm? How many multiplications?
- b) This algorithm is very inefficient because it computes each power of  $x$  from scratch. Write an improved algorithm, POLY-EVAL2, which saves the previous power and uses it to compute the next higher power. How many multiplications does POLY-EVAL2 take?
3. (2) You are given 10 gold coins and are told that all but one bad one weigh the same. You are also

given a known good gold coin, and a pan balance. Your problem is to identify the bad coin. There is a simple-minded algorithm for this: Weigh in turn each of the 10 against the test coin until the pans don't balance.

- a) Write this in algorithmic language.
- b) What is the best case outcome? Assume that the only step that takes time is doing a balance.
- c) What is the worst case outcome?
- d) What is the average case?

*Note:* There is a much smarter algorithm for this problem. Its best case is a bit worse, but its worst and average cases are much better (see Section 4.3).

4. (2) Again you have 10 gold coins and a pan balance and are told that exactly one coin has a different weight from the others. But this time you don't have a test coin. Again, there is a naive algorithm for solving the problem: Weigh two coins; if they balance they are good and you are back to the previous case; if they don't balance, weigh either one of them against another coin. Repeat a)-d) in [3] for this situation.
5. (3) Algorithm FINDPOWER2 finds the highest power of 2 which divides a positive integer  $N$ :

### Algorithm

**Input**  $N$

**Output**  $k$

**Algorithm** FINDPOWER2

```

 $k \leftarrow 0; n \leftarrow N$ 
repeat
     $q \leftarrow n/2$ 
    if  $q \neq |q|$  then exit
     $k \leftarrow k + 1$ 
     $n \leftarrow q$ 
endrepeat

```

Assume that the inputs are taken at random from 1 to  $1024 = 2^{10}$ . Also, assume that the only time-consuming step is division.

- a) What is the best case?
- b) What is the worst case?
- c) What is the average case? Hint: Determine the fraction of the number of inputs which lead to each possible number of divisions.

6. ⟨3⟩ Algorithm FINDPOWER2' is a modification of the algorithm in [5]. In some ways, FINDPOWER2' is preferable to FINDPOWER2. It has fewer lines, and it puts the exit condition at the start of the loop, where you can find and appreciate it quickly. But how does it compare in efficiency to FINDPOWER2? Answer this question by considering which operations are the most important in the two algorithms. Explain. Hint: What has to happen in order to check whether  $2 \nmid n$ ?

### Algorithm

Input  $N$

Output  $k$

#### Algorithm FINDPOWER2'

$k \leftarrow 0; n \leftarrow N$

repeat until  $2 \nmid n$

$k \leftarrow k + 1$

$n \leftarrow n/2$

endrepeat

[Where  $\nmid$  means  
“does not divide”]

7. ⟨3⟩ Write algorithms to do the following:

- a) Compute the inner product (see Section 0.5) of two vectors of length  $n$ .
- b) Compute the product of an  $m \times n$  matrix and an  $n \times 1$  column vector.
- c) Compute the product of an  $m \times n$  matrix and an  $n \times p$  matrix.
- d) Assuming that each real-number multiplication takes one step, what are the best, worst, and average case complexities of your algorithms in a)-c)?
- e) Suppose  $A$  and  $B$  are  $n \times n$  matrices and  $X$  is an  $n \times 1$  column vector. The product  $ABX$  can be computed either in the order  $(AB)X$  or  $A(BX)$ . Compare the complexities of these two alternatives.

8. ⟨2⟩

- a) Modify Algorithm MAXNUMBER to start the search from the end of the list.
- b) Suppose the numbers need not be distinct. Modify Algorithm MAXNUMBER with the search starting at the beginning of the list, so that the output is the *smallest* subscript  $j$  such that  $x_j$  equals the maximum value.

- c) Modify the algorithm again to output the *largest* subscript  $j$  such that  $x_j$  equals the maximum value.

9. ⟨3⟩ State a loop invariant for MAXNUMBER and prove that it really is a loop invariant.

10. ⟨2⟩ Compute the average number of replacements for Algorithm MAXNUMBER for  $n = 2, 3$ , and 4.

11. ⟨2⟩ There is a fable about the Indian rajah who required his subjects each year to pay tribute to him by putting one grain of gold on the first square of a checkerboard, 2 grains on the next square, 4 grains on the third square and, in general,  $2^i$  on the  $i$ th square. Why was this an impossible request and how does this fable relate to TOH?

12. ⟨2⟩ Show that  $2^n - 1 = \text{Ord}(2^n)$  using the definitions in Section 0.3.

13. ⟨3⟩ Argue informally that Algorithm HANOI not only results in the minimum number of moves to solve the TOH puzzle but that the moves generated by HANOI are the only possible set of moves to realize this minimum number. (This will be proved carefully by induction in Theorem 3, Section 2.4.)

14. ⟨3⟩ For the Euclidean algorithm, tabulate the number of divisions required for all  $n \leq 10$ . Do you see that this does not require you to consider all natural numbers for  $m$ ? For each  $n$ , what values of  $m$  in fact suffice to get all possible cases of division sequences? What pattern do you see, if any, in the results?

15. ⟨3⟩

- a) Justify our claim that, if the remainder were exactly halved every two steps of EUCLID, it would take exactly  $2 \log_2 n$  divisions by 2 to reduce  $2^n$  to 1.

- b) Show therefore that an upper bound on the number of divisions in the Euclidean algorithm is  $(2 \log_2 n) + 1$ . Compare this bound to the results of [14]. Can you improve on this bound by using the floor function?

16. ⟨3⟩ Argue that each pass through the repeat-loop of EUCLID1 in [12, Section 1.1] reduces *rem* by at least a factor of 2, so that the loop is iterated at most  $1 + \log_2 n$  times. This is half the bound we obtained for EUCLID. Does this mean EUCLID1 is a faster algorithm? What more do you feel you need to know to decide?

17. (2) What is the final value of  $k$  in each of the following two algorithm fragments?

```
n ← 3  
if  $n > 2$  then  $k \leftarrow 3$  endif  
if  $n > 1$  then  $k \leftarrow k^2$  endif
```

```
n ← 3  
if  $n > 2$  then  $k \leftarrow 3$   
   $n > 1$  then  $k \leftarrow k^2$   
endif
```

18. (3) Modify Algorithm SEQSEARCH so that if  $w$  is not in the list, it is inserted into the list in correct alphabetical order. Why is this process quite inefficient? Can you think of any way to organize the list of words so that when a new word is inserted, you don't have to "move" all the words which follow it?

19. (3)

- a) Modify Algorithm SEQSEARCH again, so that it works even if the list is *not* in alphabetical order.
- b) Give the best and worst case analyses of the revised algorithm.
- c) Give the average case analysis, assuming that the search word  $w$  is *not* in the list.
- d) Give the average case analysis, assuming that the search word has probability  $p$  of being in the list and is equally likely to be any one of the words in the list.

20. (3) Suppose the probability that  $w = w_i$  in sequential search is  $p_i$ , and the probability that  $w$  is not equal to any  $p_i$  is  $p$  (with  $\sum p_i + p = 1$ ). How must the analysis of this algorithm be modified? (Assume the list is alphabetized.)

21. (2) Our analysis that the average case of SEQSEARCH requires  $(n + 1)/2$  comparisons was not too difficult but do you see why you should have expected this result without doing any formal analysis?

22. (2) Rewrite Algorithm BINSEARCH as a recursive algorithm.

23. (2) In BINSEARCH show that, if  $w$  is not on the list, then eventually  $L - F + 1 = 0$ .

24. (2) Suppose the first step after **repeat** in Algorithm BINSEARCH is changed to

$$i \leftarrow \lceil (F + L)/2 \rceil.$$

- a) Is the algorithm still correct? If not, what must be done to correct it?

- b) How must the analysis of the algorithm be changed?

25. (3) For Algorithm BINSEARCH and  $n = 2, 3, 4$ , and 5, compute:

- a) The number of comparisons in the worst case. Can you make a conjecture from this data of the answer for a general  $n$ ? See [22, Section 2.5].
- b) The number of comparisons on average if the search is successful. Assume that all search arguments are equally likely.

## Supplementary Problems: Chapter 1

---

1. (2) Use algorithmic notation to write an algorithm which calculates the roots of the quadratic equation

$$ax^2 + bx + c = 0,$$

where  $a$ ,  $b$ , and  $c$  can be any real numbers. When you have done this, compare your solution with that in the Hints and Answers. Explain any differences between your solution and the recommended solution.

2. (3) Conversion of integers in base  $b$  to base 10. Let's analyze this problem using the recursive paradigm. Suppose you already know how to convert an integer with  $n$  digits in its base  $b$  representation and you now wish to convert an integer with  $n + 1$  digits. As usual, index the digits from left to right, as  $a_n, a_{n-1}, \dots, a_1, a_0$ .

- a) The easiest approach is to think of  $a_{n-1}, \dots, a_0$  as the sequence you already know how to convert. Write an iterative algorithm to implement this approach.

- b) Write a recursive algorithm to handle the approach in a).
- c) You could also think of  $a_n, a_{n-1}, \dots, a_1$  as the sequence we already know how to convert. To convert the entire sequence, you must “shift over” the known subsequence and then add in a new 1’s digit,  $a_0$ . Write an iterative algorithm to do this.
- d) Write a recursive algorithm to handle the approach in c).

3. (3) Conversion of integers in base 10 to base  $b$ . Let’s analyze this problem using the recursive paradigm. Suppose you are given the integer  $N$  in base 10. Suppose you already know how to convert all integers  $M < N$  to base  $b$ .

- a) You could try to determine the left-most digit of  $N$  in base  $b$  first. This depends on finding the highest power of  $b$  which is  $\leq N$ . Once you find it, you can subtract it from  $N$  the maximum number of times and then proceed to convert the difference to base  $b$ . Write an iterative algorithm based on this idea.
- b) Write a recursive algorithm to handle the approach in a).
- c) Or you could try to determine the right-most digit of  $N$  in  $b$  first. This is the remainder when  $N$  is divided by  $b$ . Why? Also, the quotient, if converted to base  $b$ , tells you the other digits of  $N$  in base  $b$ . Why? Turn these observations into an iterative algorithm to convert to base  $b$ .
- d) Write a recursive algorithm to handle the approach in c).

### Algorithm 1.15 EUCLID2

<b>Input</b> $m, n$	[Integers $\geq 0$ ]
<b>Output</b> $num$	[ $= \gcd(m, n)$ ]

#### Algorithm EUCLID2

```

 $num \leftarrow m; denom \leftarrow n$ 
repeat until  $denom = 0$ 
     $quot \leftarrow \lfloor .5 + (num/denom) \rfloor$ 
     $rem \leftarrow |num - (quot \times denom)|$ 
     $num \leftarrow denom; denom \leftarrow rem$ 
endrepeat

```

4. (3) In introducing Algorithm EUCLID in Section 1.1, we argued that  $m - kn$  and  $n$  have the same divisors as  $m$  and  $n$ , so long as  $m - kn \geq 0$ .

Actually, nowhere did we use that  $m - kn \geq 0$ . If we can make the absolute value of the remainder smaller by subtracting a little beyond zero, let’s do it. Call the revised algorithm EUCLID2. The key algebraic change is that  $\lfloor .5 + x \rfloor$  rounds  $x$  to the nearest integer, up or down. Thus if  $num/denom$  is, say, 5.8, we subtract  $denom$  6 times, thus going beyond 0.

- a) Trace EUCLID2 on the input pairs (45,26) and (21,13). Compare with the traces in [12, Section 1.1]? What do you notice?
- b) Argue that each pass through the repeat-loop of EUCLID2 reduces  $rem$  by at least a factor of 2, so that the loop is iterated at most  $1 + \log_2 n$  times. This is half the bound we obtained for EUCLID. Is EUCLID2 a faster algorithm than EUCLID?

### Algorithm

**Input**  $a_1, \dots, a_n$

**Output**  $b_i, c_i, i = 1, \dots$

[Distinct entries and counts]

#### Algorithm STRIPDUPS1

```

 $w \leftarrow 0$ 
repeat
     $k \leftarrow 1$ 
    repeat until  $a_k \neq null$  [Look for an uncounted entry]
         $k \leftarrow k + 1$ 
        if  $k = n+1$  then stop
    endrepeat
     $w \leftarrow w+1$ 
     $b_w \leftarrow a_k; c_w \leftarrow 1$ 
    for  $j = k+1$  to  $n$  [Look ahead for repeats]
        if  $a_j = b_w$  then  $c_w \leftarrow c_w + 1; a_j \leftarrow null$ 
    endfor
endrepeat

```

5. (3) Consider the following task: Given  $\{a_k\}$ , an array of length  $n$  with possible repetitions, do a tally of the *distinct* entries, that is, eliminate duplicates. Simple as this task sounds, there are different methods for it with different complexities. See Algorithms STRIPDUPS1 and STRIPDUPS2. In both methods, at the end  $b_w$  is the  $w$ th distinct entry and  $c_w$  is how many times that entry appears in  $\{a_k\}$ . In both methods, an entry can be reassigned the special value *null*, which means it has already been accounted for.

Algorithm STRIPDUPS1 does repeated sweeps of the array  $\{a_k\}$  until it is empty. Algorithm STRIPDUPS2 does a single sweep. Determine and compare the best and worst cases of these two algorithms.

## Algorithm

**Input and Output** same as in STRIPDUPS1

**Algorithm** STRIPDUPS2

```

for  $w = 1$  to  $n$ 
     $b_w \leftarrow null$ 
     $c_w \leftarrow 0$ 
endfor

for  $k = 1$  to  $n$ 
     $w \leftarrow 1$ 
    repeat
        if  $b_w = null$  then  $b_w \leftarrow a_k$ 
        if  $b_w = a_k$  then  $c_w \leftarrow c_w + 1$ ; exit
         $w \leftarrow w + 1$ 
    endrepeat
endfor
```

6. (3) The Knapsack Problem. You have  $n$  knapsacks with capacities  $c_1, c_2, \dots, c_n$ . You have  $m$  items of sizes  $s_1, s_2, \dots, s_m$ . The goal is to get all the items into the knapsacks — in fact, into as few of the knapsacks as possible. For instance, let's have  $n = 3$  knapsacks with  $c_1 = 25$ ,  $c_2 = 30$ , and  $c_3 = 27$ ; let's also have 9 items with sizes

$$20, 15, 14, 9, 8, 7, 4, 3, 2.$$

- a) One approach is “largest first fit”; i.e., stuff the largest item into the first knapsack in which it fits. “First” means lowest index, so you put the item of size 20 in knapsack 1. Then you put the next largest item (size 15) into the first (lowest index) knapsack in which it now fits, namely, knapsack 2. And so on, until all items are in the knapsacks or some item won't fit in any knapsack. Carry this algorithm out for the rest of the given data.
- b) Another approach is “smallest first fit”. Now you start by putting the smallest item into the lowest index knapsack in which it fits, then the next smallest item, and so on. Carry out this algorithm for the given data.
- c) A variation on largest first fit is to first renumber the knapsacks in order of decreasing size.

Thus  $c_1 = 30$ ,  $c_2 = 27$ , and  $c_3 = 25$ . Carry out largest first fit again with this change.

- d) Another approach might be called “largest roomiest fit”. Items are stuffed in order of decreasing size, but you use the knapsack with the most remaining room. (In case of a tie for remaining room, use the knapsack with the lower index.) This is clearly not a good algorithm if you are trying to use the minimum number of knapsacks (why?), but if the goal is simply to fit the items into the knapsacks available, it is at least as plausible as the other approaches described. Carry it out for the data given at the beginning of this problem.

- e) Is it possible to get all 9 of the given items into the 3 knapsacks? Why?

7. (2) The assignment problem. You are the manager of a large consulting firm. You have  $n$  jobs to be performed and  $n$  employees to do them. You have rated each employee for each job, with 10 being outstanding and 1 being miserable. You wish to assign the people to the jobs so that the work gets done as well as possible. The usual mathematical formulation of this is: Find the assignment which maximizes the sum of the ratings. There is an obvious, greedy approach to doing this. State it. Apply it to the following situation, where the number in the  $i$ th row and  $j$ th column is the rating of person  $i$  for job  $j$ .

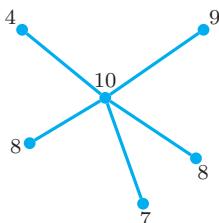
$$\begin{bmatrix} 10 & 9 \\ 8 & 1 \end{bmatrix}$$

What do you conclude about the success of the greedy algorithm for the assignment problem? (See also [18, Section 3.7].)

8. (3) Maximum-weight independent set of vertices. Imagine that you have a graph (i.e., a set of vertices and edges between some pairs of them, as in Fig. P.1 in the Prologue), with each vertex having a number.

You wish to pick a set of vertices, no two of which have an edge between them, so that the sum of the numbers on the selected vertices is as large as possible. (The vertices might represent people, an edge might mean that the people hate each other, and the goal might be to select a team for a project where close cooperation is essential. The numbers might represent the quality of the work a person does when other people aren't bothering

him or her.) There is an obvious greedy algorithm for this problem. State it and then apply it to the following graph. What do you conclude?



*Note:* There is no known efficient algorithm which solves this problem in general.

9. {3} The worst case analysis of Algorithm EUCLID did not provide the strongest upper bound possible. This is not surprising since it broke the problem into 2 uneven cases: In one case the size of the remainder goes down by  $\frac{1}{2}$  in one step, in the other case by  $\frac{1}{2}$  in two steps. Thus all we can be certain about is that the remainder halves in at most two steps. Let's consider an analysis which reduces the problem by the same factor per step. Suppose we pick some fraction  $x$  between  $\frac{1}{2}$  and 1 and break into two cases as follows: Case 1 is if  $rem < xn$ , and Case 2 is otherwise. (As in the text, we assume that  $n < m$ .) Then in Case 2 the new remainder one step later is at most  $(1 - x)n$ . Thus we either decrease the size of the smaller number by a factor of  $x$  in one step or by a factor of  $1 - x$  in two. So suppose we choose  $x$  so that  $x^2 = 1 - x$ . This makes the reduction factor *per step* the same either way.

- a) Find the  $x$  which meets this condition.
  - b) The argument in the text gave a reduction by  $\frac{1}{2}$  in two steps. What reduction does the method of this problem give in two steps?
  - c) Collect some data (more than you were asked to collect in [14, Section 1.5]) and see how close our new bound is to the actual worst case.
10. {3} Consider the problem of determining whether an input integer  $N$  is a prime. You can find out by simplifying Algorithm PRIMENUM in Example 2, Section 1.1. Namely, try dividing  $N$  by 2, and then only by the primes 3, 5, 7, 11, etc., until you find a divisor or until  $N$  itself is reached. Assume that the only step which takes time is division.

- a) What is the best case?
- b) What is the worst case, if  $N$  must be  $< 100$ ?
- c) The average case is hard. But what fraction of positive integers will require just one division? What fraction, 2 divisions? What fraction, 3 divisions? Can you use this information to determine the average case for  $N < 100$ ? Hint: At worst, every  $N < 100$  will either be shown to be composite by one of the first 4 attempted divisions, or it will end up being a prime and require every prime division from 2 to  $N$ .

11. {3} Merge Sort. Suppose you have two lists of words,  $w_1, \dots, w_m$  and  $x_1, \dots, x_n$ , and each list is in alphabetical order. You wish to merge them into a single alphabetical list. There is a simple way to do this. Start by comparing the first words in both lists and picking the one which comes first. Then what?
- a) Write up this approach in our algorithmic language. The only tricky part is what to do when one of the lists is exhausted.
  - b) Assuming that the relevant thing to count is the number of comparisons, determine the best and worst cases for this algorithm. (The average case analysis is quite tricky, for it depends on determining the probability that one list will be exhausted when there are  $k$  words left in the other. For more on the worst case, see Section 5.8.)
12. {3} Consider the following algorithm fragment, which reduces any positive integer  $n$  to 1.

```

repeat until n = 1
  if n is odd then n ← n - 1
    else n ← n / 2
  endif
endrepeat

```

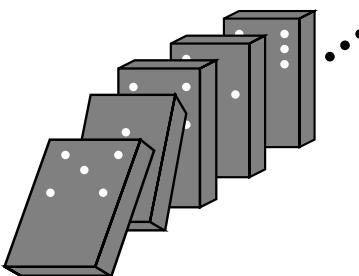
Let the size of  $n$  be the number of bits  $b$  in the binary representation of  $n$ . (This binary measure is more convenient for this problem than the number of digits in the decimal representation.) Do a best, worst, and average case analysis of the number of assignments needed to carry out this algorithm. In other words, determine the best, worse, and average case for all positive integers that require  $b$  bits.

# Mathematical Induction

## 2.1 Introduction

A row of dominoes, all standing on their ends, extends beyond the horizon. The dominoes are spaced close enough so that, if one falls over, the next falls down too. (See Fig. 2.1.) Someone knocks the first domino over.

The result: They all fall down.



**FIGURE 2.1**

The parable of the dominoes:  
The essence of mathematical induction.

This parable of the dominoes describes the essence of **mathematical induction**, the most important method of proof in discrete mathematics. Mathematical induction applies when you have an infinite sequence of statements,

$$P(1), P(2), \dots, P(n), P(n+1), \dots,$$

and you want to prove they're all true. The statements are the dominoes. To be proved true is to be knocked down. If you can show that the truth of any one statement  $P(n)$  would imply the truth of the next statement  $P(n+1)$ , then each domino is close enough to knock down the next. If you can also show that the initial statement  $P(1)$  is true (falls down), then all the statements are true (fall down).

Mathematical induction is intimately related to the inductive and recursive paradigms because it involves the same approach they use: get from one case to

the next by finding a relationship between them. So far we have thought of our two paradigms as being methods of discovering relationships, devising algorithms, and solving problems. However, we also want to be certain (prove) that our algorithms and solutions are correct. This is where mathematical induction comes in. It is the standard proof technique for both paradigms.

In this book, you will see many proof techniques. We hope you will develop a general understanding of them all. For mathematical induction we have a higher goal. We want you to understand it so well that you can *do* it yourself, even in this first course. Not only is mathematical induction the most useful proof technique in discrete mathematics but, fortunately, it is also not hard to learn. The reason is that induction has a specific form. You always have to show that the initial domino falls; then you have to show that each domino knocks over the next.

This is not to say that inductions are all the same, or that all are easy. There are variations (for some of which the domino analogy is not so useful), and we will deal with several of them as we go along. For instance, some inductions require more than one starting case, and other inductions require more than one previous case to get the next case.

This chapter proceeds as follows. Section 2.2 introduces the basic mathematical technique, using a variety of algebraic and nonalgebraic examples, but no applications to algorithms. Section 2.3 extends the technique to so-called Strong Induction, where more than the previous case is needed to prove the next case. Then the next two sections apply induction to algorithms. In Section 2.4 induction is applied to prove recursive algorithms correct; here it is easy to set up the proofs because the induction exactly parallels the recursion in the algorithm. Section 2.5 shows how to apply induction to iterative algorithms. Here there is more subtlety in setting up the induction, but often the nicest approach is to do induction on a loop invariant. Then we turn to induction as a broader idea than proof. In Section 2.6 we show how to conjecture what to prove and how inductive thinking can help in the discovery. Section 2.7 is about inductive definitions. It turns out that every inductive proof relies, usually implicitly, on an inductive definition. Finally, we wrap up with Section 2.8, which gives lots of examples of faulty inductions. You'll want to see if you've absorbed enough to catch the errors.

One last comment: The fact that this proof technique is called induction is in some ways unfortunate. In general discussions of reasoning (by lay people, psychologists, etc.), the word “induction” is contrasted with “deduction”. In such discussions deduction describes reasoning that draws airtight conclusions by logical analysis, while induction refers to the discovery of patterns that *generalize* examples; these patterns may be believed but they have not been proved.

It is in this sense of generalization that we have used the word “induction” in the phrase “*inductive paradigm*” — at least with respect to the first, discovery stage of that paradigm. It is also the sense that is meant when scientists refer to “scientific induction”. But this is not at all the sense we mean in the phrase “*mathematical induction*”. Mathematical induction is a form of rigorous proof and is thus actually a special case of deduction. Thus the word “mathematical” in front of “induction” makes a crucial difference. Nonetheless, in mathematics books, when

it is clear that a method of proof is being discussed, it is customary to drop the word mathematical and simply say induction.

## 2.2 First Examples

---

Before you can *do* inductions, you need to see a few.

### EXAMPLE 1

Prove

$$\sum_{k=1}^n k = \frac{n(n+1)}{2} \quad (1)$$

for all positive integers  $n$ . (We already proved this in Section 1.5, Example 4, while analyzing sequential search, but the proof was by a trick.)

Sums like Eq. (1) are the traditional sort of problem on which mathematical induction is used. Indeed, some people think that's *all* mathematical induction is about. Nothing could be further from the truth. The rest of this chapter shows why. However, it doesn't hurt to start on traditional ground.

Before we start the proof, though, you have a right to object: "How are we supposed to come up with the statement of results like this? What good does it do to teach us how to devise a proof if we don't know how to conjecture what to prove?"

Good questions — and in this book we promise not to ignore them. However, in this section we are only going to say a little about how to conjecture what to prove; learning how to do inductions is already a mouthful, and we don't want you to gag. We save most of our advice about how to make conjectures for Section 2.6, which is devoted entirely to that subject.

For now, let us simply say that the most straightforward way to conjecture Eq. (1) is to use the inductive paradigm. Write down the first few cases:

$$1 = 1, \quad 1 + 2 = 3, \quad 1 + 2 + 3 = 6, \quad 1 + 2 + 3 + 4 = 10, \dots,$$

and see if you can find a pattern. Had we not already told you a formula, maybe you would see the pattern, maybe you wouldn't. As we said, Section 2.6 will help.

**Solution to Example 1** There are three steps in the proof.

Step 1: Name explicitly the statement  $P(n)$  on which to do the induction. In this case it's easy. Eq. (1) is really an infinite collection of statements, one for each positive integer  $n$ :

$$\begin{aligned} P(1) : & \quad 1 = 1(2)/2, \\ P(2) : & \quad 1 + 2 = 2(3)/2, \\ P(3) : & \quad 1 + 2 + 3 = 3(4)/2, \end{aligned}$$

and so on. Thus for each  $n$ ,  $P(n)$  is just the statement  $\sum_{k=1}^n k = n(n+1)/2$  for that particular  $n$ . We wish to prove  $P(n)$  for *all*  $n$ .

( $P$  stands for *proposition*, the word in logic for a statement that is either true or false. See Chapter 7. We will use this  $P$  notation throughout our study of induction.)

Step 2: Prove the initial case, here proposition  $P(1)$ , since we wish to prove  $P(n)$  for  $n = 1, 2, \dots$ . In general, the initial case is called the **basis case**, and the second step is called the **basis step**. For Example 1, the basis step is easy: We simply check that 1 does indeed equal  $1(2)/2$ , as stated.

Step 3: Prove that *any one* statement  $P(n)$ , *should it be true*, implies the truth of the next statement  $P(n+1)$ . In other words, if  $P(n)$ , then  $P(n+1)$ . This implication is usually abbreviated as  $P(n) \Rightarrow P(n+1)$ . This third step is called the **inductive step**, and within this step the hypothesis  $P(n)$  of the implication is called the **inductive hypothesis**. To prove an implication, the most straightforward way is to start with the hypothesis, here  $P(n)$ , and manipulate it until we end up with the conclusion, here  $P(n+1)$ .

Thus in Example 1 we want to start with

$$\sum_{k=1}^n k = \frac{n(n+1)}{2},$$

do some manipulations to both sides, and end with

$$\sum_{k=1}^{n+1} k = \frac{(n+1)[(n+1)+1]}{2}.$$

To help us fill in the middle, we note that the only change above on the left-hand side (LHS) from the top line to the bottom line is the addition of one more term,  $n + 1$ . Following the Golden Rule — do unto one side as you would do unto the other — it's clear that we must also add  $n + 1$  to the right-hand side (RHS). Then, it's simply a matter of massaging this sum until it looks like the RHS of the bottom line above. The result is:

$$\begin{aligned} \sum_{k=1}^n k &= \frac{n(n+1)}{2}, \\ \left(\sum_{k=1}^n k\right) + (n+1) &= \frac{n(n+1)}{2} + n+1, \\ \sum_{k=1}^{n+1} k &= \frac{n(n+1) + 2(n+1)}{2} \\ \sum_{k=1}^{n+1} k &= \frac{(n+1)(n+2)}{2} \\ \sum_{k=1}^{n+1} k &= \frac{(n+1)[(n+1)+1]}{2}, \end{aligned}$$

where as usual each line of a display implies the next line. Thus we have shown that  $P(n) \Rightarrow P(n+1)$  for any  $n$ . In other words, we have shown that  $P(1) \Rightarrow P(2)$ ,

$P(2) \Rightarrow P(3)$ ,  $P(3) \Rightarrow P(4)$ , and so on. Since earlier we showed  $P(1)$ , from  $P(1) \Rightarrow P(2)$  we now know  $P(2)$ ; and from  $P(2)$  and  $P(2) \Rightarrow P(3)$  we also know  $P(3)$ ; and next  $P(4)$ , and so on forever. In other words,  $P(n)$  is true for all positive integers  $n$ . The proof by induction is complete. (Thinking in terms of dominoes, we have shown that all the dominoes fall down because the first fell and each one knocks over the next.) ■

*Remark 1.* We have described mathematical induction as having three steps. Often such descriptions give only two; our first step is omitted. The reason for this is that often the choice of  $P(n)$  is obvious — often the problem breaks naturally into cases  $1, 2, \dots$ , in which case  $P(n)$  is just the  $n$ th case. However, in some problems the choice of  $P(n)$  is far from obvious. Indeed, figuring out  $P(n)$  is often the hardest part of the proof, especially in proofs of the correctness of algorithms. *Always state your  $P(n)$  explicitly.*

*Remark 2.* During the inductive step we do not know that the inductive hypothesis is true; we are doing a what-if analysis — *would* the  $n$ th domino knock over the  $(n+1)$ st? Only when the whole proof is finished, including the basis step (the first domino falls), do we know that the inductive hypothesis is true.

*Remark 3.* To prove  $P(n) \Rightarrow P(n+1)$ , it is not necessary to use  $P(n)$  at the very beginning of your proof, as we did in Example 1. You must end up with  $P(n+1)$ , but you can use  $P(n)$  anywhere along the way that is convenient. For reasons that will only become clear later, in many induction proofs the best place to use  $P(n)$  is not at the start of the inductive step, but in the middle.

Let's illustrate this approach by proving Eq. (1) again, putting  $P(n)$  in the middle of the Inductive Step. Also, now that we have explained the format of induction at length, there is no need to explain it within the proof. Indeed, typically such explanations of induction are not given within proofs, so doing this proof again allows us to present a more standard format.

**Theorem 1.** For all positive integers  $n$ ,

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}.$$

**PROOF** By induction. For each  $n$ , let  $P(n)$  be the statement that  $\sum_{k=1}^n k = n(n+1)/2$ .

*Basis step.*  $P(1)$  asserts that  $1 = 1(2)/2$ , which is true.

*Inductive step.*

$$\sum_{k=1}^{n+1} k = \left( \sum_{k=1}^n k \right) + (n+1) \stackrel{P(n)}{=} \frac{n(n+1)}{2} + n+1 = \frac{n(n+1) + 2(n+1)}{2}$$

$$= \frac{(n+1)(n+2)}{2} = \frac{(n+1)[(n+1)+1]}{2}.$$

Thus the first expression equals the last, proving  $P(n+1)$  from  $P(n)$ .

Together, the basis step and the inductive step prove the theorem. ■

Note several things. First, we have once again “uglified”  $(n+2)$  to  $[(n+1)+1]$  in the last step. Such manipulations seem strange to those who have been taught only to simplify, and are often omitted, but they show that we have  $P(n+1)$  exactly. We urge you, at least for now, to show as explicitly as you can that you have obtained  $P(n+1)$  exactly.

Second, notice the  $P(n)$  sitting over the second equals sign. Putting it there is a brief way of saying that  $P(n)$  is the reason why the equality is true. Indeed, that equality is obtained by substituting  $n(n+1)/2$  for  $\sum_{k=1}^n k$ , which are equal quantities according to  $P(n)$ , into  $(\sum_{k=1}^n k) + (n+1)$ . There is another way to indicate that  $P(n)$  is the reason: Write “ $P(n)$ ” in brackets at the right-hand end of the line where it is used, in the same way we have put reasons in brackets in algorithms. When using the bracket method, it is a good idea to have just one equality (or inequality, or whatever) on that line; otherwise, how can the reader know which equality (or other assertion) on that line you referred to? In other words, in bracket format the first line of the preceding display would best be broken up as

$$\begin{aligned}\sum_{k=1}^{n+1} k &= \left(\sum_{k=1}^n k\right) + (n+1) \\ &= \frac{n(n+1)}{2} + n+1.\end{aligned}\quad [P(n)]$$

There is yet a third way to indicate the reasons for your algebra: Break up the string of equations with words at the appropriate points. In fact, this is the most common style across all mathematics. In this style the proof now includes:

*Inductive step.* By the meaning of sums,

$$\sum_{k=1}^{n+1} k = \left(\sum_{k=1}^n k\right) + (n+1).$$

By  $P(n)$ , we may substitute  $n(n+1)/2$  for  $\sum_{k=1}^n k$ , obtaining

$$\sum_{k=1}^{n+1} k = \frac{n(n+1)}{2} + (n+1).$$

Now by algebra,

$$\begin{aligned}\frac{n(n+1)}{2} + (n+1) &= \frac{n(n+1) + 2(n+1)}{2} \\ &= \frac{(n+1)(n+2)}{2} = \frac{(n+1)[(n+1)+1]}{2}.\end{aligned}$$

In summary,

$$\sum_{k=1}^{n+1} k = \frac{(n+1)[(n+1)+1]}{2},$$

which is  $P(n+1)$ .

This proof takes longer to write, but you or your instructor may find it more agreeable.

## EXAMPLE 2

Prove that

$$(1+x)^n \geq 1 + nx$$

whenever  $n$  is a nonnegative integer and  $x$  is a nonnegative real number.

**Solution** First, why should an inequality of this form be interesting? Because the LHS might be tedious or ugly to compute exactly (say, if  $x = .3972$ ); so it is nice if we can approximate or bound the value by an expression that *is* easy to compute (the RHS). Since this particular LHS shows up in lots of problems, both inequalities and equalities for it are worth obtaining. (We prove an equality, the “Binomial Theorem”, in Section 4.6.)

Why should we conjecture this particular inequality? You probably already know that

$$(1+x)^2 = 1 + 2x + x^2,$$

$$(1+x)^3 = 1 + 3x + 3x^2 + x^3,$$

so in the cases  $n = 2$  and  $n = 3$  the first two terms of the exact expansion are  $1 + nx$ . Since every term in the expansion is nonnegative when  $x$  is, the inequality  $(1+x)^n \geq 1 + nx$  seems reasonable.

In any event, for each  $n$  let  $P(n)$  be the statement that  $(1+x)^n \geq 1 + nx$  when  $x \geq 0$ . Note that the basis case this time is  $P(0)$ , not  $P(1)$ , because we were asked for a proof for all nonnegative  $n$ .

*Basis step.*  $P(0)$  states that

$$(1+x)^0 \geq 1 + 0x, \quad \text{or} \quad 1 \geq 1. \checkmark$$

(The  $\checkmark$  is a useful shorthand for “which is true”.)

*Inductive step.* We start with the LHS of  $P(n+1)$ , use  $P(n)$  in the middle, and obtain the RHS of  $P(n+1)$ .

$$\begin{aligned} (1+x)^{n+1} &= (1+x)^n(1+x) \\ &\geq (1+nx)(1+x) \tag{[P(n)]} \\ &= 1 + nx + x + nx^2 = 1 + (n+1)x + nx^2 \\ &\geq 1 + (n+1)x. \tag{[Since  $x^2 \geq 0$ ]}\end{aligned}$$

Thus

$$(1+x)^{n+1} \geq 1 + (n+1)x,$$

which is  $P(n+1)$ . ■

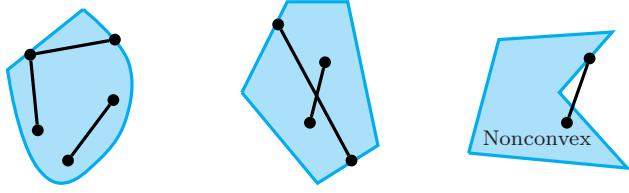
**Note 1.** The only hard part in the inductive step for Example 2 is the first equality. Of all the ways that we might manipulate  $(1+x)^{n+1}$ , why do we choose  $(1+x)^n(1+x)$ ? Because we know that we have to use  $P(n)$  somewhere, so we must “break down” what we start with in order that some key aspect of the statement  $P(n)$  shows. Since the key aspect of this  $P(n)$  is the expression  $(1+x)^n$ , the factorization  $(1+x)^n(1+x)$  is a good thing to try first.

**Note 2.** The statement we proved,  $(1+x)^n \geq 1 + nx$ , is a statement in two variables,  $n$  and  $x$ . However, only one of them,  $n$ , is restricted to integers, so it was clear that we had to do induction on  $n$ . That is, the statement was broken into cases based on the value of  $n$ , not  $x$ . In some later multivariable problems, the choice of the inductive variable won’t be so obvious.

### EXAMPLE 3

Find and prove a formula for the sum of the angles of a convex polygon.

**Solution** A **convex set** is a set such that every line segment connecting points of the set lies entirely in the set. A **convex polygon** is a polygon whose perimeter and interior together form a convex set. See Fig. 2.2. A polygon has at least  $n = 3$  sides (a triangle) and in a triangle the sum of the angles is  $180^\circ$ , a result whose truth we assume you know. The breakdown approach allows us to both discover the more general formula and prove it. As the convex polygon in Fig. 2.3 shows, we can always break down an  $(n+1)$ -gon into a triangle and an  $n$ -gon. Therefore the (interior) angles in an  $(n+1)$ -gon sum to  $180^\circ$  more than the angles of an  $n$ -gon. Thus the formula for the angle sum is  $180(n-2)$ , because this formula is correct for a triangle and increases by  $180$  for each additional side.



**FIGURE 2.2**

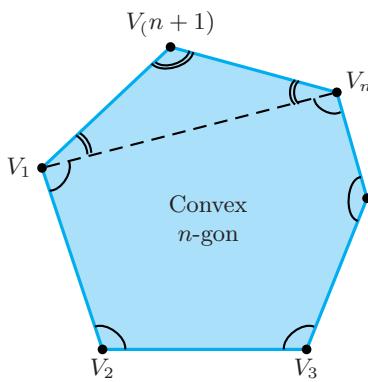
A convex set, a convex polygon, and a nonconvex polygon.

We may summarize this analysis in the form of a tight and careful proof.

**PROOF** For each  $n$  let  $P(n)$  be the statement that any convex  $n$ -gon has  $180(n-2)$  degrees as the sum of its interior vertex angles. Clearly  $n$  is restricted to  $n \geq 3$ , so  $n = 3$  is the basis case. Furthermore, we have already noted the truth of Statement  $P(3)$ . For the inductive step, given any arbitrary  $(n+1)$ -gon, slice it into an  $n$ -gon and a triangle with segment  $V_1V_n$  as in Fig. 2.3. By  $P(n)$ , the sum of the angles in the  $n$ -gon is  $180(n-2)$ . The sum of the angles in the triangle is  $180^\circ$ . The sum of both is the sum of the interior angles for the  $(n+1)$ -gon. This sum is

$$180(n-2) + 180 = 180[(n+1)-2].$$

Thus  $P(n)$  implies  $P(n+1)$ . ■



**FIGURE 2.3**

Breakdown of an  $(n+1)$ -gon into an  $n$ -gon and a triangle.

Did you notice that we actually used more than one previous case to prove  $P(n+1)$ ? In addition to  $P(n)$  we used  $P(3)$ . Such multiple use should not surprise you; in the recursive paradigm one often has to relate the next case to several previous cases. However, we have to admit that this use is not in keeping with the specific formulation we gave to mathematical induction in Section 2.1 — there each domino was strong enough *by itself* to knock over the next. So is this proof valid?

Yes, it is. We have shown  $P(3)$  and  $[P(3) \text{ and } P(n)] \implies P(n+1)$ . Substituting  $n = 3$  in the implication, we conclude that  $P(4)$  is true. Now substitute  $n = 4$ . Since we know both  $P(3)$  and  $P(4)$  at this stage, we get  $P(5)$ . Next,  $P(3)$  and  $P(5)$  imply  $P(6)$ , and so on. We have another valid induction principle.

There are many other variants of induction — three previous cases are needed, a different two previous cases are needed, and so on. Fortunately, they can be lumped together by allowing all previous cases to be used. This results in the principle of **strong induction**, which we discuss in the Section 2.3.

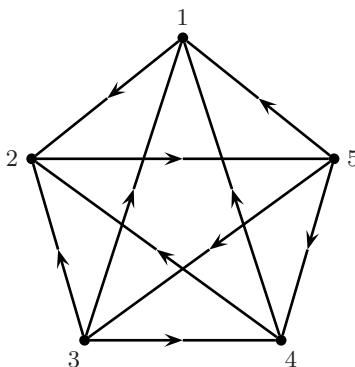
## Induction and Tournaments

We devote a subsection to tournaments because they provide a fine opportunity to break away from the idea that induction is solely a method for proving formulas. In fact, this subsection contains no algebraic computation at all. Tournaments also provide another preview of the broad usefulness of graphs, the subject of Chapter 3.

We will discuss ranking in round-robin tournaments. A **round-robin tournament** is one in which each team plays every other team exactly once. There is a real-world difficulty with ranking teams in such tournaments: How many games a team wins might not be the best indication of relative strength. (Why?) Consequently, there has been considerable research into other possible ways to rank teams. One approach that seems quite reasonable (but see [31]) is to look for a “chain of command” — an ordering of the teams so that each team beats the next team in the order. If such an order exists, one might declare this order to be the ranking. You might suspect, however, that in general no such ordering exists, especially if bizarre things happen in the tournament, like team  $A$  beating team  $B$  but then losing to every team that  $B$  beats. Surprisingly, such an order *must* exist:

**Theorem 2.** Suppose that  $n$  teams play a round-robin tournament in which there are no tie games. No matter what the individual game outcomes are, one can always number the teams  $t_1, t_2, \dots, t_n$ , so that  $t_1$  beat  $t_2$ ,  $t_2$  beat  $t_3$ , and so on, through  $t_{n-1}$  beat  $t_n$ .

Figure 2.4 illustrates this phenomenon for a particular tournament with five teams. The tournament is represented by a directed graph. The teams are the vertices, and the arrows on the edges point from winners to losers. For instance, 1 beat 2. Sure enough, there is a ranking of the sort that Theorem 2 claims: 5,3,4,1,2. (That is,  $t_1 = 5$ ,  $t_2 = 3$ , and so on.) The theorem claims that this is not luck — any round-robin tournament with any number of teams will have such a ranking!



**FIGURE 2.4**

A particular tournament with five teams.

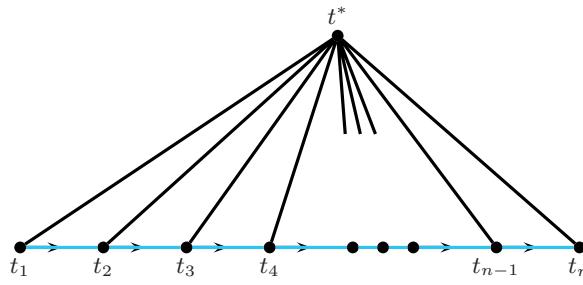
**PROOF OF THEOREM 2** For each  $n$ , let  $P(n)$  be the statement of the theorem for  $n$  teams. The smallest meaningful value of  $n$  is 2, for a tournament consisting of one game. Thus the basis case  $P(2)$  is obviously true: Call the winner of that game  $t_1$  and the loser  $t_2$ .

The heart of the proof is, as usual, the inductive step. The inductive hypothesis is: Any tournament with  $n$  teams has the desired ranking. We must show that any tournament with  $n+1$  teams does also. So, imagine that somebody presents us with the results of an arbitrary  $(n+1)$ -tournament (tournament with  $n+1$  teams). We must break down the new case to make the old case show. How? Temporarily, ignore any one team, call it  $t^*$ . The games played between the remaining teams form an  $n$ -tournament, so by  $P(n)$  there is a ranking  $t_1, \dots, t_n$  such that  $t_i$  beat  $t_{i+1}$  for  $i = 1, 2, \dots, n-1$ . This is illustrated by the directed path of arrows along the bottom of Fig. 2.5.

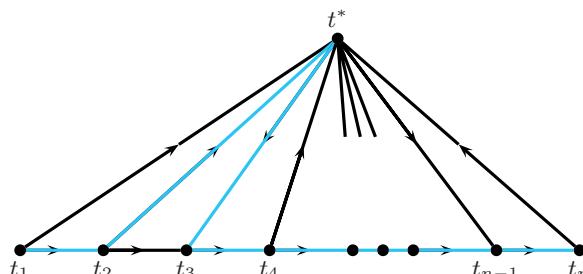
Now remember  $t^*$ . It plays against each of the other teams; we just don't know which of those games it won. Thus we have indicated those games in Fig. 2.5 as lines without arrows. In terms of the figure, the question is: Is there some directed path which includes all the vertices? For instance, is there somewhere we can insert

**FIGURE 2.5**

The graph of an arbitrary  $(n+1)$ -tournament, assuming that all  $n$ -tournaments have rankings.



What if we move  $t^*$  into the directed path along the bottom and keep all the arrows going the way they should? Fig. 2.6 illustrates a particular way the games involving  $t^*$  might have gone, and an insertion which works in that case (follow the path printed in color). But we must give an argument which demonstrates the existence of an insertion no matter how the arrows in Fig. 2.5 are filled in.



**FIGURE 2.6**

One possibility for inserting  $t^*$ .

Suppose that the arrow between  $t^*$  and  $t_1$  points down. Then we are done: Insert  $t^*$  at the start of the ranking. So consider the remaining case where the arrow points up. Look from left to right at the other edges from  $t^*$ . If we ever find an arrow pointing down from  $t^*$ , then we are in business: Let  $t_i$  be the first vertex for which the edge from  $t^*$  is directed down and insert  $t^*$  between  $t_{i-1}$  and  $t_i$  in the ranking. If there aren't any arrows pointing down from  $t^*$ , we are also in business: Use the last arrow to insert  $t^*$  at the end of the ranking. Since we have found a ranking in every case, we have shown that  $P(n) \Rightarrow P(n+1)$ , completing the inductive step. The basis step and the inductive step together prove the theorem. ■

*Remark 1.* Note the intimate relation between this proof and the recursive paradigm: We suppose we know the truth of the previous case (when, in fact, we don't yet understand at all why it should be true) and then we show that this assumption implies the truth of the current case. In short, after the initial case of an inductive proof we jump right into the middle—the recursive paradigm. Heretofore, we have used that paradigm to compute the solution to problems, usually by an algorithm. Now we find we can use it to do proofs. Indeed, we can use the recursive paradigm to do both at once. For instance, now that we have proved Theorem 2,

wouldn't it be nice to have an algorithm that actually finds the ranking? The recursive relationship between cases  $n$  and  $n+1$  in the proof should suggest an algorithm to you [43, 44].

*Remark 2.* Note that at no point in the proof did we have to think about all the games in the  $n$ -tournament or in the  $(n+1)$ -tournament. The inductive step (recursive paradigm) forced us to look only at what the  $n$ th conclusion provides, namely, the results in the bottom row of Fig. 2.5. Given this limited information about the  $n$ -tournament, we are led to look for a directed  $(n+1)$ -path that differs from the  $n$ -path only by an insertion. If we had been left to imagine the whole tournament and look for any sort of directed path, we would have been incapacitated by the lack of focus.

Finally, the proof above is quite long. This time it's because we tried to motivate our approach. You don't have to provide the motivation with your proofs (although, if your approach is particularly clever, motivation will help even wise old professors read it). To help you see what good, succinct mathematical writing style is, we provide an appropriately short rewrite.

**TIGHT PROOF OF THEOREM 2** By induction. Let  $P(n)$  be the statement of the theorem for  $n$ . Thus  $P(2)$  is clear: Rank the winner of the one game before the loser. For the inductive step, consider an arbitrary  $(n+1)$ -tournament. Temporarily ignore some one team,  $t^*$ , leaving an  $n$ -tournament. By  $P(n)$ , there is a ranking  $t_1, t_2, \dots, t_n$  of the remaining teams so that  $t_i$  beat  $t_{i+1}$  for all  $i = 1, 2, \dots, n-1$ . Now consider the games between  $t^*$  and the others. Either (1)  $t^*$  beat  $t_1$ ; (2) some  $i > 1$  is the first  $i$  for which  $t^*$  beat  $t_i$ ; or (3) every other team beat  $t^*$ . In the first case, insert  $t^*$  at the start of the ranking. In the second, insert  $t^*$  between  $t_{i-1}$  and  $t_i$ . In the third, insert  $t^*$  at the end. ■

Like our earlier proof of Theorem 2, this tight version could well have continued along the lines:

Thus we have shown  $P(n) \Rightarrow P(n+1)$ . Therefore the inductive step is complete, so, combining it with the basis case, we conclude that  $P(n)$  is true for all  $n \geq 2$ .

Working mathematicians tend to leave these last sentences out because — once the writer has announced that the proof is by induction — this part of the argument is understood. For now, though, it may help you get a grasp on what is going on to leave such sentences in.

In any event, please note that, even in this tight version, most of it is words. *Mathematical arguments do not consist of unconnected calculations.*

## Summary and Further Remarks

The Principle of Mathematical Induction (in the so-called “Weak” or “First” form) is the following, and covers most of the examples so far.

---

## The Principle of Mathematical Induction, Weak Form

To prove the proposition  $P(n)$  for all integers  $n \geq n_0$ , it suffices to prove both

- i) the Basis Step:  $P(n_0)$ , and
  - ii) the Inductive Step: for all  $n \geq n_0$ ,  $P(n) \Rightarrow P(n+1)$ .
- 

There are a lot of subtleties in mathematical induction. We now discuss three key ones. You may want to come back to this discussion later when you grapple with harder examples or get confused.

*Remark 1.* Remember, in induction,  $P(n)$  is the name of a *proposition*.  $P(n)$  is *not* a number, nor a polynomial function, nor an algebraic expression. Thus in a proof of Theorem 1 it is *not* acceptable to write something like “Thus  $P(n) = n(n+1)/2$ .” In other contexts it is perfectly reasonable for  $P(n)$  to be the polynomial  $n(n+1)/2$ , but not here. In induction it is a statement, and statements can’t ever equal numbers.

*Remark 2.* When one writes a statement in mathematics, like  $x + y = y + x$ , or  $\sum_{k=1}^n k = n(n+1)/2$ , one usually means to claim it *for all* values of the variables. (See the discussions of implicit quantifiers on p. 61 and p. 667.) However, in induction, when we define  $P(n)$  we mean it to represent the claim for a *single value* of  $n$ . Thus, continuing our sum example, just as  $P(3)$  is the single claim  $1 + 2 + 3 = 3(3+1)/2$ , so is  $P(n)$  a single claim for one particular but arbitrary positive integer  $n$ . In short,  $n$  here is a *generic particular* as explained on p. 62. Sometimes we shall make it explicit that  $P(n)$  refers only to a particular  $n$  but more often we shall not because, probably unfortunately, this omission is traditional in mathematical writing.

But why make this fuss distinguishing between a particular  $n$  and all  $n$ ? The reason for the fuss is because in induction we don’t directly prove  $P(n)$  but rather that  $P(n) \Rightarrow P(n+1)$ . If, in this implication,  $P(n)$  did mean the claim for all  $n$ , then induction would be an invalid method because it really would assume what it claims to prove.

Proving  $P(n) \Rightarrow P(n+1)$  is often much easier than proving  $P(n)$ , because proving that one thing implies a similar thing is usually easier than proving something from scratch. That’s why the method of induction often succeeds when direct attacks fail.

This discussion is much easier to present, and perhaps to understand, with the use of the quantifier symbolism of Chapter 7 (e.g.,  $\forall$ ). Readers who already know that symbolism might turn to the discussion of Quantification and Induction starting on p. 668.

## Problems: Section 2.2

1. ⟨2⟩ Discover and prove a formula for the sum of the first  $n$  odd positive integers.
2. ⟨2⟩ By doubling the equation in Theorem 1, we get a formula for the sum of the first  $n$  even positive integers. For practice, prove this formula directly by induction.
3. ⟨2⟩ A sequence  $u_1, u_2, u_3, \dots$  is defined by  $u_1 = 1$  and, for  $n \geq 1$ ,  $u_{n+1} = u_n + 8n$ . Compute the first few terms in the sequence. Guess a closed formula for  $u_n$ . Prove it by induction.

4. ⟨2⟩ An arithmetic sequence with initial term  $a$  and difference  $d$  is a sequence  $a, a+d, a+2d, \dots$ , where each term is  $d$  more than the preceding term. Show by induction that the  $n$ th term is  $a + (n-1)d$ . (This fact is pretty obvious; the point is to practice your induction skills.)

5. ⟨3⟩ Compare  $\sum_{k=1}^n k^3$  with  $(\sum_{k=1}^n k)^2$ . Conjecture a relationship and prove it by induction.

6. ⟨2⟩ Prove by induction the standard formula for the sum of the first  $n$  terms of an arithmetic series:

$$\sum_{k=1}^n [a + (k-1)d] = an + \frac{1}{2}n(n-1)d.$$

Then prove it again using Theorem 1 and facts about sums from Section 0.4.

7. ⟨2⟩ Prove by induction the standard formula for the sum of the first  $n+1$  terms of a geometric series with ratio  $r \neq 1$ :

$$\sum_{k=0}^n ar^k = \frac{a(r^{n+1}-1)}{r-1}.$$

8. ⟨2⟩ If you write out  $\sum_{k=1}^n (a_{k+1}-a_k)$  for  $n = 1, 2, 3, 4$ , you'll see that, in general, the sum simplifies to  $a_{n+1}-a_1$ . This simplification is called the **telescoping series** property. Prove it rigorously using induction.

9. ⟨3⟩ Use the telescoping series property to find a simple expression for  $\sum_{n=1}^m 1/((n+1)n)$ . You will need to reexpress  $1/((n+1)n)$  in the form  $a_{n+1}-a_n$ .

10. ⟨2⟩ Prove the product formula

$$\prod_{j=1}^n \frac{j}{j+1} = \frac{1}{n+1}.$$

(We hope this formula is fairly obvious, but proving it is a good first example of using induction with products.)

11. ⟨3⟩ Find and prove a formula for

$$\prod_{n=2}^m \left(1 - \frac{1}{n^2}\right).$$

12. ⟨3⟩ Formulate and prove a **telescoping product** property; see [8].

13. ⟨2⟩ In Section 0.6 we discussed direct and indirect proofs. So far all our induction proofs have been direct, but they don't have to be. Redo the inductive step of Example 1 indirectly. That is, assume the equality  $P(n+1)$  is false and use this inequality to contradict  $P(n)$  or prove that  $P(n)$  is false.

14. ⟨1⟩ State the formula for the sum of angles in a convex polygon (Example 3) using radians. Redo the inductive proof.

15. ⟨2⟩ Given a line segment of unit length, show that you can construct with straightedge and compass a segment of length  $\sqrt{n}$  for each natural number  $n$ . To start, recall how you would construct  $\sqrt{2}$ .

16. ⟨2⟩ Discover and prove a formula for

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^n.$$

17. ⟨2⟩ Prove that  $(1+x)^n > 1+nx$  for all  $x > 0$  and  $n > 1$ .

18. ⟨2⟩ Prove by induction that  $2^n > n+1$  for all integers  $n \geq 2$ . Then prove it again as a special case of [17].

19. ⟨3⟩ In the second line of the inductive step in Example 2, the fact that  $(1+x)^n \geq 1+nx$  was applied to  $(1+x)^n(1+x)$  to obtain

$$(1+x)^n(1+x) \geq (1+nx)(1+x).$$

This is a substitution of an inequality to obtain an inequality, but such substitutions are more subtle than substitutions of equalities to obtain equalities.

a) It is simply not always true that

$$a > b \implies ac > bc.$$

For what numbers  $a, b, c$  is it true? Why?

b) For what real numbers  $a, b, c$  does  $a \geq b \implies ac \geq bc$ ?

c) What can you conclude about  $ac$  and  $bc$  if  $a > b$  and  $c < 0$ ?

d) What conditions must be put on  $x$  and  $y$  so that  $x > y \implies 1/x < 1/y$ ?

20. ⟨3⟩ For  $n > 1$ , consider

$$1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{3}} + \cdots + \frac{1}{\sqrt{n}}.$$

Compare this sum to  $\sqrt{n}$ ; then conjecture a relationship and prove it.

21. ⟨4⟩ Let  $x_1 = 1$ , and for  $n \geq 1$  let

$$x_{n+1} = x_n + \frac{1}{x_n}.$$

Prove: For  $n > 1$ ,  $x_n > \sqrt{n}$ .

22. ⟨2⟩ Consider the sequence

$$x, \quad x^x, \quad x^{(x^x)}, \quad x^{\left(x^{(x^x)}\right)}, \dots$$

Prove: If  $x > 1$ , then this sequence is increasing.

Hint: For each  $n$ , let  $P(n)$  be “ $a_{n+1} > a_n$ ”, where  $a_n$  is the  $n$ th term.

23. ⟨3⟩ Show that, for each integer  $n > 1$ ,

$$\underbrace{\sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \cdots}}}}}_{n \text{ 1's}}$$

is irrational. The innermost radical sign contains only the integer 1. You may assume that  $\sqrt{2}$  is irrational; if you have never seen a proof of this, get someone to show you.

24. ⟨3⟩ Fact:  $n^3 - n$  is divisible by 3 for all integers  $n$ .

a) Prove this fact for nonnegative integers by induction.

b) Prove it for nonpositive integers by **downward induction**—pxx[24]; that is, show  $P(0)$  and then prove  $P(n) \implies P(n-1)$ .

c) Show that the nonpositive case follows from the nonnegative case by algebra alone — a second induction is unnecessary. Hint: Let  $n$  be any positive integer. You know that

$n^3 - n$  is divisible by 3 and want to show that  $(-n)^3 - (-n)$  is divisible by 3.

d) Prove this fact all at once for all integers  $n$  by factoring.

25. ⟨3⟩ Prove that the sum of the cubes of any three consecutive integers is divisible by 9. See [24] for several methods.

26. ⟨3⟩ Prove that if  $n$  is even, then  $13^n + 6$  is divisible by 7.

27. ⟨3⟩ Let  $a$  and  $b$  be integers. Prove that  $a^n - b^n$  is a multiple of  $a - b$  for all positive integers.

28. ⟨2⟩ Conjecture a formula for the number of subsets of an  $n$ -set and prove it by induction.

29. ⟨3⟩ Prove by induction that the number of even cardinality subsets of an  $n$ -set is equal to the number of odd-cardinality subsets.

30. ⟨3⟩ A famous puzzle goes like this: Three men are lined up facing forward, one behind the other, so that the last can see the heads of the first two, the second can see the head of the first, and the first can't see the others. They close their eyes, and hats are put on their heads. They are told that each hat is either red or white, and at least one hat is red. The man in back is asked if he can determine the color of his hat. He says No (and the others can hear him). The middle man is asked if he can now determine the color of his hat, and he says No. The front man is asked if he can now determine the color of his hat. He says Yes. The question is: what color hat did the front man have and how did he know? You are to assume the men are “logicians”, that is, they will correctly deduce any conclusion that can be deduced from the data.

In fact, there is an  $n$ -man version of this puzzle. Find it and prove its solution correct by induction.

31. ⟨2⟩ Theorem 2 says less than you might think.

a) For instance, it does not claim that the ranking is unique. Find another such ranking for the tournament of Fig. 2.4.

b) It also does not say that each team beats *all* teams further down in the ranking, or even most teams further down. Find a ranking in Fig. 2.4 in accord with Theorem 2 and in which the first team in the ranking beats *only* the second team in the ranking.

32. ⟨1⟩ Suppose that tie games are allowed in a

round-robin tournament. Prove that there is a ranking so that each team either beat or tied the next team.

*Note:* To obtain this proof, you don't have to prove a new theorem. You just have to apply Theorem 2 in a new way, by redefining what a directed edge represents. While in a certain sense that makes this problem very easy, it hardly makes it insignificant. It is a very important point about applying mathematics that once you find the right *model* (i.e., representation of the applied problem with mathematical concepts), sometimes the information you seek falls out as known results.

33. (1) Again suppose that ties are allowed, but that we still insist on a ranking in which each team beat the next team. Give an example where such a ranking does not exist.

34. (1) In an alternative form of round robin, each team plays every other team  $k$  times, where  $k$  is some fixed positive integer. (In many professional sports leagues, each season consists of such a round robin.) We say that team  $i$  "won its series" with team  $j$  if  $i$  won more than half the games against  $j$ . Prove that, if  $k$  is odd, there is always a ranking such that each team won its series against the next team.

35. (1) A tournament is not a round robin if some teams never play each other. Show by example that a ranking as in Theorem 2 need not exist if a tournament is not a round robin.

36. (3) Show that, if a round-robin tournament has an odd number of teams, it is possible for every team to win exactly half its games. *Hint:* Do induction by 2; i.e., assuming there is a tournament verifying  $P(2n-1)$ , show how to add two players and fix their games so that you have a tournament verifying  $P(2n+1)$ .

37. (3) Show that, if a round robin has an even number of teams, then a possible outcome is that, for every team, the number of games it wins and the number it loses differ by 1.

38. (4) Can you prove the results of [36, 37] by a single induction?

39. (4) A round-robin  $n$ -tournament is **linear** if there is a ranking  $t_1, t_2, \dots, t_n$  so that  $t_i$  beat  $t_j$  iff  $i < j$ . In other words,  $t_1$  beats all others,  $t_2$  beats all others except  $t_1$ , and so on. Next, a **3-cycle** in a tournament is a set of teams  $t, t', t''$  such that  $t$  beats

$t'$ ,  $t'$  beats  $t''$ , and  $t''$  beats  $t$ . A  **$k$ -cycle**, or just **cycle** (length not specified) is defined similarly.

- a) Prove: If a tournament is linear, it has no 3-cycles. (This is easy.)  
b) Prove the converse. Use induction on the number of teams in the tournament.

40. (2) Prove: If a tournament has a cycle, then it has a 3-cycle. *Hint:* Do induction on the length of the known cycle.  
41. (3) Call a ranking  $t_1, t_2, \dots, t_k$  of some of the teams of an  $n$ -tournament *compatible* with the tournament if  $t_1$  beat  $t_2$ ,  $t_2$  beat  $t_3, \dots, t_{k-1}$  beat  $t_k$ . With this language, Theorem 2 says that every tournament has a compatible ranking of all the teams.

This language also allows us to pinpoint the essence of the inductive step of the proof of Theorem 1: Any compatible ranking of some teams can be extended to a compatible ranking with one more team. By iterating this argument (adding another team over and over), we obtain the following

**Fact:** Any compatible ranking of some  $k$  teams of an  $n$ -tournament extends to a compatible ranking of all the teams, where *extends* means that the order of the original teams does not change as more teams are added. (To turn the informal argument given above for this fact into a formal proof, see [42].)

- a) Use the Fact to prove: If tournament  $T$  contains a 3-cycle (or indeed, any cycle), then  $T$  has more than one compatible ranking of all the teams.  
b) Use [39] to prove the converse of part a). *Hint:* Use proof by contradiction. Suppose  $T$  had more than one compatible ranking and no 3-cycle. What's the contradiction?  
42. (3) While we hope the argument for the Fact in [41] was evident, it was an "and so on" argument, and these are always implicit inductions. In this case, though, it's a little tricky to formalize the induction. Do it each of the following ways.  
a) Fix  $k$  and let  $P(j)$  be the statement: Any compatible ranking of  $k$  teams in an  $n$ -tournament, where  $n \geq j$ , can be extended to a compatible ranking of  $j$  teams. The basis case is  $P(k)$ .

b) Fix  $n$  and let  $P(j)$  be the statement: Any compatible ranking of  $j$  teams in an  $n$ -tournament can be extended to a compatible ranking of all  $n$  teams. The basis case is now  $P(n)$  and you do a *downward* induction to  $P(k)$ .

43. {4} Write an iterative algorithm that accepts as input the outcomes of the games of an  $n$ -tournament, and outputs a ranking as in Theorem 2. *Suggestion:* Input the data as an  $n \times n$  0–1 matrix, with  $a_{ij} = 1$  iff team  $i$  beat team  $j$ . The hard part to do efficiently is the updating of the ranking. If you know about pointers, use them.

44. {4} Same as [43], but make the algorithm recursive.

45. {3} Here is one of the simpler sorts of fair division problems.  $N$  campers must share a cake, and each one wants to feel that she got at least  $1/N$  of the cake. There is an algorithm that ensures this result, called the “moving knife” algorithm. A referee (the camp counselor) slowly passes a knife over the cake from left to right. Each time a camper yells “stop”, the referee stops moving the

knife across, cuts the cake at this point, and gives the piece to the left of the cut to the camper who yelled.

(If several campers yell at once, the referee picks one of them arbitrarily to get the piece.) Then the process begins again (a new “round”) with one fewer campers and the remaining cake. In each round, each camper is supposed to yell as soon as she thinks  $1/m$  of the remaining cake has passed under the knife, where  $m$  is the number of remaining campers. (Do you see that there is strong incentive to yell when you are supposed to, even though if you wait you might get an even bigger piece of cake?) Anyway, the following is a theorem:

**Theorem.** If there are  $N$  campers, if the moving knife procedure is followed, and if each camper yells when she is supposed to (as defined above), then each camper will get what she thinks is at least  $1/N$  of the cake (that is, at least her fair share).

Prove this theorem.

## 2.3 Strong Induction and Other Variants

---

We used an analogy to dominoes in our presentation in Section 2.2 of the weak form of the principle of mathematical induction. But suppose to knock down the  $n$ th domino we need to have more than just the  $(n-1)$ st domino falling against it. That is, suppose that mathematically we need to assume the truth of more than just  $P(n-1)$  in order to prove that  $P(n)$  is true. Is it valid to use more cases? Would the induction still be valid?

Yes. In general, we *have* more than  $P(n-1)$  when we wish to prove  $P(n)$  is true. Indeed, when we are ready to prove  $P(n)$  for any specific  $n$  we already know not just that  $P(n-1)$  is true but also that *all* the propositions  $P(i)$ ,  $1 = n_0, n_0 + 1, \dots, n - 2$  are true (where  $P(n_0)$  is the basis case) since they will have been proved on our way from  $n_0$  to  $n - 1$ . Furthermore, it is always legitimate to use everything we have already proved. We hope this argument convinces you of the **strong form** of mathematical induction, in its usual form stated below. However, as we will show later in this section, an even stronger form is needed to cover all the sorts of cases that actually arise.

The strong form states: To prove  $P(n)$  for all integers  $n \geq n_0$ , it suffices to prove both

- i) the Basis Step:  $P(n_0)$ ; and

- ii) the Inductive Step: for all  $n > n_0$ , the truth of all  $P(j)$ ,  $j = n_0, \dots, n-1$ , implies the truth of  $P(n)$ .

We need strong induction because, as noted in Example 3, Section 2.2, ordinary (weak) induction doesn't always do the trick. It's nice if we can get the conclusion of the inductive step by assuming just the immediately preceding case — we have fewer balls to juggle — but sometimes that case doesn't give us enough punch. Strong induction is the opposite extreme: We assume every preceding case.

Often, we don't need that much punch; usually just a few preceding cases in ii) will do. Fortunately, further intermediate induction principles are not needed. The inductive step of strong induction *allows* us to use all preceding cases, but we are not *required* to use them. Any situation where we use one or more preceding cases is therefore justified by strong induction. For instance, ordinary weak induction is actually a special instance of strong induction (only case  $P(n-1)$  is used from ii). In other typical instances, we need

- the immediately preceding case and one early case; or
- one or more preceding cases, but you don't know which ones; or
- the  $j$  immediately preceding cases ( $j$  is usually 2).

We now give three examples, one for each situation.

## EXAMPLE 1

### The Generalized Distributive Law

Assuming the distributive law

$$a(b + c) = ab + ac, \quad (1)$$

prove the generalized distributive law: For  $n \geq 2$ ,

$$a \sum_{i=1}^n b_i = \sum_{i=1}^n ab_i. \quad (2)$$

**Solution** Let  $P(n)$  be the  $n$ th case of Eq. (2). The basis,  $P(2)$ , is Eq. (1), and we were told to assume it. Inductive step: We break down  $\sum_{i=1}^n b_i$  into  $(\sum_{i=1}^{n-1} b_i) + b_n$ . Denoting the sum to  $n-1$  as a single term  $B$ , we have the start of a proof beginning with the LHS of  $P(n)$ :

$$a \left( \sum_{i=1}^n b_i \right) = a(B + b_n).$$

Our immediate impulse is to distribute on the right, after which we can expand  $aB$  using  $P(n-1)$ . That's fine, but note that in distributing we are using  $P(2)$ . So the entire argument is

$$\begin{aligned}
 a\left(\sum_{i=1}^n b_i\right) &= a\left(\sum_{i=1}^{n-1} b_i + b_n\right) \\
 &\stackrel{P(2)}{=} a \sum_{i=1}^{n-1} b_i + ab_n \\
 &\stackrel{P(n-1)}{=} \sum_{i=1}^{n-1} ab_i + ab_n = \sum_{i=1}^n ab_i,
 \end{aligned}$$

where again we have inserted  $P$ 's over the equal signs to indicate where we have used the assumptions of the truth of earlier cases. In other words, this is a strong induction using one early case,  $P(2)$ , and the immediately preceding case,  $P(n-1)$ . ■

*Remark.* Both the distributive law and the generalized distributive law are probably equally obvious (or nonobvious!) to you, in which case you probably feel that we should treat them equally — either assume both without proof, or prove both. We agree with you! However, we prove only one for the following reasons. We prove the generalized law as an illustration of the fact that many mathematical facts involving two terms have generalizations to  $n$  terms, and the proof of the generalization is usually a straightforward strong induction, if you assume the two-term form. See, for example, [1, 2, 7, 11]. We don't prove the basic distributive law ( $P(2)$  above) because its proof is very subtle. Indeed, the more elementary the arithmetic, the harder it is to prove things rigorously — we have less to work with. For a proof, take a course with a title like “Foundations of Analysis” or “Philosophy of Mathematics”. Rest assured, though, its proof involves induction — aided by many other things.

## EXAMPLE 2

### Prime factorization

Most every kid in middle school knows that every positive integer greater than 1 can be factored into primes, where if a number is prime, factoring it means leaving it alone. For example,

$$7 = 7, \quad 18 = 2 \cdot 3 \cdot 3, \quad \text{and} \quad 1001 = 7 \cdot 11 \cdot 13.$$

Prove this fact by strong induction.

**Solution** Define  $P(n)$  to be the statement that  $n$  can be factored into primes. The basis case is  $P(2)$ , which is true because 2 is a prime. As for the inductive step, suppose that all numbers before  $n$  have prime factorizations. Consider  $n$ . If it is a prime, we are done. If  $n$  is not prime, then by definition it has some factorization  $n = rs$ . We do not claim that  $r$  and  $s$  need to be primes, but clearly they are less than  $n$ . Therefore, by  $P(r)$  and  $P(s)$ , they themselves have prime factorizations  $r = \prod p_i$  and  $s = \prod q_j$ . By strong induction

$$n = \left(\prod p_i\right)\left(\prod q_j\right)$$

is a prime factorization of  $n$ . ■

*Remark 1.* You may know an even stronger theorem, the *unique* factorization theorem for integers. Its proof also involves induction, but is much harder. We discuss it further in Section 2.8.

*Remark 2.* The difference between Example 2 and Example 1 is that here we don't know the relation of  $r$  and  $s$  to  $n$ . One of them may be small and the other large, or they may both be middling — we don't know and don't care. This sort of situation arises frequently in number theory. It also arises in recursive algorithms; think of the recursive EUCLID algorithms from Section 1.3. When we do know what previous cases are needed (say, we need just case  $n-1$  to handle case  $n$ ), then we would usually devise an iterative algorithm instead of using recursion.

### EXAMPLE 3

Suppose we are given a sequence  $a_1, a_2, \dots$ , where for  $n > 1$ ,

$$a_{n+1} = 2a_n + a_{n-1}. \quad (3)$$

Show that if  $a_1$  and  $a_2$  are integers, then all terms are integers.

**Solution** This conclusion should be pretty obvious and we prove it simply as a first example of its type. It's obvious because, when you add integers to get new terms, you get integers. But this is just the gist of the inductive step. Realizing that, let's go right to a proof.

*Definition.* For each  $n$ , let  $P(n)$  be the statement that  $a_n$  is an integer.

*Basis step.*  $P(1)$  and  $P(2)$  are given.

*Inductive step.* Assume that all terms through  $a_n$  are integers. To show  $P(n+1)$ , we simply note that  $a_{n+1}$  is an integer by Eq. (3). ■

Did you notice that this valid induction does not quite fit the strong form of mathematical induction? Why? Because that form, as stated, requires each case after  $n_0$  (here 1) to be provable using just previous cases. In particular, it requires  $P(2)$  to be provable from  $P(1)$ . But  $P(2)$  is not provable from  $P(1)$ , because the recurrence (3) requires two previous cases and so doesn't kick in until  $n = 3$ . That's why Example 3 states that both  $a_1$  and  $a_2$  are integers. In short, we need two basis cases for this induction to work.

Here is another example which requires more than one basis case, after which we will state a stronger induction principle that covers all the examples in this section.

### EXAMPLE 4

Prove that any item costing  $n > 7$  kopecks can be bought using only 3-kopeck and 5-kopeck coins.

**Solution** The key idea is: If you can buy an item costing  $n-3$  kopecks with just such coins, you can also buy an item costing  $n$  kopecks — use one more 3-kopeck coin. So as long as we can get back to a basis case by multiples of 3, we can prove the  $n$ th case. But that means we need three basis cases.

Here's a proof:

*Definition.* Let  $P(n)$  be the statement that  $n$  kopecks can be paid using 3-kopeck and 5-kopeck pieces only.

*Basis step.*  $P(8)$  is true since  $8 = 3+5$ .  $P(9)$  is true since  $9 = 3+3+3$ .  $P(10)$  is true since  $10 = 5+5$ .

*Inductive step.* For  $n > 10$ , assume  $P(n-3)$ . With this assumption pay out  $n - 3$  kopecks using 3- and 5-kopeck pieces. Now use one more 3-kopeck piece and you have paid  $n$  kopecks. ■

We promised an induction principle that covers all cases so far. Let's call it the final strong form. This is not a standard name because, alas, most books state only the strong form given earlier without noting that it doesn't cover all the cases they need. In actuality, all inductions that are not weak are called strong, whether they need the strong form or our final strong form.

---

### The Principle of Mathematical Induction, Final Strong Form

To prove  $P(n)$  for all integers  $n \geq n_0$ , it suffices to prove for some positive integer  $k$  both

- i) the Basis Step: the  $k$  statements  $P(n_0), P(n_0+1), \dots, P(n_0+k-1)$ ; and
  - ii) the Inductive Step: for all  $n \geq n_0+k$ , the truth of all  $P(j)$  from  $P(n_0)$  to  $P(n-1)$  implies the truth of  $P(n)$ .
- 

Example 3 is an instance of this principle with  $k = 2$  and  $n_0 = 1$ , while for Example 4 we have  $k = 3$  and  $n_0 = 8$ .

The main issue for you in *using* induction should not be the exact wording of the principles, but rather should be deciding what principle to use and how to use it correctly. In the inductive step, to prove  $P(n)$  will you need more than  $P(n-1)$ ? The statement you are trying to prove will usually decide that for you. But when do you need more than one basis case? The answer is that, if your inductive step *always* needs more than one previous case, or always needs a case that is not the immediately previous case, then you will need more than one basis case to get started.

There is one more aspect of induction that we should mention. When you have a problem involving several variables, and you can't figure out how to do an induction on just one of them, you have to do a **multiple induction**. Most commonly, there are two variables, in which case this variant is called **double induction**. We postpone a discussion of double induction until we first need it, in Example 5, Section 5.3. For those who want to figure out double induction for themselves, see [21, Supplement].

# Problems: Section 2.3

1. ⟨2⟩ Almost every rule of algebra has a generalized form. Assuming the standard form, use strong induction to prove the following generalized forms. The sums and products below are assumed to run from 1 to  $n$ . The standard form is the case  $n = 2$ .

a)  $(\sum x_i)/m = \sum(x_i/m)$

b)  $(\prod x_i)^\alpha = \prod(x_i^\alpha)$

c)  $b^{\sum a_i} = \prod b^{a_i}$

d)  $\log(\prod x_i) = \sum(\log x_i)$

2. ⟨2⟩ For  $n > 2$ , prove  $\sum_{i=1}^n |x_i| \geq |\sum_{i=1}^n x_i|$ , assuming the case  $n = 2$ . To see how to prove case  $n = 2$ , look at [3, Section 0.2].

3. ⟨2⟩ It is well known that in a triangle with sides  $a, b, c$ , we have  $c < a+b$ . State and prove a generalization of this inequality to arbitrary convex polygons.

4. ⟨2⟩ It is a fact that if a prime  $p$  divides the product  $ab$ , where  $a$  and  $b$  are integers, then either  $p$  divides  $a$  or  $p$  divides  $b$ . Assuming this fact, prove that if  $p$  divides  $\prod a_i$ , where each  $a_i$  is an integer, then  $p$  divides at least one of the  $a_i$ 's.

5. ⟨2⟩ Every odd integer  $\geq 3$  has an odd prime divisor. Given the Prime Factorization Theorem (Example 2), this statement is obvious and easy to prove. (What integers have even factors only?) For practice, prove it from scratch by strong induction.

6. ⟨2⟩ Let  $Q(n)$  be the statement “Every positive integer from 2 to  $n$  has a prime factorization.” Prove that every positive integer has a prime factorization by *weak* induction on  $Q$ .

7. ⟨3⟩ Explain why

$$\lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x+y \rfloor \leq \lfloor x \rfloor + \lfloor y \rfloor + 1.$$

State and prove a generalization that bounds  $\lfloor \sum_{i=1}^n x_i \rfloor$ .

8. ⟨3⟩ For the ceiling function, find and prove similar results to those in [7].

9. ⟨2⟩ Let  $A$  and  $B_1, B_2, \dots$  be sets. Assuming

$$A \cap (B_1 \cup B_2) = (A \cap B_1) \cup (A \cap B_2),$$

prove

$$A \cap \left( \bigcup_{i=1}^n B_i \right) = \bigcup_{i=1}^n (A \cap B_i).$$

10. ⟨2⟩ Prove the dual of [9]; that is, prove the statement with unions and intersections interchanged.

11. ⟨2⟩ Assuming De Morgan's first law of sets,

$$\overline{A \cup B} = \overline{A} \cap \overline{B},$$

state and prove its generalized form. Recall from Section 0.1 that  $\overline{A}$  is the complement of  $A$  relative to some universal set  $U$ .

12. ⟨2⟩ State and prove the generalization of

$$\overline{A \cap B} = \overline{A} \cup \overline{B}.$$

13. ⟨3⟩ Use [9,11,12] to give another proof of [10] that does not directly use an induction.

14. ⟨2⟩

- a) Assuming that

$$\lim_{x \rightarrow c} (f(x) + g(x)) = \lim_{x \rightarrow c} f(x) + \lim_{x \rightarrow c} g(x),$$

prove that

$$\lim_{x \rightarrow c} \sum_{i=1}^n f_i(x) = \sum_{i=1}^n \lim_{x \rightarrow c} f_i(x).$$

- b) Assuming that

$$\lim_{x \rightarrow c} (f(x)g(x)) = \left( \lim_{x \rightarrow c} f(x) \right) \left( \lim_{x \rightarrow c} g(x) \right),$$

prove that

$$\lim_{x \rightarrow c} \prod_{i=1}^n f_i(x) = \prod_{i=1}^n \lim_{x \rightarrow c} f_i(x).$$

15. ⟨2⟩ In Section 2.1 we gave a domino analogy for weak induction. That physical analogy doesn't work for strong induction: to knock over the 100th domino, domino 1 is neither needed nor present (it is already knocked over much earlier in the line).

- a) Can you think of a good physical analogy for the first form of strong induction?  
b) Can you think of a good physical analogy for the final form of strong induction?

16. ⟨1⟩ Sometimes a statement  $P(n)$  is proved for all positive integers by splitting into odd and even

cases. If so, one way to prove it is with a **split induction**. For the odd case, one proves the basis  $P(1)$  and shows  $P(n) \Rightarrow P(n+2)$  for the inductive step. (This is effectively a weak induction with *step size 2*.) For the even case, one proves  $P(2)$  and again shows  $P(n) \Rightarrow P(n+2)$ .

a) Show how such a split induction is a special case of the final form of strong induction.

b) Explain how Example 4 can be viewed as a split induction.

17. ⟨1⟩ Prove the result of Example 4 again, using a 5-kopeck piece in the inductive step.

18. ⟨2⟩ Suppose you can prove that the truth of  $P(n)$  implies the truth of  $P(n+2)$  for all positive  $n$ . What else must you be able to prove before you can assert that  $P(n)$  is true for all positive  $n$ ?

19. ⟨2⟩ Let  $P(n)$  be a proposition defined for each positive integer  $n$ .

a) Suppose you can prove that  $P(n+1)$  is true for  $n > 3$  if *all* of  $P(n)$ ,  $P(n-1)$ ,  $P(n-2)$  and  $P(n-3)$  are true. What else must you prove to assert that  $P(n)$  is true for all positive  $n$ ?

b) Suppose you can prove that  $P(n+1)$  is true if *any* of  $P(n)$ ,  $P(n-1)$ ,  $P(n-2)$  or  $P(n-3)$  are true. What else must you prove to assert that  $P(n)$  is true for all positive  $n$ ?

20. ⟨3⟩ Suppose that a sequence of numbers  $a_1, a_2, a_3, \dots$  has the property that for each  $n > 1$ ,

$$a_{n+1} = 2a_n + a_{n-1}. \quad (4)$$

a) Prove: If *any* two consecutive terms in the sequence are integers, then every term is an integer.

b) Prove: If  $a_1$  and  $a_2$  are increased, and all other terms are changed to maintain Eq. (4), then all terms are increased.

21. ⟨4⟩ Suppose that a sequence  $a_1, a_2, \dots$  satisfies

$$a_{n+1} = 2a_n - a_{n-1}. \quad (5)$$

Prove: If  $a_1$  is increased,  $a_2$  is increased more than  $a_1$  is increased, and all other terms are changed so that Eq. (5) remains true, then every term is increased.

*Caution:* This proof is much trickier than that in [20b]. If you make the obvious choice for  $P(n)$  — “ $a_n$  is increased” — you won’t be able to do the inductive step, even assuming the two preceding cases. Try it! Then find a “good”  $P(n)$ .

22. ⟨3⟩ Suppose that a function  $T$  is defined on the nonnegative integers by

$$\begin{aligned} T(0) &= 0, \\ T(n) &= T(\lfloor n/3 \rfloor) + T(\lfloor n/5 \rfloor) + \\ &\quad T(\lfloor n/7 \rfloor) + n, \quad n > 0. \end{aligned}$$

Prove that  $T(n) < 4n$  for  $n > 0$ .

23. ⟨3⟩ Suppose a sequence  $a_0, a_1, \dots$  is defined by  $a_0 = a$ ,  $a_1 = b$  and

$$a_n = 1 + \max\{a_{\lfloor n/2 \rfloor}, a_{\lceil n/2 \rceil}\}, \quad n \geq 2.$$

Prove: If  $a \leq b$ , then  $\{a_n\}$  is an increasing sequence. (This problem is relevant to all sorts of binary division algorithms, e.g., BINSEARCH in Example 5 of Section 1.5. Do you see why?)

24. ⟨3⟩ How many strictly increasing sequences of integers are there that begin with 1 and end with  $n$ ? Prove it. (A strictly increasing sequence  $a_1, a_2, \dots$  is one in which, for all  $i$ ,  $a_i < a_{i+1}$ . For instance, when  $n = 6$ , two of the strictly increasing sequences from 1 to  $n$  are 1, 2, 6 and 1, 3, 5, 6. Notice that different lengths are allowed.)

25. ⟨3⟩ Consider the directed graph with vertex set  $V = \{1, 2, \dots, n\}$  such that there is an edge from vertex  $i$  to vertex  $j$  iff  $i < j$ . (Our diagrams of tournaments in Section 2.2, for instance, Fig. 2.4, are directed graphs.) How many directed paths are there from 1 to  $n$ ? Discover a pattern and prove it. Then look at [24].

26. ⟨4⟩ Round-robin tournaments revisited. Here is another attempt to come up with a way to determine the strength of a team in a round robin. Let a team be called a “Real Winner” if, for every other team, it either beat that team or beat some third team which beat that team. Prove that every tournament has a Real Winner. Try both weak and strong inductions.

27. ⟨3⟩ In [26], it happens to be easier to prove the theorem without induction. Given a round robin, show that any team which wins the most games is necessarily a Real Winner.

28. ⟨4⟩ Alas, the real winner concept of [26] is not very satisfactory, for there can be too many winners. Indeed, it’s a theorem that, for every  $n > 4$ , there exists an  $n$ -tournament in which *every* team is a Real Winner. Prove this theorem. *Suggestion:* It is possible, but tricky, to prove this by

ordinary induction. It turns out to be much simpler to prove it by the induction principle of [18], modified to start at  $n = 5$  instead of  $n = 1$ .

29. ⟨3⟩ Let  $\theta$  be an angle for which  $\sin \theta$  and  $\cos \theta$  are rational. Prove that  $\sin n\theta$  and  $\cos n\theta$  are rational for all integers  $n$ .

30. ⟨3⟩ Fact: Given two squares, it is possible to dissect them into a finite number of polygonal pieces (all sides straight) that can be reassembled into a single square. Fig. 2.7 shows one way to do it: If the squares have sides  $a$  and  $b$ , with  $a \leq b$ , then don't cut the smaller square at all, cut the larger square into four pieces as shown, and reassemble.

- a) Verify that this method works, i.e., the reassembly really does fit together just right (no overlaps or gaps) and forms a square.  
b) Prove that *any* number of squares may similarly be dissected and reassembled into one square.

- c) Make three squares of sides 4, 5 and 6, respectively, from different colors of construction paper. Actually cut them up into the polygonal pieces indicated by your proof, and reassemble. (No, this isn't art class, but there is something to learn from this hands-on approach about the relation of inductive proofs to explicit constructions.)

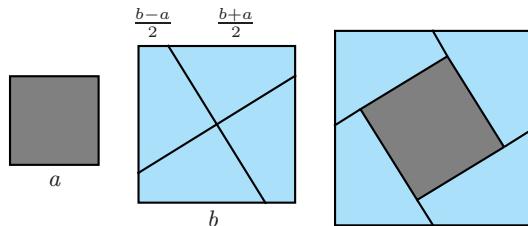


FIGURE 2.7

Dissection of two squares into one.

## 2.4 Induction and Recursive Algorithms

Algorithms are so important in this book that we devote two sections in this chapter to the use of induction to analyze algorithms and prove them correct. In this section we discuss recursive algorithms since the setup of recursive algorithms so closely parallels the setup of induction. Thus it is always straightforward to apply induction to these algorithms — or at least to get started. In Section 2.5 we apply induction to iterative algorithms, for which there is often more difficulty, but also often more options in setting up the inductions. In both sections we focus primarily on proving algorithms correct, but we also discuss the use of induction to prove other properties of algorithms.

This section will use two sets of examples, one about Towers of Hanoi (TOH), the other about Euclid's algorithm.

### Towers of Hanoi

When we discussed TOH before (Example 3, Section 1.3), we gave a recursive algorithm, HANOI, for solving it. Though we motivated the algorithm at length, we never really *proved* that it works. Now we will.

Temporarily ignore the algorithm. Let's first prove by induction that the puzzle has a solution. Then we turn back to Algorithm HANOI (restated in this section as Algorithm 2.1) and prove, again by induction, that it constructs a solution. Comparing the two proofs will illustrate an important point about recursion.

---

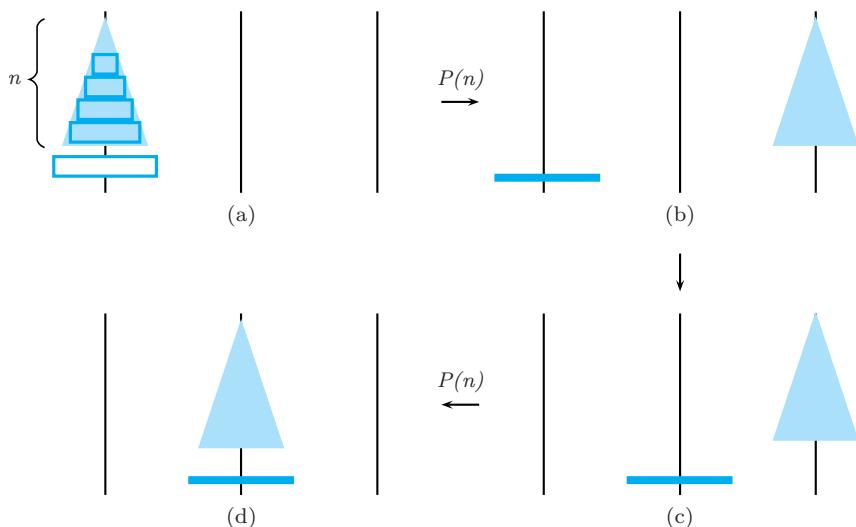
**Theorem 1.** Towers of Hanoi can always be solved.

---

**PROOF** For the first time, the definition of  $P(n)$  does not stare us in the face from the statement of the theorem. The theorem appears to be a single assertion rather than a sequence of related assertions. But it's not hard to split up. For each  $n$ , let  $P(n)$  be the proposition that TOH can be solved when there are exactly  $n$  rings — independently of the pole they start on or where they are supposed to go. If we prove  $P(n)$  for all  $n \geq 1$ , we have proved the theorem.

$P(1)$ . It's obvious: Move the ring directly to the finish pole.

$P(n) \implies P(n+1)$ . We are given  $n+1$  rings on some initial pole and are told to move them to another final pole. See Fig. 2.8, in which we illustrate the case with the initial pole on the left and the final pole in the middle. Temporarily ignore the bottom ring and think of the others as a unit — shown as a triangle in the figure. (It's all right to ignore the bottom ring since it is biggest. The rules say that, no matter where it is at any given time, it does not affect what moves the other rings may make.) By the standard induction assumption that  $P(n)$  is true, somehow there exists a way to move the remaining  $n$  rings to *either* other pole. Imagine they are moved to the *third* pole (Fig. 2.8b). Now remember ring  $n+1$ . At this point it may be moved to the middle pole (Fig. 2.8c). Finally, by  $P(n)$  again, the top  $n$  rings may be moved from the third pole to the middle, on top of ring  $n+1$  (Fig. 2.8d). Thus, assuming  $P(n)$ , we have shown  $P(n+1)$ . ■



**FIGURE 2.8**

The inductive step in the proof that Towers of Hanoi is solvable.

At this point you should look at Algorithm 2.1. We have made one change from the statement of the algorithm in Chapter 1. We have given the Input variables the

same names as the defining variables in the recursive procedure  $H$ , namely  $n, r, s$ . (In Chapter 1 the Input variables were  $num, Pinit, Pfin$ .) By using the same names, we don't have to worry whether we are talking about the main algorithm or the procedure and then worry whether we are using the right names. (As we explained in Section 1.4, Inputs and defining variables have different roles, and it is usually good to distinguish them, but in this case we will be better off using the same names.)

## Algorithm 2.1

### Hanoi

<b>Input</b>	$n$	[Number of rings]
	$r$	[Initial pole; $1 \leq r \leq 3$ ]
	$s$	[Final pole; $1 \leq s \leq 3; s \neq r$ ]
<b>Output</b>	The series of commands to the robot	
	to move the rings from pole $r$ to pole $s$	
<b>Algorithm HANOI</b>		
<b>procedure</b> $H(\text{in } n, r, s)$		
<b>if</b> $n = 1$ <b>then</b> $\text{robot}(r \rightarrow s)$		
<b>else</b> $H(n-1, r, 6-r-s)$		
$\text{robot}(r \rightarrow s)$		
$H(n-1, 6-r-s, s)$		
<b>endif</b>		
<b>endpro</b>		
$H(n, r, s)$	[Main algorithm]	

**Theorem 2.** Algorithm HANOI works.

The approach we will use is the standard approach for validating algorithms that rely on a recursive procedure. We do induction on the statement that the recursive procedure does what it is supposed to. From there it is usually easy to show directly that the main algorithm is also correct.

**PROOF OF THEOREM 2** For each  $n \geq 1$ , let  $Q(n)$  be the statement that procedure  $H(n, r, s)$  gives a correct sequence of moves whenever there are  $n$  rings on an arbitrary start pole  $r$ , and the goal is to get them to another arbitrary pole  $s \neq r$ . Since the main algorithm of HANOI is just a single call of  $H$  for the actual input values, proving  $Q(n)$  correct for all  $n$  will prove that HANOI is correct.

*Basis step.* To prove  $Q(1)$ , simply walk through the procedure when  $n = 1$ . Since the **then** clause takes effect instead of the **else** section, the robot moves the ring

from pole  $r$  to pole  $s$ , and then the procedure stops. The correct “sequence” of one move has been made.

*Inductive step.* This time we assume  $Q(n-1)$  and prove  $Q(n)$ , because that’s the way the indices in this typical recursive algorithm are set up. Clearly, proving  $Q(n-1) \Rightarrow Q(n)$  for  $n > 1$  is the same as proving  $Q(n) \Rightarrow Q(n+1)$  for  $n \geq 1$ , because both claim that, starting with  $Q(1)$ , each proposition  $Q$  implies the next proposition  $Q$ . This is so small a change in the induction format that we won’t even refer to it as a variant form.

So suppose we already knew that procedure  $H$  works whenever it is called for  $n-1$  rings, where  $n-1 \geq 1$ . Let us invoke the procedure with  $n$  rings, and any  $r$  and  $s$ , and walk through it to see what happens. Since  $n > 1$ , the **then** clause is skipped. The first thing that happens in the **else** section is that  $H(n-1, r, 6-r-s)$  is called. By  $Q(n-1)$ , the result of this call is a sequence of moves that somehow correctly gets the top  $n-1$  rings from pole  $r$  to pole  $6-r-s$ . See Fig. 2.8b again! The next line of the algorithm says to move the top ring currently on pole  $r$  to pole  $s$ . This is ring  $n$ , since it is the only ring on pole  $r$ . So now we have Fig. 2.8c. The next line calls  $H(n-1, 6-r-s, s)$ . By  $Q(n-1)$  again, this call correctly moves the  $n-1$  rings onto pole  $s$ ; see Fig. 2.8d. This completes the call of  $H(n, r, s)$ , and the rings are just where we want them to be. ■

In the last paragraph of this proof, we assumed and needed that  $n$  has the same value when each of the three commands of procedure  $H$  is invoked and similarly for  $r$  and  $s$ :

$$\begin{aligned} & H(n-1, r, 6-r-s) \\ & \quad \text{robot}(r \rightarrow s) \\ & H(n-1, 6-r-s, s) \end{aligned} \tag{1}$$

It is such an obvious understanding in mathematics that a letter keeps its meaning in consecutive expressions that it is rarely mentioned. However, here the *algorithm* has control over the values of the variables, and each call of  $H$  involves many subcalls in which the values of the variables  $n, r, s$  are reassigned. So we need to argue that each variable has the same value for each of the commands in (1) above. The reason they have the same values is the convention about local variables discussed on p. 111: all subcalls use different copies of the variables, even if the variable names are the same, and when we return to any call its variables are restored. All three lines of (1) are in the same call, the main call. Thus, for proving the correctness of recursive algorithms, the variable protection scheme of procedures and functions is essential (even if rarely mentioned).

Fig. 2.8 illustrates both proofs at once. Indeed, although we marked the key transitions in the figure with  $P(n)$ , we could just as well have marked them by  $Q(n-1)$ . Our two proofs illustrate that, if you discover how to do something recursively, then you will usually know how to prove your method, how to write an algorithm to do it, and how to prove the algorithm correct — they are related facets of recursion.

Now that we know that Algorithm HANOI solves TOH, let’s use induction to verify various other properties of the algorithm. In Example 2, Section 1.5, we

concluded that HANOI takes  $2^n - 1$  steps (ring moves) to move  $n$  rings from one pole to another. We also concluded that it is a minimum algorithm: no algorithm could take fewer steps. If you review those arguments, you will see that they were inductions, but we did not have the constructs to state them as carefully (“formally”) as we can now. In [1, 2] we ask you to redo those two proofs more formally. The rewrite of these two proofs using induction is quite easy. To prove the step count, for each  $n$  let  $P(n)$  be the claim that  $H(n, r, s)$  takes  $2^n - 1$  steps, for any choices of  $r, s$ . Then continue by identifying the basis and inductive steps in the argument in Section 1.5.

But now let’s use our induction formalism to extend our knowledge of TOH.

---

**Theorem 3.** The minimum solution to Towers of Hanoi is *unique*, in that, for each choice of  $n, r, s$ , there is exactly one sequence of moves that gets an  $n$ -tower from pole  $r$  to pole  $s$  ( $\neq r$ ) in the minimum number of legal moves.

---

In other words, there may be algorithmic formulations other than HANOI that win the game in  $2^n - 1$  moves, but they all do exactly the same sequence of moves. (Henceforth, when analyzing TOH, we will alternate as convenient between the language of *solving* a puzzle and the language of *winning* a (solitaire) game.) For a very different looking, but still minimum, TOH algorithm, see [28, Section 2.5]

**PROOF** We will show how to do this proof without even knowing that the minimum number is  $2^n - 1$ . All that we need is that the minimum number depends only on  $n$ , not on  $r$  and  $s$ . This is true “by symmetry”. That is, since the legality of a move is independent of the positions of the poles (it depends only on what rings are on which poles), any minimum sequence of moves between  $r$  and  $s$  becomes a minimum sequence between any other pair,  $r'$  and  $s'$ , by simply rearranging the poles.

So let  $P(n)$  be the statement that, for each pair  $r \neq s$ , the minimum TOH sequence that gets an  $n$ -tower from pole  $r$  to pole  $s$  is unique. We need to prove  $P(n)$  for all  $n \in N^+$ .

*Basis step,  $P(1)$ .* Clearly the minimum for 1 ring is 1 step, and since the move must go from pole  $r$  to pole  $s$ , it is unique.

*Inductive step,  $P(n-1) \Rightarrow P(n)$ .* Let  $h_n^*$  be the minimum number of moves for the  $n$ -game, that is, the version that starts with  $n$  rings on one pole. The largest ring must move *at least* once (off of pole  $r$ ) and in order for it to move the first time the  $(n-1)$ -game must be played first to get all  $n - 1$  smaller rings out of the way. By symmetry, it must take the same number of moves to move the  $n - 1$  rings to pole  $s$  or to the third pole  $t$ . It must be best to move them to  $t$  for then the largest ring can move to  $s$  and never need move again, after which the  $(n-1)$ -game is played once more to move the rings from  $t$  to  $s$ . (Note that if playing the  $(n-1)$ -game

took different numbers of steps for different poles, then it might save steps to move the top  $n-1$  rings first to goal pole  $s$ , move the largest ring to pole  $t$  and then have to move them all again; but by symmetry we don't have to consider such tricky alternatives in this version of the game.)

Therefore, *every* way to play the  $n$ -game must take at least  $2h_{n-1}^* + 1$  steps. Now, by the induction hypothesis, there is *only one* way to play each of the two minimum  $(n-1)$ -games within this sequence. Therefore, assuming  $P(n-1)$ , we have proved  $P(n)$ , that the  $n$ -game minimum sequence is unique.

Having proved the basis and inductive steps, we have proved  $P(n)$  for all  $n$ . ■

In the problems, we introduce several other versions of TOH — this game makes a very rich example — and ask you to prove various sort of things about them: correctness, run time, uniqueness, etc. All of the proofs are by weak induction, since all the versions can be analyzed recursively in terms of the immediately previous case.

## Euclid's Algorithm

In Section 1.3 we presented two recursive versions of Euclid's algorithm, one which used a recursive function (EUCLID-RECFUNC, reproduced as Fig. 2.9), and another which used a recursive procedure (EUCLID-RECPRO, Fig. 2.10). We now prove by induction that both algorithms output the greatest common divisor of their inputs. In each case, as usual, we do induction on the recursive function or procedure within the algorithm.

---

**Theorem 4.** For all nonnegative integer inputs  $m, n$ , EUCLID-RECFUNC outputs the greatest common divisor of the inputs.

---

<b>Input</b> $m, n$	[Integers $\geq 0$ ]
<b>Output</b> <i>answer</i>	[ $= \text{gcd}(m, n)$ ]
<b>Algorithm</b> EUCLID-RECFUNC	
<b>function</b> $\text{gadf}(i, j)$	
<b>if</b> $j = 0$ <b>then</b> $\text{gadf} \leftarrow i$	
<b>else</b> $\text{gadf} \leftarrow \text{gadf}(j, i - j[i/j])$	[Recursive call]
<b>endif</b>	
<b>endfunc</b>	
<i>answer</i> $\leftarrow \text{gadf}(m, n)$	[Main algorithm]

**FIGURE 2.9**

Algorithm EUCLID-RECFUNC repeated.

**PROOF** Since EUCLID-RECFUNC outputs  $\text{answer} = \text{gadf}(m, n)$ , and since the values of  $m, n$  are assigned to  $i, j$  in the function definition, we need to show, for

all  $i, j \in N$ , that

$$\text{gcdf}(i, j) = \gcd(i, j). \quad (2)$$

We prove this by strong induction on  $j$ . (Recall that the second variable has been the right one to use in all our previous arguments about Euclid's algorithm.) For each fixed  $j$ , let  $P(j)$  be the statement that Eq. (2) holds for all  $i \in N$ .

*Basis step.*  $P(0)$ . By the if-line of the definition of  $\text{gcdf}$  in EUCLID-RECFUNC,  $\text{gcdf}(i, 0) = i$  for any  $i$ . As noted when  $\gcd$  was defined in Example 4, Section 1.1,  $\gcd(i, 0) = i$ . Therefore  $\text{gcdf}(i, 0) = \gcd(i, 0)$  for all  $i \in N$ , proving  $P(0)$ .

*Inductive step.* For  $j > 0$  we want to deduce  $P(j)$  assuming all previous cases. First, recall that in Example 4, we proved

$$\gcd(m, n) = \gcd(n, m - kn).$$

In particular, then,

$$\gcd(i, j) = \gcd(j, i - j \lfloor i/j \rfloor).$$

Now, if  $j > 0$ , then by the recursive call in the definition of function  $\text{gcdf}$ ,

$$\text{gcdf}(i, j) = \text{gcdf}(j, i - j \lfloor i/j \rfloor).$$

Next, since  $0 \leq (i - j \lfloor i/j \rfloor) < j$ , by the strong inductive hypothesis we have

$$\text{gcdf}(j, i - j \lfloor i/j \rfloor) = \gcd(j, i - j \lfloor i/j \rfloor).$$

Putting these last three equations together, we get Eq. (2) for this  $j$  and all  $i$ . That is, we have deduced  $P(j)$  assuming previous cases (actually just one case,  $i - \lfloor i/j \rfloor j$ ), completing the inductive step.

Since we have proved the basis and the inductive steps, by strong induction we have proved  $P(j)$  for all  $j \in N$ . That is, we have proved Eq. (2) for all  $i, j \in N$  and thus have proved Theorem 4. ■

Note how our decision back in Section 1.3 to use different names for  $\gcd$  and  $\text{gcdf}$  has paid off. If we had not had such foresight, we would now be in the embarrassing and confusing position of proving, and explaining why we needed to prove, that  $\gcd(m, n) = \text{gdf}(m, n)$ !<sup>†</sup>

All right, let's turn to EUCLID-RECPRO. Look for similarities and differences between the proof below and the proof above.

**Theorem 5.** For all nonnegative integer inputs  $m, n$ , EUCLID-RECPRO outputs the greatest common divisor of the inputs.

<sup>†</sup>Truth in advertising compels us to admit that it wasn't foresight at all. Only after we reached this section did we realize the problem, and only after heated discussion of several options did we choose to go back and rewrite earlier sections. But in fact, the need for such backtracking when writing mathematics comes with the territory.

<b>Input</b> $m, n$	[Integers $\geq 0$ ]
<b>Output</b> <i>answer</i>	[ $= \text{gcd}(m, n)$ ]
<b>Algorithm</b> EUCLID-REC <sub>PRO</sub>	
procedure gcdp( <b>in</b> $i, j$ ; <b>out</b> <i>result</i> )	
<b>if</b> $j = 0$ <b>then</b> <i>result</i> $\leftarrow i$	
<b>else</b> gcdp( $j, i - j\lfloor i/j \rfloor$ ; <i>result</i> )	[Recursive call]
<b>endif</b>	
<b>endpro</b>	
gcdp( $m, n; answer$ )	[Main algorithm]

**FIGURE 2.10**

Algorithm EUCLID-REC<sub>PRO</sub> repeated.

**PROOF** EUCLID-REC<sub>PRO</sub> outputs *answer*, which is the value returned to the main algorithm by the procedure call gcdp( $m, n; answer$ ). Furthermore,  $m, n$ , *answer* are assigned, respectively, to the defining variables  $i, j, result$ . Therefore, we need to show, for all  $i, j \in N$ , that the value returned to the calling level as *result* by procedure gcdp( $i, j; result$ ) satisfies

$$result = \text{gcd}(i, j) \quad (3)$$

We prove this by strong induction on  $j$ . For each fixed  $j$ , let  $Q(j)$  be the statement that Eq. (3) holds for all  $i \in N$ .

*Basis step*,  $Q(0)$ . By the if-line of the definition of gcdp in EUCLID-REC<sub>PRO</sub>, when  $j = 0$  then *result*  $\leftarrow i$ . Next, we know that  $\text{gcd}(i, 0) = i$ . Therefore, for all  $i$ , when gcdp( $i, 0; result$ ) is called the value of *result* returned satisfies *result* =  $\text{gcd}(i, 0)$ , proving  $P(0)$ .

*Inductive step*. For  $j > 0$  we want to deduce  $Q(j)$  assuming all previous cases. As in the previous proof, we know that

$$\text{gcd}(i, j) = \text{gcd}(j, i - j\lfloor i/j \rfloor).$$

Now, if  $j > 0$ , then by the recursion in the definition of procedure gcdp, the value of *result* passed *into* the call gcdp( $i, j; result$ ) from below is the value computed by the call gcdp( $j, i - j\lfloor i/j \rfloor$ ). Next, since  $0 \leq (i - j\lfloor i/j \rfloor) < j$ , by the strong inductive hypothesis we have that this value of *result* equals  $\text{gcd}(j, i - j\lfloor i/j \rfloor)$ , which, by the previous display, also equals  $\text{gcd}(i, j)$ . Finally, for  $j > 0$  (so that the then-clause is skipped) nothing in the definition of procedure gcdp recomputes *result*. Therefore, the value of *result* passed *out of* gcdp( $i, j; result$ ) still equals  $\text{gcd}(j, i - j\lfloor i/j \rfloor) = \text{gcd}(i, j)$ . In other words, Eq. (3) holds for this  $j$  and all  $i$ , which is  $Q(j)$ , completing the inductive step.

Since we have proved the basis and the inductive steps, by strong induction we have proved  $Q(j)$  for all  $j \in N$ . That is, we have proved Eq. (3) for all  $i, j \in N$  and thus have proved Theorem 5. ■

# Problems: Section 2.4

1.  $\langle 1 \rangle$  Prove that Algorithm HANOI moves individual rings  $2^n - 1$  times. (This was already argued by induction in Example 2, Section 1.5, but it was an informal presentation because we hadn't explained induction yet. Now do it carefully.)

2.  $\langle 2 \rangle$  Prove that Algorithm HANOI solves TOH in the minimum possible number of steps. (Compare with the proof in Section 1.5.)

3.  $\langle 2 \rangle$  To solve [1] you probably showed that the recurrence

$$h_1 = 1,$$

$$h_n = 2h_{n-1} + 1,$$

has the closed-form solution  $h_n = 2^n - 1$ . Now prove that the inequality recurrence

$$h_1^* = 1,$$

$$h_n^* \geq 2h_{n-1}^* + 1,$$

has the closed form consequence  $h_n^* \geq 2^n - 1$ . This fact was used in Example 2, Section 1.5.

4.  $\langle 3 \rangle$  Let a TOH **clutter** be any arrangement of different sized rings on the three poles so that a larger ring is not above a smaller ring. Show that, using the rules of the game, it is always possible to reassemble a clutter into a single pile.

5.  $\langle 3 \rangle$  STOH (Straightline TOH). In this version the poles must be in a straight line, and there is an additional rule: In a single step, a ring can be moved only to an adjacent pole. Since the order of the poles is now important, it makes a difference where we start and finish. Let's say we want to move the rings from one end pole to the other.

- a) Prove that STOH can be solved.  
b) Write an algorithm that solves it.  
c) Prove the algorithm correct.  
d) Determine how many steps your algorithm takes. That is, find a formula in terms of the number of rings,  $n$ , for the the number of ring moves.  
e) Prove your algorithm takes the minimum number of moves for STOH.  
f) Show that the minimum sequence of moves for STOH is unique.
6.  $\langle 4 \rangle$  Repeat [5], but now move the rings from an end pole to the middle.

7.  $\langle 3 \rangle$  Answer [4] again for Straightline TOH.
8.  $\langle 2 \rangle$  Consider TOH when there is an infinite number of poles. Your task is still to get a tower of  $n$  rings from pole 1 to pole 2 following the rules. Find the minimum number of moves to do this, where moving one ring between any two poles continues to count as one step. Prove that your number is minimum. (It's easy to figure out the moves; the point is to practice giving a minimality proof. This is one of those rare cases where one can, and it's not too hard.)

9.  $\langle 2 \rangle$  A claim that an algorithm is correct should state for what values of the inputs the algorithm is correct. For instance, we certainly wouldn't expect Algorithm HANOI to work for noninteger values of the input variables. Now, the Towers of Hanoi puzzle can be solved for 0 rings as well as all positive integer number of rings. For  $n = 0$  rings you can get all 0 rings from any start pole to any finish pole by doing nothing. Yet HANOI does not work for  $n = 0$ .
- a) Why not?  
b) Give a precise statement of the values of the inputs for which Algorithm HANOI works. Point out where in the proof these restrictions come into play.
10.  $\langle 2 \rangle$  Consider procedure TOH, a variant of the recursive procedure in HANOI.

## Algorithm 2.2. TOH

```
procedure TOH( $n, r, s$ )
  if  $n > 0$  then
    TOH( $n-1, r, 6-r-s$ )
    robot( $r \rightarrow s$ )
    TOH( $n-1, 6-r-s, s$ )
  endif
endpro
```

- a) How does this differ from  $H(n, r, s)$  in the text? What is the significance of the differences?  
b) Give an inductive proof of correctness of this procedure for all *nonnegative* integers.

11. ⟨2⟩

- a) Consider Algorithm TOH? from Problem [12, Section 1.4]. Assume it starts with  $n$  rings in proper TOH order on pole  $r$ . Once you decide what the algorithm accomplishes, prove it by induction. For instance, if you conclude that it ends up with the  $n$  rings in proper TOH order on pole  $s$ , prove it for all  $n \geq 1$ . (Don't worry if the steps of the algorithm violate TOH rules; just concentrate on the final configuration.)
- b) Same thing for Algorithm TOH?? from [13, Section 1.4].

12. ⟨2⟩ Our proof of Theorem 4 (that EUCLID-REFUNC works) used induction on the second input variable,  $n$ . You can also prove that  $\text{gcd}(m, n)$  works by doing induction on the first variable, but it is a little messier. Do enough of the induction to show why.

13. ⟨3⟩ In this problem we compare the results of running Algorithm EUCLID-REFUNC with inputs  $(m, n) = (c, d)$  and  $(m, n) = (kc, kd)$ , where  $k$  is some positive integer. Call the two cases run A and run B, respectively.

- a) Prove: For each pair of nonnegative integers  $c, d$ , run A and run B make the same number of calls to function  $\text{gcdf}$ .
- b) Prove: for all nonnegative integers  $c, d$ ,  $\text{gcd}(kc, kd) = k \text{gcd}(c, d)$ . (This result is easy to see using common knowledge of factoring, but the goal here is to prove it by doing an induction that analyzes runs A and B.)

### Algorithm 2.3. EUCLIDROUND-FUNC

---

**Input**  $m, n$  [Integers  $\geq 0$ ]  
**Output**  $answer$  [ $= \text{gcd}(m, n)$ ]  
**Algorithm** EUCLIDROUND-FUNC

```

function gcdf3(i, j)
    if  $j = 0$  then gcdf3  $\leftarrow i$ 
    else roundrem  $\leftarrow |i - \lfloor .5 + \frac{i}{j} \rfloor j|$ 
        gcdf3  $\leftarrow \text{gcdf3}(j, roundrem)$ 
    endif
endfunc
answer  $\leftarrow \text{gcdf3}(m, n)$  [Main alg.]
```

---

14. ⟨3⟩ In doing Euclid's algorithm, when reducing the pair  $i, j$ , we might subtract  $j$  from  $i$  more than  $\lfloor i/j \rfloor$  times, if that leads to a smaller remainder. Algorithm EUCLIDROUND-FUNC does this. In

fact, it amounts to a recursive version of Algorithm EUCLID1 from [12, Section 1.1].

- a) Run EUCLIDROUND-FUNC on  $(m, n) =$
- (45, 26)
  - (21, 13)
  - (70, 29)
- b) Prove that EUCLIDROUND-FUNC is correct.

15. ⟨3⟩ EUCLIDROUND-PRO is (we claim) the same as EUCLIDROUND-FUNC in [14] except written using a procedure. Prove that this variant is correct.

### Algorithm 2.4. EUCLIDROUND-PRO

---

<b>Input</b> $m, n$	[Integers $\geq 0$ ]
<b>Output</b> $answer$	[ $= \text{gcd}(m, n)$ ]
<b>Algorithm</b> EUCLIDROUND-PRO	
procedure gcdp3( <b>in</b> $i, j$ ; <b>out</b> $result$ )	
if $j = 0$ then $result \leftarrow i$	
else $roundrem \leftarrow  i - \lfloor .5 + \frac{i}{j} \rfloor j $	
gcpd3( $j, roundrem$ ; $result$ )	
endif	
endpro	
gcpd3( $m, n$ ; $answer$ )	
[Main alg.]	

---

16. ⟨3⟩ In another variant of Euclid's algorithm, one always rounds *up*, that is, one subtracts  $j$  from  $i$  until  $i - kj$  is nonpositive, then drops the negative sign to get the new remainder. EUCLID-ROUNDUP is a recursive function version of this variant.

- a) Run EUCLIDROUNDUP on  $(m, n) =$
- (45, 26)
  - (21, 13)
  - (8, 7)
- b) Prove that EUCLIDROUNDUP is correct.

### Algorithm 2.5. EUCLID-ROUNDUP

---

<b>Input</b> $m, n$	[Integers $\geq 0$ ]
<b>Output</b> $answer$	[ $= \text{gcd}(m, n)$ ]
<b>Algorithm</b> EUCLID-ROUNDUP	
function gcpd4( $i, j$ )	
if $j = 0$ then $gcpd4 \leftarrow i$	
else $gcpd4 \leftarrow \text{gcpd4}(j, \lceil i/j \rceil j - i)$	
endif	
endfunc	
answer $\leftarrow \text{gcpd4}(m, n)$	
[Main alg.]	

---

17. (3) Write and prove correct a version of EUCLIDROUNDUP from [16] that uses a recursive procedure instead of a function.

18. (2) The following procedure, Coins, finds a way to give  $n$  kopecks change using 3- and 5-kopeck pieces, as in Example 4, Section 2.3. Prove that it works.

```
procedure Coins(in n)      [n ∈ N, n ≥ 8]
    if n = 8 then print 3, 5
    n = 9 then print 3, 3, 3
    n = 10 then print 5, 5
    else print 3
        Coins(n-3)
    endif
endpro
```

19. (3) Prove that the following procedure PF (Prime Factors) prints all the prime factors of  $n$ , with multiplicities. For example, PF(12) prints 2,2,3. You don't have to prove that PF prints the factors in increasing order, though it does.

```
procedure PF(in n)      [n ∈ N, n ≥ 2]
    k ← 2
    repeat while k ≤ √n
        if k|n then PF(k)
            PF(n/k)
            exit      [Exits repeat-loop]
        k ← k+1
    endrepeat
    if k > √n then print n endif
endpro
```

20. (2) Consider Algorithm SUM-RECUR from [21b, Section 1.4]. Prove by induction that it outputs the sum of its inputs.

21. (2) Consider Algorithm MIN1 from [14, Section 1.4]. Prove by induction on  $k$  that the value of function Min( $k$ ) is the minimum of the first  $k$

inputs to the algorithm. Explain why the algorithm as a whole therefore outputs the minimum of all the inputs.

22. (2) Consider Algorithm MULT2 from [10, Section 1.4]. Prove by induction on  $q$  that when the procedure MR( $p, q; prod$ ) within MULT2 outputs  $prod$ , the value of  $prod$  is  $pq$ . Explain why the algorithm as a whole therefore outputs  $xy$ , the product of its inputs, so long as  $y$  is a nonnegative integer.

23. (3) There is a subtlety in proving Theorems 1 and 2. In Theorem 1, our inductive hypothesis was  $P(n)$ , that we can solve the problem when there are  $n$  rings on the board, but in fact there are  $n+1$  because we want to prove  $P(n) \Rightarrow P(n+1)$ . For Theorem 2 the inductive hypothesis is  $Q(n-1)$ , that the procedure  $H$  works when there are  $n-1$  rings, but we use it when there are  $n$  rings, because we want to prove  $Q(n-1) \Rightarrow Q(n)$ . So we need to show that  $P(n)$  and  $Q(n-1)$  apply, even though the situations they refer to do not quite obtain.

- a) They do apply, and this was actually shown in the proof of Theorem 1. Where?
- b) Was a similar argument given in our proof of Theorem 2? If not, put it in.
- c) There is another approach to proving these theorems: Change  $P$  and  $Q$  so that  $P(n)$  and  $Q(n-1)$  apply directly (without a special argument). Define  $P^*(n)$  to be: Whenever there are  $N \geq n$  rings on the board in a legitimate configuration, and the smallest  $n$  are together on one pole, those  $n$  may be moved to any other pole without moving any other rings. Prove Theorem 1 by induction on  $P^*(n)$ .

- d) Why do we have to allow any number of additional rings under the  $n$ -stack in  $P^*(n)$ ? Aren't we only concerned with the case that there is just one?
- e) Come up with a  $Q^*(n)$  that works directly for Theorem 2.

## 2.5 Induction and Iterative Algorithms

We now turn to inductive proofs about iterative algorithms. As we have said before, in many cases the most natural way to prove correctness for algorithms with iteration is with loop invariants, and such proofs hinge on induction, though it's often implicit. There are other inductive proofs as well, often closer to the

proofs for recursive versions of the same algorithms. So, we will begin with proofs using loop invariants (mostly proofs of correctness) and then do proofs with other sorts of inductions, about correctness and other properties. We will spell things out in much more detail than is typical so that you can really see what is going on.

## Loop Invariant Proofs

Our first example will be to prove carefully the correctness of Algorithm EUCLID, our original iterative version of the Euclidean Algorithm from Example 4, Section 1.1. Try to think (before we say it) how the general remarks immediately below apply to that algorithm.

A **loop invariant** for a loop in an algorithm is an assertion that satisfies the following conditions:

1. It is true upon first entry to the loop (usually obviously so);
2. It is *invariant*, that is, if it is true upon entry to the loop for one iteration, it is true upon loop entry for the next iteration.
3. It is true upon final exit from the loop (if there is one).

The reason for the last parenthetical comment will become clear later in this section in Example 3.

The phrase *upon entry* in these conditions has a precise meaning; it means just before the **repeat** or **for** statement that begins the loop. For every iteration of the loop after the first, “upon entry” can also be thought of as just after the **endrepeat** or **endfor** statement that ends the previous iteration — why?

Why not just say that a loop invariant should be true *throughout* each iteration of the loop instead of focusing on the entry point of the loop? The reason is that in many cases the loop invariants we need simply won’t be true throughout the loop; [3] shows that EUCLID itself is an example and indicates why such examples are common.

Conditions 1 and 2 imply by an easy induction (Theorem 1) that a loop invariant is true for every iteration of the loop. This fact usually makes proving Condition 3 easy, as we will see by example in a moment.

A loop invariant is most useful when, in combination with whatever condition causes the loop to terminate (e.g.,  $denom = 0$  in EUCLID), it implies (usually in an easy fashion) that the loop computes correctly what we wish it to compute. This implies we will always wish to prove (if it’s true!) that a loop terminates. Proof of termination is often obvious, but in any event it is usually another induction. So induction normally shows up in two places when using a loop invariant to prove the correctness of an algorithm.

### EXAMPLE 1

Prove by a loop invariant argument that Algorithm EUCLID (repeated as Fig. 2.11) correctly computes the gcd of its inputs  $m, n$ .

<b>Input</b> $m, n$	[Integers $\geq 0$ ]
<b>Output</b> $num$	[ $= \gcd(m, n)$ ]
<b>Algorithm</b> EUCLID	
$num \leftarrow m; denom \leftarrow n$	[Initialize $num$ and $denom$ ]
<b>repeat until</b> $denom = 0$	
$quot \leftarrow \lfloor num/denom \rfloor$	[Quotient]
$rem \leftarrow num - (quot \times denom)$	[Remainder]
$num \leftarrow denom; denom \leftarrow rem$	[New $num$ and $denom$ ]
<b>endrepeat</b>	

**FIGURE 2.11**

Algorithm EUCLID  
repeated.

**Solution** We present a typical loop invariant proof, which doesn't necessarily verify the properties in the loop invariant definition in the order given. So we will point out in brackets what part of the definition is being considered.

**PROOF** First we prove that EUCLID terminates. It does so because, whatever  $num$  and  $denom$  are, so long as the latter is not zero,  $rem$  is assigned a nonnegative integer less than  $denom$  (why?). Since  $denom$  in pass  $i+1$  through the loop is  $rem$  from pass  $i$ , the values of  $denom$  form a strictly decreasing nonnegative sequence that cannot end at a positive number. Therefore  $denom$  must eventually reach 0. When it does, EUCLID terminates.

Now we use a loop invariant to complete the proof that EUCLID is correct. The loop invariant for the repeat-loop in EUCLID is: Upon each entry to the loop,

$$\gcd(num, denom) = \gcd(m, n). \quad (1)$$

Once we show that this statement is indeed a loop invariant [meets Conditions 1–3], we will have validated EUCLID. because, when the loop terminates with  $denom = 0$ , the loop invariant says  $\gcd(num, 0) = \gcd(m, n)$ . From the definition of gcd,  $\gcd(num, 0) = num$ , and  $num$  at that point is exactly what is output. So EUCLID outputs  $\gcd(m, n)$  and is, therefore, correct.

Why is (1) a loop invariant? Because, first, it is true upon first entry to the loop: at that point,  $num = m$  and  $denom = n$ , so the invariant merely asserts the obvious identity  $\gcd(m, n) = \gcd(m, n)$ . [We have verified Condition 1.] Second, if (1) is true at the start of some pass  $P$ , it's true at the start of the next by our old friend

$$\gcd(m, n) = \gcd(n, m-kn), \quad m, n, k \in N. \quad (2)$$

(Let  $m$  in Eq. (2) be the value of  $num$  at the start of pass  $P$ , let  $n$  be the value of  $denom$  then, and let  $k$  be the value of  $\lfloor num/denom \rfloor$  computed during  $P$ .) [Condition 2 verified]. Thus, we now know that (1) is true upon every entrance to the loop, in particular, just before the *last* visit to the first line, **repeat until**  $denom = 0$ . Since the termination point is *at* this line, and no variables change value on that line, (1) is true at termination [Condition 3].

This completes the proof that (1) is a loop invariant and since we have also proved that EUCLID terminates, we have therefore, proved the correctness of EUCLID. ■

Well, this is a fine proof, but it doesn't actually show the inductions! The induction for loop invariants is always the same, so let's do it once in generality (Theorem 1) and then you don't have to do it again. Termination arguments are somewhat more varied, so you do have to do them again whenever you face a new type. The argument for termination in Example 1 occurs often enough in various parts of mathematics that it has a name: the **Finite Descent Principle**. While this principle is pretty obvious, we don't have to simply assert it as a new principle, because it does have a proof, by induction. We will outline the induction for Finite Descent after Theorem 1 and leave the details to a problem.

---

**Theorem 1.** Any assertion that satisfies Conditions 1 and 2 in the loop invariant definition will be true at entry to every iteration of the loop.

---

**PROOF** By weak induction. For each  $k$ , let  $P(k)$  be the claim: if the loop is entered a  $k$ th time, then the loop invariant is true then.

*Basis Step.* That  $P(1)$  is true is just Condition 1 in the definition.

*Inductive Step.* This is Condition 2.

Thus by induction,  $P(k)$  is true for all  $k \in N^+$ , that is, the loop invariant is true at the start of every iteration of the loop. ■

To prove finite descent, let  $Q(k)$  be the assertion: If we start a finite descent process at  $k$ , where for any  $k > 0$  the process results in a smaller nonnegative integer and for  $k = 0$  the process immediately terminates, then the process terminates at 0. We must prove  $Q(k)$  for all  $k \in N^+$ . The basis case is  $Q(0)$ . Why is this true? Just read the statement out loud! Now finish the proof [1].

*Note:* How do loop invariant proofs work when the iteration structure is a for-loop which is controlled by a counter which takes on the values  $Cstart, Cstart+1, \dots, Cstop$  (see Section 1.2)? Perhaps surprisingly, for-loop arguments require a bit more subtlety than repeat-loops. The answer is that at entry to such a for-loop, the counter has implicitly the value  $Cstart - 1$  and at a normal exit from the loop the value of the counter is  $Cstop$ . These values can be used to prove that a purported loop invariant really is one [4]. (Since more often than not  $Cstart = 1$ , the implicit value of the counter at loop entry is often 0.) When counting goes down (i.e., from  $Cstart$  to  $Cstart - 1$  at the first iteration), the situation is similar [5]. For for-loops depending on set membership, things are a bit trickier and we won't pursue them.

## EXAMPLE 2

Prove Algorithm BINSEARCH (repeated from Section 1.5 as Fig. 2.12) correct by a loop invariant argument. Recall that BINSEARCH keeps cutting its list of words in half until it finds the search word or exhausts the list.

<b>Input</b> $n$	[ $n > 0$ ; number of items in list]
$w_i, i = 1, \dots, n$	[Alphabetized list of items]
$w$	[Search word]
<b>Output</b> $i$	[Index of word if in list]
or	
‘failure’	[If word not in list]
<b>Algorithm</b> BINSEARCH	
$F \leftarrow 1; L \leftarrow n$	[Initialize pointers to first ( $F$ ) and last ( $L$ ) items on list]
<b>repeat</b>	
$i \leftarrow \lfloor (F + L)/2 \rfloor$	[Index of approximate midpoint]
<b>if</b> $w = w_i$ <b>then print</b> $i$ ; <b>exit</b>	[Success]
<b>if</b> $w < w_i$ <b>then</b> $L \leftarrow i - 1$	[Leave case-if; search first half]
<b>if</b> $w > w_i$ <b>then</b> $F \leftarrow i + 1$	[Leave case-if; search 2nd half]
<b>endif</b>	
<b>if</b> $L - F + 1 = 0$ <b>then print</b> ‘failure’; <b>exit</b>	
<b>endrepeat</b>	

**FIGURE 2.12**

Algorithm BINSEARCH repeated.

**Solution** Often the hardest part of a loop invariant proof is identifying the invariant. As always we want an assertion that remains true from iteration entry to iteration entry. For Algorithm BINSEARCH the assertion must have something to do with finding the search word if it is to be useful in proving the algorithm correct. If you had to explain intuitively to someone why BINSEARCH works, you might say that it keeps narrowing its search to an appropriate sublist. The right loop invariant is obtained by stating this idea carefully:

Statement  $S$ : Either the search word  $w$  is in the current sublist (from  $w_F$  to  $w_L$  inclusive, for the current values of  $F$  and  $L$ ), or else  $w$  is not on the entire list at all.

For variety, this time we proceed with our proof in the following order: We verify that Statement  $S$  is a loop invariant by checking the three conditions, then we prove termination, and finally we show why the two conclusions together prove BINSEARCH correct.

1. Statement  $S$  is certainly true upon first entrance to the loop, because at this point  $S$  says that either the word is on the entire list or it is not (because  $F = 1, L = n$ ).
2.  $S$  is true at entry to the next pass if it is true at entry to the current pass because each time through the current pass BINSEARCH either finds the search word  $w$  (and terminates, so that the issue of entry to the next pass is

moot) or else it reduces the sublist to that half of the previous sublist where  $w$  must be if it is on the list at all — why? — and then begins the next pass).

3. We must show that  $S$  is true at exit (if there is one) from the repeat-loop. There are only two ways that BINSEARCH can terminate: Either (a)  $w$  has just been found — in which case the first clause of  $S$  before the “or” is true — or (b)  $L-F+1 = 0$  at the end of the repeat-loop — in which case  $S$  is true because it’s always true at the end of the repeat-loop. (Recall that just after the **endrepeat** is the same as just before entry, and by the standard induction from Conditions 1 and 2 we already know that  $S$  always holds at entry. Since no variables are changed in the line before **endrepeat**,  $S$  is also true at every pass through this line.) Note that we haven’t explained why  $L-F+1 = 0$  would ever be true or important, but it’s not relevant in this step of the proof, because here we need only show that  $S$  is true *if* we terminate. The significance of  $L-F+1 = 0$  comes out in the termination argument just below.

So  $S$  is indeed a loop invariant.

Why must BINSEARCH terminate? If it finds  $w$  it clearly terminates; we must also show that if it doesn’t find  $w$  it terminates. The key insight is: The length of the (sub)list under consideration is always  $L - (F-1)$ . (Why?) So long as the list is nonempty (i.e.,  $L-F+1 \geq 1$ ), the list will be cut approximately in half with each pass through the repeat-loop, because  $w$  and all words on one side of it will be eliminated. That is, the length will be reduced. But as soon as the length is 0 (i.e.,  $L-F+1 = 0$ ), the loop will terminate. (Why couldn’t  $L-F+1$  ever be less than 0?) So, by the Finite Descent Principle, if  $w$  is not found the length *will* reach 0 and the algorithm will terminate.

Finally, why is BINSEARCH correct? If it finds  $w$ , it is certainly correct. If it doesn’t actually find  $w$ , we now know it terminates from the line before **endrepeat**, at which point it prints “failure”, the list is empty, and Statement  $S$  is true. Statement  $S$  is an or-statement, and its first clause can’t be true when the current list is empty. Therefore, at this sort of termination the second clause is true:  $w$  is not on the list. Therefore, since BINSEARCH is correct in the only two possible cases, it is a correct algorithm. ■

To summarize this subsection, a proof of algorithm correctness using a loop invariant has three parts: a proof that a certain loop invariant holds, a proof that the loop will terminate, and an explanation of why the truth of the invariant at the termination point implies correctness. The first two parts usually involve inductions, perhaps disguised.

Proofs of iterative algorithms are usually easy, *if* you find the right loop invariant. But how do you find it? There is no simple answer. For instance, in the problems we challenge you to prove correct an iterative TOH algorithm for which we don’t know a simple loop invariant [28].

We have, however, two pieces of advice about finding loop invariants. The first we’ve already given: Familiarize yourself with the loop until you see intuitively why it works and then try to formalize your intuition as a statement about invariance.

Second and more important: In real life the algorithms for whose correctness you are most responsible are the ones you create yourself. You should think about invariants for each repetitive portion of an algorithm *before* you write the algorithm. In other words, try to think of a solution method that works because, in each loop to be constructed, some property is repeatedly true. Then make the algorithm fit the method.

For instance, if you were inventing a gcd algorithm, you might say, can I find a smaller pair of numbers with the same gcd? Once you figure out how to create such a smaller pair, then if you choose to write an iterative algorithm, you should write a loop where the invariant is that the gcd of the current pair at each entry to the loop is the original gcd. This way, you'll have your proof of correctness as soon as you have your algorithm.

## Loop Invariants and Non-Termination

### EXAMPLE 3

Show that the following is *not* an algorithm.

```
Algorithm WHAT'S-WRONG
    a ← 4
    repeat while a ≠ 2
        a ← (a+2)/2
    endrepeat
```

**Solution** What's wrong with WHAT'S-WRONG? It looks like an algorithm. And even if it isn't, can we break the assertion that it's not into an infinite set of statements  $P(n)$  to be proved by induction?

Recall that part of our definition of an algorithm is that it must terminate in a finite number of steps. We will show that WHAT'S-WRONG does not. This time let's use the inductive paradigm to figure out why. Walk through the "algorithm" for a while. You will find that  $a$  gets a new value each time through the loop, and that the first few values are

$$4, \quad 3, \quad 2.5, \quad 2.25, \quad 2.125.$$

The pattern should be clear [11], but the only important thing about the pattern is that the values keep *not* being exactly 2 (although they get ever closer). Thus it appears that the termination condition for the loop is never met.

Put another way, it looks like  $a \neq 2$  is a loop invariant. Since loop invariants only prove algorithms correct if the algorithm terminates, you might think that loop invariants are useless when an algorithm is "bad" and doesn't terminate. But to the contrary, a loop invariant is usually just the thing that *proves* that the purported algorithm doesn't terminate — when as here the termination condition ( $a = 2$ ) contradicts the proposed loop invariant ( $a \neq 2$ ).

Of course, we have to prove that  $a \neq 2$  is a loop invariant for WHAT'S-WRONG, by verifying the three conditions in the loop invariant definition. Is  $a \neq 2$  true on

first entrance to the repeat-loop? Yes, because  $4 \neq 2$ . Is  $a \neq 2$  invariant from one loop entry to the next? Yes, for the following reason. Let  $A$  be the value of  $a$  when the loop is entered for the  $n$ th time; then the value of  $a$  when the loop is entered for the  $(n+1)$ st time is  $(A+2)/2$ . Thus we must show that  $[A \neq 2] \implies [(A+2)/2 \neq 2]$ . This is easily shown by adding 2 to both sides and dividing:

$$\begin{aligned} A &\neq 2 \\ A + 2 &\neq 4 \\ \frac{1}{2}(A + 2) &\neq 2. \end{aligned} \tag{3}$$

Condition 3 is satisfied because it's vacuously true: This condition says that something should happen *if* the loop terminates, but this loop doesn't terminate. By Conditions 1–2, the loop invariant is always true at entry, which is also the termination point, and so the termination condition is never met. ■

*Remark.* In computer science, you usually want to prove that things *do* terminate (and compute what you claim they compute!). In some sense, then, proofs of algorithms are “finite inductions” — you are not trying to prove something for an infinite number of iterations, as in WHAT’S-WRON. But the fundamental ideas of basis step and inductive step are the same. This connection between finite induction and algorithm verification is pursued in more depth in Section 7.4.

## Proofs for Iterative Algorithms without Loop Invariants

We prove again, but without using a loop invariant, that Algorithm EUCLID works, and then prove a bound on its worst case performance.

### EXAMPLE 4

Prove that EUCLID is correct without using a loop invariant.

**PROOF** We prove that EUCLID terminates and outputs  $\text{num} = \gcd(n, m)$ , that is, one argument will prove both claims, by proving by induction that the following proposition is true for all  $j$ .

$P(j)$ : Whenever the loop of EUCLID is entered with  $\text{num} = i$  and  $\text{denom} = j$ , then EUCLID terminates with output  $\text{num} = \gcd(i, j)$ .

Once we have proved this for all  $j$ , we have proved EUCLID correct because we can invoke  $P(n)$  for whatever  $n$  is input.

*Basis Step,  $P(0)$ .* In this case, the loop is immediately exited and the algorithm terminates by outputting  $\text{num} = i$ , which is  $\gcd(i, 0)$ .

*Inductive Step, by strong induction.* If  $j > 0$  and the loop is entered with  $\text{num} = i$ ,  $\text{denom} = j$ , a pass through the loop is then completed and the loop is re-entered with  $\text{num} = j$ ,  $\text{denom} = i - \lfloor i/j \rfloor j$ . Since now  $0 \leq \text{denom} < j$ , by the inductive hypothesis EUCLID terminates and outputs  $\gcd(j, i - \lfloor i/j \rfloor j)$ . By (2), this equals  $\gcd(i, j)$ .

Since we have proved a basis step and the inductive step, we have proved  $P(j)$  for all  $j \in N$ . ■

*Remark.* Notice that this proof is very similar to the proof in Section 2.4 of EUCLID-RECFUNC. Effectively, we have come up with a proof by thinking of EUCLID as if it were recursive instead of iterative (with each new iteration of the loop treated as if it were another function call in a recursive algorithm). Another way to think about the difference of this proof from the loop invariant proof is that here the inductive variable is one of the variables manipulated within the loop, whereas in the loop invariant method the inductive variable is the number of times the loop has been entered. Which variable within the algorithm is the best one to use for induction depends on the algorithm. For instance, perhaps the best way to prove BINSEARCH correct without using a loop invariant is to do induction on the length of the sublist that is still under consideration [10].

We now formalize the upper bound argument for the worst case of EUCLID from Section 1.5. Recall that we measured the algorithm's work by counting the number of divisions,  $quot \leftarrow [num/denom]$ , which is the same as the number of times EUCLID passes completely through its repeat-loop (as opposed to entering and immediately exiting, which it does when  $denom = 0$ .)

## EXAMPLE 5

Prove that with inputs  $m \in N$  and  $n \in N^+$ , Algorithm EUCLID performs division at most  $1 + 2 \log_2 n$  times. (We leave the case  $n = 0$  out of the theorem because  $\log 0$  is undefined. Besides, we know that in this case the algorithm enters its loop once and immediately terminates, without doing any divisions.)

**PROOF** Let  $P(j)$  be the statement: If the repeat-loop is entered with  $denom = j$ , then EUCLID will terminate after at most  $1 + 2 \log_2 j$  additional divisions. We prove  $P(j)$  for all  $j$  by strong induction starting with  $j = 1$ . Once we prove this, the theorem then follows from the case  $P(n)$ , where  $m, n$  are the inputs to EUCLID, because the repeat-loop is entered with this  $n$  immediately, so the  $1 + \log_2 n$  additional divisions are *all* the divisions.

*Basis Step.* If the loop is entered at any time with  $denom = 1$ , then, at the end of the resulting pass,  $denom = 0$  (the division had no remainder) and so the algorithm terminates at the start of the next pass. Therefore, there is 1 more division, and, sure enough,  $1 = 1 + \log_2 1$ .

*Inductive Step.* Assume  $denom = j > 1$  and that  $P(k)$  is true for  $1 \leq k < j$ . To prove  $P(j)$ , we assume we enter the loop with  $denom = j$ . By the argument in Example 3, Section 1.5, after at most 2 more divisions, the algorithm has either terminated or else we reenter the loop with  $denom = k \leq j/2$ . By  $P(k)$ , after that we will do at most  $1 + 2 \log_2 k$  more divisions. Therefore, from where we are now (with  $denom = j$ ), the remaining divisions will number at most

$$\begin{aligned} 2 + (1 + 2 \log_2 k) &\leq 2 + 1 + 2 \log_2(j/2) && [\log \text{ is an increasing function}] \\ &= 1 + 2(1 + \log_2(j/2)) && [\text{algebra}] \\ &= 1 + 2 \log_2 j, && [\log_2(x/2) = -1 + \log_2 x] \end{aligned}$$

proving  $P(j)$ . ■

This argument proves that EUCLID is an  $O(\log n)$  algorithm, but it doesn't prove that in the worst case it is  $\text{Ord}(\log n)$ , because the argument was by inequalities. In fact, it turns out that for  $n > 1$  there is no input pair  $(m, n)$  for which EUCLID takes exactly  $1 + 2 \log_2 n$  divisions. Nonetheless, the order is right. We won't show it, but the worst case is  $\text{Ord}(\log n)$ .

A similar argument to the one in the proof above can be applied to BINSEARCH. Based on the discussion in Example 5, Section 1.5, one can carefully prove that BINSEARCH takes  $\leq 1 + \log_2 n$  comparisons [20], that for this algorithm there is a sequence of values of  $n$  for which this bound is exact [21], and that there is even a formula that is exact for the worst case for all  $n \geq 1$  [22].

## Problems: Section 2.5

---

1. ⟨1⟩ Finish the proof of the Finite Descent Principle.

2. ⟨1⟩ Prove that if we start with  $u = 0$  and repeatedly add 1, eventually we can reach any positive integer. *Hint:* Let  $P(n)$  be the statement that we can reach  $n$ . (What you are asked to prove is obvious, but since incrementing a counter is done repeatedly in algorithms, the fact that any  $n$  can be reached is implicit in the proof that many loops terminate. For instance, see [4]. Thus it is worthwhile to appreciate that this fact boils down to an induction.)

3. ⟨1⟩ The invariant for the gcd Algorithm EUCLID was that  $\gcd(\text{num}, \text{denom}) = \gcd(m, n)$ .

- a) Find a place in the repeat-loop of EUCLID where the loop invariant becomes false (because of changes to  $\text{num}$  and/or  $\text{denom}$ ).

- b) Find the place later in the repeat-loop of EUCLID where the loop invariant becomes true again.

- c) Explain why this phenomenon — an invariant failing and then being restored — would not be uncommon.

4. ⟨2⟩ Use the conventions about for-loop counters to help you prove with a loop invariant that when the following algorithm fragment terminates,  $S$  is the sum of the first  $n$  positive integers.

```
 $S \leftarrow 0$ 
for  $k = 1$  to  $n$ 
   $S \leftarrow S + k$ 
endfor
```

5. ⟨2⟩ Let's sing 99 Bottles of Beer on the Wall. Assume we already have a procedure Beer( $k$ ) which correctly sings the verse that begins " $k$  bottles of beer on the wall". Prove that the following for-loop correctly sings the whole song, that is, it sings all 99 verses correctly, in order, and then stops.

```
for  $k = 99$  downto 1
  Beer( $k$ )
endfor
```

6. ⟨2⟩ Figure out what the following algorithm fragment computes, and prove you are right with a loop invariant.

```
 $M \leftarrow 0$ 
for  $k = n$  downto 1
   $M \leftarrow M - k$ 
endfor
```

7. ⟨1⟩ Does the loop invariant proof of Algorithm BINSEARCH (either the correctness part or the termination part) depend on the algorithm at each iteration checking a middle word? Could it check any word in the current list?

8. ⟨2⟩ Here is another non-loopinvariant way to prove EUCLID correct. Let  $Q(k)$  be the statement: If EUCLID terminates at the start of the  $k$ th pass through its repeat-loop, then its output is  $\gcd(m, n)$ .

- a) Prove  $Q(k)$  for all  $k \in N^+$  by induction.
- b) Do you need to prove anything else to prove EUCLID correct?

9. ⟨3⟩ Consider Algorithm EUCLID1 from [12, Section 1.1].

- a) Prove it correct by a loop invariant argument.
- b) Prove it correct by a different induction. As in the proof of EUCLID in Example 4, do induction on a variable manipulated within the loop, not on the number of times the loop has been entered.
10. (3) Prove BINSEARCH correct by strong induction on  $Q(k)$ : If the end of the repeat-loop of BINSEARCH is ever reached at a point where  $k = L - (F-1)$  (for the current values of  $L$  and  $F$ ), then the algorithm eventually terminates, outputting the index of  $w$  if  $w$  is on the sublist  $w_F, \dots, w_L$  and outputting ‘failure’ if  $w$  is not on this sublist. (Notice that once again, the non-loopinvariant method does not split the proof into two inductions, one for the invariant and the other for termination.)
11. (2) State a formula for the value of  $a$  at the end of the  $n$ th time through the loop of Algorithm WHAT’S-WRON. Prove it by induction.
12. (1) We have proved that, as a theoretical procedure, WHAT’S-WRON doesn’t terminate. What actually happens if you implement it on a real computer? Why?
13. (2) Let  $Q(n)$  be the claim “When WHAT’S-WRON is run, the loop is entered at least  $n$  times.” Prove that WHAT’S-WRON is not an algorithm by proving  $Q(n)$  for  $n > 0$  by induction.
14. (2) In the loop invariant proof for WHAT’S-WRON, display (3) is the inductive step of the induction proof that  $a \neq 2$  is really invariant. Redo this inductive step as an indirect proof. This is perhaps a more natural example than [1, Section 2.2] for an indirect induction proof, since arguing indirectly in this case allows you to work with equalities.
15. (3) Suppose we changed WHAT’S-WRON replacing

**repeat while**  $a \neq 2$

with

**repeat while**  $a \neq b$

and replacing

$$a \leftarrow (a+2)/2$$

with

$$a \leftarrow (a+b)/2,$$

where the user gets to input both the initial value

of  $a$  and the value of  $b$ . Now, when does the algorithm terminate? Prove your answer with a loop invariant.

16. (2) Prove that the following is not an algorithm:

**Algorithm** ERROR

```

 $a \leftarrow 4$ 
repeat while  $a \neq 2$ 
     $a \leftarrow \sqrt{a+2}$ 
endrepeat

```

17. (2) Prove by induction that, for all  $m, n \in N^+$ , EUCLID-REFUNC (p. 163) makes at most  $1 + 2\log_2 n$  calls to function gcf.
18. (3) Prove that EUCLID takes strictly fewer than  $1 + 2\log_2 n$  divisions to compute  $\gcd(m, n)$ . This proof requires pushing the argument in Example 3, Section 1.5 a little bit harder.
19. (3) Find and prove a bound on the worst case number of iterations of the repeat-loop for EUCLID1 from [12, Section 1.1].
20. (3) In this problem we get a bound on the worst case number of comparisons for BINSEARCH in terms of the number  $n$  of words on its list. Recall that by one comparison for BINSEARCH we meant in Example 5, Section 1.5, that the 3-way case-if statement in the repeat-loop is entered. The count is the same whether 1, 2 or all 3 lines of this case-if are read.
- a) Explain why, for each  $n$ , not having the search word on the list is always a worst case (maybe not the unique worst case).
- b) Explain why, if the search word is not on the current list of  $n$  words, then the sublist the algorithm will look at next has length either  $\lfloor (n-1)/2 \rfloor$  or  $\lceil (n-1)/2 \rceil$ .
- c) Come up with a recurrence for  $C_n$ , the worst case number of comparisons.
- d) Use this recurrence and strong induction to prove that  $C_n \leq 1 + \log_2 n$ .
- Caution:* Since your recurrence uses two previous cases, you will need two basis cases.
21. (3) Find an infinite sequence of values of  $n$  for which, if the search word is not on the list, the bound in [20] is exact. Prove that you are right.
22. (4) Find an exact formula for  $C_n$ , the worst case number of comparisons for BINSEARCH on a list of  $n$  words. Prove that you are right. If you don’t

already have a conjecture from [25, Section 1.5], then [20–21] above should help. You should expect logs and floors or ceilings to appear. Be prepared for heavy use of the algebraic properties of these functions.

23. ⟨3⟩ The iterative Algorithm SEQSEARCH from Example 4, Section 1.5 finds whether  $w$  is on a list by simply marching straight through it. Prove the algorithm correct by a loop invariant.

24. ⟨2⟩ Consider Algorithm 1.12, MAXNUMBER, on p. 119.

- a) Prove it correct by a loop invariant.
- b) Rewrite it recursively, and prove this version correct by induction.

25. ⟨2⟩ We want to take a piece of chocolate (or peanut brittle) and break it into  $n$  pieces. We do this by repeatedly taking a piece and snapping it in two. Prove two ways, as described below, that it will take  $n - 1$  snaps.

- a) Do induction on  $P(k)$ : To break a piece of chocolate into  $k$  pieces takes  $k - 1$  snaps.
- b) Do induction on this loop invariant: Whenever a piece of chocolate has been snapped into pieces a number of times, there is always one more piece than snaps (i.e.,  $j$  snaps gives  $j + 1$  pieces).

The point of this problem is: the two ways of using induction to prove an iterative algorithm valid can both apply to situations that are not immediately thought of as algorithms (e.g., sharing chocolate).

26. ⟨3⟩ An urn contains 75 white balls and 150 black balls. Repeatedly, you pick out 2 balls at random and do the following:

- i) If at least one is black, throw out a black and put the other ball back in.
- ii) If both are white, throw them both out and put in a new black ball.

It turns out that, when exactly one ball is left in the urn, its color is always the same. What color is it? Why? *Hint*: loop invariants.

27. ⟨3⟩ Consider the following divide-and-conquer algorithm for finding the maximum of  $n$  numbers.

Divide the set of numbers in half as nearly as possible, find the maximum of each subset by recursion, and then compare the two maximum values.

- a) Prove this algorithm correct.
- b) Find a recurrence for  $C_n$ , the number of comparisons of inputs this algorithm requires when given  $n$  numbers.
- c) Discover an explicit formula for  $C_n$  and prove it by induction.

### Algorithm 2.6. ITERATIVE-TOH

**Input** all rings on one pole

**Output** all rings on another pole

**Algorithm** ITERATIVE-TOH

Move smallest ring clockwise

**repeat until** all rings are on one pole

    Move 2nd smallest exposed ring to the one  
        place it can go

    Move smallest ring clockwise

**endrepeat**

28. ⟨4⟩ If we put the three poles of Towers of Hanoi on the circumference of a circle, we can describe which pole to move to next by saying “move clockwise” or “move counterclockwise”. Algorithm 2.6, ITERATIVE-TOH, makes use of this circular setup. Not only is it iterative, but it is easy for a person to act out, say, at parties. The reason is that it requires almost no memory, whereas the recursive algorithm HANOI requires keeping track of where you are in a lengthy series of procedure calls.

While performing ITERATIVE-TOH is simple, proving it correct is not. Try to give a complete proof.

29. ⟨4⟩ Find and prove the explicit solution to

$$L(n) = 1 + \min_{n/2 \leq k < n} \{L(k)\}, \quad L(1) = 0.$$

This recurrence arises in finding a lower bound for the number of multiplications needed to compute  $x^n$ .

30. ⟨4⟩ Find the solution to

$$L(n) = 1 + \max_{n/2 \geq k \geq 1} \{L(k)\}, \quad L(1) = 0.$$

## 2.6 How to Conjecture What to Prove

---

So far, we've emphasized *how* to prove things by induction. But how do you conjecture *what* to prove?

Unfortunately, there is no sure-fire answer. There are some classic works on the issue, for instance, Polya's film "Let Us Teach Guessing" and his books on problem solving, such as *How to Solve It*. Books on problem solving are relevant because the first step in problem solving must be to formulate the problem, that is, to make a conjecture. In this regard, another book, *The Art of Problem Posing* by Brown and Walter, is also relevant. In any event, the art of conjecturing is learned over time by experience. The best we can do, and these references can do, is amass a good variety of examples in order to "speed up" your experience.

Fortunately, in this section we have a limited task: We are trying to teach conjecturing only for results that can be proved by induction, that is, for results that naturally break into cases  $n = 1, 2, \dots$ . We won't have to address such hard issues as "How do you make up conjectures about the distribution of primes or the properties of derivatives?"

Still, while conjecturing claims to be proved by induction may be a limited task, that doesn't mean it is necessarily easy. The easier instances are where there is just one thing to look at for each value of  $n$ . For instance, suppose we want to find a formula for a sequence, say the sequence  $\{S_n\}$ , where  $S_n = \sum_{k=1}^n k$ . Then there is only one number to look at for each  $n$  and a pattern might emerge fairly quickly using the "try some cases" part of the inductive paradigm. Or, when we can find a formula relating  $S_{n+1}$  to  $S_n$ , the recursive paradigm may suggest the solution.

Often there are many things to look at for each  $n$ . For instance, for the TOH clutter problem [4, Section 2.4], for each  $n$  there are many different clutters to consider with  $n$  rings. Or for the tournament ranking problem, for each  $n$  there are many different round-robin tournaments with  $n$  teams. For such problems trying cases to seek a pattern rapidly gets out of hand. However, the recursive paradigm as well as some other tools can help.

One more initial point. Are we talking about conjecturing or about guessing? Actually, to mathematicians these words mean about the same thing. Mathematicians never do wild guessing. Their guessing is *directed*, based on carefully chosen special cases, or particular ways to organize the computations, or analogies, or visual images, or at least some sort of intuition. Furthermore, guesses are immediately tested and modified by the additional data that the testing suggests. In the examples that follow, look to see how the methods exemplify these claims.

Some of the methods to follow may be completely new to you. That's good; it means you are learning something completely new. Once you see a method enough, and use it yourself in problems, you will have internalized it, and then you have become a more powerful problem solver.

**EXAMPLE 1**

Conjecture a closed form formula for  $S_n = \sum_{k=1}^n k$ . (This was our first example of an induction proof in Section 2.1, where we seemingly just pulled the answer out of a hat, so it will be nice to see that this wasn't just legerdemain.)

**Solution** We are asking for a formula for the sequence

$$1, \quad 3 = 1+2, \quad 6 = 1+2+3, \quad 10, \quad 15, \quad 21, \quad \dots \quad (1)$$

Maybe you can just see a formula from the data, but most people can't. There are various ways to look for a pattern in this data. We present four here, all of them useful in other situations.

*Inductive Paradigm approach.* Well, if we're going to look for a pattern in this data, what kind of pattern? A good general rule in mathematics is to look first for simple patterns and, only if this fails, look for more complex ones. What is the simplest pattern (i.e., function) that we could try to fit to the data above? Without wishing to get into any deep discussion of what "simplest" means, we claim that a polynomial function is about as simple as we can get. Anyhow let's try this.

---

**TABLE 2.1**  
**Sum of the First  $n$  Integers**

---

$n$	$S_n = \sum_{i=1}^n i$	$n^2$	$n^2/2$	$S_n - (n^2/2)$
1	1	1	$\frac{1}{2}$	$\frac{1}{2}$
2	3	4	2	1
3	6	9	$4\frac{1}{2}$	$1\frac{1}{2}$
4	10	16	8	2
5	15	25	$12\frac{1}{2}$	$2\frac{1}{2}$
6	21	36	18	3

---

Look first at the left-most two columns of Table 2.1. These just list the data from (1). Now the simplest polynomial function (other than a constant) is a linear polynomial ( $an + b$ ). But the second column should make you doubt that the sum of the first  $n$  integers is growing linearly in  $n$ . Well, how about a quadratic ( $an^2 + bn + c$ )? If it is a quadratic, the  $n^2$  term will eventually dwarf the other terms, so we can try to ignore  $bn + c$  at first and just ask if  $S_n$  is roughly  $an^2$  for some  $a$ , that is, if  $S_n$  and  $n^2$  look roughly proportional as  $n$  increases. Comparing column 3 with column 2, this looks more hopeful. At least the numbers in both columns appear to be growing similarly. A good way to test this hypothesis is to compute the ratio of successive values in each column. For the  $S_n$  column these ratios are, respectively,  $3, 2, 1\frac{2}{3}, 1\frac{1}{2}, 1\frac{2}{5}$ , and for the  $n^2$  column they are  $4, 2\frac{1}{4}, 1\frac{7}{9}, 1\frac{9}{16}, 1\frac{11}{25}$ . Promising, but,

if correct, we need a value for  $a$ , which surely isn't 1. Again opting for simplicity, let's try  $a = \frac{1}{2}$  as in column 4. Much better! If this is correct, then the difference  $S_n - n$  must be  $bn + c$  (why?). Indeed, taking the difference of columns 2 and 4, as shown in column 5, and comparing this with column 1, it's clear that  $b = \frac{1}{2}$  (and that  $c = 0$ ). So using the inductive paradigm we conjecture that the sum of the first  $n$  integers is  $n^2/2 + n/2 = n(n+1)/2$ , the conjecture we proved correct in Section 2.1 without ever saying where it came from.

*Recursive approach.* We know that

$$S_n = S_{n-1} + n; \tag{2}$$

why? Now, if instead  $S_n$  increased by the same amount each time (i.e., if  $S_n$  equaled  $S_{n-1} + c$ ), then  $S_n$  would have to be the linear function  $S_n = cn + S_0$ . Since, as we noted using the inductive paradigm approach,  $S_n$  is instead growing by an increasing amount,  $S_n$  has to grow faster than linear. As above, the next simplest thing is quadratic. So we guess

$$S_n = an^2 + bn + c, \tag{3}$$

and then proceed as before to get  $S_n = n^2/2 + n/2$ . Remember that this formula is still a guess because the formula for  $S_n$  doesn't *have* to be the next simplest thing (i.e., a quadratic). But that's what we should guess — both because it is the next simplest form to test and because things in mathematics *ought* to work out simply, and often do!

*Geometric approach.* A useful approach for analyzing some sequences is to figure out how to represent the terms as geometric shapes. If we represent the number  $n$  by  $n$  dots in a row, then the fact that  $S_n = S_{n-1} + n$  means that the shape for  $S_n$  should have one more row of dots than the shape for  $S_{n-1}$ , and each time we go from one shape to the next the new row gets one dot longer. If we line the rows up side by side, we get Figure 2.13. Aha! Similar triangles. The number of dots seems proportional to the area and the area grows as the square of the base, in fact  $1/2$  the square of the base (since base equals height for these triangles). Hence we strongly suspect  $S_n$  is  $\frac{1}{2}n^2$  plus some correction term (since the triangle argument is not exact). It's reasonable to assume the correction is a lower order polynomial. In short, we have now conjectured on geometrical evidence that  $S_n$  has the form in (3), probably with  $a = \frac{1}{2}$ . To determine  $a, b, c$ , we could proceed as in Table 2.1 again, but we would like to introduce an algebraic method. Namely, simply set  $n = 1, 2, 3$  in Eq. (3) using the known values  $S_1, S_2, S_3$  and solve the resulting three linear equations [1]. You will get  $a = b = \frac{1}{2}$  and  $c = 0$ , so once again  $S_n = \frac{1}{2}n^2 + \frac{1}{2}n + 0 = \frac{1}{2}n(n+1)$ . And once again this doesn't *prove* anything yet (because we have assumed without proof that Eq. (3) holds for all  $n$ , not just the values  $n = 1, 2, 3$  for which we have forced it to work by our choice of  $a, b, c$ ). However, we *have* conjectured what to prove.

*The Method of Undetermined Coefficients.* All three approaches above arrive at some point at the assumption that  $S_n$  is a quadratic polynomial. Fairly obvious question: Why not just assume right away a polynomial form for  $S_n$ ? Of course,

## FIGURE 2.13

The terms of  $\sum_{k=1}^n k$

$$S_k = \sum_{k=1}^n k$$

represented as similar triangles.



then we wouldn't know what degree polynomial to choose except that surely a constant polynomial would not be correct (why?). So we might try  $S_n = an + b$ . Using the algebraic method introduced in the previous paragraph, substitute the known values  $S_1 = 1$  and  $S_2 = 3$  into  $S_n = an + b$  and solve the resulting two equations for  $a$  and  $b$ , obtaining [3]  $a = 2, b = -1$ . Can  $S_n = 2n - 1$  be correct? Well, for  $n = 3$  it says that  $S_3 = 5$ . Oops — no good since we know that  $S_3 = 6$ .

Since  $S_n$ , therefore, isn't linear, let's try a quadratic:  $S_n = an^2 + bn + c$ . Then setting  $n = 1, 2, 3$ , we would solve the resulting three equations for the three unknowns  $a, b, c$ , as we already did in the geometric method, obtaining  $a = 1/2, b = 1/2, c = 0$ . Moreover, we can check easily enough that the formula obtained this way for  $S_n$  does equal  $\sum_{k=1}^n k$  for  $n = 4, 5$  up to any value we wish. That it works for any such finite set of values doesn't, of course, *prove* anything. As usual the way to prove correctness is by induction.

The method we have just used is called the **Method of Undetermined Coefficients**. If the degree of the polynomial we are seeking was, say, 4 or 5 or greater, solving the successive sets of equations for  $n = 1, 2, \dots$  until we found the one that works would become very tedious by hand. Luckily in practice you don't have to do this. In Section 5.11, we show an easier method to find the correct degree of the polynomial — indeed, to find a formula for the polynomial — if the function you are looking for is a polynomial. (See Theorem 6 there.) Actually, whatever method you use, a CAS can remove the tedium. ■

## EXAMPLE 2

Find a closed form for  $P_n = \prod_{k=2}^n \frac{k^2 - 1}{k^2}$ , for  $n = 2, 3, 4, \dots$

**Solution** If you do the calculations for  $n = 2, 3, 4, \dots$ , canceling common factors in the normal way, you get that the sequence  $P_2, P_3, \dots$  is

$$\frac{3}{4}, \frac{2}{3}, \frac{5}{8}, \frac{3}{5}, \frac{7}{12}, \frac{4}{7}, \frac{9}{16}, \frac{5}{9}, \dots$$

This is most peculiar; there is a clear pattern for every other term (see it?), but not for the whole sequence. Do we really need two separate formulas? One would not expect so — again one should first look for something simple. Do you see what to do? The fact that every other denominator is smaller suggests we have canceled out too much. Double numerator and denominator in every other fraction and we get

$$\frac{3}{4}, \frac{4}{6}, \frac{5}{8}, \frac{6}{10}, \frac{7}{12}, \frac{8}{14}, \dots$$

Sweep your eyes from left to right across the numerators. Now sweep across the

denominators. Remember that the first term listed,  $3/4$ , is  $P_2$ , not  $P_1$ . It seems that  $P_n = (n+1)/2n$ , and this is easy to prove by induction [11, Section 2.2].

*Moral:* Look first for a single pattern. What usually counts as simplified form may not be the right form for seeing the pattern. ■

### EXAMPLE 3

Find a closed form for the  $n$ th term of the sequence defined by

$$a_1 = a_2 = 1, \quad a_n = a_{n-1} + 2a_{n-2} \quad (n \geq 3).$$

**Solution** Again, begin by computing the first several terms. They are 1, 1, 3, 5, 11, 21, 43. If you just look at the individual numbers themselves, they probably won't look familiar, but if you look at the relationship between each term and the next, something should jump out. It seems that  $a_n \approx 2a_{n-1}$ . This suggests that the sequence should be approximately  $c2^n$  for some  $c$ . But, if so, how do we find  $c$ ? If  $a_n \approx c2^n$ , then  $a_n/2^n \approx c$ . Therefore, let's compute the ratios  $a_n/2^n$ . For  $n = 1$  to 7 they are

$$.5, \quad .25, \quad .375, \quad .3125, \quad .3438, \quad .3281, \quad .3359,$$

where we have written the ratios as decimals instead of fractions because decimals are easier to compare (and we have rounded to 4 places). Indeed, believing that a simple problem like this ought to have a simple pattern, we guess that these ratios have a limit,  $1/3$ . So now we believe that  $a_n \approx 2^n/3$ . But we want the exact value of  $a_n$ . So as in the right-most column of Table 2.1, consider a “residual” sequence, this time  $d_n = a_n - (2^n/3)$ . The first several terms are

$$d_1 = \frac{1}{3}, \quad d_2 = -\frac{1}{3}, \quad d_3 = \frac{1}{3}, \quad d_4 = -\frac{1}{3}.$$

We don't have to go any further. Our conjecture is

$$a_n = 2^n/3 + (-1)^{n+1}/3 = \frac{2^n + (-1)^{n+1}}{3}.$$

This conjecture is correct, as you can prove by induction [10]. The correction term involves  $(-1)^{n+1}$ , not  $(-1)^n$ , because the *even-indexed* terms are the ones where  $d_k$  is negative. ( $n - 1$  would serve just as well as  $n + 1$ .)

*Moral:* Look at the relationship between successive values of a sequence, not just at the individual computed values. (This is a key point in the recursive paradigm, but it is also important when, as in this example, we follow the inductive paradigm and begin by computing values. ■

A key step in the last example was looking for a limit to  $a_n/b_n$ , where  $\{a_n\}$  is the sequence for which we desire a formula and  $\{b_n\}$  is some known sequence, in this case  $\{2^n\}$ . In fact, there is a systematic exploratory procedure, based on such ratio analysis, that works for finding a formula for most sequences involving exponentials. We outline that method in [26].

Why not give this method more prominence? First, because in Chapter 5 we will also show you how to find formulas for the same sequences without any exploration at all. (This better method also proves the formula correct, so you don't need to follow your discovery with an induction.)

Second, there is now a lovely online resource to help guess formulas for any sequence, so long as the terms are integers: The Online Encyclopedia of Integer Sequences, at [www.research.att.com/~njas/sequences/](http://www.research.att.com/~njas/sequences/). At that site, you type in the first few numbers in your sequence and you get back any sequences in the database that start that way, or contain that subsequence — along with formulas, a brief description of the problems that lead to these sequences, and references to the literature.

Try it. But beware. This is a research tool, listing sequences of interest to experts. We just typed in 1, 2, 4, 8, 16; we got back 30 different sequences, *none* of which was  $\{2^n\}$ . So, if there are several matches with your sequence, and none of the descriptions given are for your exact problem, then you will need to compute more terms or try to show that one of the problems described is equivalent to yours. In any event, the Encyclopedia will have given you fresh leads for investigating your sequence.

## EXAMPLE 4

Suppose someone handed you Algorithm EUCLID-RECFUNC (p. 96) without any explanation or hints and asked you to figure out what it does. How would you do it?

**Solution** The thing absolutely *not* to do is to randomly select a lot of pairs  $m, n$  and compute the output. It is highly unlikely you will see any pattern. Instead, this problem illustrates the value of directed exploration.

First look at the algorithm itself. It hinges on a recursive function gcdf for which  $\text{gcd}(i, j) = \text{gcd}(j, i - j[i/j])$ . If you recognize that  $i - j[i/j]$  is the remainder when  $j$  divides  $i$ , you have a head start on understanding this algorithm. You'll be able to compute your data with the algorithm more quickly, and you'll already be asking yourself “What do remainders have to do with the outputs I am getting?”

Now, vary just one of the variables at a time. But which one? There is no obvious answer but  $j$  looks a little more important in the algorithm so we'll try it. So first let  $j = 0$  and vary  $i$ . You rapidly see that  $\text{gcd}(i, 0) = i$ , but what does this mean? Keep going while you keep that question open. Now fix  $j = 1$ . Probably you will see with your first choice of  $i$  that  $\text{gcd}(i, 1) = i$  for all  $i$ . Again, what does that mean? Next try  $j = 2$  and again vary  $i$ . After just the first few values of  $i$  you see that  $\text{gcd}(i, 2)$  is 1 or 2, depending on whether  $i$  is odd or even. Now try  $j = 3$  with  $i = 1, 2, 3, 4, \dots$ . Probably you will soon see that the output is 1 if  $3 \nmid i$  and 3 otherwise. At this point you might suspect that gcdf has something to do with common factors. Then you might go back to remainders and ask what remainders have to do with common factors. Eventually you will see that remainders preserve common factors, and so you will be well on your way to both determining what EUCLID-RECFUNC does *and* why it works.

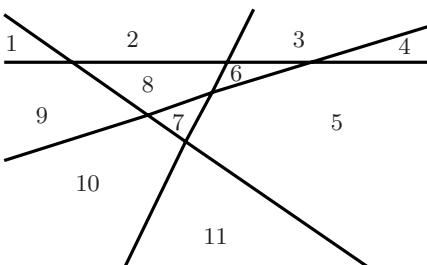
Each person will need a different amount of exploring before figuring out what this (or any) algorithm is doing. But a systematic approach will benefit anyone. ■

But would anyone ever really hand you an algorithm and make you figure out its purpose? Yes, this is not so uncommon! When maintaining or enhancing a piece

of software written by someone else (not available for questioning) and not well-documented, you often have to figure out what portions of the software are doing, i.e., what algorithm the software implements. On other occasions, the person who makes you figure out an algorithm may be *yourself*. You are given a problem and think you have an algorithm to solve it. You run the algorithm and it doesn't do what you expected. Then you need to figure out what your own algorithm is actually doing in order to know how to change it.

## EXAMPLE 5

Into how many regions do  $n$  straight lines divide the plane?



**FIGURE 2.14**

A plane divided by four straight lines.

**Solution** Figure 2.14 shows four straight lines dividing the plane into 11 regions. You should imagine all the lines going off to infinity in both directions.

Collect some data. When  $n = 1$  there are 2 regions. When  $n = 2$  there are already complications. If the lines intersect there are 4 regions, but if they are parallel there are only 3. Hmm, the answer does not depend on  $n$  alone. Does this indeterminacy get worse as  $n$  increases? Well, what can happen with 3 lines? None, two or three can be parallel, with different numbers of regions in each case. And now there is another issue. If all three lines are **concurrent** (intersect in a common point), then there are 6 regions, but if each pair intersects at a different point there are 7 regions. Check out what we've said by drawing all these cases yourself.

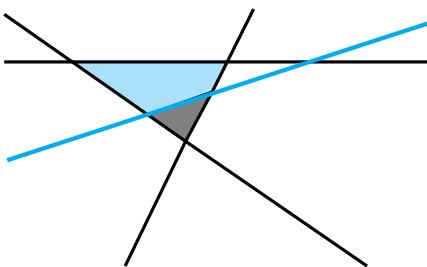
Clearly the problem is getting out of hand, and so far  $n$  is only 3. To make progress, we should (at least temporarily) *restrict* the question. If we threw down the lines on the plane like pick-up sticks, parallel lines and concurrent lines would be rare exceptions. Therefore, a set of lines is said to be in **general position** if no two are parallel and no three are concurrent. If we restrict ourselves to  $n$  lines in general position, will the number  $r_n$  of regions depend only on  $n$ ? Draw some more cases with  $n = 3$  and 4 to convince yourself that the answer seems to be Yes [14].

Let's first try the recursive approach. So suppose we already know  $r_{n-1}$  (including knowing that it is independent of the exact position of the lines, so long as they are in general position). How can we figure out  $r_n$  (including showing that it depends on  $n$  alone)? Well, imagine someone gives us an arbitrary  $n$ -line configuration. We need to see an  $(n-1)$ -line configuration in it, since we are assuming we know all about those. Temporarily color one of the  $n$  lines invisible. The remaining lines are still in general position, so by assumption there are  $r_{n-1}$  visible regions.

Now make the  $n$ th line visible again, that is, imagine redrawing it from one side of the picture to the other, splitting visible regions as it is redrawn. Since it crosses each of the  $n - 1$  previous lines separately (it is parallel to none, and crosses no two simultaneously), it necessarily passes through exactly  $n$  previous regions, dividing each of them in two. (See Fig. 2.15, where  $n = 4$  and the fourth line is in color.) Hence  $r_n$  is also uniquely determined and

$$r_n = r_{n-1} + n. \quad (4)$$

But wait, this is exactly the recurrence for the sum of the first  $n$  positive integers, except that  $r_1 = 2$  instead of  $S_1 = 1$ . That is, the starting values of the two sequences differ by 1, but the additions to get later terms are all the same. Surely, then,  $r_n = S_n + 1 = \frac{1}{2}(n+1)n + 1$ . We can now confirm this with a proof by induction, in fact two different ways [15].



**FIGURE 2.15**

Putting back the  $n$ th line.

This technique of deleting and reinserting the  $n$ th piece in a configuration is very powerful. It is really just an instance of the recursive paradigm.

Now we do this example again using the inductive paradigm. We noted above that  $r_n = 2, 4, 7, 11$  for  $n = 1, 2, 3, 4$ , respectively. It's not that hard to draw the picture for  $n = 5$  which shows that there are 16 regions in this case. Does this sequence — 2, 4, 7, 11, 16 — suggest anything? Perhaps not. But if you compare it with the data in the second column of Table 2.1, you should see immediately that each datum for  $n$  regions is one greater than the corresponding datum for the sum of the first  $n$  integers. So it would be reasonable to conjecture that  $r_n = n^2/2+n/2+1$ . Now how would you prove this by induction? Well, as above in the recursive approach, by either of the two ways in [15]. ■

*Moral:* When looking for a pattern in a sequence, always think about similar but not identical sequences for which you know the answer.

## EXAMPLE 6

Circles are drawn in the plane so that each pair intersect (thus in 2 points) and no three intersect at the same point. If there are  $n$  circles, into how many regions do they divide the plane? For instance, when there is just one circle, there are 2 regions, the inside and the outside.

**Solution** First, we admit that it's hardly obvious that you *can* draw  $n$  mutually intersecting circles for every  $n$ , although it is true. Still, it is not hard to draw pictures for  $n = 1, 2, 3$ , and the number of regions are, respectively, 2, 4, and 8. It's

pretty obvious what to guess next: 16, and in general (assuming the configurations exist)  $r'_n = 2^n$ . That is, we are once again asserting that the number of regions depends only on  $n$ , and this time by a very simple formula. Note also the use of  $r'_n$  instead of  $r_n$ , which we just used for another count.

Should we gather more data or should we go right to trying to prove our conjecture? Since it's a little tedious to actually draw 4 mutually intersecting circles and count the regions, and increasingly difficult to draw examples for  $n > 4$ , let's go right to trying an inductive proof. We know the basis is correct, 2 regions when  $n = 1$ . For the inductive step, we need to show that the number of regions doubles with each new circle.

This leads us to a key question: How are new regions formed out of old when a new circle is added (or, if you like, *if* a new circle can be added)? In other words, what's the recurrence relating  $r'_n$  to  $r'_{n-1}$ ? So by analogy to the argument in Example 5, imagine we have  $n$  circles on the plane meeting the conditions, and we temporarily color one of them invisible. (See Fig. 2.16, where  $n = 3$  and the third circle is in color.) By assumption there are  $r'_{n-1}$  visible regions. Now make circle  $n$  visible again, starting, say, at point  $p$  and coloring the circle clockwise. Each time you reach another intersection point, you have finished cutting another old region in two. The last time you finish such a cut is when you return to  $p$ . Now, since the  $n$ th circle intersects every other circle twice, and all these pairs of points are distinct, we count  $2(n-1)$  points in the clockwise traversal. Therefore we have split  $2(n-1)$  old regions and

$$r'_n = r'_{n-1} + 2(n-1). \tag{5}$$

With this recurrence, it is easy to compute  $r'_4$ :

$$r'_4 = r'_3 + 6 = 8 + 6 = 14 \neq 16 (!)$$

So our conjecture was false:  $2^n$  is the wrong sequence.

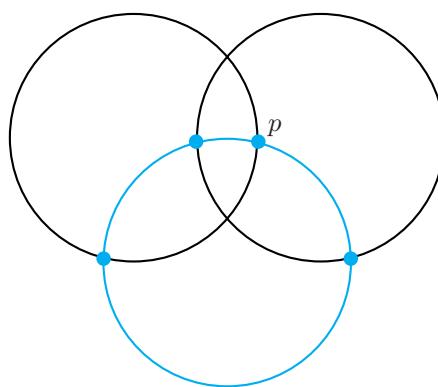
*Moral:* Don't jump to conclusions. Recurrences are more useful for finding patterns than just guessing from a few numbers. If we had turned to developing a recurrence right at the start of this problem, we would have never come up with the wrong conjecture that  $r'_n = 2^n$ .

We still haven't found the right formula for  $r'_n$ . But recurrence (5) is almost the same as (2) and (4). This analogy should be enough for you to figure out a closed formula. In fact, in [22] we will show two approaches, just as for Example 5. ■

## EXAMPLE 7

How might one be led to discover the statement of Theorem 2, Section 2.2, that every round-robin tournament has an ordering in which each team beat the next? As noted at the start of this section, even laying out the data for small  $n$  is difficult. For each value of  $n > 2$  there are several different tournaments, and the number grows rapidly.

**Solution** One way to make such discoveries is to make *some* conjecture, even with the flimsiest evidence, and try to prove it. (In effect, we did this in Example 6, but here we are suggesting doing it without first collecting any data at all!) For



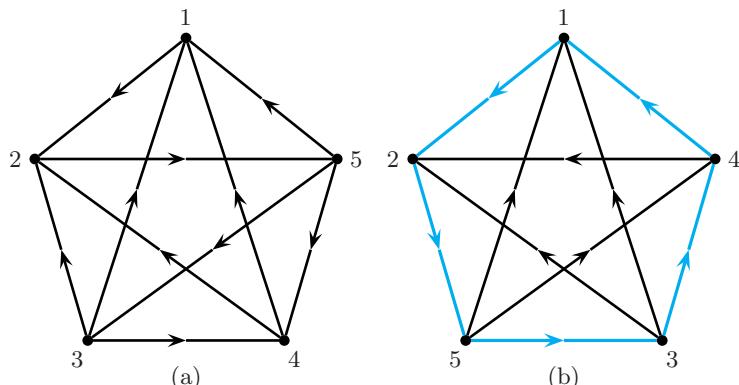
**FIGURE 2.16**

Putting back the  $n$ th circle.

instance, if we are interested in rankings, it is natural to *want* an ordering in which  $t_i$  beats  $t_{i+1}$  for each  $i$ . So we say: “I’ll try to prove this. If I succeed, I have a surprising result; if I don’t, the place where my attempt breaks down may suggest a more restricted situation where the ordering I want exists, or a weaker ordering result that is always true.” In other words, when it is hard to make conjectures from data, mathematical induction (or other forms of deduction) may be used as a *discovery technique*.

Specifically, when we try to prove the ranking theorem, we very quickly get to “Suppose that I already knew it was true for  $n$ -tournaments”. As soon as we say those magic words, we get Fig. 2.5, Section 2.2 and the proof follows. But if we hadn’t dared to try a proof, we would never have said the magic words.

Is this daredevil approach useful when it fails? Watch and see. Look at Fig. 2.17a, which repeats Fig. 2.4 from Section 2.2. We pointed out there that in the order 5, 3, 4, 1, 2, each team beat the team to its right. But more is true: Team 2 also beat team 5. In other words, if we wrap the ordering 5, 3, 4, 1, 2 counterclockwise around a circle, as in Fig. 2.17b, each team beats the team directly counterclockwise from it. So maybe we should have conjectured more: that every tournament has such a “circular order”.



**FIGURE 2.17**

A tournament and a circular ranking for it.

Rather than draw more tournaments, we jump right in and try to prove this conjecture. For each  $n$  let  $P(n)$  be: Every  $n$ -tournament has a circular order. Right away the basis case,  $P(2)$ , shows that this conjecture is false — the unique 2-tournament has no cycles because there is only one game.

All right, our conjecture is wrong. But maybe it's wrong only for small tournaments. Maybe it's true for all tournaments with at least some number  $n_0$  of teams. Let's turn to the inductive step, where the basis value  $n_0$  is irrelevant.

So suppose that  $P(n)$  is true for some  $n$ ; will  $P(n+1)$  be true? Let's see. Consider any  $(n+1)$ -tournament. Deleting any one team  $t^*$ , we are left with an  $n$ -tournament, so by hypothesis there is a circular order  $t_1, t_2, \dots, t_n$ . Can  $t^*$  be inserted into this circle? Yes, if and only if there is some  $i$  such that  $t_i$  beat  $t^*$  and  $t^*$  beat  $t_{i+1}$ . (When  $i = n$ ,  $i+1$  means 1.) Well, so long as  $t^*$  beat some teams and lost to others, there must be such a  $t_i$  and we would be done. But what if  $t^*$  lost to *all* other teams, or beat all others? We could certainly make up such a tournament, so this attempted inductive step is invalid.

Moreover, this reasoning shows that the conjecture is false for every  $n$ . Whenever some team wins all its games or loses all its games, there cannot be a circular order involving that team, for such an order requires it to both win and lose.

But we have hardly reached a dead end. The inductive step broke down because of a special case. If  $n$  is large, you wouldn't expect to find a team which loses all its games or wins them all. In other words, maybe almost all  $n$ -tournaments have a circular order, or maybe all those without "total winners" or "total losers" do. In short, there is much more to look into. We won't pursue this (though some of it is addressed in [6, Section 2.8]), because our purpose is not to understand tournaments themselves but rather to convince you that trying to prove things is a good discovery method. ■

Let us summarize the message of this section. Mathematicians use all sorts of methods to develop conjectures. They are not shy about experimenting, collecting data, drawing pictures, and looking for analogies. However, mathematicians are also not shy about trying general reasoning — the development of recurrences and attempts at proofs — before they have much evidence. You should be equally flexible.

## Problems: Section 2.6

1.  $\langle 2 \rangle$  Finish the geometric approach to Example 1 by finding  $a, b, c$  that make the conjecture  $\sum_{k=1}^n k = an^2 + bn + c$  correct for  $n = 1, 2, 3$ . For instance, when  $n = 1$  this conjecture becomes

$$1 = a1^2 + b1 + c = a + b + c.$$

Substituting  $n = 2$  and 3 as well gives you three linear equations in three unknowns. Solve them.

2.  $\langle 2 \rangle$  In Example 1, geometric approach, suppose we had more faith in our initial belief that  $a = \frac{1}{2}$ . Then we need only find  $b, c$  and thus need only match two values of  $S_n$ , say for  $n = 1, 2$ . This is much less work and leads to the same formula (since our belief in  $a$  is correct).

a) Do this work.

b) But suppose our belief about  $a$  had been

wrong; say we believed  $a = 1$ . We could still solve for  $b, c$  on that assumption. If you do so, you get the conjecture that  $S_n = n^2 - n + 1$ . Now we would try to prove it. How would we find out that we are wrong?

3. ⟨1⟩ Show that if a formula for  $S_n = \sum_{k=1}^n k$  were  $S_n = an+b$ , then the known values  $S_1 = 1, S_2 = 3$  would imply by algebra that  $a = 2, b = -1$ .

4. ⟨2⟩ Discover and prove formulas for the following sums:

- $\sum_{k=1}^n k^2$
- $\sum_{k=1}^n k(k+1)$
- $\sum_{k=1}^n \frac{1}{k(k+1)}$

5. ⟨2⟩ Find and prove a formula for

$$\frac{1}{1 \cdot 4} + \frac{1}{4 \cdot 7} + \frac{1}{7 \cdot 10} + \cdots + \frac{1}{(3n-2)(3n+1)}.$$

6. ⟨2⟩ Find and prove a formula for

$$1 \cdot 1! + 2 \cdot 2! + 3 \cdot 3! + \cdots + n \cdot n!$$

7. ⟨3⟩ Discover and prove formulas for the following products:

- $\prod_{k=1}^n \left(1 - \frac{4}{(2k-1)^2}\right)$
- $\prod_{k=2}^n \frac{k^2+k}{k^2+k-2}.$

8. ⟨3⟩ In addition to using geometric figures to help solve problems given algebraically as in Example 1, we can generate problems directly from geometric figures. Fig. 2.18 shows the first three **hexagonal numbers**. For instance,  $h_2 = 7$  because there are 7 dots in the second figure. Find a formula for the hexagonal numbers.

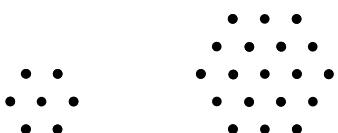


FIGURE 2.18

The hexagonal numbers.

9. ⟨2⟩ When humans play TOH, the first hard part comes right at the beginning: where to move the first ring, to the goal pole or the spare pole? Experiment with different numbers of rings until you

have a conjecture about the right choice (assuming you want to win in the minimum number of steps). Prove your conjecture by induction.

10. ⟨2⟩ Finish Example 3 by using induction to prove: If the sequence  $\{a_n\}$  is defined by  $a_1 = a_2 = 1$  and by  $a_n = a_{n-1} + 2a_{n-2}$  for  $n \geq 3$ , then  $a_n = \frac{1}{3}[2^n + (-1)^{n+1}]$ .

11. ⟨2⟩ Change the initial conditions in Example 3 to  $a_1 = 2, a_2 = 1$ . Now determine a formula for  $a_n$ .

12. ⟨3⟩ What does Algorithm MYSTERY below do? Prove it.

```
Input n, m           [Integers with n ≥ 0]
Output F(n, m)
Algorithm MYSTERY
    function F(n, m)
        if n = 0 then F ← 0
        else   F ← (m+n-1) +
                F(n-1, m-1)
        endif
    endfunc
    print F(m, n)
```

13. ⟨2⟩ Consider Algorithm MYSTERY2 below. Conjecture an explicit formula for the output. Then prove you are right.

```
Input n           [n ≥ 0]
Output answer
Algorithm MYSTERY2
    function foo(n)
        if n = 0 then foo ← 1
        n = 1 then foo ← 2
        else   foo ←  $\frac{4}{3}(foo(n-1) + foo(n-2))$ 
        endif
    endfunc
    answer ← foo(n)
```

14. ⟨1⟩ Draw all the cases of 3 lines dividing the plane into regions that we discussed in Example 5 and find the number of regions in each case. (That is, the lines need not be in general position.)

15. ⟨2⟩ In Example 5 we showed that the number of regions  $r_n$ , when  $n$  lines in general position divide up the plane, satisfies the recurrence

$$r_n = r_{n-1} + n, \quad r_1 = 2.$$

We conjectured that  $r_n = \frac{1}{2}(n^2 + n) + 1$ . Prove the conjecture by induction two ways:

- a) Let  $P(n)$  be the statement that

$$r_n = \frac{1}{2}(n^2 + n) + 1.$$

b) Let  $Q(n)$  be the statement that  $r_n = S_n + 1$ , where  $S_n$  is the sum in Example 1. Recall that  $S_n$  satisfies the same recurrence but a different initial condition,  $S_1 = 1$ . When you are finished with this induction, you will have shown that  $r_n = S_n + 1$  for all  $n \geq 1$ . Why does that prove the desired result?

16. {1} We started most of the examples in this section with  $n = 1$ : sum one integer, draw one line, draw one circle. Often one can start with  $n = 0$ : sum no numbers, count the regions of the plane when there are no lines, etc. For the rest of this problem, we apply this approach to Example 5 only.

- a) What is  $r_0$ ?
- b) Does the formula we came up with work for  $n = 0$  as well?
- c) Is there any advantage to using  $n = 0$  as the basis case in the proof of the induction?

17. {5} In Example 5 we chose to limit ourselves to lines in general position so that the number of regions depends only on  $n$ . But this should only be a temporary restriction. Once we understand the basic situation we should tackle the variants.

One could now handle the special case where the only “nongeneral” aspect is one pair of parallel lines. Once one answers that completely, one could handle the special case where the only non-general aspect is exactly 3 lines concurrent. But such an approach is of limited value, as the number of special cases grows without bound as  $n$  increases. One only treats individual special cases if one doesn’t see how to handle them all at once, or as a step towards finding a general pattern.

Do as much as you can to settle the entire problem of determining the number of regions when  $n$  lines divide the plane. *Hint:* Recursion is still the key. As you add the next line, what happens?

18. {2} Consider again the regions formed by  $n$  lines in general position on the plane. Now, instead of counting the regions, we merely want to color them — so that no two regions sharing a common boundary are the same color. (Regions that share just a point in common — for instance, “opposite” regions at a point where two lines cross — need not be different colors.) What is the minimum number of colors necessary when there are  $n$  lines? Prove it.

19. {2} Same as [18], but now the lines need not be in general position; several may share a point in common or be parallel.

20. {2} Consider again the regions formed by  $n$  lines in general position on the plane. Now, instead of counting the regions, we want to count the number of segments into which the lines divide themselves. For instance, 2 lines divide themselves into 4 segments. Determine if the number of segments depends only on  $n$ . If it does, conjecture and prove a formula for it. If it doesn’t, show an example and you’re done.

21. {1} Same as [20], but now the lines need not be in general position.

22. {3} In this problem we suggest two ways to find a closed form for  $r'_n$ , the number of regions formed by  $n$  intersecting circles in Example 6.

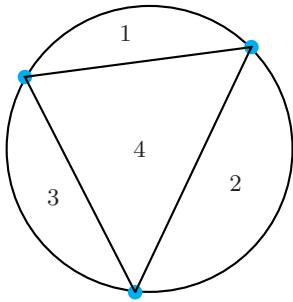
- a) Relate the recurrence (5) to the recurrence (2) for summing the first  $n$  integers, and use this connection to find a closed form.
- b) Use geometric reasoning, as in Example 1. Can you get figures as nice as the triangles there?

23. {5} Suppose we generalize the circles in Example 6 to ovals. Is the number of regions dependent only on  $n$ ? If not, can you restrict the problem in one or more ways in order to solve it. Can you solve it completely?

24. {2} By a **sphere** we mean the surface of a ball. A **great circle** is a largest possible circle on a sphere. (An equivalent definition: a great circle is the intersection of the sphere with any plane that passes through its center; e.g., the equator, or a longitude circle, but not a latitude circle other than the equator). Let  $R_n$  be the number of regions the sphere is divided into by  $n$  great circles, no three of which meet at the same point.

- a) Determine  $R_0, R_1, R_2$  and  $R_3$ . You can do this by brute force if you want, that is, drawing pictures.
- b) Find a recurrence for  $R_n$ .
- c) Find a closed formula for  $R_n$ .

25. {4} When  $n \geq 2$  points are picked randomly on a circle, and all chords between them are drawn, these chords divide the inside of the circle into several regions. For instance, when  $n = 3$ , the 3 chords form 4 regions, as shown in Fig. 2.19. When  $n = 4$ , the 6 chords form — well, we leave it to you below to figure out the number of regions.



**FIGURE 2.19**

Chords dividing a circle into regions.

- a) We said “randomly” because if the positioning is too regular, the number of regions can decrease. Give an example of an  $n$  and two configurations of  $n$  points on a circle so that the number of regions is different for the two configurations.

- b) Come up with a definition of general position for this problem so that, for  $n$  points in general position, the number of regions formed depends only on  $n$ . Call the number  $A_n$  (for areas; we’ve overused  $r$  and  $R$ ).

- c) Determine  $A_n$  for  $n = 2, 3, 4, 5$ .

- d) Make a conjecture about  $A_n$ .

- e) Test your conjecture, either by computing at least one more value or by devising a recurrence.

26. ⟨3⟩ In this problem we explore the ratio method for determining if a sequence is essentially exponential and, if so, how to determine its formula. We will use some growth concepts from Section 0.3.

As usual, a sequence is exponential if it is of the form  $Ab^n$ , where  $A$  is a nonzero constant and  $b \neq 0$  or 1. We say that a sequence  $\{a_n\}$  is **essentially exponential with dominant term  $Ab^n$**  if  $Ab^n$  is exponential and  $a_n = Ab^n + c_n$  where  $c_n = o(b^n)$ . The part  $c_n$  is called the **lower order term(s)**.

- a) Prove: if  $\{a_n\}$  is essentially exponential with dominant term  $Ab^n$ , then  $\lim_{n \rightarrow \infty} (a_{n+1}/a_n)$  exists and is not 0 or 1. In other words, if  $\{a_n\}$  is essentially exponential, you can gather evidence for this by determining if the ratios  $a_{n+1}/a_n$  have a limit (other than 0 or 1). This

is the first test in the systematic exploratory method.

- b) Show that the converse of the theorem in a) is false by determining the limit of  $a_{n+1}/a_n$  for  $a_n = n2^n$  (which is not an essentially exponential sequence because of the nonconstant coefficient  $n$ ). Still, the theorem in a) is very useful because if  $a_{n+1}/a_n$  appears to have a limit, we can still *guess* that  $\{a_n\}$  is essentially exponential.

- c) Prove: if  $\{a_n\}$  is essentially exponential with dominant term  $Ab^n$ , then  $A = \lim_{n \rightarrow \infty} a_n/b^n$ . In other words, once we find  $b$  from looking at the limit of  $a_{n+1}/a_n$ , then if we are right in guessing that  $a_n$  is essentially exponential, we can then go on and find  $A$  too.

At this point we know the dominant term exactly. If we want to determine  $c_n$  exactly, we study the sequence  $a_n - Ab^n$ , since this is  $c_n$ . It might be that  $c_n$  is itself exponential, in which case we iterate the procedures in a) and c). For instance, if  $a_n = 3^n + 2^n$  then  $c_n$  would be the exponential  $2^n$  (which is lower order than the dominant term  $3^n$ ). If  $c_n$  turned out to be polynomial, there would be other procedures.

- d) Guess by systematic exploration a closed formula for the sequence  $\frac{1}{2}, 4, 7, 31, 73, 259, 697, \dots$

- e) Part of what we proved in a) is this: If  $\lim_{n \rightarrow \infty} (a_{n+1}/a_n) = 1$ , then  $a_n$  is not essentially exponential. Prove a bit more: If  $a_n$  is polynomial, then this limit is 1. Thus, even if you are not sure whether a sequence is exponential or polynomial (often it is one or the other), looking at  $\lim_{n \rightarrow \infty} (a_{n+1}/a_n)$  is still the right thing to do first if you are exploring numerically.

27. ⟨2⟩ Using the method of [26], or otherwise, come up with formulas for the following sequences. *Note:* In one sense there is no right answer for these sequences, because we give you only a finite number of terms. It is a theorem that, for any finite sequence, there are infinitely many formulas that correctly describe it. (The formulas would start giving different values for terms after those given.) But we have in mind a simple formula for each sequence. With the methods of this section, you should be able to find it. Assume that the first term corresponds to  $n = 1$ .

- a) 3, 6, 11, 20, 37, 70, 135, 264, 521, 1034
- b) 1, 0, -1, 0, 7, 28, 79, 192, 431, 924
- c) 1, 5, 13, 41, 121, 365, 1093, 3281, 9841, 29525
- d) 0, 0, 6, 24, 60, 120, 210, 336, 504, 720
- e) 2, 8, 24, 64, 160, 384, 896, 2048, 4608, 10240

28. ⟨1⟩ Find and prove a formula for  $a_n$  if  $a_1 = 1$  and

$$a_n = a_{\lceil n/2 \rceil} + a_{\lfloor n/2 \rfloor} \quad \text{for } n > 1.$$

29. ⟨3⟩ Find the solution to

$$L(n) = 1 + L(\lceil n/2 \rceil), \quad L(1) = 0.$$

30. ⟨3⟩ Find the solution to

$$L(n) = 1 + L(\lfloor n/2 \rfloor), \quad L(1) = 0.$$

31. ⟨2⟩ A **chord** in a polygon (also called a **diagonal**) is a line segment between two nonadjacent vertices. Chords are **nonintersecting** if they have no points in common, except perhaps an endpoint. A polygon is **triangulated** if it is divided by nonintersecting chords into triangles. For instance, a rectangle with one diagonal drawn is triangulated into two triangles, and a triangle is triangulated using zero chords.

a) Triangulate a hexagon various ways; same for an 7-gon.

b) Make a conjecture about triangulations of convex polygons. How many triangles are obtained? How many diagonals are used?

c) Prove your conjectures by induction. (But see also [8, Section 4.5] for a proof without induction.)

32. ⟨3⟩ A **quadrangulation** is a partition of an  $n$ -gon into quadrilaterals using nonintersecting diagonals.

a) Conjecture by experiment which convex  $n$ -gons can be quadrangulated.

b) For those convex  $n$ -gons that can be quadrangulated, conjecture relationships between  $n$ ,  $q$  and  $d$ , where  $q$  is the number of quadrilaterals formed and  $d$  is the number of diagonals used.

c) Prove your result in part a) by induction. *Hint:* You could do one induction over those  $n$  for which there are quadrangulations and another over  $n$  for which there are not; but perhaps you can do one induction that covers both cases.

*Note:* Part b) of this problem can be dispatched much more crisply by the sort of combinatorial argument introduced in Chapter 4. See [9, Section 4.5].

## 2.7 Inductive Definitions

Let's look again at the very first induction in this chapter, the proof that  $\sum_{k=1}^n k = (n+1)n/2$ . The key step, breaking down case  $n+1$  to get  $n$ , was accomplished by using the equality

$$\sum_{k=1}^{n+1} k = \left( \sum_{k=1}^n k \right) + (n+1). \tag{1}$$

Why is this equation correct?

It's correct by the definition of sigma (summation) notation. However, the only "definition" of sigma notation that we've given is that

$$\sum_{k=1}^n a_k = a_1 + a_2 + \cdots + a_n. \tag{2}$$

While this is quite satisfactory for sigma notation, for more complicated notation it might be clearer if we avoided ellipses (dot dot dot definitions).

Note that Eq. (2) is really an infinite number of equations, one for each value of  $n$ . Trying to deal with this infinity of cases is what led us to the description with an ellipsis. Have we ever had to treat a problem with an infinite number of cases

before? Of course: in proving inductions. So, can the “basis and inductive step” approach be applied to definitions as well and thus avoid ellipses? Yes. From the perspective of induction, sigma notation is *defined* by the following display:

$$\sum_{k=1}^n a_k = \begin{cases} a_1 & \text{if } n = 1 \\ \left(\sum_{k=1}^{n-1} a_k\right) + a_n & \text{if } n > 1. \end{cases}$$

Such a display, with the notation to be defined on the left and a large brace followed by alternatives on the right, is said to be a *cases display*. There is an alternative, used most often for defining sequences, where the basis case(s) is presented as a separate equation, for instance,

$$\begin{aligned}\sum_{k=1}^1 a_k &= a_1, \\ \sum_{k=1}^n a_k &= \left(\sum_{k=1}^{n-1} a_k\right) + a_n.\end{aligned}$$

This format is also more common when, in the inductive part of the definition, you don’t define case  $n$ , but rather  $n+1$  or some other case; see Example 2 below.

Not surprisingly, with either format this sort of thing is called an **inductive definition** or a **recursive definition**.

And the answer to why Eq. (1) is correct is now, simply, “by definition”: Eq. (1) is just the inductive part of the definition with  $n+1$  substituted for  $n$ .

Many other notations can be defined recursively.

## EXAMPLE 1

### Factorials

One often sees  $n!$  defined by

$$n! = n(n-1) \cdots 3 \cdot 2 \cdot 1,$$

but one can avoid the dots by writing

$$n! = \begin{cases} 1 & \text{when } n = 1 \\ n[(n-1)!] & \text{when } n > 1. \end{cases}$$

In fact, it is common to define  $0! = 1$  (see [9]), in which case we may restate the inductive definition as

$$n! = \begin{cases} 1 & \text{when } n = 0 \\ n[(n-1)!] & \text{when } n > 0. \end{cases}$$

Just as proofs by induction come in a variety of forms, so do inductive definitions. The next example shows a case where there are two terms in the basis. The thing being defined is a sequence. We’ve given recursive definitions of sequences earlier in this book, but we do it again to indicate that they are similar in spirit to recursive definitions of symbols and of operations, as introduced in this section.

## EXAMPLE 2

### Fibonacci numbers

Define the sequence  $\{f_n\}$  by

$$\begin{aligned} f_0 &= f_1 = 1, \\ f_{n+1} &= f_n + f_{n-1}, \quad n \geq 1. \end{aligned}$$

Thus

$$\begin{aligned} f_2 &= f_1 + f_0 = 1 + 1 = 2, \\ f_3 &= f_2 + f_1 = 2 + 1 = 3, \end{aligned}$$

and so on. This is a famous sequence, called the Fibonacci sequence, which we will return to in Section 5.2. ■

In this book, we first introduced recursion in the context of algorithms. Thus you should not be surprised to learn that recursive definitions can easily be put into algorithmic form. For instance, here is  $\sum$  written as a recursive function. It is assumed that the main algorithm in which it sits has  $a_1, a_2, \dots, a_n$  among its inputs:

```
function Sum(n)
    if n = 1 then Sum ← a1
    else Sum ← Sum(n-1) + an
    endif
endfunc
```

In the case of  $\sum$  it is easy to write an equivalent iterative fragment as well:

```
S ← a1
for k = 2 to n
    S ← S + ak
endfor
```

The appearance of  $k = 2$  in the iterative version may seem a little odd; the only two numbers one might expect are 1 and  $n$ . This suggests a slight revision:

```
S ← 0
for k = 1 to n
    S ← S + ak
endfor
```

The recursive version of Sum can also be rewritten to correspond to this second iterative version [3].

This second iterative version suggests that there is a reasonable way to define the empty sum  $\sum_{k=1}^0 a_k$ , or more generally,  $\sum_{k \in \emptyset} a_k$ . It should be defined as 0,

because if you want to get the right answer in the end, that's what you must start with before you add anything. Perhaps this point seems obvious — of course you have 0 before you do anything — but actually it's more subtle than that. Consider the empty product [7, 8] and the empty factorial [9] and you will see that you *don't* always have 0 before you do anything.

## Inductively Defined Sets

So far, we have used the basis and inductive step approach to define *values* for expressions and functions. The same approach can be used to define sets. Our goal here is to show you through examples how to interpret and create inductive set definitions.

### EXAMPLE 3

Devise an inductive definition for  $N$ , the nonnegative integers.

**Solution** The essence of  $N$  is that it is all the numbers you get (and only the numbers you get) by starting with 0 and forever adding 1. In other words, we have:

---

**Definition 1.** The set  $N$  of **nonnegative integers** is the smallest set such that

- i)  $0 \in N$ , and
  - ii) if  $n \in N$ , then  $(n+1) \in N$ .
- 

This definition is typical of the form of inductively defined sets. There is a basis — one or more specific objects are declared to be in the set — and there is an inductive step — one or more **production rules** are given, which say that if such-and-such is in the set, then so-and-so must also be in the set. (In the preceding definition of  $N$ , there is just one specific object, 0, and just one production rule: If  $n$  is in the set, then  $n+1$  is in the set too.) Finally, there is also the assertion that the set in question is the smallest set satisfying the basis and the inductive step. But what does “smallest” mean here? The intuitive idea is that the smallest set should contain just those elements that have to be there to satisfy the basis and production rules. For instance, the real numbers  $R$  satisfy i)-ii) above, but  $R$  isn't the smallest set because we can throw out all the noninteger reals and i)-ii) are still satisfied. It turns out that a nice way to identify the smallest set is to insist that it be a proper subset of every other set that satisfies the basis and production rules — there always is a smallest set in this sense. For another approach, see [23].

Recursive definitions of sets are especially valuable whenever you have an infinite set that is hard to describe in a direct way. Consider, for instance, any computer language you know. There are quite exacting rules about what is a legitimate “sentence” in that language, yet there are infinitely many legitimate sentences, and you would be hard put to define them directly. If you study language design at some

point, you will see that their definitions are recursive — with a big basis and lots of production rules. To give the idea, we introduce some simple “toy languages” in Examples 4 and 5 below. In any event, the subject of **formal languages** is an area of interest to both computer scientists and linguists, for it has a bearing on natural language, too. We might add that constructing the languages is the easy part; figuring out how to parse expressions in them is the bigger challenge. (To parse means to figure out how the expression was put together, so that you then can figure out what it means.)

## EXAMPLE 4

Let  $S$  be the set of items defined recursively by

- every lowercase roman letter is in  $S$ , and
- if  $A$  and  $B$  are expressions in  $S$ , then so is  $(A+B)$ .

Describe the set  $S$  directly.

*Note 1.* The phrase “the smallest set” does not appear above, but in a recursive definition of a set it is always understood.

*Note 2.* In (b),  $A$  and  $B$  are generic names for any two beasts in  $S$ ;  $A$  might be  $(x+(x+y))$ . There is no claim that the capital italic letters  $A$  and  $B$  are symbols in  $S$ . On the other hand, the parentheses and plus sign in “ $(A+B)$ ” mean exactly themselves. That is, the production rule requires that they be included, even in cases where parentheses are optional in ordinary algebra.

**Solution** Let’s look at some typical elements of  $S$ . Applying (b) once to the elements from (a), we get expressions like

$$(x+z), \quad (a+b), \quad (b+c), \quad (n+n).$$

Applying (b) again to everything we now know to be in  $S$ , we get things like

$$(a+(b+c)), \quad ((a+b)+c), \quad ((x+z)+(n+n)).$$

In the next round we get things like

$$\begin{aligned} &((a+(b+c))+d), \quad ((a+b)+((x+z)+(n+n))), \\ &(((x+z)+(n+n))+((x+z)+(n+n))). \end{aligned}$$

Therefore we can describe  $S$  by saying it consists of all “fully parenthesized” expressions involving addition and using lowercase roman letters. (See Section 7.2 for further discussion of fully parenthesized expressions.) ■

## EXAMPLE 5

### Palindromes

A palindrome is an expression that reads the same backwards and forwards, e.g., “abcba” or “able was I ere I saw elba”. (If we ignore spaces, capitalization, and punctuation, there are even more impressive palindromes in English: “A man, a plan, a canal, Panama” and “Go hang a salami! I’m a lasagna hog!”) Let  $P$  be the set of all palindromes using only lowercase roman letters and no blanks. Devise an inductive definition for  $P$ .

**Solution** An inductive step will require that we express longer palindromes in terms of shorter ones. This is easy: The first and last symbol of a palindrome must be the same, and if we strip them off we still have a palindrome (if anything is left). Since this stripping operation relates a palindrome of length  $n$  to a palindrome of length  $n-2$ , we need two basis cases: palindromes of lengths 1 and 2. Therefore we have the following definition:  $P$  is the set of symbol strings defined recursively by

- a) Single and double roman lowercase letters are in  $S$  (e.g., “q” and “hh”), and
- b) If  $A$  is in  $S$ , and  $*$  represents a single lowercase roman letter, then  $*A*$  is in  $S$ . ■

## Problems: Section 2.7

---

1.  $\langle 1 \rangle$  Write the recursive definition of  $n!$  using  $n$  and  $n+1$  instead of  $n-1$  and  $n$ . Let  $n = 0$  be the basis case.
2.  $\langle 1 \rangle$  Write  $n!$  as a recursive function  $f(n)$  in our algorithmic language. Let  $n = 0$  be the basis case.
3.  $\langle 1 \rangle$  Using our algorithmic language, write a recursive function for  $\sum_{i=0}^k a_i$  that has  $k = 0$  as its base case.
4.  $\langle 2 \rangle$  We don't have to invoke algorithms to explain why mathematicians have defined  $\sum_{k=1}^0 a_k$  to be 0. Let  $A$  and  $B$  be disjoint, nonempty sets. Then clearly
 
$$\sum_{i \in A \cup B} a_i = \sum_{i \in A} a_i + \sum_{i \in B} a_i.$$
 How should  $\sum_{i \in \emptyset} a_i$  be defined so that the identity is true even if either  $A$  or  $B$  is empty?
5.  $\langle 1 \rangle$  Write a recursive function for  $\prod_{k=1}^n a_k$ 
  - a) in our algorithmic language. Use  $n = 0$  as the base case.
  - b) using a recursive definition as was done for  $\sum_{k=1}^n a_k$  in the text.
6.  $\langle 1 \rangle$  Write an iterative algorithm fragment to compute  $\prod_{k=1}^n a_k$  so that the loop goes from  $k = 1$  to  $n$ .
7.  $\langle 1 \rangle$  Arguing on the basis of [5, 6], figure out how mathematicians define  $\prod_{k=1}^0 a_k$ .
8.  $\langle 2 \rangle$  Using an argument similar to that in [4], figure out how mathematicians define  $\prod_{k=1}^0 a_k$ .
9.  $\langle 2 \rangle$  Why should  $0!$  be defined as 1?
10.  $\langle 1 \rangle$  Rewrite the recursive definition of the Fibonacci numbers from Example 2 in the cases format.
11.  $\langle 2 \rangle$  Consider Algorithm FIB-RECUR from [21a, Section 1.4]. Prove by strong induction on  $n$  that the value of function Fib( $n$ ) is  $f_n$  of Example 2 in this section, and thus that the algorithm correctly outputs  $f_n$ .
12.  $\langle 2 \rangle$  Write the definition of Fibonacci numbers in Example 2 as an iterative algorithm fragment. Prove your algorithm correct.
13.  $\langle 1 \rangle$  Give a recursive definition for the set of even nonnegative integers.
14.  $\langle 2 \rangle$  The **empty string** (no symbols) is often considered to be a palindrome, but it is not included in the set defined in Example 5. Revise the definition to include it. (The empty string is usually denoted by  $\Lambda$ , the Greek capital letter Lambda. One solution is simply to add  $\Lambda$  to the list in condition (a) of the definition, but you can do better.)
15.  $\langle 3 \rangle$  Modify the recursive definition of palindromes so that symmetrically positioned single spaces are allowed within expressions (as in “able was I ere I saw elba”). *Caution:* You don't want a definition that allows spaces at the ends (as in “ere”) or consecutive spaces. So you can't just make space a new symbol in the alphabet.
16.  $\langle 2 \rangle$  Modify the recursive definition of palindromes further (in addition to spaces as in [15]) so that capital letters can match up with lower case (as in “Able was I ere I saw Elba”).

17.  $\langle 3 \rangle$  Devise a recursive definition of the set of palindromes if single spaces are allowed between words and these spaces need not match other spaces (as in “a man a plan a canal panama”).

18.  $\langle 2 \rangle$  Let  $m$  and  $n$  be specific integers. Define the set  $U$  recursively by

- i)  $m, n \in U$ , and
- ii)  $p, q \in U \implies p+q \in U$ .

Give a direct (i.e., nonrecursive) description of  $U$  in terms of  $m$  and  $n$ . Give a direct, numerical description of  $U$  when  $m = 7$  and  $n = 11$ .

19.  $\langle 3 \rangle$  Give a direct description of the set of numbers  $V$  defined recursively by

- i)  $2, 10 \in V$ , and
- ii) if  $p$  and  $q$  are in  $V$ , then so are  $pq$  and  $p^q$ .

20.  $\langle 2 \rangle$  Assume that we are building the foundations of mathematics and have succeeded in defining and proving all the standard properties of addition for nonnegative integers. The next step is to define multiplication inductively as follows:

- i)  $0 \cdot m = 0$ ,
- ii)  $(n+1) \cdot m = n \cdot m + m$ .

Prove from this definition that  $3m = m + m + m$  for all nonnegative integers  $m$ .

21.  $\langle 2 \rangle$  Assuming the generalized distributive law

$$a \sum_{i=1}^n b_i = \sum_{i=1}^n ab_i$$

(Example 1, Section 2.3), prove the further generalization:

$$\sum_{i,j} a_i b_j = \left( \sum_i a_i \right) \left( \sum_j b_j \right).$$

What definition of the symbol  $\sum_{i,j}$  is implicit in your proof?

22.  $\langle 2 \rangle$  Use the inductive definition of the nonnegative integers  $N$  (Example 3) to prove the Principle of Weak Induction (as stated at the end of Section 2.2, but you may assume that  $n_0 = 0$ ). Hint: Whatever proposition  $P$  is, let  $S$  be the set of real numbers  $x$  for which  $P(x)$  is true. Show that  $S$  satisfies conditions (i) and (ii) from the inductive

definition of  $N$ . Since  $N$  is the smallest set satisfying these conditions, what can we conclude? (This is a typical example of a proof using an inductive set definition.)

23.  $\langle 4 \rangle$  (Alternative form of inductive set definition) For any basis set  $B$  and production rules  $\mathcal{P}$ , define

$$S_0 = B$$

and

$$S_k = \text{all objects obtained by applying } \mathcal{P} \text{ once to items in } \bigcup_{i=0}^{k-1} S_i.$$

Thus, in Example 4,  $S_5$  is the set of all expressions  $(A+B)$  where  $A, B \in \bigcup_{i=0}^4 S_i$ . Then the recursively defined set  $S$  is declared to be

$$S = \bigcup_{k=0}^{\infty} S_k.$$

When each production rule acts on single items (e.g., the production rule for  $N$  acts on single numbers  $n$  to produce  $n+1$ ), you may replace  $\bigcup_{i=0}^{k-1} S_i$  in the definition of  $S_k$  by  $S_{k-1}$ . Thus, for the nonnegative integers,  $S_k = \{n+1 \mid n \in S_{k-1}\}$ .

- a) Explain why the nonnegative integers turn out to be what they should be with this alternative definition. Hint: What is  $S_k$ ?
- b) Describe  $S_1$  and  $S_2$  for Example 4. Were these sets already discussed in the text?
- c) In general, why is the alternative definition closer in style to the recursive definitions of values in the first examples in this section?
- d) Argue that  $S$  in the alternative definition above meets the two conditions in the original form of inductively defined set:
  - i)  $B \subset S$ ,
  - ii) If objects  $A_1, \dots, A_n$  are in  $S$ , and  $A'$  is obtained by applying a production rule once to  $A_1, \dots, A_n$ , then  $A' \in S$ .
- e) Argue that every set that meets the basis condition and the production condition of the original form of inductively defined set contains  $S$  as defined in this problem.

Parts d) and e) together show that  $S$  is the *smallest* set meeting the basis condition and production rules. You have shown that the original form of inductive set definition and the form in this problem are equivalent.

## 2.8 Faulty Inductions

One theory of pedagogy holds that an instructor should never deliberately put something false on the board, even if it's announced. The students will mindlessly copy it in their notebooks and memorize it!

Maybe — but it's hard to grasp precisely what something is without contrasting it with similar things that it is not. This is particularly true when the thing in question is subtle. In this vein, we provide several arguments that look like successful induction proofs but are not. The first is a classic — every mathematician knows some version of it. The others are our own contributions.

### EXAMPLE 1

Prove that all women are blondes.

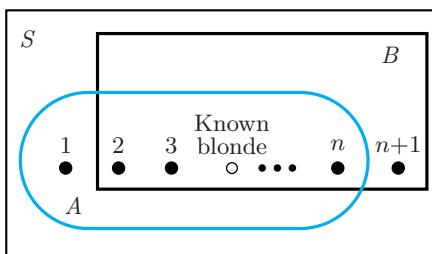
“Proof”: Let  $P(n)$  be the statement: For any set of  $n$  women, if any one of them is blonde, then all  $n$  of them are blonde. First, we will prove this for all  $n$  by induction. Then, taking  $n$  to be the number of women on earth, we get the special case: For the set of all women on earth, if any one of them is blonde, then all of them are. Since it is well known that at least one woman is blonde (look around, or maybe you are), we conclude that all women are blonde.

Now for the induction.  $P(1)$  asserts: For any set of 1 woman, if any one of them is blonde, then all 1 of them is blonde. This is just a contorted way to say “a blonde is a blonde”, and is clearly true.

Next, assume  $P(n)$ . We must show that, given an arbitrary set  $S$  of  $n+1$  women, including a blonde, all the women in the set are blonde. To apply  $P(n)$  we must look for sets of  $n$  women, including a blonde, within the  $n+1$  set. Figure 2.20 shows how to do it. We let  $A$  be some subset of  $S$  that contains the known blonde and that has  $n$  members. We let  $B$  be any other  $n$ -subset of  $S$  that contains the known blonde and that includes the one member left out of  $A$ . By  $P(n)$ , set  $A$  is all blonde, and by  $P(n)$  again, set  $B$  is all blonde. Therefore  $A \cup B = S$ . But  $A \cup B = S$ , so we are done. ■

FIGURE 2.20

All women are  
blondes, by  
induction?



Where's the flaw? We've hidden the answer later in this book, so you might as well argue about it in class! The explanations of the next three faulty inductions are given after Example 4.

## EXAMPLE 2

Prove that every integer  $> 1$  has a unique prime factorization. (We really should say unique *up to order*: The factorizations  $5 \cdot 2 \cdot 3 \cdot 2$  and  $2 \cdot 2 \cdot 3 \cdot 5$  are not identical factorizations of 60, but they contain the same primes each used the same number of times, so they are considered the same “up to order”.)

This time the claim is true — but the following proof is wrong. Why?

“Proof”: Let  $P(n)$  be the statement that  $n$  has a unique factorization. For the basis,  $P(2)$  is clearly true since 2 is prime. By strong induction, assume  $P(j)$  for  $j < n$  and consider  $n$ . If  $n$  is prime, by definition there is no other way to factor it and we are done. Otherwise,  $n$  factors somehow as  $n = rs$ , where  $r < n$  and  $s < n$ . Therefore, by assumption,  $r$  has a unique factorization  $\prod p_i$  and  $s$  has a unique factorization  $\prod q_j$ . Thus  $\prod p_i \prod q_j$  is one prime factorization of  $n$ , and since none of the factors in either piece can be changed, none of the factors of the whole thing can be changed. That is, the prime factorization of  $n$  is unique. ■

## EXAMPLE 3

Prove that in a regular  $n$ -gon, the sum of the internal angles is  $180(n-2)$  degrees.

“Proof”: Let  $P(n)$  be the statement for a particular  $n$ . The basis case is  $P(3)$ , which is true by the famous theorem that a triangle has  $180^\circ$ . For the inductive step, given a regular  $n$ -gon with vertices  $v_1, v_2, \dots, v_n$  in order around the  $n$ -gon, lop off vertex  $v_2$  by cutting along the diagonal from  $v_1$  to  $v_3$ . By assumption, the remaining  $(n-1)$ -gon has an interior angle sum of  $180(n-3)$  degrees. Now put back the triangle, which has  $180^\circ$ . Adding, we get  $180(n-2)$  degrees. ■

Didn’t we do the preceding proof in Section 2.2? So where’s the flaw?

## EXAMPLE 4

Consider 0–1 sequences in which 1’s may not appear consecutively, except in the right-most two positions. For instance, 0010100 and 1000011 are sequences of this sort, and 0011000 is not. Prove that there are  $2^n$  “allowed” sequences of length  $n$ .

“Proof”: Let  $s_n$  be the number of such sequences of length  $n$  and let  $P(n)$  be the statement that  $s_n = 2^n$ . The basis case,  $P(1)$ , asserts that  $s_1 = 2$ . This assertion is clearly correct: Both 0 and 1 are allowed binary sequences of length 1. Now, to show that  $P(n) \Rightarrow P(n+1)$ , take any allowed sequence of length  $n$ . We may append either 0 or 1 at the right end — in the latter case, we may create 11 in the last two positions, but that’s okay. Therefore,

$$s_{n+1} = 2s_n = 2 \cdot 2^n = 2^{n+1},$$

finishing the proof.

Unfortunately,  $2^n$  is the number of *all* binary sequences of length  $n$ . (Such facts will be justified at length in Chapter 4.) Yet some binary sequences (e.g., 01101) are not allowed because of the consecutive 1’s. Where did we go wrong? ■

Here are our explanations for the flaws in Examples 2–4.

**Example 2 Explained.** The flaw is a **breakdown error**, one that is easy to fall into if you are asked to prove that something is unique or optimal. We broke

$n$  down into  $rs$ . There was nothing wrong with that. There was also nothing wrong with asserting by induction that none of the factors in either  $r$  or  $s$  can be changed. But who says that the only factorization of  $n$  into two factors is  $r \times s$ ? Maybe  $n$  also equals  $tu$ , and maybe the prime factorizations of  $t$  and  $u$  are also unique, but the product of those factorizations looks nothing like the product of prime factorizations of  $r$  and  $s$ . To be sure, such a situation can't happen for the integers, but the argument in Example 2 doesn't prove it — and indeed it *does* happen in some other number systems. See [15, 19, 20].

*Moral:* If the pieces are uniquely constructed, that doesn't mean the whole thing is uniquely constructed. The decomposition into pieces may not be unique.

**Example 3 Explained.** The kicker here is the word “regular”. It has resulted in another sort of breakdown error, one that is sometimes very subtle, although here it was pretty blatant. When we take a regular  $n$ -gon and lop off a corner, the resulting  $(n-1)$ -gon isn't regular and so  $P(n-1)$  doesn't apply. That's the error: not actually obtaining the previous case when you break down the current case. Indeed, in this example it's hard to imagine what you could do to a regular  $n$ -gon that would make a regular  $(n-1)$ -gon appear naturally.

But, you object, the angle formula *is true* for the  $(n-1)$ -gon we obtain by deleting a corner, so what's wrong with using it? What's wrong is that in the method of induction you are allowed to use only the previous case(s) of the  $P$  you are trying to prove, and  $P(n-1)$  is the statement that a *regular*  $(n-1)$ -gon has an angle sum of  $180(n-3)$  degrees. True, we are allowed to use general knowledge and previous theorems as part of the argument in an induction, as we may in any proof. However, if we are going to use the general result about polygons from Example 3, Section 2.2, then we wouldn't now be giving a separate inductive proof for regular polygons; we would merely note that this special case follows immediately from the general result.

Indeed, we don't know any way to fix this faulty induction, despite the fact that the result is true. The only way to use induction is to prove the stronger result in Example 3, Section 2.2. This is an instance of **strengthening the inductive hypothesis**. It points up a curious subtlety about induction: In choosing what  $P(n)$  to prove by induction in order to get the desired theorem, it's not always obvious how “strong”  $P$  should be. With other sorts of proofs, if you can't succeed in proving that  $Q$  implies  $R$ , you either strengthen  $Q$  (assume more) or weaken  $R$  (aim for less) and try again. But in induction,  $P$  is both the conclusion (in the case  $n$ ) and the hypothesis (in the case  $n-1$ ), and so you have a delicate balancing act. If you strengthen  $P$ , you have more to work with, but you also have more to prove. If you weaken  $P$ , you have less to prove, but you also have less to work with. In this example, our initial choice of  $P$  turned out to be too weak. In another case it might be too strong.

How do you decide the right strength for  $P$ ? As always, experience helps. Noting carefully how you get stuck also helps. For instance, suppose you really didn't know that the interior angle formula holds for all convex polygons and you were trying to prove it for regular polygons only. Presumably, you would start writing down a proof as in Example 3. However, seasoned by the experience gained

from this section, you would catch the flaw in the inductive step. Since you would see that lopping off a corner does not preserve regularity or other special properties, you would suspect that the only way to make the induction work is to strengthen  $P$  by not assuming any special properties. Once you got the idea to strengthen  $P$ , you might think to strengthen it all the way to arbitrary polygons.

This example illustrates the important point that generalizations are sometimes easier to prove than special cases.

**Example 4 Explained.** The flaw this time is a **buildup error**. We asserted that, by appending either 0 or 1 to all the allowed sequences of length  $n$ , (1) all the allowed binary sequences of length  $n+1$  are obtained and (2) only the allowed ones are obtained. Claim (2) is false. For example, take  $n = 3$  and the allowed sequence 011. Appending either 0 or 1 gives a disallowed sequence. In the problems there are some examples where building up case  $n+1$  from case  $n$  leads to too few things instead of too many.

It is now time to explain the comment made earlier (in Remark 3 on p. 139) that the best place to use the previous case in the inductive step is generally in the middle, not at the beginning.

In the simplest inductions,  $P(n)$  is a statement about a single object. For instance, in the proof that  $\sum_{k=1}^n k = n(n+1)/2$ ,  $P(n)$  is one specific equation, the equation just given for that particular  $n$ . Therefore, in proving  $P(n) \Rightarrow P(n+1)$ , there is no problem about starting with  $P(n)$  and working up to  $P(n+1)$ . There is one specific equation you are heading for.

However, in many inductions,  $P(n)$  is a statement about a large set of things, even when  $n$  is fixed. For instance,  $P(5)$  might be a statement about the set of all round-robin tournaments with 5 teams or the set of all allowed binary sequences of length 5. To prove something about all items in a set, it is necessary to argue about an *arbitrary* item in the set. (See the discussion of “any” and “generic particular” on p. 62.) That is why we have often started our inductive steps with lines such as “so imagine that somebody presents us with the results of an arbitrary  $(n+1)$ -tournament” (from the proof of Theorem 2 of Section 2.2), or “so imagine someone gives us an arbitrary  $n$ -line configuration” (from the proof for Example 5, Section 2.6). If instead we start the inductive step by looking at the previous case  $P(n)$  and building up to  $P(n+1)$ , we are taking control of how example  $n+1$  is constructed, and there is a danger that we will not construct an arbitrary example. If our construction does not generate *all* items of type  $n+1$ , then even if we correctly argue that proposition  $P$  holds for what we construct, we have not proved  $P(n+1)$ . See [6].

Similarly, if we want to count the elements of a set (as opposed to proving that some property holds for all elements of a set), it is necessary that we consider all the elements and nothing else. If we start the inductive step by looking at the previous  $n$ -set and building up to the  $(n+1)$ -set, there is a danger that we will not construct all the elements or will construct other objects as well, as in the faulty induction of Example 4.

To avoid falling into such traps, we strongly recommend avoiding the buildup approach entirely. If  $P(n+1)$  is about all objects of “type”  $n+1$ , begin the inductive

step by considering an arbitrary such object. Break it down to obtain an object of type  $n$ , use  $P(n)$ , and work back up. For instance, had we started the inductive step in Example 4 by considering an arbitrary allowed sequence of length  $n+1$  and had we worked *down* by lopping off the final bit, then we would have realized that the remaining sequence can't have consecutive 1's *anywhere* and that therefore the simple relation between  $s_n$  and  $s_{n+1}$  required to obtain  $s_n = 2^n$  is false. (There *is* a formula for  $s_n$ , but deriving it depends on finding a relation between  $s_{n+1}$  and two previous terms. Stay tuned for Chapter 5.)

Even when  $P(n)$  is a single formula, you can use this “down and back” approach. We illustrated this with the second proof in Section 2.2 of  $\sum_{k=1}^n k = n(n+1)/2$  (the proof after the statement of Theorem 1). Namely, start with the LHS of  $P(n+1)$ , break down to the LHS of  $P(n)$ , assert the RHS of  $P(n)$  by induction, and then build back to the RHS of  $P(n+1)$ .

## Problems: Section 2.8

---

Each of problems 1–18 consists of an assertion and a “proof”. Most of the proofs are invalid, or at least badly muddled. Nonetheless, some of the assertions are true. For each problem, identify any errors in the proof. If there are errors but the assertion is true, provide a good proof.

- 1.**  $\langle 2 \rangle$  Assertion: Every positive integer is either a prime or a perfect square.

“Proof”: The number 1 is a square. By strong induction, assume the theorem for  $m < n$  and consider  $n$ . If it is prime, we are done. If not, it has some factorization  $n = rs$ . By assumption,  $r$  and  $s$  are squares, say,  $r = u^2$  and  $s = v^2$ . Thus

$$n = (u^2)(v^2) = (uv)^2,$$

which shows that  $n$  is a perfect square.

- 2.**  $\langle 1 \rangle$  Assertion: Every integer  $> 6$  can be represented as a sum of 7's and 11's.

“Proof”: Let  $P(n)$  be the statement that  $n$  can be represented as a sum of 7's and 11's. Basis step:  $P(7)$  is clearly true. Inductive step: By strong induction, we may assume that  $P(n-7)$  is true. Add one more 7 and  $n$  is represented as a sum of 7's and 11's as well.

- 3.**  $\langle 1 \rangle$  Assertion:  $\sum_{k=1}^n 2^k = 2^{n+1}$ .

“Proof”: Let  $P(n)$  be the case  $n$  of the assertion. Assume  $P(n-1)$  and add  $2^n$  to both sides:

$$\begin{aligned}\sum_{k=1}^{n-1} 2^k &= 2^n \\ \sum_{k=1}^{n-1} 2^k + 2^n &= 2^n + 2^n \\ \sum_{k=1}^n 2^k &= 2 \cdot 2^n = 2^{n+1},\end{aligned}$$

which is  $P(n)$ .

- 4.**  $\langle 2 \rangle$  Assertion: If the sequence  $\{a_n\}$  satisfies

$$a_0 = 2, \quad a_1 = 3, \quad \text{and}$$

$$a_n = 2a_{n-1} - a_{n-2} \quad \text{for } n > 1,$$

then no term of the sequence equals 1.

“Proof”: Clearly  $a_0 \neq 1$ . Now, for the inductive step, it suffices to show that  $\{a_n\}$  is increasing, since it starts at 2. Assume that  $a_{n-1} > a_{n-2}$ . Then

$$a_n = 2a_{n-1} - a_{n-2} > a_{n-1} > 1,$$

so  $a_n \neq 1$ .

- 5.**  $\langle 3 \rangle$  Assertion: Define  $\{x_n\}$  by

$$x_1 = 1, \quad x_2 = 3, \quad \text{and}$$

$$x_n = \frac{1}{2}(x_{n-1} + x_{n-2}) \quad \text{for } n > 2.$$

Starting with  $n = 2$ ,  $\{x_n\}$  is strictly monotonic decreasing (that is,  $x_{n+1} < x_n$ ).

“Proof”: We prove by induction that each term is smaller than the preceding term. Since  $x_2 = 3$  and  $x_3 = \frac{1}{2}(x_1 + x_2) = 2$ , the basis is true. Now

we want to show that  $x_{n+1} < x_n$ . Substituting the definition twice, we get

$$\begin{aligned} x_{n+1} &= \frac{1}{2}(x_n + x_{n-1}) \\ &= \frac{1}{2}\left(\frac{1}{2}[x_{n-1} + x_{n-2}] + x_{n-1}\right) \\ &< \frac{1}{2}\left(\frac{1}{2}[x_{n-2} + x_{n-2}] + x_{n-1}\right) + x_{n-1} \\ &\quad [\text{Since } x_{n-1} < x_{n-2}] \\ &= \frac{1}{2}(x_{n-1} + x_{n-2}) \\ &= x_n. \end{aligned}$$

6. ⟨3⟩ In a round-robin tournament, call a team a “Total Winner” if it wins every game. Call it a “Total Loser” if it loses every game.

Assertion: If a (round-robin) tournament with  $n \geq 3$  teams has no Total Winners or Total Losers, then there is an ordering of the teams so that Team 1 beat Team 2, 2 beat 3, ...,  $n-1$  beat  $n$ , and  $n$  beat 1.

“Proof”: Let  $P(n)$  be the assertion for  $n$  teams. The basis case is easy to check: Each team must win one game and lose one, which forces a cycle. As for the inductive step, start with an arbitrary  $n$ -tournament  $T$  without Total Winners or Losers; by  $P(n)$  there is an ordering  $t_1, t_2, \dots, t_n$  so that each team beats the next cyclically.

Now add an  $n+1$ st team  $t^*$ . Since there must not be any Total Winners or Losers in the extended tournament  $T'$ ,  $t^*$  must win at least one game and lose at least one. Therefore, as we go around the cycle in  $T$ , there must be some consecutive pair where the first team beats  $t^*$  and the second team loses to  $t^*$ . Inserting  $t^*$  between them we have a cyclical ordering in  $T'$ .

7. ⟨3⟩ Same assertion as in [6], but this time our proof does not use a buildup argument — those are always suspicious.

“Proof”: Let  $P(n)$  be the assertion for  $n$  teams. The basis case  $P(3)$  is true, as shown before. Now assume  $P(n)$  and consider an arbitrary  $(n+1)$ -tournament without Total Winners or Losers. Temporarily ignore one Team,  $t^*$ . By induction, there is an ordering  $t_1, t_2, \dots, t_n$  of the remaining teams so that each beats the next cyclically. Consider  $t^*$  again. Since  $t^*$  is neither a Total Winner nor a Total Loser,  $t^*$  beats some  $t_i$  and loses to some  $t_j$ . Go around the cycle of  $n$  teams starting with  $t_j$ . At some point we reach for the first time a team  $t_k$  that  $t^*$  beats. Then insert  $t^*$  in the

cycle after  $t_{k-1}$  and before  $t_k$ . (If  $k = 1$ , then  $k-1$  means  $n$ .)

8. ⟨1⟩ Assertion: Any adult can lift at one time all the sand in the world.

“Proof”: Let  $P(n)$  be: Any adult can lift  $n$  grains of sand at one time. We prove this by induction. The assertion is simply the case where  $n$  is the total number of grains of sand in the world. Basis step: Even the weakest adult can lift 1 grain of sand. Inductive step: Consider any adult. By assumption s/he can lift  $n-1$  grains of sand. One more grain is an imperceptible difference, so s/he can lift  $n$  grains as well.

9. ⟨3⟩ Assertion: In any convex polygon, the longest side is shorter than the sum of the lengths of the other sides.

“Proof”: Let  $P(n)$  be: In a convex  $n$ -gon, the longest side is shorter than the sum of the others. The basis case,  $P(3)$ , is the well-known fact that every side of a triangle is shorter than the sum of the other two. To show that  $P(n) \Rightarrow P(n+1)$ , take any convex  $(n+1)$ -gon  $P$ . Label the sides  $1, 2, \dots, n+1$  in clockwise order, with side 1 the longest. Slice off sides 2 and 3, replacing them by a single side  $2'$ , and thus obtain a convex  $n$ -gon  $P'$ . By  $P(n)$ , side 1 is shorter than the sum of the other sides of  $P'$ . If we now replace the length of side  $2'$  by the sum of the lengths of sides 2 and 3, the sum of the lengths of sides other than 1 gets even greater.

10. ⟨2⟩ Let  $a_n$  be the number of ways to place  $n$  distinct balls into three distinct boxes, if each box must have at least one ball.

Assertion: For  $n \geq 3$ ,  $a_n = 6 \cdot 3^{n-3}$ .

“Proof”: For the base case  $n = 3$ , the claim is that  $a_n = 6$ . This is right because exactly one ball must go in each box. There are 3 choices for which ball to put in box 1, times 2 choices for which of the remaining balls to put in box 2. As for  $P(n) \Rightarrow P(n+1)$ , since  $n \geq 3$ , each box already has at least one ball when there are  $n$  balls, so ball  $n+1$  may go in any of the three boxes. Therefore  $a_{n+1} = 3a_n = 3(6 \cdot 3^{n-3}) = 6 \cdot 3^{(n+1)-3}$ .

11. ⟨2⟩ Let  $b_n$  be the number of sequences of  $n$  digits in which no two consecutive digits are the same.

Assertion:  $b_n = 10 \cdot 9^{n-1}$ .

“Proof”: Clearly  $a_1 = 10$ , because any one of the

10 digits may appear by itself. That's the basis. As for the inductive step, for any  $n$ -long sequence without equal consecutive digits, we may add on as the  $(n+1)$ st digit any digit except the one which is currently last. That is, for each  $n$ -long sequence there are nine  $(n+1)$ -long sequences. Therefore

$$b_{n+1} = 9b_n = 9(10 \cdot 9^{n-1}) = 10 \cdot 9^{(n+1)-1}.$$

- 12.**  $\langle 3 \rangle$  A **derangement** of order  $n$  is a listing of the integers 1 to  $n$  so that no integer is in its “right” place. For instance, 312 is a derangement of order 3 because 1 is *not* in the first position (from the left), 2 is not in the second, and 3 is not in the third. Let  $d_n$  be the number of derangements of order  $n$ .

Assertion: For  $n > 1$ ,  $d_n = (n-1)!$

“Proof”: The basis,  $P(2)$ , says that  $d_2 = 1$ , and this is correct since 21 is the only derangement of order 2. To show  $P(n-1) \Rightarrow P(n)$ , take any derangement of order  $n-1$  and place  $n$  at the right end. This is not a derangement, but we may now switch  $n$  with *any* of the  $n-1$  previous numbers. Afterward, the integer  $n$  will surely be in the wrong place, and so will the number it switched with. All other numbers remain unchanged and thus are in the wrong place. Therefore, there are  $n-1$  derangements of order  $n$  for each derangement of order  $n-1$ . Thus

$$d_n = (n-1)d_{n-1} = (n-1)[(n-2)!] = (n-1)!$$

- 13.**  $\langle 3 \rangle$  Assertion: For every positive integer  $n$ ,  $n = n+1$ .

“Proof”: Let  $n+1$  be called the successor of  $n$ , or  $s(n)$  for short. Let  $M$  be the set consisting of 0 and all positive integers that equal their successors. Let  $P(n)$  be the statement that  $n \in M$ . We prove the assertion by induction on  $P(n)$ . Basis step:  $P(0)$  is true by definition of  $M$ . Inductive step: Suppose  $P(n)$ . This means  $n$  is in  $M$ . That is,  $n = n+1$ . But that means  $n+1$  is in  $M$ , so  $P(n+1)$  is true.

- 14.**  $\langle 2 \rangle$  Assertion: Every positive integer  $n > 1$  has a unique prime factorization.

“Proof”: Let  $P(n)$  be the statement that  $n$  has a unique factorization.  $P(2)$  is true since 2 is prime. As for the inductive step, assume that every integer  $< n$  has a unique factorization and consider  $n$ . By Example 2, Section 2.3, we know that  $n$  has

at least one prime factorization. Thus it suffices to show that any two prime factorizations are the same.

Let  $q$  be a prime that appears in the first factorization. Thus  $q$  divides  $n$ . By [4, Section 2.3],  $q$  divides one of the factors in the second factorization. However, all the factors in the second factorization are primes, and the only way for  $q$  to divide a prime is for  $q$  to be that prime. Thus both factorizations have at least one  $q$ . Temporarily delete one  $q$  from both factorizations; that makes them factorizations of  $n/q$ . By assumption, the prime factorization of  $n/q$  is unique, so these two factorizations of  $n/q$  are the same. Putting back the  $q$ , we find that both factorizations of  $n$  are the same.

- 15.**  $\langle 2 \rangle$  A polygon is **triangulated** if it is divided by nonintersecting chords into nonoverlapping triangles. (Chords were defined in [31, Section 2.6].) For instance, a rectangle with one chord drawn in is triangulated, and a triangle is triangulated using zero chords.

Assertion: Every convex polygon has a unique triangulation. (That is, of the various ways to put in nonintersecting chords, there is only one set of chords that results in a triangulation.)

“Proof”: Let  $P(n)$  assert that each convex  $n$ -gon has just one triangulation. The basis,  $P(3)$ , is true because a triangle is already triangulated, and there are no chords you could draw to make other triangulations. For  $n > 3$  use strong induction. Given an  $n$ -gon  $N$ , draw in one chord, forming an  $r$ -gon  $R$  and an  $s$ -gon  $S$ , with  $r, s < n$ . By induction,  $R$  and  $S$  have unique triangulations, thus so does  $N$ .

*Remark:* Note the similarity in the *form* of this argument to that in Example 2.

- 16.**  $\langle 2 \rangle$  An old question is: If you are waiting for the elevator on some floor of a skyscraper, what are the chances that the first time an elevator reaches your floor it will be going the way (up or down) you want to go?

Assertion: The elevator will *always* be going the way you want to go.

“Proof”: The basis case is  $n = 2$ , and in this case, whichever floor you are on, both you and the elevator have only one other floor to go to, so you are both going the same way.

Inductive step. Assume there are  $n + 1$  floors.

If you are on any of the first  $n$  floors, then the claim is true by induction. If you are on floor  $n+1$ , then both you and the elevator have only one choice, namely down. Therefore, the claim is true for all  $n+1$  floors.

- 17.** **(2)** Assertion: If the sequence  $a_0, a_1, \dots$  is defined recursively by

$$a_0 = 1, \quad a_n = 4a_{n-2},$$

then  $a_n = 2^n$  for all  $n = 0, 1, 2, \dots$

“Proof”: For each  $n$ , let  $P(n)$  be the claim that  $a_n = 2^n$ . The basis,  $P(0)$ , is true by definition, since  $2^0 = 1$ . As for the inductive step,

$$\begin{aligned} a_n &= 4a_{n-2} && [\text{Def.}] \\ &= 4(2^{n-2}) && [P(n-2)] \\ &= 2^n. \end{aligned}$$

- 18.** **(3)** Assertion: For any  $n \geq 1$ , any Hanoi clutter can be moved to a single pile by STOH rules. (Clutters were defined in [4, Section 2.4].)

“Proof”: By induction. For each  $n$ , let  $P(n)$  be the claim for that  $n$ .

Basis step,  $P(1)$ . Do nothing: any clutter is already a pile.

Inductive step,  $P(n) \Rightarrow P(n+1)$ . Consider any clutter of  $n+1$  rings. The largest is necessarily at the bottom, whichever pole it is on, so temporarily regard it as part of the floor. The remaining rings form an  $n$ -clutter. So by  $P(n)$  we may move them into a pile on top of the largest ring, making a single pile of all the rings.

- 19.** **(2)** If proving uniqueness through decomposition and induction is faulty in Example 2 and in [15], then why is it valid for TOH (Theorem 3, Section 2.4)?

- 20.** **(4)** Let  $E^+$  be the set of positive even integers. We will show that prime factorization is *not* unique within  $E^+$ .

If  $n, m \in E^+$ , then we say  $n$  is a factor of  $m$  *within*  $E^+$  if  $m/n \in E^+$ . Thus 2 is not a factor of 6 within  $E^+$  because  $6/2 = 3 \notin E^+$ . However, 2 is a factor of 12. A number is a prime *within*  $E^+$  if it has no factors within  $E^+$ . (Note: We don't have to say “except 1 and itself”; why?)

- a)** Show that every  $n \in E^+$  has a prime factorization within  $E^+$ . (Induction!)
- b)** Show that 60 has two different prime factorizations within  $E^+$ .

- c)** What are all the primes within  $E^+$ ?
- d)** What numbers *do* have unique prime factorizations within  $E^+$ ?

This example of nonunique factorization may seem like a cheap shot. Perhaps it's not surprising that familiar properties like unique factorization fail if we restrict ourselves to a set with glaring gaps — how do you expect factorization to work right in a set without 1? However, there are *extensions* of the integers which don't have any obvious omissions and still don't have unique factorization, e.g., all real numbers of the form  $m + n\sqrt{5}$ ,  $m, n \in \mathbb{Z}$ . Take abstract algebra and find out more.

- 21.** **(1)** Show that the following is a faulty inductive definition of the set  $P$  of all palindromic strings of roman lowercase letters:

1. Any single roman letter is in  $P$ ;
2. If  $A$  is a string in  $P$ , and  $*$  is a single roman letter, then  $*A*$  is in  $P$ .

- 22.** **(2)** Critique the following claim about the basic format of mathematical induction: In the inductive step it is all right to prove  $P(n-1) \Rightarrow P(n)$ , but it is invalid to prove  $P(n) \Rightarrow P(n+1)$ . The latter is invalid because it assumes the very thing the whole mathematical induction is trying to show, namely,  $P(n)$ . The former is okay because it concludes  $P(n)$  rather than assuming it, thus completing the whole proof.

- 23.** **(2)** Suppose that  $P(n)$  is of the form  $Q(n) \Rightarrow R(n)$ . For instance,  $P(n)$  might be: “If  $S$  is a set of  $n$  integers, then the difference between some two integers in  $S$  is even.” If you want to attempt a proof of  $P(n)$  by induction, with inductive step  $P(n) \Rightarrow P(n+1)$ , explain how to use the “down and back” approach when  $P(n)$  is of this if-then form. Specifically, in the inductive step, what is the best thing to assume in your first sentence? Should it be  $Q(n)$ , or  $Q(n) \Rightarrow R(n)$ , or  $Q(n+1)$ , or  $R(n+1)$ , or what?

- 24.** **(2)** In [36, Section 2.2] , we asked you to prove that for any odd  $n$  there exists an  $n$ -tournament in which every team wins the same number of games. We told you to show this by assuming such a tournament for  $n = 2k-1$  and constructing from it such a tournament for  $n = 2k+1$ . Isn't this precisely the sort of buildup argument that we now say is all wrong? Why not?

25.  $\langle 3 \rangle$  Here are parts of a student solution for [3, Section 2.2].

a) The student first wrote

$$P(n) = u_n = (2n - 1)^2.$$

What is wrong with this and how is it corrected?

- b) What is wrong with the following proof of the inductive step?

$$\begin{aligned} u_{n+1} &= u_n + 8n \\ (2(n+1) - 1)^2 &= (2n - 1)^2 + 8n \\ 4n^2 + 4n + 1 &= (4n^2 - 4n + 1) + 8n \\ 4n^2 + 4n + 1 &= 4n^2 + 4n + 1 \quad \checkmark \end{aligned}$$

26.  $\langle 2 \rangle$  Consider the sequence  $a_0 = 1, a_1 = 2, a_3, \dots$  defined recursively by

$$a_n = \frac{4}{3}(a_{n-1} + a_{n-2}).$$

Which of the following proofs is correct, if either?

- a) Claim:  $a_n = 2^n$  for all  $n \geq 0$ . Proof by strong induction. The basis  $P(0)$  is true since  $2^0 = 1$  and it is given that  $a_0 = 1$ . For the inductive step,

$$\begin{aligned} a_n &= \frac{4}{3}(a_{n-1} + a_{n-2}) \\ &= \frac{4}{3}(2^{n-1} + 2^{n-2}) \\ &= \frac{4}{3}(3 \cdot 2^{n-2}) = 4(2^{n-2}) = 2^n. \end{aligned}$$

- b) Claim:  $a_n = (-\frac{2}{3})^n$  for all  $n \geq 0$ . Proof by strong induction. The basis  $P(0)$  is true since  $(-\frac{2}{3})^0 = 1$  and it is given that  $a_0 = 1$ . For the inductive step,

$$\begin{aligned} a_n &= \frac{4}{3}(a_{n-1} + a_{n-2}) \\ &= \frac{4}{3}\left((- \frac{2}{3})^{n-1} + (- \frac{2}{3})^{n-2}\right) \\ &= \frac{4}{3}(- \frac{2}{3} + 1)(- \frac{2}{3})^{n-2} = \frac{4}{3} \cdot \frac{1}{3}(- \frac{2}{3})^{n-2} \\ &= \frac{4}{9}(- \frac{2}{3})^{n-2} = (- \frac{2}{3})^n. \end{aligned}$$

27.  $\langle 3 \rangle$  Consider the sequence defined recursively by

$$a_n = a_{\lfloor n/2 \rfloor} + a_{\lceil n/2 \rceil}, \quad a_0 = 0.$$

Which of the following proofs is correct, if either?

- a) Claim:  $a_n = n$  for all  $n \geq 0$ . Proof by strong induction. The basis  $P(0)$  is given. For the inductive step,

$$\begin{aligned} a_n &= a_{\lceil n/2 \rceil} + a_{\lfloor n/2 \rfloor} && [\text{Definition}] \\ &= \lceil n/2 \rceil + \lfloor n/2 \rfloor \\ &= P(\lceil n/2 \rceil), P(\lfloor n/2 \rfloor)) \\ &= n. \end{aligned}$$

- b) Claim:  $a_n = 2n$  for all  $n \geq 0$ . Proof by strong induction. The basis  $P(0)$  is given, since  $2 \cdot 0 = 0$ . For the inductive step,

$$\begin{aligned} a_n &= a_{\lceil n/2 \rceil} + a_{\lfloor n/2 \rfloor} && [\text{Definition}] \\ &= 2\lceil n/2 \rceil + 2\lfloor n/2 \rfloor \\ &= P(\lceil n/2 \rceil), P(\lfloor n/2 \rfloor)) \\ &= 2(\lceil n/2 \rceil + \lfloor n/2 \rfloor) \\ &= 2n. \end{aligned}$$

28.  $\langle 1 \rangle$  Find an unsuspecting friend and convince him or her that all men are bald; that all pigs are yellow.

**Example 1 Explained.** (You didn't really think we would make this too hard to find, did you?) Since  $P(1)$  is true, but in general  $P(n)$  is false, the error has to be in the inductive step. Figure 2.20 shows that the inductive step is all right generally, so it must break down in some particular early case. In fact, it must already break down for  $P(1) \Rightarrow P(2)$ , because  $P(2)$  is false — take any set consisting of a blonde and a brunette. So let's try to draw Fig. 2.20 in the particular case  $n = 1$ . We can't! If the known blonde is in both sets  $A$  and  $B$ , making them both 1-sets as they are supposed to be, then the other woman can't be in either set and so we cannot conclude that she is blonde. See Fig. 2.21.

*Moral:* Make sure your inductive argument really works for all  $n$  beyond the basis case(s).

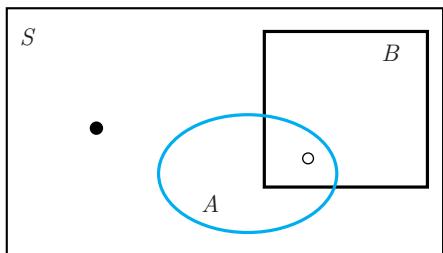


FIGURE 2.21

Why not all women are blondes.

## Supplementary Problems: Chapter 2

1. ⟨2⟩ Thought question. We seem to have said that induction is the technique to use when you have infinitely many cases to prove; induction allows you to prove them by proving them one at a time. But why not prove them all at once? We do that all the time in mathematics: when we prove that all equilateral triangles are equiangular, when we prove that  $(x+y)(x-y) = x^2 - y^2$  for all  $x, y$  by simple algebra, etc. So why waste time on mathematical induction?
2. ⟨2⟩ Many writers denigrate induction as a proof technique as follows. They say: while induction shows results to be true, it doesn't *explain* them. The proof by induction that  $\sum_{i=1}^n k = n(n+1)/2$  is often given as an example. The critics say: The reader of this proof doesn't have any better feel for why this identity is true after seeing the inductive proof than before; furthermore, one can't discover the identity by induction, but must already know it (by some more informative means) in order to apply the proof.

We feel this view sells induction short, especially when one considers the application of induction to algorithms. We have said that inductive thinking can not only be used to prove, but also to discover. If it helps you to discover, we think it also helps you to explain.

  - a) Take some algorithm we have proved by induction in this chapter, and decide which side you come down on. Does the proof help explain why the algorithm is correct or not? Give your reasons.
  - b) The traditional induction proof of  $\sum_{i=1}^n k = n(n+1)/2$  is a pretty good example for the critics of induction, and our proof at the beginning of this chapter was pretty close to standard (since it was at the beginning). We nonetheless feel that this proof does provide some explanation, or can be redone so that it does. Agree or disagree with us. Give your reasons.
3. ⟨2⟩ Let the three poles for TOH be on a circle, and suppose we add a new rule that a piece can only move to the pole immediately clockwise from its current position. Can this game be won for every  $n$ ? Give a proof or counterexample.
4. ⟨4⟩ Here is another iterative algorithm for ordinary TOH. Color the rings alternately red and white from top to bottom of the original tower. Also color the platform under the largest ring differently from that ring, color the second platform differently from the first platform, and color the third platform the same as the first. In addition to the usual rules, never move the same piece twice in a row, and never let a ring move directly on top of another ring or platform of the same color. Prove that these rules force a unique sequence of moves that result in a winning game.
5. ⟨4⟩ Let  $f$  be a function that, for every pair of real numbers  $x$  and  $y$ , satisfies
$$f(x+y) = f(x) + f(y). \quad (1)$$
  - a) Name such a function; check to see whether you are right by substituting into Eq. (1).

For any such function
  - b) Prove by induction that for integers  $n > 0$ ,
$$f(nx) = n f(x).$$
  - c) Prove that  $f(x/n) = f(x)/n$  for integers  $n > 0$ . *Hint:* Induction isn't helpful directly (try it and see why). Instead, substitute  $y$  for  $x/n$ ; now induction is helpful — in fact, you've already done it.
  - d) Show that  $f(\sum x_i) = \sum f(x_i).$
  - e) See what else you can discover about such functions. For instance, what is  $f(0)$ ? How does  $f(-x)$  relate to  $f(x)$ ?
6. ⟨4⟩ Consider the curious function defined as follows:
$$f(n) = \begin{cases} n-3 & \text{if } n \geq 1000 \\ f(f(n+6)) & \text{if } n < 1000. \end{cases}$$

What is  $f(1)$ ?
7. ⟨2⟩ The argument in Section 2.8 that all women are blonde breaks down going from  $n = 1$  to  $n = 2$  in the inductive step. Therefore, the following modification of the argument, which starts at  $n = 2$  and is clearly true there, must avoid that problem. Is the argument therefore correct? If not, why not?

Theorem. All women in the world are blonde.

**Proof.** Let  $Q(n)$  be the statement: In any set of  $n$  women, if at least two of them are blonde, then all of them are blonde. We will prove  $Q(n)$  for all  $n \geq 2$ . Now let  $N$  be the number of women in the world.  $Q(N)$  makes a claim about a lot of sets, but in particular it claims that if the set of women in the world contains at least two blondes, then all women in the world are blonde. Since there are clearly at least two blondes, and  $Q(N)$  is true (as proved below), then all women are blonde.

**Basis step ( $n = 2$ ).**  $Q(2)$  says: In any set of 2 women, if at least 2 of them are blonde, then they are all blonde. This is true because it is simply a roundabout way to say “Any 2 blonde women are blonde women.”

**Inductive step,  $Q(n-1) \implies Q(n)$ .** Take any set  $S$  of  $n$  women with at least 2 known blondes. Label the women  $w_1, w_2, \dots, w_{n-1}, w_n$  with the blondes in the middle somewhere. Now define  $T = \{w_1, w_2, \dots, w_{n-1}\}$  and  $T' = \{w_2, \dots, w_{n-1}, w_n\}$ . By  $Q(n-1)$ , both  $T$  and  $T'$  are all blondes. Since  $S = T \cup T'$ , we conclude that  $S$  is all blondes.

- 8. ⟨2⟩** Here’s another attempt to fix the original proof that all women are blonde. The reason this proof broke down was that  $P(n)$  was too tricky. (Recall that  $P(n)$  was the peculiar statement “For any set of  $n$  women, if at least one of them is blonde, then all of them are blonde.”) Let’s use the simpler statement  $Q(n)$ : Any set of  $n$  women are blonde. Now the following inductive step is valid, even when  $n + 1 = 2$ :

**Inductive step:  $Q(n) \implies Q(n+1)$ .** Take any set  $W$  of  $n + 1$  women. Consider the first  $n$  women in  $W$  to be the set  $S$  and the last  $n$  women to be the set  $T$ . Then by  $Q(n)$  all women in  $S$  are blonde and again by  $Q(n)$  all women in  $T$  are blonde. Thus all women in  $W = S \cup T$  are blonde. Thus any set of  $n + 1$  women are blonde.

- a)** Why doesn’t it matter in this inductive argument whether or not  $S$  and  $T$  overlap? (Recall that the original proof about blonde women broke down if  $S \cap T = \emptyset$ .)
- b)** But the conclusion of the induction is still false: not all women are blonde! So what is wrong with this fix?

- 9. ⟨3⟩** Consider the unusual English sentence

Buffalo buffalo buffalo.

This is a proper English sentence: The individual

words have meanings so that the whole sentence follows the rules of English grammar and makes sense. (For non-native English speakers: Buffalo is a plural noun in English referring to a certain big land animal. Buffalo is also a verb — it is a regular verb, so without an s it is plural — meaning to push around.)

Prove by induction that

Buffalo buffalo buffalo (that buffalo buffalo)\*.

is a proper English sentence. When a phrase is in parentheses and followed by a star, that means that the phrase in parentheses may appear any number of times, including 0. So, we have the cases

$n = 0$ : Buffalo buffalo buffalo.

$n = 1$ : Buffalo buffalo buffalo that buffalo buffalo.

$n = 2$ : Buffalo buffalo buffalo that buffalo buffalo that buffalo buffalo.

etc.

**Suggestion:** In your proof use the notation  $(\dots)^k$  to mean the phrase in parentheses is repeated exactly  $k$  times.

- 10. ⟨2⟩ (Assumes calculus)** In Section 2.2 we proved

$$(1+x)^n \geq 1+nx \quad (2)$$

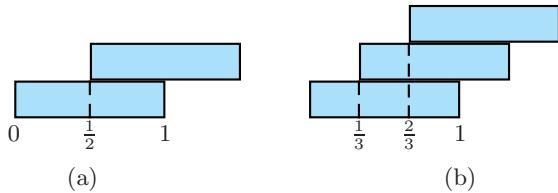
for  $x \geq 0$  and  $n$  a nonnegative integer by induction. In this case there is also a proof without induction. Indeed, there has to be, because (2) is true whenever  $n$  is a *real number*  $\geq 1$ . That is,  $n$  doesn’t have to be an integer, and induction doesn’t work for real numbers.

Use calculus to prove (2) for all reals  $n \geq 1$ . *Hint:* tangent lines and 2nd derivatives.

- 11. ⟨5⟩** You are given  $n$  bricks, all the same rectangular shape, and all with the same uniform distribution of weight. If you pile them up, one on each level, but use no mortar, what is the maximum overhang you can achieve? Prove it.

(The relevant principle from physics is: The configuration is stable if, for each brick  $B$ , the center of gravity of the set of bricks on top of  $B$  is directly over some point of  $B$ . For instance, in the left-hand configuration in Fig. 2.22, the center of gravity of all 1 bricks above the bottom brick is directly over the point marked 1, which is a point on the bottom brick (just barely!). In the right-hand configuration, the center of gravity of the

bricks above the bottom brick is also over point 1. Also, the center of gravity of the top brick is directly over point  $7/6$ , and therefore over the middle brick, which extends from  $1/3$  to  $4/3$ . Thus, if we declare that the length of each brick is one unit, we find from the figure that for  $n = 2$  the maximum overhang is at least  $1/2$ , and for  $n = 3$  it is at least  $2/3$ . Is it possible to do better? How much?)

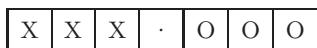


**FIGURE 2.22**

12. **<4>**  $N$  identical cars are placed randomly along a long circular track. Their gas tanks are empty. Then just enough gas for any one of these cars to make exactly one circuit of the track is divided up in a different random way and distributed to the cars. You are given a siphon and challenged to get all the way around the track clockwise in the cars. You get to pick which car you want to start in, and if you reach the next car, you may siphon gas between them before starting out again. You are told the positions of the cars and the amount of gas each has received.

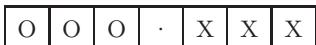
- a) Prove that challenge can always be met.
- b) Devise a recursive algorithm for determining which car to start in. Prove your algorithm correct.
- c) Devise and verify an iterative algorithm to determine which car to start in.

13. **<5>** Jump Solitaire (Frogs). You have a board consisting of  $2n+1$  squares in a row. The left-most  $n$  squares have one sort of piece on them, call it an X, and the right-most  $n$  squares have another sort of piece on them, call it O. The middle square (marked below by a dot) is empty. For instance, with  $n = 3$  you have



(Or, think of black frogs and white frogs facing each other on a log.) The pieces can move like

checker pieces — by sliding into an empty square or jumping over a single adjacent piece of the opposite type into an empty square. However, X's can only move right and O's can only move left. The object is to reverse the arrangement, i.e., to obtain



- a) Prove that the game can be won for every  $n$ . (But try the next part first; if you succeed you will have a proof by algorithm that subsumes this part.)
- b) Devise a recursive algorithm that wins Jump Solitaire and prove it correct.
- c) Devise and verify an iterative algorithm for Jump Solitaire.

14. **<5>** The **Arithmetic-Mean/Geometric-Mean Inequality** (AGI for short) says that, for any positive real numbers  $a$  and  $b$ ,

$$\sqrt{ab} \leq \frac{a+b}{2}.$$

- a) Prove this inequality by algebra. Start by considering the square of the inequality.
- b) The **generalized AGI** says: For positive numbers  $a_i$ ,

$$\sqrt[n]{\prod a_i} \leq \frac{1}{n} \sum a_i.$$

Try to prove this inequality directly from (a) by a standard induction —  $P(2)$  and  $P(n)$  together imply  $P(n+1)$ . If you succeed, let us know — we don't see how. But see [15].

15. **<3>** The generalized AGI can be proved by induction by using the following ingenious approach devised by the famous French mathematician Cauchy. Let  $P(n)$  be the  $n$ th case. We wish to prove  $P(n)$  for all  $n \geq 2$ . Note that the basis,  $P(2)$ , has already been proved in [14].

- a) Show that  $[P(2) \text{ and } P(n)] \implies P(2n)$ .
- b) Show that  $P(n) \implies P(n-1)$  as follows:
  - i) Given  $a_1, \dots, a_{n-1}$ , define

$$a_n = \left( \sum_{i=1}^{n-1} a_i \right) / (n-1).$$

Show that, therefore,

$$\frac{1}{n} \left( \sum_{i=1}^n a_i \right) = a_n.$$

Note: This sum goes to  $n$ , not  $n-1$ .

- ii) Write down  $P(n)$ , raise both sides to the  $n$ th power, and simplify.

Why does all this prove  $P(n)$  for all  $n \geq 2$ ?

16. <4> The **Cauchy-Schwarz Inequality** says: For any  $n$ -tuples  $(a_1, a_2, \dots, a_n)$  and  $(b_1, b_2, \dots, b_n)$  of real numbers,

$$\left( \sum a_i^2 \right) \left( \sum b_i^2 \right) \geq \left( \sum a_i b_i \right)^2.$$

In words, the product of the sums of the squares is greater than or equal to the square of the sum of the products. Try to prove this inequality by induction. It can be done, but the algebra is very involved. There are much simpler proofs without induction, but you need to know some vector algebra to do them.

17. <3> The ( $n$ -dimensional) **triangle inequality** states:

$$\sqrt{\sum a_i^2} + \sqrt{\sum b_i^2} \geq \sqrt{\sum (a_i + b_i)^2}.$$

Why Triangle Inequality? Because (taking  $n = 2$  for convenience) if we consider the points  $D = (0, 0)$ ,  $A = (a_1, a_2)$  and  $C = (a_1 + b_1, a_2 + b_2)$ , then this inequality says that in triangle  $ACD$ ,  $DA + AC \geq DC$ . Show, using just algebra, that the Triangle Inequality follows from the Cauchy-Schwarz Inequality.

18. <3> (Assumes calculus) Let  $[n] = \{1, 2, \dots, n\}$ . Assuming the derivative product formula  $(fg)' = f'g + fg'$ , prove the generalization

$$\left( \prod_{i=1}^n f_i \right)' = \sum_{i \in [n]} \left( f'_i \prod_{j \neq i} f_j \right).$$

19. <2> (Assumes calculus) Rolle's Theorem says:

If  $g$  is differentiable on  $[a, b]$  and  $g(a) = g(b) = 0$ , then there exists a point  $c$  between  $a$  and  $b$  where  $g'(c) = 0$ .

The Generalized Rolle's Theorem says:

If  $g$  is  $n$  times differentiable on  $[a, b]$ ,  $g^{(k)}(a) = 0$  for  $k = 0, 1, \dots, n-1$ , and  $g(b) = 0$ , then there is a point  $c$  between  $a$  and  $b$  where  $g^{(n)}(c) = 0$ .

Prove the Generalized Rolle's Theorem by induction from Rolle's Theorem.

20. <3> Define the **harmonic numbers** recursively by

$$H(1) = 1 \\ H(n) = H(n-1) + (1/n).$$

Show that

- a)  $\sum_{i=0}^m 1/(2i+1) = H(2m+1) - \frac{1}{2}H(m)$ .  
 b)  $(m+1)H(m) - m = \sum_{i=1}^m H(i)$ .  
 c)  $H(2^m) \geq 1 + (m/2)$ .

21. <4> You are given an increasing sequence of numbers  $u_1, u_2, \dots, u_m$  ( $u$  for "up") and a decreasing sequence of numbers  $d_1, d_2, \dots, d_n$ . You are given one more number,  $C$ , and asked to determine if  $C$  can be written as the sum of one  $u_i$  and one  $d_j$ .

The brute force approach of comparing all  $mn$  sums  $u_i + d_j$  to  $C$  works, but there is a much more clever way. See Algorithm SUM-SEARCH.

- a) Figure out what's going on in SUM-SEARCH and explain it.  
 b) Use induction to prove that SUM-SEARCH is correct. (This is an opportunity to invent double induction.)

### Algorithm 2.7. SUM-SEARCH

```

Input  $u_1, \dots, u_m$  increasing,  

       $d_1, \dots, d_n$  decreasing,  $C$   

Output pair [Indices of summands of  $C$ ,  

            or  $(0, 0)$  if  $C$  not obtainable]  

Algorithm SUM-SEARCH  

i  $\leftarrow m$ ; j  $\leftarrow n$   

pair  $\leftarrow (0, 0)$  [C not found yet]  

repeat until i = 0 or j = 0  

  if  $u_i + d_j = C$  then pair  $\leftarrow (i, j)$  [3-case if]  

    exit  

   $u_i + d_j < C$  then j  $\leftarrow j-1$  [Peel off a  $d_j$ ]  

   $u_i + d_j > C$  then i  $\leftarrow i-1$  [Peel off a  $u_i$ ]  

  endif  

endrepeat
```

- 22.**  $\langle 2 \rangle$  This problem should suggest yet another form of induction, **simultaneous induction**. Let sequences  $\{s_1, s_2, \dots\}$  and  $\{t_1, t_2, \dots\}$  be defined by

$$\begin{aligned}s_n &= t_{n-1} - t_{n-2} + 4, \\ t_n &= t_{n-1} + s_{n-1}, \\ s_1 &= 3, \quad s_2 = 5, \quad \text{and} \quad t_1 = 1.\end{aligned}$$

Discover formulas for  $s_n$  and  $t_n$  and prove them by induction.

In [16, Section 1.4] you were asked to prove that  $\gcd(r_1, r_2, \dots, r_k) = \gcd(\gcd(r_1, r_2, \dots, r_{k-1}), r_k)$ .

This is not hard if you know that  $a$  divides all of  $r_1, r_2, \dots, r_{k-1}, r_k$  iff the factors of  $a$  are common factors of  $r_1, r_2, \dots, r_{k-1}, r_k$ . However, this fact needs to be proved, and the easiest proof uses unique factorization theorem for integers, which we have also not proved. The next three problems offer an alternative proof using Euclid's algorithm (which is the key tool for proving unique factorization as well).

- 23.**  $\langle 3 \rangle$  It is a fact that for any  $m, n \in N$ ,  $\gcd(m, n)$  can be written in the form  $am + bn$ ,  $a, b \in Z$ . (Recall  $Z$  is all integers, including negatives.)

- a) Prove this by induction on  $n$ .  
 b) Modify some version of the gcd algorithm to find  $a, b$  while finding  $\gcd(m, n)$ .

- 24.**  $\langle 2 \rangle$  Prove:  $k|m, n \implies k|\gcd(m, n)$ . That is, not only is  $\gcd(m, n)$  the largest common divisor of  $m$  and  $n$  (that's the definition), but it is actually a *multiple* of every other common divisor. Prove this using [23].

- 25.**  $\langle 3 \rangle$

- a) Show that  $\gcd(a, b, c) = \gcd(\gcd(a, b), c)$ .  
*Hint:* Use the definition of gcd to prove  $\gcd(a, b, c) \geq \gcd(\gcd(a, b), c)$ . Then use [24] to prove  $\gcd(a, b, c) \leq \gcd(\gcd(a, b), c)$ .  
 b) Use the idea in a) to find an algorithm for  $\gcd(a_1, a_2, \dots, a_k)$  and prove it correct by induction.

- 26.**  $\langle 3 \rangle$  Recall that we defined the gcd for any integers, not just nonnegative ones. For  $m, n \in Z$ , we defined  $\gcd(m, n)$  to be the largest *positive* integer that divides evenly into both  $m$  and  $n$ . For instance,  $\gcd(9, -6) = 3$ . (We defined  $\gcd(0, 0) = 0$  as a special case.)

However, the *algorithms* we wrote to compute the gcd don't always work for negative integers. Indeed, the inputs were carefully restricted to

$m, n \in N$ , and the proofs of correctness used this restriction.

- a) Explain why  $\gcd(m, n) = \gcd(|m|, |n|)$ . Therefore, we *could* use any of our previous Euclid algorithms to find  $\gcd(m, n)$  for any  $m, n \in Z$  by simply taking absolute values of the inputs at the start.

However, it will allow us to make an instructive point about induction if we don't take absolute values at the start. EUCLIDZ is an algorithm that finds  $\gcd(m, n)$  without taking absolute values until the end.

- b) Apply EUCLIDZ to  $(m, n) = (45, -26)$ . To  $(m, n) = (-45, 26)$ .  
 c) Prove that EUCLIDZ is correct. Here is the problem: You can't do induction directly on  $Z$  the way you can on  $N$ . So, what aspect of the variables  $m, n$  can you do induction on?

### Algorithm 2.8. EUCLIDZ

<b>Input</b> $m, n$	[Any integers]
<b>Output</b> $answer$	[ $= \gcd(m, n)$ ]
<b>Algorithm</b> EUCLIDZ	
<b>function</b> gcdz( <b>in</b> $i, j$ )	
<b>if</b> $j = 0$ <b>then</b> $\text{gcdz} \leftarrow  i $	
<b>else</b> $\text{gcdz} \leftarrow \text{gcdz}(j, i - \lfloor i/j \rfloor j)$	
<b>endfunc</b>	
$answer \leftarrow \text{gcdz}(m, n)$	[Main alg.]

- 27.**  $\langle 4 \rangle$  The **associative law** of addition says that, for all real numbers  $x, y$  and  $z$ ,

$$x + (y + z) = (x + y) + z.$$

The **generalized associative law** says that, no matter how you parenthesize the addition of real numbers  $x_1, x_2, \dots, x_n$ , the sum is the same — so long as the numbers appear in the same order. Thus, for instance, the generalized law says that

$$(x+y) + (z+w) = [(x+y) + z] + w$$

and

$$\begin{aligned}(x_1 + (x_2 + (x_3 + (x_4 + x_5)))) &= \\ (x_1 + x_2) + ((x_3 + x_4) + x_5).\end{aligned}$$

Of course, equality would still be true if the order of numbers was changed, but that involves the generalized commutative law [28].

Assuming only the associative law, prove the generalized associative law. (Despite how familiar these facts are to us, the proof is subtle.)

**28.**  $\langle 4 \rangle$  The **commutative law** of addition says that, for any real numbers  $x$  and  $y$ ,  $x+y = y+x$ . The **generalized commutative law** says that, no matter how you order any real numbers  $x_1, x_2, \dots, x_n$ , the sum  $x_1 + x_2 + \dots + x_n$  is the same. (Note that we must assume the generalized associative law from [27] just to state the generalized commutative law, for we have left out all parentheses.) Assuming the commutative law, prove the generalized commutative law.

**29.**  $\langle 2 \rangle$  Is this following principle of “strong induction for the *real* numbers” valid?

In order to prove  $P(x)$  for all real numbers  $x \geq 0$ , it suffices to prove

- i)  $P(0)$ , and
- ii) for all  $x > 0$ ,  
 $[P(y) \text{ is true for all } y \text{ in } 0 \leq y < x] \implies P(x)$ .

**30.**  $\langle 3 \rangle$  The **Well Ordering Principle** says that every nonempty set of nonnegative integers has a least element. In an advanced course, all of induction is typically developed from this Principle. Prove that the Well Ordering Principle is equivalent to the Weak Induction Principle of Section 2.2 as follows:

a) Prove that Well Ordering  $\implies$  Weak Induction by supposing that Weak Induction is false and getting a contradiction. So suppose that there is some sequence  $P(0), P(1), \dots$ , of propositions such that Conditions (i) and (ii) from p. 147 hold, yet the set

$$S = \{n \in N \mid P(n) \text{ is false}\}$$

is nonempty. By the Well Ordering Principle,  $S$  contains some smallest number  $n'$ . Then  $P(n'-1)$  is true. Apply Condition (ii).

b) Prove that Weak Induction  $\implies$  Well Ordering by contradiction. Suppose that the non-negative integers contained a nonempty subset  $S$  without a least element. Let  $P(n)$  be the proposition that none of the integers from 0 to  $n$  is in  $S$ . Show that  $P(0)$  is true and that  $P(n) \implies P(n+1)$  is true. By Weak Induction, what contradiction can you obtain about  $S$ ?

c) Reprove [1 and 2, Section 2.5] using the Well Ordering Principle. These two examples show especially well that Well Ordering provides a context in which it is natural to do induction by indirect proofs. (See the discussion of indirect proofs on p. 64.)

# Graphs and Trees

## 3.1 Examples and Terminology

---

One way to judge the importance of a branch of mathematics is by the variety of unrelated problems to which it can be applied. By this measure, the subject of this chapter is surely one of the most important in discrete mathematics. To illustrate, we begin by presenting three quite different problems to which graph theory can be applied. One of these is a children's puzzle; one is a problem on the properties of sequences; and the last is a common problem in computer science. (By the way, the first example in this book (in the Prologue) also was an example of graph theory.)

In presenting these three problems we'll introduce informally some of the terminology of graph theory. We think you will understand these examples without much elaboration of the terminology. Later in this section we'll formalize the terminology used in the examples as well as some of the other terminology of graph theory.

First we'll state the three problems and then consider their solutions.

### EXAMPLE 1

#### The Wolf, Cabbage, Goat and Farmer Problem

A farmer is bringing a wolf, a cabbage and a goat to market. The farmer arrives with all three at one side of a river that they need to cross. The farmer has a boat which can accommodate only one of the three. (It's a big cabbage.) But if the wolf is left alone with the goat, the wolf will eat the goat. And if the goat is left alone with the cabbage, the goat will eat the cabbage. (The wolf can be left alone with the cabbage because wolves don't like cabbage.) How can all three get across the river intact?

### EXAMPLE 2

When checking into a hotel nowadays, sometimes a guest receives a four-digit "combination", not a key. On the door of each room is a keypad. Any time those four

digits are entered in the proper sequence (regardless of what has been entered previously), the door opens. The problem is this: A burglar wishes to break into a certain room. What is the minimum number of digits the burglar must use to be *certain* of entering the correct four-digit key in sequence?

### EXAMPLE 3

You are presented with a sequence of distinct words or names, one at a time, and wish to alphabetize them. Design an algorithm to do so.

These are three quite different problems. But all can be approached using a similar mechanism — *graphs* — as we shall now demonstrate. (*Caution.* “Graph” as used in this chapter has a quite different meaning from the familiar one that depicts the values of a function, say  $f(x)$ , for a range of values of  $x$ .)

**Solution to Example 1** This problem can be solved without too much trouble by informed trial and error. But we are interested here, as throughout this book, in systematic (i.e., algorithmic) approaches. Denote the four principals by W, C, G and F. According to the rules of the problem, the following combinations are legal on either side of the river:

WCGF	WC	W	
WCF	GF	C	
WGF		G	
CGF		$\emptyset$	(1)

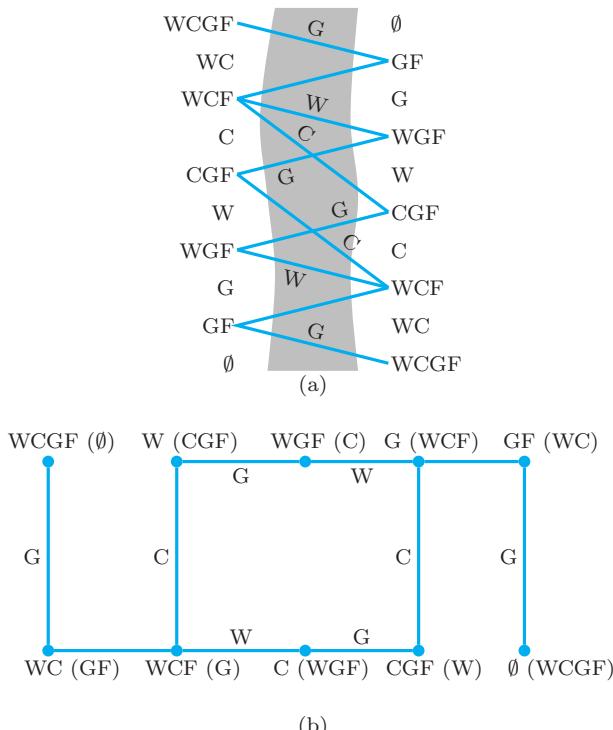
where  $\emptyset$ , the empty set, represents the situation where none of the four is on one side of the river (and therefore all four are on the other side). Note that WF and CF are not included because, while both are allowable, WF on one side implies CG — which is illegal — on the other side, and CF implies WG, which is also illegal.

Figure 3.1 displays our solution. In part (a) we show the 10 legal combinations on each side of the river with lines showing how a trip in the boat with the farmer and, perhaps, a passenger leads from one configuration to another. In each case the trip could proceed in either direction. For example, the line from WCF on the left side to CGF on the right side could represent the cabbage being transported in either direction.

In part (b) we display the same information without showing the river explicitly. Each point or *vertex* in (b) represents one of the 10 cases in (1) on the near bank of the river where all four start. Also shown for each vertex is the state on the far bank of the river at the same time, assuming that no one is in transit across the river. Each line or *edge* represents a trip by the farmer from one side to the other with or without a passenger (e.g., the edge between WGF and G represents the farmer taking the wolf from one bank to the other). The label on the edge designates who besides the farmer is in the boat. Since we start with WCGF on the near bank and wish to end with  $\emptyset$  on the near bank, our object must be to find a *path* in (b) from WCGF to  $\emptyset$  because this will be a solution to the problem [1].

From Fig. 3.1b, you can see that there are two solutions:

$$\text{WCGF} \rightarrow \text{WC} \rightarrow \text{WCF} \rightarrow \text{W} \rightarrow \text{WGF} \rightarrow \text{G} \rightarrow \text{GF} \rightarrow \emptyset,$$



The Wolf, Cabbage, Goat, and Farmer Problem. (a) The allowable states are shown on both sides of the river. Each label in the river represents the passenger in the boat besides the farmer. (b) Representation of allowable states as a graph. The label at each vertex is a state on the near bank. The label in parentheses corresponds to the state on the far bank. Each edge label represents a passenger in the boat besides the farmer.

and

$$\text{WCGF} \rightarrow \text{WC} \rightarrow \text{WCF} \rightarrow \text{C} \rightarrow \text{CGF} \rightarrow \text{G} \rightarrow \text{GF} \rightarrow \emptyset.$$

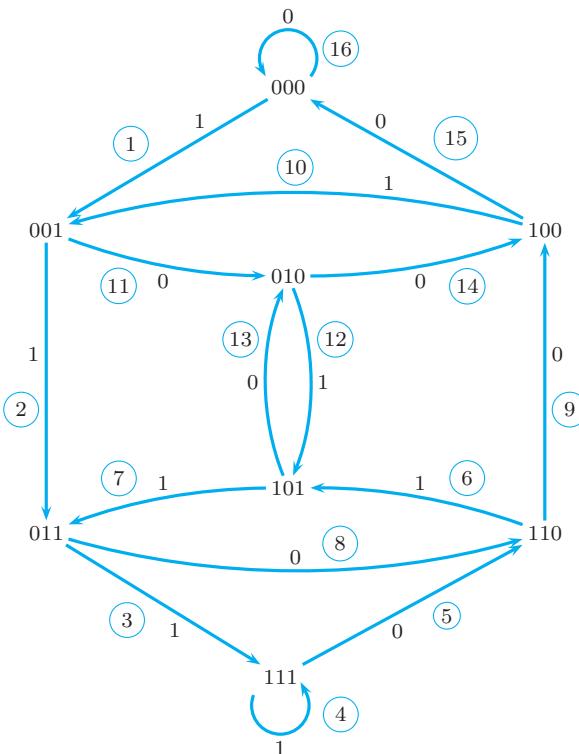
The structure shown in Fig. 3.1b is a *graph* (an *undirected* graph because the edges have no arrows on them). Structures analogous to this are the subject of this chapter. The graph in Fig. 3.1b is usually called a **state graph** because the vertices represent the various configurations (states) possible, and two vertices are joined by an edge if a single move gets you from one state to the other. In this example, each state describes the whereabouts of the four participants, and each edge represents a single boat trip. ■

**Solution to Example 2** Since there are 10,000 four-digit combinations (0000 to 9999), the total number of digits in all the combinations is 40,000. But surely the burglar doesn't have to enter all 40,000 digits to get into the room. For example,

**FIGURE 3.1**

after dialing, say, 6083, then by dialing a 7, the burglar can check whether the correct combination is 0837. This idea of overlapping combinations should make it possible to test all the combinations with a lot fewer than 40,000 digits. But what is the minimum?

To make the discussion easier, let's simplify the problem by restricting the digits in the combination to 0's and 1's (the *binary* case). (This simplification means that the hotel could have at most 16 rooms because there are only 16 four-digit binary numbers!) We denote the binary digits in the combination by the usual contraction *bits*.



**FIGURE 3.2**

The Hotel Burglar Problem (binary case).

Figure 3.2 displays a graph in which each vertex has a label, which is one of the eight three-bit sequences of 0's and 1's. Each edge has a direction, so we call this graph a directed graph, or *digraph*. Each edge also has a label which is 0 or 1. The digraph is constructed so that there is an edge linking two vertices if and only if the right-most two bits of the label of the vertex at the tail of the arrow are the same as the left-most two bits at the vertex at the head of the arrow. The label on each edge is the last bit of the vertex label at the head of the arrow. This digraph contains two *loops*, which are edges that begin and end at the same vertex.

Now consider the edges in the graph in the order given by the sixteen circled numbers. This gives us a path which traverses each edge of the digraph once and only once and ends back at the same vertex (000) from which it started. (A path

which ends where it begins is called a cycle.) The bits of the labels of the consecutive edges from 1 to 16 are

$$1111\ 0110\ 0101\ 0000. \quad (2)$$

The interesting thing about sequence (2) is that it contains all 16 four-bit binary sequences (i.e., room numbers in the hotel) if we allow wraparound from the end of the sequence to the beginning (e.g., 1001 is in the middle; 0001 starts at the end and ends at the beginning). It follows therefore that to open a door whose “combination” is a four-bit sequence, we need only dial the 19-bit sequence:

$$1111\ 0110\ 0101\ 0000\ 111$$

(19 because, of course, wraparound isn’t possible on an actual lock). Do you see that 19 bits is the shortest possible sequence for testing all 16 combinations [8]?

Can this solution be generalized to the problem as originally stated where the digits 0, 1, 2, …, 9 are allowed? Can the idea of using cycles on graphs be generalized to any set of digits and any length sequence of the digits? The answer to both questions is Yes. We don’t discuss the general case in this book, but we shall return to the binary case in Section 3.3. The sequences that we’ve been discussing in this example are usually called **de Bruijn cycles**, or full cycles.

Since we’re interested in sequences of four bits, could we have used a graph whose vertices have four-bit labels instead of the graph shown in Fig. 3.2, which has three-bit labels? Sure, but it would have had 16 vertices instead of 8 (because there are 16, four-bit binary numbers) and 32 edges instead of 16. The graph would therefore have been quite a bit more complicated than in Fig. 3.2. ■

**Solution to Example 3** Consider the list of words in Fig. 3.3a. To sort these words we have constructed (Fig. 3.3b) a special kind of graph called a *tree*, a structure we introduced in Example 3, Section 1.1. Since all the edges in the tree in Fig. 3.3 implicitly point down, we might argue that a tree is really a digraph. We’ll return to this point in the subsection below on terminology.

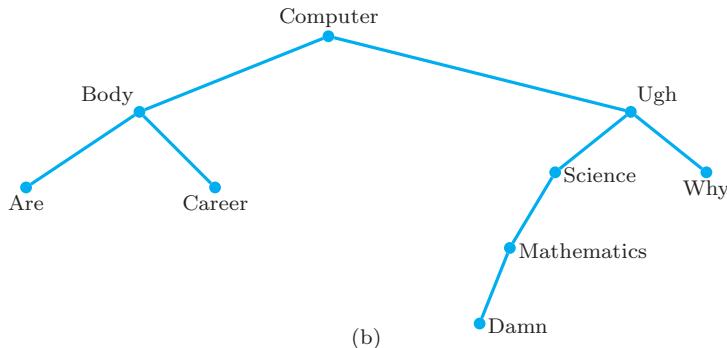
Algorithm 3.1 shows how to construct a tree like the one in Fig. 3.3b. Fig. 3.3c illustrates the “growth” of the tree at various stages.

How can we use the tree in Fig. 3.3b to alphabetize the list? All we have to do is list the words in the tree from left to right regardless of their height (or *level*) on the tree. We hope you’re not very impressed with this demonstration since the success of this idea depends on a carefully drawn tree in which, for example, the “computer” node is to the left of the “mathematics” node. How could we make this idea work if there were hundreds or thousands of words in our list instead of just 9? Actually building trees in order to sort a list is an important method of sorting; in Section 3.7 we’ll present an algorithm to alphabetize the words which doesn’t depend on how carefully the tree is drawn.

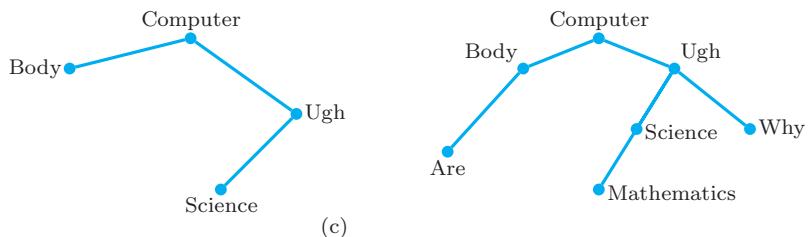
The alphabetizing method illustrated in this example is only one of many possible ways of *sorting* a list, a problem which is ubiquitous in computer science and which we’ll return to in Section 7.7. The method shown here is the basis of important and efficient techniques of sorting. ■

Computer  
Ugh  
Science  
Body  
Mathematics  
Are  
Why  
Damn  
Career

(a)



(b)



(c)

**FIGURE 3.3**

Tree structure for words to be sorted: (a) list of input words; (b) complete tree; (c) tree at intermediate stages.

These three very different problems are only representative of those that can be solved — or, at least, approached — using the structures we call graphs. In the subsequent sections of this chapter we shall discuss the main properties of graphs and consider various problems and algorithms related to the use of graphs. But first, in the subsection below, we'll define carefully the terminology used in the previous examples as well as other terminology that will be used later in this chapter.

## Terminology

We begin by defining graphs and digraphs and their constituents more carefully than we did in our examples.

*Caution.* There is considerable variation in the terminology of graph theory from book to book. The terminology we'll use here is, if not universal, at least quite common and accepted.

### Algorithm 3.1

#### BuildTree

<b>Input</b> $n$	$[n > 0; \text{number of words}]$
$w_i, i = 1, \dots, n$	$[\text{Words to be alphabetized}]$
<b>Output</b> Tree of words	$[\text{As in Fig. 3.3b}]$
<b>Algorithm</b> BUILDTREE	
<b>for</b> $i = 1$ <b>to</b> $n$	
<b>if</b> $i = 1$ <b>then</b> construct a tree root	$[\text{Root is top-most vertex}]$
and place $w_1$ at root	
<b>else</b> starting at the root	$[w_i < \text{ word means prior}$
move left if $w_i <$ word	$\text{alphabetically}]$
at vertex and right	
otherwise until a	
place with no vertex	
is reached; construct	
a new vertex and	
place $w_i$ at it	
<b>endif</b>	
<b>endfor</b>	

---

**Definition 1.** A **graph** or **digraph**  $G$  is a finite set  $V$  of points called **vertices** and a finite set  $E$  of **edges** such that each edge links a pair of vertices or, in the case of a **loop**, a vertex with itself. In a graph the edges have no indicated direction; in a digraph each edge is directed from one vertex to the other.

---

From the definition it follows that a sensible – and often useful – notation instead of  $G$  is  $G(V, E)$ .

Definition 1 does not say what notation we shall use for individual vertices and edges. Typically vertices are  $u, v, w, \dots$  or  $v_1, v_2, \dots, v_n$ . Similarly, edges are sometimes just  $e$  or  $e_1, e_2, \dots, e_m$ . However, it is important to know which vertices each edge links. Therefore, we usually write

$$e_k = \{v_i, v_j\} \quad (3)$$

when  $e_k$  links  $v_i$  and  $v_j$  in a graph, and

$$e_k = (v_i, v_j) \quad (4)$$

when  $e_k$  links  $v_i$  to  $v_j$  in a digraph. The point is that the set notation (3) is unordered, corresponding to the undirected nature of edges in a graph, while the ordered pair notation (4) corresponds to the directed nature of edges in a digraph.

Note, in particular, that  $\{v_i, v_j\}$  and  $\{v_j, v_i\}$  represent the same edge in a graph but that  $(v_i, v_j)$  and  $(v_j, v_i)$  represent two distinct edges in a digraph (why?).

Our convention is that for  $e = (u, v)$ ,  $u$  is the **tail** of  $e$  and  $v$  is the **head**, that is, the edge goes from  $u$  to  $v$ . Fig. 3.4b illustrates a digraph, with directions shown by arrows.

For a graph or digraph with  $n$  vertices and  $m$  edges, we may naturally write

$$V = \{v_1, v_2, \dots, v_n\} \quad \text{and} \quad E = \{e_1, e_2, \dots, e_m\}.$$

As an example of this formalism, the graph in Fig. 3.4a has a vertex set

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

and an edge set

$$\begin{aligned} E &= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} \\ &= \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_5\}, \{v_2, v_5\}, \{v_2, v_4\}, \{v_3, v_6\}, \{v_4, v_6\}\}. \end{aligned}$$

Similarly for the digraph in Fig. 3.4b,

$$V = \{v_1, v_2, v_3, v_4, v_5\}$$

and

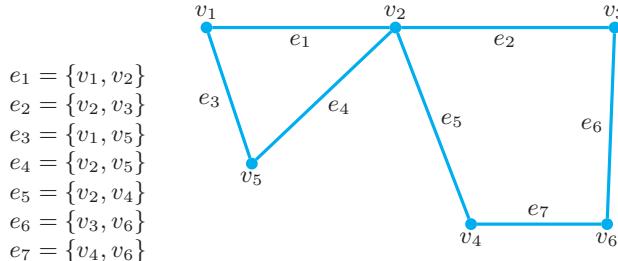
$$\begin{aligned} E &= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\} \\ &= \{(v_1, v_2), (v_2, v_3), (v_3, v_2), (v_4, v_1), (v_4, v_3), (v_1, v_5), (v_5, v_1), (v_4, v_5)\}. \end{aligned}$$

When  $V$  is empty and therefore  $E$  is also empty, we have a **null graph**. As with the empty set, we denote it by  $\emptyset$ . In this chapter we shall call a member of  $V$  a **vertex** when discussing graphs but a **node** when discussing trees. Similarly, a member of  $E$  will be called an **edge** in a graph and a **branch** in a tree.

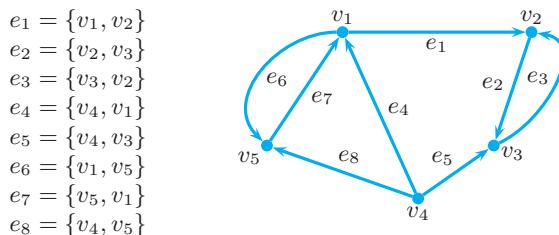
Some books use the notation  $v_i v_j$  for an edge linking  $v_i$  and  $v_j$  in either a graph or a digraph (with  $v_i$  the tail when the edge is in a digraph). Although context usually indicates whether a graph or digraph is intended, we won't use this notation because of the ambiguity.

When our definitions and theorems apply to both graphs and digraphs, as they often do, it becomes tedious to say "graph or digraph" so we'll use instead (di)graph to refer to both. There is a problem with representing undirected edges as sets. The usual definition of set (as in Section 0.1) does not allow repeated elements, so how do we represent loops? The answer: we allow edges to be multisets; thus  $\{v, v\}$  is a loop at  $v$ .

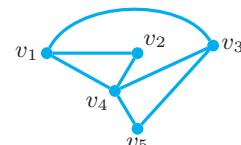
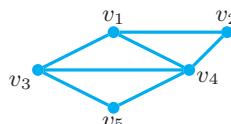
Similarly, since Definition 1 says that  $E$  is a *set* of edges, our definition only allows a single edge between a pair of vertices in a graph or a single edge in each direction for a digraph. (Di)graphs with this restriction and no loops are called **simple graphs**. Sometimes **multigraphs**, in which there can be more than one edge between a pair of vertices in a graph or in the same direction between two vertices of a digraph, are useful. Our definition will also provide for multigraphs if we allow  $E$  to be a multiset. But for multigraphs our notation for the edges of a graph does not work since two or more edges linking the same two vertices would be designated the same way. So, for multigraphs we would only refer to the edges linking the same two vertices as  $e_1, e_2, \dots$ .



(a)



(b)



Edges for both graphs:  $\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_4\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_5\}$

(c)

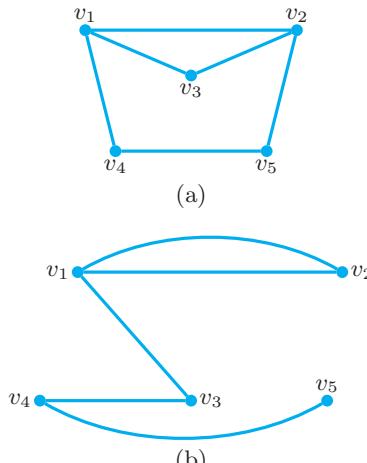
**FIGURE 3.4**

(a) An undirected graph with six vertices and seven edges; (b) a directed graph with five vertices and eight edges; (c) two isomorphic graphs.

Our convention in the remainder of this chapter is that we do not intend to exclude multigraphs unless we say so, even though we will mostly use the  $\{v_i, v_j\}$  notation that is only fully satisfactory for simple graphs. Fig. 3.5 shows an example of a simple graph and two multigraphs.

When are two (di)graphs, which look different when they are drawn, really the same? We say that two graphs are *isomorphic* (*iso* meaning equal; *morph*, structure) if they have the same number of vertices and if the vertices of the two (di)graphs may be assigned labels so that the edge sets are identical. Fig. 3.4c shows a pair of isomorphic graphs.

Back in Section 0.1 we introduced the notion of a binary relation. A digraph may be used to represent any binary relation on a finite set. Conversely, any digraph may be interpreted as a binary relation [18]. The vertices of the digraph represent the elements in the set and the edges represent the relation itself. Thus if  $a$  and  $b$



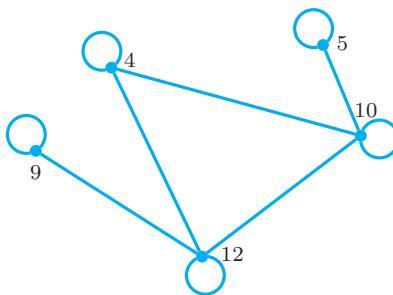
**FIGURE 3.5**

- (a) A simple undirected graph;  
 (b) an undirected multigraph; (c) a directed multigraph with a loop.

are two elements of a set, then there is an edge from vertex  $a$  to vertex  $b$  if and only if  $(a, b) \in R$  for the relation  $R$ . A reflexive relation is one for which there is a loop at every vertex. A symmetric relation is one for which, whenever there is an edge from  $a$  to  $b$ , there is a corresponding edge from  $b$  to  $a$ . Thus a symmetric relation may be represented by a graph instead of a digraph, as illustrated in Fig. 3.6. How would the transitivity of a relation be expressed graphically [17]? Because of the advantages of pictorial representations generally, depicting a relation as a (di)graph is often a way to glean insights about the relation not discernible from its definition as a set of ordered pairs.

We say that two vertices of a graph,  $v_i$  and  $v_j$  are **adjacent** or **linked** if  $\{v_i, v_j\} \in E$ . For a digraph two vertices are adjacent if either  $(v_i, v_j)$  or  $(v_j, v_i)$  is in  $E$ . For example, in Fig. 3.4a, vertices  $v_1, v_3, v_4$  and  $v_5$  are adjacent to vertex  $v_2$  and, in Fig. 3.4b, vertices  $v_1$  and  $v_3$  are adjacent to vertex  $v_2$ . If  $v$  is any vertex of a (di)graph, we define  $A(v)$  to be the set of vertices adjacent to  $v$ . Thus in Fig. 3.4a  $A(v_2) = \{v_1, v_3, v_4, v_5\}$ . The vertices in  $A(v)$  are also called the **neighbors** of  $v$ .

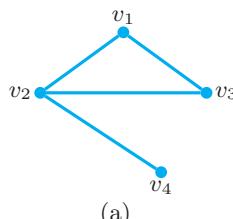
Let  $G(V, E)$  be a (di)graph. Then  $G'(V', E')$  is a **subgraph** of  $G$  if  $V'$  is a subset of  $V$  and  $E'$  is a subset of  $E$  whose edges join only vertices in  $V'$ . If either  $V' \neq V$  or  $E' \neq E$ , then the subgraph is said to be **proper**. If  $V' = V$ , the subgraph is said to **span**  $G$ . (Recall the mention of a spanning tree in Example 3,



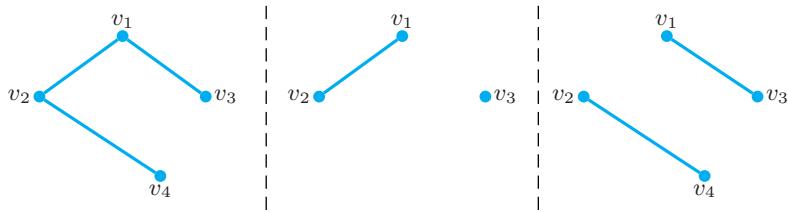
A symmetric relation as a graph. Let  $S = \{4, 5, 9, 10, 12\}$  and the relation  $R$  be such that  $(a, b) \in R$  holds only when  $a$  and  $b$  have a prime factor greater than 1 in common. This graph represents the relation  $R$ .

**FIGURE 3.6**

Section 1.1, which was about building a fiber optics network. We shall discuss spanning trees in Section 3.7.) Fig. 3.7 illustrates the idea of a subgraph.



(a)



(b)

**FIGURE 3.7**

- (a) A graph and  
(b) three of its  
subgraphs.

The **degree** of a vertex  $v$  in a (di)graph is the number of edges *incident* on it (i.e., the number of edges for which  $v$  is one of the pair of vertices which define the edge). For a digraph we have also the **indegree** and the **outdegree** of a vertex which are, respectively, the number of edges which terminate on or originate at the vertex. In Fig. 3.5c, the vertex  $v_5$  has degree 3, indegree 2, and outdegree 1. We will sometimes denote the degree of a vertex  $v$  by  $\deg(v)$ . When  $\deg(v) = 0$ , the vertex is said to be **isolated**.

But how does our definition of degree apply to loops? Does a loop at a vertex add one to the degree (only one edge is involved) or does it add two (each end of the loop impinges on the vertex)? There's no obvious correct answer to this question. In mathematics generally you can define things as you wish, the only caveat being that the definitions should be sensible and useful and shouldn't conflict with other definitions. In this chapter we choose to say that a loop adds two to the degree of a vertex because that decision turns out to be convenient for most purposes, in particular in the theorem just below. To summarize: The degree of a vertex is the number of non-loop edges incident on that vertex, plus 2 for each loop at the vertex. For instance, vertex  $v_2$  in Fig. 3.5c has degree 5, indegree 3, and outdegree 2. An important result which follows from our definition of degree is:

**Theorem 1.** Let  $G(V, E)$  be a (di)graph, where  $V = \{v_1, v_2, \dots, v_n\}$ .

Then

$$2|E| = \sum_{i=1}^n \deg(v_i). \quad (5)$$

**PROOF** Since each edge of a (di)graph joins two vertices, if we add up the degrees of all the vertices (the right-hand side of Eq. (5)), we will have counted each edge twice. Hence this sum equals the left-hand side of Eq. (5). ■

Simple, isn't it? Yes, but only because of the way we have defined the degree of a loop. If we had defined this so that a loop only added one to the degree of a vertex, Theorem 1 would be true only for loopless (di)graphs [20].

To see whether you really understand what Eq. (5) says, show that the number of people at a party who shake hands with an odd number of people must be even. Let each vertex of a graph represent a person. Then, to answer this question, what should an edge of the graph represent? This problem is explored further in [19].

We can now use the basic vocabulary of graphs to define some important concepts in graph theory, some of which we introduced informally in the previous examples.

**Definition 2.** A **path**  $P$  in a simple (di)graph is a sequence of vertices

$$(v_0, v_1, v_2, \dots, v_n)$$

such that

$$\{v_i, v_{i+1}\} \in E \quad (\text{for a graph}) \quad \text{or} \quad (v_i, v_{i+1}) \in E \quad (\text{for a digraph}),$$

for  $i = 0, 1, \dots, n - 1$ .

In this definition the edge  $\{v_i, v_{i+1}\}$  or the directed edge  $(v_i, v_{i+1})$  is said to be *on*  $P$  and  $P$  is said to **traverse** the edge. (Why doesn't this definition work for multigraphs [22]?) We also speak of "traversing a path", when we think of someone or something traveling the edges in order,  $\{v_0, v_1\}$ ,  $\{v_1, v_2\}$ , etc. More formally, such a traversal is a listing of the edges in order.

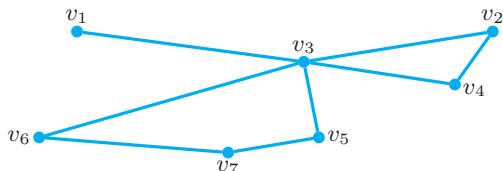
A path with no loops or repeated edges is called **simple**. If  $v_0 = v_n$  in Definition 2, then the path is a called a **cycle**. A simple cycle is one where there are no repeated edges and only the vertex at which the cycle begins and ends is repeated. For example, in Fig. 3.4a,  $(v_1, v_2, v_3, v_2)$  is a path but not a simple one because the edge  $\{v_2, v_3\}$  is repeated. Similarly  $(v_2, v_3, v_2)$  is a cycle but not a simple cycle for the same reason. Note that every graph with an edge has a cycle like  $(v_2, v_3, v_2)$ . Consequently, a graph is said to be **acyclic** if it has no *simple* cycles. How do you suppose you would define acyclic for digraphs [25]?

In a graph, if  $(v_i, \dots, v_j)$  is a path, then so is  $(v_j, \dots, v_i)$ . We say that  $(v_i, \dots, v_j)$  can be traversed in either direction. But in a digraph, if  $(v_i, \dots, v_j)$  is a path, that says nothing about  $(v_j, \dots, v_i)$ . In a digraph, a path traversal must go in the direction of the arrows. Fig. 3.8 illustrates the terminology of paths and cycles, including some terms (Eulerian, Hamiltonian) to be discussed in Section 3.3.

Here's an example that illustrates how the idea of a cycle can be useful. Suppose that you had some data from biological experiments that indicates when two segments of the same strand of some species' DNA overlap (i.e., intersect). Suppose further that you want to test the hypothesis that the DNA is, in fact, a *linear* structure (i.e., that it has no branchings or crossing portions). You are interested, therefore, in the *intersection structure* of the DNA, that is, in determining which segments overlap which others. As "intersection" is just a relation between sets, you may represent the known segments as vertices of a graph with an edge between two vertices meaning that the segments do overlap (i.e., intersect).

The intersection structure of linear sets (that is, sets that are segments of a line) is limited. That is, if segment  $A$  intersects segment  $B$  which intersects segment  $C$  but  $A$  does not intersect  $C$ , then no segment  $D$  can intersect  $A$  and  $C$  without also intersecting  $B$  (Fig. 3.9a). This is the same as saying that the graph of the intersection relation among linear sets cannot contain a cycle with four edges (a four-cycle) unless there is also a cycle with three edges involving the same sets (Fig. 3.9b). To see whether your biological data is consistent with the hypothesis of linearity, you could draw the intersection graph to find out whether there are any four-cycles with no related three-cycles. If you found a four-cycle without a related three-cycle, that would mean the structure of the DNA could not be linear. What if you didn't find such a four-cycle? Actually, this would *not* prove that the structure *is* linear. A few more conditions (which we won't state here) must be added before you have a necessary and sufficient condition for linearity.

We now come to a group of definitions about connectedness.

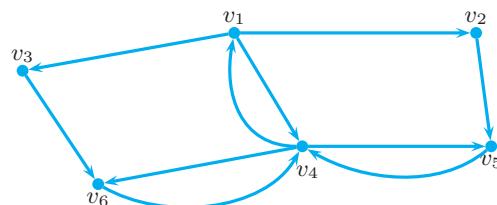


Some paths between  $v_1$  and  $v_5$ :  
 $(v_1, v_3, v_5)$   
 $(v_1, v_3, v_6, v_7, v_5)$   
 $(v_1, v_3, v_2, v_4, v_3, v_6, v_7, v_5)$

Some cycles:  
 $(v_3, v_2, v_4, v_3)$   
 $(v_3, v_6, v_7, v_5, v_3)$

An Eulerian path (all edges):  $(v_1, v_3, v_6, v_7, v_5, v_3, v_4, v_2, v_2, v_3)$

(a)



Some paths between  $v_1$  and  $v_4$ :  
 $(v_1, v_4)$   
 $(v_1, v_3, v_6, v_4)$

Some cycles:  
 $(v_1, v_4, v_1)$   
 $(v_4, v_6, v_4)$   
 $(v_1, v_3, v_6, v_4, v_1)$   
 $(v_4, v_5, v_4)$

A Hamiltonian path (all vertices):  $(v_3, v_6, v_4, v_1, v_2, v_5)$

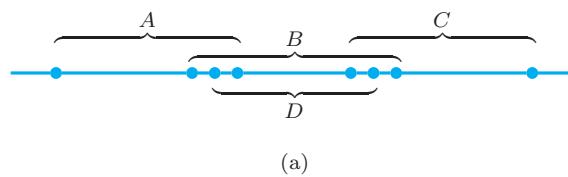
(b)

**FIGURE 3.8**

Paths and cycles:  
(a) in a graph;  
(b) in a digraph.

**Definition 3.** A graph is **connected** if for every pair of vertices  $u, v$  with  $u \neq v$  there is at least one path between them. A digraph is **strongly connected** if for each such pair there is a path in each direction. A digraph is **weakly connected** (or, sometimes, just connected) if the **underlying graph** (obtained by deleting all the directions) is connected.

Fig. 3.10 illustrates the definition of connected (di)graphs. Note that being connected is a property of graphs, not of pairs of vertices. For instance, if  $\{u, v\}$  is an edge, we say that  $u$  and  $v$  are adjacent but not that they are connected.

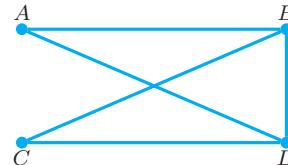


(a)

**FIGURE 3.9**

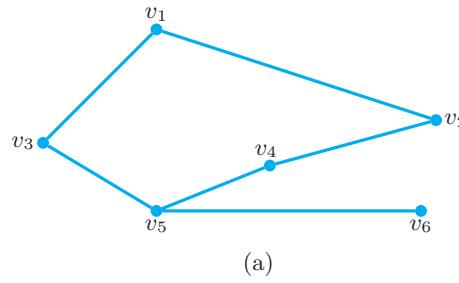
Linear sets.

(a) Four overlapping segments on a straight line; (b) a graph in which vertices represent segments and an edge between two vertices means that the segments overlap.



(b)

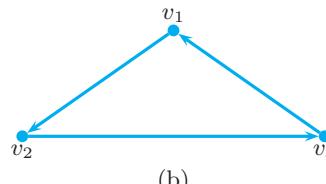
$(A, B, C, D, A)$  is a four-cycle.  
 $(A, B, D, A)$  and  $(B, C, D, B)$  are three-cycles.



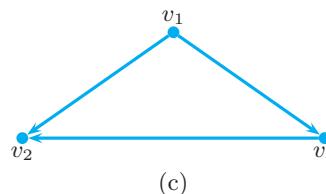
(a)

**FIGURE 3.10**

Connected (di)graphs. (a) A connected simple graph. If the edge  $(v_5, v_6)$  were deleted, the vertex  $v_6$  would be isolated and the graph would no longer be connected. (b) A strongly connected digraph. (c) A weakly connected digraph.



(b)



(c)

## Trees

Trees, although only a special type of graph, are so important that we give them equal billing in the title of this chapter.

There are, in fact, two different definitions of trees, a digraph version and a graph version. The digraph definition is the one most often used by computer scientists; the graph version is most often used by mathematicians.

The digraph definition is that a digraph is a tree if

- with edge directions removed, the resulting graph is connected and acyclic;
- every vertex  $v$  except one, called the *root*, has indegree 1; and the root has indegree 0.

Actually, requiring both connectedness and no cycles is redundant; we leave the explanation to [29].

As in Fig. 3.3, we will almost always display trees “upside down” with the root at the top and with all edges proceeding from top to bottom. We don’t do this just to be perverse, but because it is convenient to “grow” trees from top to bottom. So even though a tree as just defined is a digraph, there is no ambiguity if we omit the arrows. Mathematicians often call the digraph version of a tree a **rooted directed tree**.

Trees as digraphs can be used to represent a wide variety of situations. One example is the structure of a game in which the root represents a position in the game and each vertex represents a later position after a move by one player. (See [34–39, Section 3.7] and [49–53, Supplement].) Another familiar use of trees is for family trees. In this case, a tree is used to represent relations (quite literally!) among people.

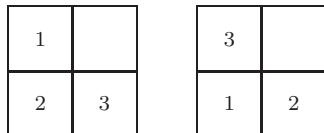
The graph definition of a tree is: A tree is a connected, acyclic graph. In this case requiring a tree to be both acyclic and connected is *not* redundant [29c]. Computer scientists often call this kind of tree a **free tree**. It will always be clear from context whether the digraph or graph version of a tree is being discussed.

## Problems: Section 3.1

---

1. <1> Were you convinced by our argument that a path in Fig. 3.1 from WCGF on one bank to WCGF on the other solves the problem? If during transit of a passenger from one bank to the other, a disallowed state occurs on either bank, then our “solution” would not be correct. Argue that this situation can never occur.
2. <2> Suppose that we add a lion to the wolf, cabbage, goat, and farmer problem. The lion eats wolves and goats but not cabbages (and, of course, not farmers either). Can you find a way to get all four to the far side of the river or prove that it can’t be done?
3. <2> The following diagram is a much simplified version of the “15 puzzle”, shown in two positions.

Each square with a number in it is a little chip of plastic and the blank square is an empty space. A move consists of shifting one of the adjacent plastic chips into the empty space. You are given the game in the configuration on the left. The goal is to get it into the configuration on the right.



You can probably solve this game in your head, but the purpose here is to develop a general approach using graph theory. Define a “state” of the game and what it means for two states to be adjacent.

Draw the associated state graph. Use it to show that winning the game is possible. What is the minimum number of moves needed to win?

4. (2) The Jealous Husbands Problem. Two married couples come to a river, which they must cross in a boat big enough to carry one or two people. (It can't be empty because there's no ferryman.) The men are very jealous. Neither will allow his wife to be with the other man unless he himself is there (either on shore or in the boat and whether or not the other man's wife is also present). Using graph theory, determine whether all four can get across the river. If so, what is the minimum number of crossings?

5. (3)

- a) Suppose that there are three couples, with everything else the same as in [4]. Again, no husband allows his wife to be with one or more other men if he is not there, too.

b) Repeat a) for four couples.

6. (2) The game of tic-tac-toe has plenty of different states but the number becomes manageable if the symmetry of the three-by-three array is taken into account. Draw all states which are not related to each other by symmetry after the first move and after the second move and draw the digraph in which these states are vertices and an edge going from one state to another indicates that a move can take you from the first state to the second. (See also [37, Section 3.7].)

7. (2) Define and draw a state graph for ordinary TOH with two rings. Use it to show that, under TOH rules, you can get from any configuration to any other configuration.

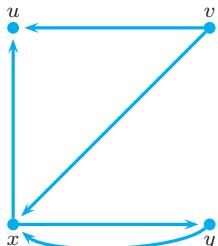
8. (1) Show that 19 bits is the shortest possible binary sequence that, without wraparound, contains as subsequences all possible 4-bit sequences.

9. (2) Find another sequence besides (2) which includes all 16 four-bit sequences (allowing wraparound) by traversing a path in Fig. 3.2. This sequence should *not* be just a circularly permuted version of (2) (i.e., one in which digits are moved from the end to the beginning as in 0110 0101 0000 1111).

10. (2) Suppose that the hotel problem specifies three-digit instead of four-digit "combinations", and we restrict ourselves to the binary case. For

this case draw a graph corresponding to Fig. 3.2 and, by finding a cycle, display a full cycle.

11. (2) Display the tree structures corresponding to Fig. 3.3b if the input words are:
- a) Discrete, Mathematics, Is, Easier, Than, Continuous, Mathematics. (Make sure that "Mathematics" appears twice in your tree.)
  - b) You, Have, To, Be, Crazy, To, Study, Mathematics, Or, Computer, Science. In this part don't put repeated words on the tree twice. Show how Algorithm BUILDTREE must be modified to accomplish this.
12. (2) For a given set of words, Algorithm BUILDTREE builds different trees depending on the order in which the words arrive. For simplicity, assume that there are just three distinct words in the set and that they arrive in any order with equal likelihood.
- a) Draw all possible trees that can be built.
  - b) Define the *height* of a tree to be the number of edges in the longest path from the root. Determine the average height of the tree built by BUILDTREE for three words by considering the trees for all possible orderings of the words.
- ### Problems for Terminology Section
13. (1)
- a) Draw a graph  $G(V, E)$ , with  $V = \{u, v, x, y\}$  and  $E = \{\{u, v\}, \{v, x\}, \{x, u\}, \{v, y\}\}$ .
  - b) For the graph below, what are the sets  $V$  and  $E$ ?
- ```
graph LR; u((u)) --- v((v)); u --- x((x)); v --- x; v --- y((y)); x --- y;
```
14. (1)
- a) Draw a digraph  $G(V, E)$ , with  $V = \{u, v, x, y\}$  and  $E = \{(u, v), (v, x), (x, v), (y, y)\}$
  - b) For the following digraph, what are the sets  $V$  and  $E$ ?



15. ⟨3⟩

- a) Draw all the simple, undirected, nonisomorphic graphs having four vertices.
- b) Draw all the simple, nonisomorphic directed graphs having four vertices, at most three edges and no edges in both directions between any pair of vertices.

All these graphs can be drawn without having edges intersect except at vertices; to save the grader from eye strain, avoid edge crossings.

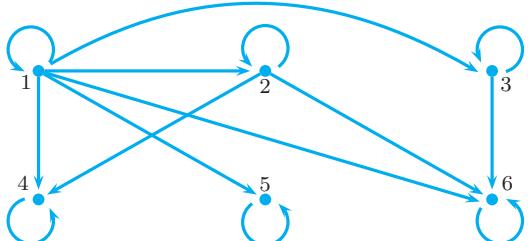
16. ⟨3⟩

- a) Draw all the simple, undirected, nonisomorphic graphs having six vertices for which the degrees of the vertices are 1, 1, 1, 2, 2, and 3.
- b) Similarly, draw all the simple directed graphs having five vertices for which the outdegrees and indegrees of the vertices are (2, 1), (0, 2), (3, 0), (1, 3), and (1, 1) (i.e., the first vertex has two edges leading from it and one going toward it).

17. ⟨2⟩ Suppose that  $G$  is a digraph representing a relation  $R$ . How would you determine from the graph whether the relation is transitive?

18. ⟨2⟩ Consider the digraph below.

- a) For the binary relation  $R$  defined by this graph, list all pairs  $(a, b) \in R$ . Is the relation reflexive, symmetric or transitive?
- b) What familiar relation between integers is represented by this graph for the integers 1 to 6?



19. ⟨2⟩

- a) Use Theorem 1 to show that any graph has an even number of vertices of odd degree.
- b) Use this result to show that, in any group of people shaking hands in pairs, an even number of people each shake hands with an odd number of people.

20. ⟨2⟩ Suppose we had defined the degree that a loop adds to vertex to be 1. Then

- a) How would the statement of Theorem 1 have had to be changed?
- b) How would the proof have had to be changed?

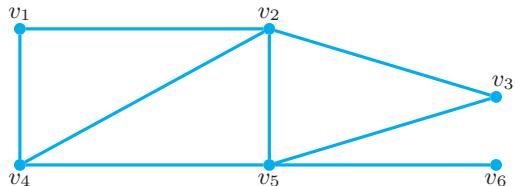
21. ⟨2⟩ Prove that a simple graph without loops has at least two vertices of the same degree. Hint: Let  $G$  have  $n$  vertices. Then what is the largest degree that any vertex can have? If this maximum is attained, what is the smallest degree that some other vertex can have? What happens when there are isolated vertices? Your argument will boil down to the **Pigeonhole Principle**: If  $p$  pigeons are put in  $q < p$  holes, then some hole contains at least two pigeons (see also Section 4.10). (Note: Induction might seem like a natural for this problem, but we don't see how to make it work. If you do, let us know!)

22. ⟨2⟩ The definition of a path in Definition 2 doesn't work for multigraphs. Why not? Come up with a definition of a path in a multigraph that does work.

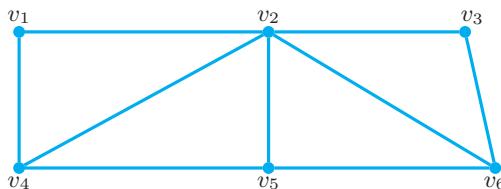
23. ⟨2⟩ If  $G$  is a simple graph with  $n$  vertices and no two edges share a vertex, what is the maximum number of edges that  $G$  can have?

24. ⟨2⟩

- a) Find all the simple cycles in the following graph.



- b) In the following graph, find three different simple cycles, each of which has at least five edges.



25. (2) When defining an acyclic digraph is it enough just to say that it has no cycles or do we have to limit ourselves to saying that there are no simple cycles? Why don't we have to be concerned about cycles of the form  $(v_1, v_2, v_1)$  as we were with graphs?

26. (2) Cut-Sets. If the vertices of a graph are divided into two sets  $V_1$  and  $V_2$ , the set of edges which join a vertex in  $V_1$  to a vertex in  $V_2$  is called the **cut-set** of the graph relative to  $(V_1, V_2)$ .

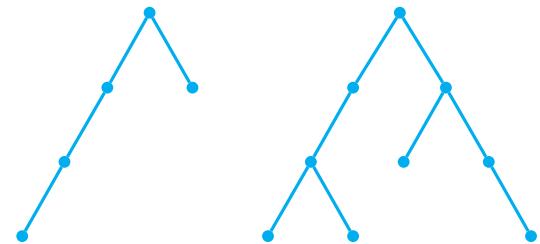
- a) For the graph in [24a], with  $V_1 = (v_1, v_2)$  and  $V_2 = (v_3, v_4, v_5, v_6)$ , what is the cut-set?
- b) For the graph in [24b], with  $V_1 = (v_1, v_3, v_5)$  and  $V_2 = (v_2, v_4, v_6)$ , what is the cut-set?

27. (2) A **tournament** is a digraph in which, for each pair of vertices, there is exactly one directed edge between them. (See Induction and Tournaments in Section 2.2.)

- a) Draw all the nonisomorphic tournaments having three vertices.
- b) Is there a tournament on five vertices in which the outdegrees of the vertices are 3, 2, 2, 2, and 2? Why?
- c) Is there a tournament on five vertices in which the outdegrees of the vertices are 3, 3, 3, 1, and 0? Why?

28. (3) We may ask when our definition of a tree as a digraph describes trees that are weakly or strongly connected.

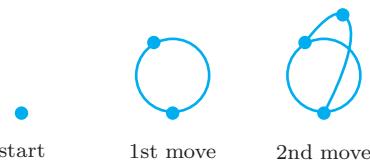
- a) Except for the tree consisting of only the root vertex, no tree is strongly connected. Why not?
- b) Very few trees are weakly connected. Can you describe all those that are?
- c) For each of the two trees shown, what is the minimum number of directed edges which must be added between existing vertices to make the tree weakly connected? Strongly connected? (Of course, they will no longer be trees.)



29. (2)

- a) Show that a consequence of our definition of a tree as a digraph is that there is a path from the root to every vertex in a tree.
- b) Why is it redundant in the definition of a tree as a digraph to require that a tree be both connected and acyclic? That is, if either "connected" or "acyclic" is deleted from the definition but the remainder of the definition is unchanged, the same graphs are defined.
- c) But why is it not redundant to require that a tree as an undirected graph be both acyclic and connected?
- 30. (2) The following sentences are *mathematically* ungrammatical because they misuse graph terminology. However, by changing just a word or two, they can be made mathematically grammatical. Do it.
  - a) The number of degrees of vertex  $v$  (in some graph) is six.
  - b) Edge  $e$  is adjacent to vertex  $v$ .
  - c) To say that vertices  $u$  and  $v$  are connected means that  $\{u, v\}$  is an edge.
- 31. (3) The game of *baby sprouts* is played as follows. A few dots (vertices) are put on paper. Two players alternately put edges between the dots, following just two rules: i) no vertex can ever get degree more than 3; ii) Whenever an edge is drawn, a new vertex is created in its middle. If at some point it is not possible to draw another edge, the person who drew the last edge wins.
 

The figure below shows what happens when you start with one dot. The second player wins. Notice that loops are allowed and that edges can cross each other without forming new vertices. (The crossing could have been avoided in this case, but not always.)

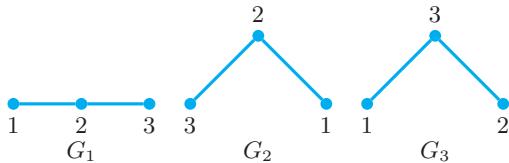


It turns out that, for baby sprouts, the winner is determined solely by the number  $d$  of original dots. In terms of  $d$ , figure out who wins and why.

In the real game of sprouts, there is one more rule: edges can't cross. Real sprouts is much harder.

- 32.** (2) Labeled graphs. A graph is **labeled** if each vertex has a name. (We have often given vertices in our graphs names to make it easy to refer to them but the names have been arbitrary. Here, however, we mean that the labels are fixed symbols which are parts of the graphs.) Two labeled graphs  $G$  and  $H$ , with the same set of vertex names, are isomorphic if and only if, for any two vertices  $u$  and  $v$ , whenever  $\{u, v\}$  is an edge in  $G$  then  $\{u, v\}$

is an edge in  $H$  and vice versa. Thus in the following figure,  $G_1$  and  $G_2$  are isomorphic, but  $G_3$  isn't isomorphic to either of them.



- a) Draw all the nonisomorphic labeled simple graphs on the vertex set  $\{u, v, w\}$ .
  - b) If the names of the vertices are now erased (so that the graphs are *unlabeled*, that is, ordinary graphs), how many nonisomorphic graphs are left?
33. (2) The **divisibility graph**  $D(n)$  is defined by  $V = \{1, 2, \dots, n\}$ ,  $E = \{\{i, j\} \mid i \text{ divides } j\}$ , where “ $i$  divides  $j$ ” means that there must be no remainder. Draw  $D(12)$ .

## 3.2 Paths, Cycles and the Adjacency Matrix

Two of the three examples in Section 3.1 involved finding paths or cycles in a graph. Practical applications of graphs are often related to paths or cycles because, by their very nature, graphs describe in some way how data associated with one vertex is related to data associated with other vertices. How else could such relationships be explored or exploited except by studying the connections — that is, the paths — between vertices? In this section and the next two we consider various aspects of paths and cycles.

### Reachability and Representations

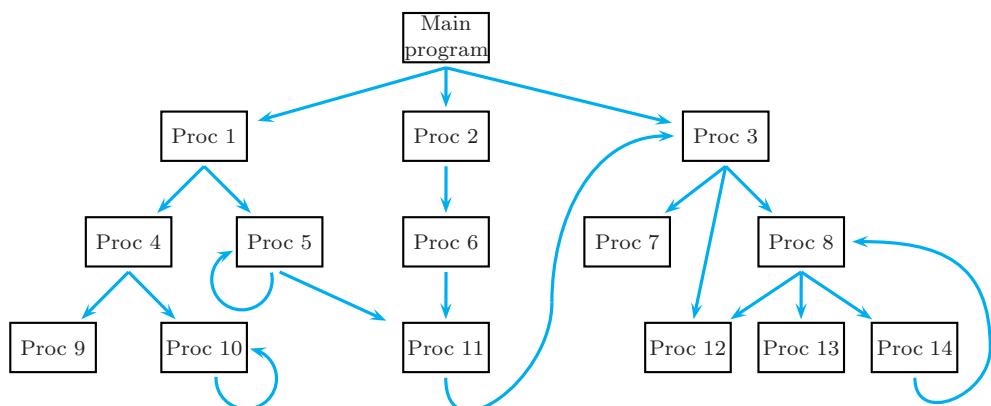
An obvious question to ask about a graph  $G(V, E)$  is whether you can find a path from one vertex to another. For any of the graphs illustrated thus far in this chapter, you can easily determine by inspection whether there is a path from one vertex to another. But suppose that you were confronted with a graph having hundreds of vertices. It would probably not be drawn at all, because the page would be too cluttered, but rather might be presented by enumerating the sets  $V$  and  $E$ . Answering a question about paths for such a graph would be decidedly nontrivial.

One vertex of a graph (or digraph) is said to be **reachable** from a second vertex if there is a path from the second vertex to the first vertex. The reachability

of one vertex from another is what the wolf, cabbage, goat, and farmer problem (Example 1, Section 3.1) is all about.

Reachability is also an important concept in all sorts of *network* problems. For example, in a communications network you are interested in whether a message can get from one vertex to another and, perhaps, you may wish to know how many different (communication) paths there are from one vertex to another. A second example is a road system linking cities when you may wish to find the shortest route from one city to another. A third and quite different example concerns a food web, which is a directed graph in which the vertices represent species and there is an edge from vertex  $u$  to vertex  $v$  if species  $u$  eats species  $v$ . Then the question of whether vertex  $w$  is reachable from vertex  $u$  is the same as asking whether the two species represented by  $u$  and  $w$  are on the same food chain.

As a final example, consider a large and complex computer program organized as a main program and procedures which call one another (sometimes recursively, sometimes not — see Fig. 3.11). This graph is known as the **call graph** of the program. Often you may need to know whether there is a path in the call graph from one procedure to another and, if so, how many separate paths there are. The result may determine, for example, how the data in the program should be organized. Or you may need to know whether there is a path from a procedure back to itself, that is, whether there is a cycle (e.g., Proc 8 to Proc 14 and back to Proc 8 in Fig. 3.11, which results in **indirect recursion**). Typically, all you know to begin with is, for each procedure, the procedures it may call. That is, for each vertex you know all the edges leading from it. From these data you then must construct the overall call graph and determine which vertices are reachable from which others. When the program is very large and complicated, this can be difficult.



**FIGURE 3.11**

A typical program call graph.

Finding the best *representation* of any problem is often crucial to finding an effective way to solve the problem. Nowhere is there a more graphic (pun intended) example of this than in problems in graph theory. It may have occurred to you that if a graph has hundreds of vertices and edges, long lists of them may not be the most useful way to deal with the graph. Nor is a drawing of the graph very useful.

A representation of graphs that is often very effective when you're *thinking* about graph problems and which is also often a good one to use when you're *computing* with graphs is called the **adjacency matrix**<sup>†</sup>.

For a graph  $G(V, E)$  with  $n$  vertices, its adjacency matrix  $A = [a_{ij}]$  is the square matrix in which  $a_{ij}$  is the *number* of edges from  $v_i$  to  $v_j$  (with a loop counting as a single edge). This number will be 1 or 0 for simple graphs but could be greater than 1 for multigraphs. Using our algorithmic notation, we can easily define the adjacency matrix for a simple graph with vertex set  $V = \{v_1, v_2, \dots, v_n\}$  as follows:

```

for  $i = 1$  to  $n$ 
  for  $j = 1$  to  $n$ 
    if  $\{v_i, v_j\} \in E$  then  $a_{ij} \leftarrow 1$ 
    else  $a_{ij} \leftarrow 0$ 
    endif
  endfor
endfor
```

This definition works for digraphs, too, if we replace the set braces by parentheses. For graphs,  $A$  is **symmetric** (i.e.,  $a_{ij} = a_{ji}$  for all  $i$  and  $j$ ), because there is an edge from  $u$  to  $v$  if and only if there is one from  $v$  to  $u$ . Thus for graphs we need only to know the **main diagonal** ( $a_{ii}, i = 1, \dots, n$ ), which gives the loops in the graph, and either the triangle below it (the **lower** triangle) or the one above it (the **upper** triangle) to know the entire matrix. For digraphs, the adjacency matrix is not, in general, symmetric (when will it be?) and must usually be given in its entirety.

Fig. 3.12 displays three graphs and their corresponding adjacency matrices. For a digraph the sum of the entries in the adjacency matrix is the size of the set  $E$  (why?). (For the matrix of Fig. 3.12c you can easily verify this.) But for a graph (see Figs. 3.12a,b), the size of the set  $E$  is the sum of the entries on the main diagonal and one of the triangles (why?).

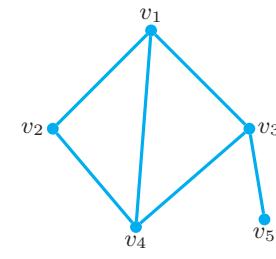
How does the definition of the adjacency matrix get us any closer to determining the reachability of one vertex from another? To answer this question, let's first define **path length** to be the number of edges on a path. If the same edge appears in a path  $m > 1$  times, it is counted  $m$  times in the path length. Thus the path length is one less than the number of vertices in Definition 2, Section 3.1. When the path is a simple cycle, the number of distinct vertices on the path equals the path length.

It follows from the definition of path length that the shortest possible path consists of a single edge and has length 1. This path could be a loop which is just a path of length 1 from a vertex to itself. (Sometimes, however, considering a vertex itself to be a path of length 0 from itself to itself is convenient. The usefulness of such **null paths** is explored in [17].)

---

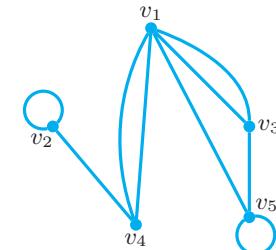
<sup>†</sup>Since the adjacency matrix is often **sparse** (i.e., it contains few nonzero entries), it is often stored in a computer not as a matrix but using some more efficient data structure such as a linked list.

|       | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ |
|-------|-------|-------|-------|-------|-------|
| $v_1$ | 0     | 1     | 1     | 1     | 0     |
| $v_2$ |       | 0     | 0     | 1     | 0     |
| $v_3$ |       |       | 0     | 1     | 1     |
| $v_4$ |       |       |       | 0     | 0     |
| $v_5$ |       |       |       |       | 0     |



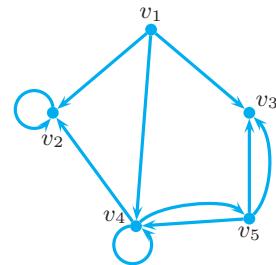
(a)

|       | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ |
|-------|-------|-------|-------|-------|-------|
| $v_1$ | 0     | 0     | 2     | 2     | 1     |
| $v_2$ |       | 1     | 0     | 1     | 0     |
| $v_3$ |       |       | 0     | 0     | 1     |
| $v_4$ |       |       |       | 0     | 0     |
| $v_5$ |       |       |       |       | 1     |



(b)

|       | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ |
|-------|-------|-------|-------|-------|-------|
| $v_1$ | 0     | 1     | 1     | 1     | 0     |
| $v_2$ | 0     | 1     | 0     | 0     | 0     |
| $v_3$ | 0     | 0     | 0     | 0     | 0     |
| $v_4$ | 0     | 1     | 0     | 1     | 1     |
| $v_5$ | 0     | 0     | 2     | 1     | 0     |



(c)

**FIGURE 3.12**

Adjacency matrices for three graphs.

Thus, one way of interpreting the element in the  $i$ th row and  $j$ th column of an adjacency matrix is that it represents the number of paths of length 1 from  $v_i$  to  $v_j$ . This observation is generalized by the following theorem.

**Theorem 1.** Let  $A$  be the adjacency matrix of a graph  $G$ . Then the element in the  $i$ th row and  $j$ th column of  $A^m$  is equal to the number of paths of length  $m$  from vertex  $i$  to vertex  $j$ .

The notation  $A^m$  means the  $m$ th power of the matrix  $A$ , that is,  $m$  factors of  $A$  multiplied together, with multiplication defined as in Section 0.5. Thus,  $A^m = A^{m-1}A = AA^{m-1}$ . In fact, the laws of exponents work with powers of a matrix just as they do with powers of a number.

**PROOF BY INDUCTION** We have already noted that  $a_{ij}$  is the number of paths of length 1 from  $v_i$  to  $v_j$ . Therefore the theorem is true for  $m = 1$ , which provides the basis step for our induction.

For the inductive step we write the adjacency matrix as

$$A = [a_{ij}].$$

Similarly, we write for the  $r$ th power of  $A$  ( $r > 1$ ):

$$A^r = [a_{ij}^{(r)}].$$

Then, using the definition of matrix multiplication in Eq. (3), Section 0.5, the element  $a_{ij}^{(2)}$  in the  $i$ th row and the  $j$ th column of  $A^2$  is

$$a_{ij}^{(2)} = \sum_{k=1}^n a_{ik} a_{kj}, \quad (1)$$

and in general, since  $A^m = AA^{m-1}$ ,

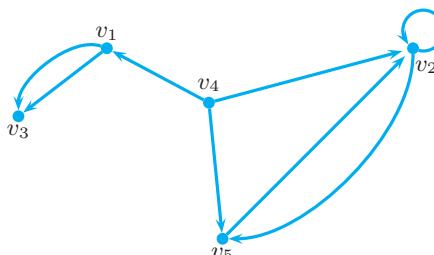
$$a_{ij}^{(m)} = \sum_{k=1}^n a_{ik} a_{kj}^{(m-1)}. \quad (2)$$

Now let the induction hypothesis be that the theorem is true for  $m - 1$  and consider each term in the summation in Eq. (2). The product of  $a_{ik}$  and  $a_{kj}^{(m-1)}$  gives the number of paths of length  $m$  from  $v_i$  to  $v_j$  which go through  $v_k$  as the second vertex on the path. Why? Because  $a_{kj}^{(m-1)}$  is, by the induction hypothesis, the number of paths of length  $m - 1$  from  $v_k$  to  $v_j$ . If  $a_{ik} = 1$  (i.e., there is an edge from  $v_i$  to  $v_k$ ), then appending this edge at the beginning of each path of length  $m - 1$  gives a path of length  $m$  with  $v_k$  the second vertex. Are there any paths of length  $m$  not counted by the sum in Eq. (2)? No, because *any* path of length  $m$  may be viewed as a path of length 1 to *some* vertex  $v_k$  adjoined to a path of length  $m - 1$ , and the sum counts all of them. Are any paths counted more than once by Eq. (2)? No, again, because each term in Eq. (2) counts only those paths starting at  $v_i$  and going to a particular  $v_k$  as the second vertex. Therefore  $a_{ij}^{(m)}$  gives all the paths of length  $m$  from  $v_i$  to  $v_j$ . ■

You can easily verify that the theorem is also true for digraphs and for multigraphs [11].

## EXAMPLE 1

Find the number of paths of length 3 in the directed multigraph of Fig. 3.5c, which we reproduce here.



**Solution** Directly from the figure we have

$$A = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Multiplying  $A$  by itself we obtain

$$A^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Multiplying again by  $A$  we have

$$A^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 2 \\ 0 & 2 & 0 & 0 & 1 \end{bmatrix}.$$

Thus there are, for example, three cycles of length 3 from  $v_2$  to itself ( $a_{22}^{(3)} = 3$ ). Referring to the illustration above, we determine that these cycles are

$$\begin{aligned} & (v_2, v_2, v_2, v_2), \\ & (v_2, v_5, v_2, v_2), \\ & (v_2, v_2, v_5, v_2). \end{aligned}$$

Also, there are two paths of length 3 from  $v_4$  to  $v_5$  ( $a_{45}^{(3)} = 2$ ), which are

$$\begin{aligned} & (v_4, v_5, v_2, v_5), \\ & (v_4, v_2, v_2, v_5). \end{aligned}$$

We can use Theorem 1 to determine whether vertex  $v$  is reachable from vertex  $u$ , that is, whether there is *any* path from  $u$  to  $v$ . Let the vertex set  $V = \{v_1, v_2, \dots, v_n\}$ . Then we first define the **path matrix**  $P = [p_{ij}]$  algorithmically as follows:

```

for  $i = 1$  to  $n$ 
  for  $j = 1$  to  $n$ 
    if there is any path from  $v_i$  to  $v_j$ 
      then  $p_{ij} \leftarrow 1$ 
      else  $p_{ij} \leftarrow 0$ 
    endif
  endfor
endfor

```

Note that the entries on the main diagonal,  $p_{ii}$ , are 1 if and only if there is a path from  $v_i$  to itself (which could be a loop). Like the adjacency matrix for simple graphs,  $P$  is a **Boolean matrix**, that is, one whose entries are all 0's and 1's.

It isn't hard to show [15] that, if a graph with  $n$  vertices has no path of length  $n$  or less between  $u$  and  $v$ , then there is no path at all between  $u$  and  $v$ . Thus to find the path matrix, we could just compute  $A^2$ ,  $A^3$ , ...,  $A^n$ . The path matrix would have a 1 in any position where any of these matrices or  $A$  itself has a nonzero quantity.

Are you satisfied? We discussed why knowing the path matrix may be important. Then we defined a representation — the adjacency matrix — which is a convenient starting point for computing the path matrix. Next we proved a theorem to show how the adjacency matrix may be used to find the number of paths of any length between two vertices. Finally, we showed how this theorem may be used to design a simple algorithm to compute the path matrix. Isn't that enough?

No, it's not. It's one thing to have *an* algorithm to accomplish a task; it's something else to have a *good* (i.e., efficient) algorithm. The one we presented just above is not a good algorithm if all we want to know is what is reachable from what. As Example 1 implies, for all but the smallest values of  $n$ , the computation of powers of  $A$  is quite tedious. In fact, if there are  $n$  vertices, each matrix multiplication takes  $n^3$  multiplications and  $n^2(n - 1) = \text{Ord}(n^3)$  additions. Since there may be  $n - 1$  matrix multiplications to obtain  $A^2, A^3, \dots, A^n$ , we may have to do  $\text{Ord}(n^4)$  operations, a very large number for all but quite small  $n$ . Is there a better way? Yes. It is embodied in an algorithm designed by S. Warshall.

**Warshall's algorithm** also starts with the adjacency matrix. Then, one at a time, we add vertices from  $V$  to a set of intermediate vertices  $V'$  *through which paths are allowed to pass* when going from one vertex in  $V$  to another. Putting it another way, suppose that we already knew, for any pair of vertices and some  $k$ , when one was reachable from the other by a path which was allowed to pass only through vertices  $v_1, v_2, \dots, v_k$ . If there is no such path, then we try allowing a path which could also pass through  $v_{k+1}$ .

Thus at the zeroth stage (when  $V'$  is empty and so *no* intermediate vertices are allowed), for each ordered pair of vertices  $(u, v)$ , we consider only edges from  $u$  to  $v$ , that is, paths with no intermediate vertices. Then, at the first stage we add  $v_1$  to  $V'$  and, for each ordered pair of vertices  $(u, v)$ , we consider edges from  $u$  to  $v$  and paths from  $u$  to  $v$  which have only  $v_1$  as an intermediate vertex. At the next stage we allow paths which pass through  $v_1$  or  $v_2$  or both. In this way, we can avoid much of the computation required to compute powers of  $A$ .

To implement this idea we first define the **modified adjacency matrix**  $A^{(1)}$  to be the adjacency matrix with all terms greater than 1 replaced by 1. For a simple graph, the adjacency matrix and the modified adjacency matrix are the same. Then we define a sequence of matrices

$$P_0 (= A^{(1)}), P_1, P_2, \dots, P_n$$

where each  $P_k = [p_{ij}^{(k)}]$ ,  $k = 0, 1, \dots, n$ , is defined by

```

for  $i = 1$  to  $n$ 
  for  $j = 1$  to  $n$ 
    if there is a path from  $v_i$  to  $v_j$  with all intermediate
    vertices in the set  $\{v_1, v_2, \dots, v_k\}$  [Path could be just an edge]
      then  $p_{ij}^{(k)} \leftarrow 1$  (3)
      else  $p_{ij}^{(k)} \leftarrow 0$ 
    endif
  endfor
endfor

```

This algorithm fragment considers at each stage all pairs of the  $n$  vertices in  $V$ , but at the  $k$ th stage, allows paths to go through *only* the vertices in the set  $\{v_1, v_2, \dots, v_k\}$ .

It should be clear from this algorithm fragment that

$$P_n = P, \quad (4)$$

where  $P$  is the path matrix. As  $k$  gets close to  $n$ , most values of  $p_{ij}$  that will eventually be 1 are 1 already. This fact enables us to reduce the total amount of computation considerably. Let's see how we can embody this idea in an algorithm.

If  $P_k$  is to have the property defined by the algorithm fragment above, then in going from  $P_{k-1}$  to  $P_k$ , the only time  $p_{ij}^{(k)}$  will differ from  $p_{ij}^{(k-1)}$  is when there is no path from  $v_i$  to  $v_j$  including at most vertices from

$$\{v_1, \dots, v_{k-1}\}$$

as intermediates (i.e.,  $p_{ij}^{(k-1)} = 0$ ) but there is a path from  $v_i$  to  $v_j$  including at most vertices in

$$\{v_1, v_2, \dots, v_{k-1}, v_k\}.$$

This latter case can occur only when there is a path from  $v_i$  to  $v_k$  including only vertices in  $\{v_1, \dots, v_{k-1}\}$  (i.e.,  $p_{ik}^{(k-1)} = 1$ ) *and* a path from  $v_k$  to  $v_j$  including only vertices in this same set (i.e.,  $p_{kj}^{(k-1)} = 1$ ). Therefore  $p_{ij}^{(k)}$  is defined by:

```

if  $p_{ij}^{(k-1)} = 1$  then  $p_{ij}^{(k)} \leftarrow 1$ 
  else  $p_{ij}^{(k)} \leftarrow p_{ik}^{(k-1)} p_{kj}^{(k-1)}$ 

```

with the product after **else** equal to 1 only when the addition of  $v_k$  to the set of vertices allows a path from  $v_i$  to  $v_j$ , which was not possible previously.

Putting all these ideas into algorithmic form gives Algorithm WARSHALL.  
Remarks:

**Algorithm 3.2****Warshall**

|                                                                     |                                     |
|---------------------------------------------------------------------|-------------------------------------|
| <b>Input</b> $n$                                                    | [Size of $V$ ]                      |
| $A^{(1)}$                                                           | [Modified adjacency matrix of $G$ ] |
| <b>Output</b> $P = [p_{ij}]$                                        | [Path matrix]                       |
| <b>Algorithm</b> WARSHALL                                           |                                     |
| $P \leftarrow A^{(1)}$                                              |                                     |
| <b>for</b> $k = 1$ <b>to</b> $n$                                    | [Index for $P^{(k)}$ ]              |
| <b>for</b> $i = 1$ <b>to</b> $n$                                    | [Row index]                         |
| <b>for</b> $j = 1$ <b>to</b> $n$                                    | [Column index]                      |
| <b>if</b> $p_{ij} = 0$ <b>then</b> $p_{ij} \leftarrow p_{ik}p_{kj}$ |                                     |
| <b>endfor</b>                                                       |                                     |
| <b>endfor</b>                                                       |                                     |
| <b>endfor</b>                                                       |                                     |

- We dropped the previous superscripts in WARSHALL to indicate that you can store successive  $P_k$  on top of each other. But wait a minute. Suppose that in the  $k$ th pass through the outer loop  $p_{ik}$  and  $p_{kj}$ , which should be values in  $P_{k-1}$ , have already been changed to their values in  $P_k$  earlier in this pass through the outer loop. Could this cause a problem? The answer is No [18].
- When  $i = k$ , then  $p_{kj}^{(k-1)}$  is defined by

```

if  $p_{kj}^{(k-1)} = 1$  then  $p_{kj}^{(k)} \leftarrow 1$ 
else  $p_{kj}^{(k)} \leftarrow p_{kk}^{(k-1)}p_{kj}^{(k-1)}$ 

```

so that  $p_{kj}^{(k)}$  always has the same value as  $p_{kj}^{(k-1)}$  (why?). A similar remark holds when  $j = k$ . Therefore we could have tested whether  $i$  or  $j$  are not equal to  $k$  at the same time we tested whether  $p_{ij} = 0$ . We didn't do this in order to keep the statement of the algorithm as simple as possible.

- Once  $p_{ij} = 1$  for any  $i$  and  $j$ , it is not necessary to consider this  $i$  or  $j$  for any further values of  $k$ . Thus the test " $p_{ij} = 0$ " may be made many times for a given  $i$  and  $j$ . But it is easier to perform this test than it is to keep track of (and test) those values of  $i$  and  $j$  for which  $p_{ij}$  has already been set to 1.
- For graphs rather than digraphs the algorithm can be simplified because  $p_{ij} = p_{ji}$  [25].

Have we convinced you that WARSHALL really calculates the path matrix? Perhaps, but our discussion has been so informal that we really should give a proof. As with so many proofs of the correctness of algorithms, the crucial step is to find the loop invariant. Here, finding the loop invariant is pretty easy because our idea was that, after the  $k$ th pass through the loop in WARSHALL,  $P$  should be  $P_k$  as

defined in algorithm fragment (3). Then when we exit from this loop the last time,  $P$  will be  $P_n$ , the path matrix, which is what we want.

Therefore we hypothesize that the loop invariant is: after  $k$  passes through the loop,

$$P = P_k.$$

We prove this by induction (of course!). The basis step is that, before the first entry into the loop (really the  $k = 0$  case),  $P = P_0$ . But this is true because we set  $P = A^{(1)}$  before entry into the loop and  $A^{(1)}$  is just the modified adjacency matrix, which by definition is  $P_0$ .

Now for the inductive step, we assume that, after exit from the  $(k-1)$ st trip around the loop,  $P = P_{k-1}$ . That is, we assume that, at that point,  $p_{ij}$  tells us whether there is a path from  $v_i$  to  $v_j$  using at most the set

$$S = \{v_1, v_2, \dots, v_{k-1}\}$$

as intermediate vertices. If the value of  $p_{ij}$  is 1 (yes, there is a path), then it remains 1, as it should, during the next pass through the loop. If  $p_{ij} = 0$  at the beginning of the pass, then it should only change to a 1 if there is a path from  $v_i$  to  $v_j$  with  $v_k$  on it. This can only happen if there is a path from  $v_i$  to  $v_k$  and a path from  $v_k$  to  $v_j$  with both paths using at most vertices from  $S$ . This is the same as saying that  $p_{ij}$  will change from 0 to 1 if and only if

$$p_{ik}p_{kj} = 1,$$

which is just what the algorithm says is computed during the  $k$ th pass. Thus after the  $k$ th pass,  $P$  does equal  $P_k$ , which is what we wanted. This completes the proof. ■

What remains to be done is to verify that WARSHALL really is more efficient than the brute force approach we outlined earlier. A worst case analysis is both easy and instructive. Clearly, the total computation for this algorithm depends entirely on how many times the statement

$$p_{ij} \leftarrow p_{ik}p_{kj} \tag{5}$$

is executed. The worst case of WARSHALL is a graph with no edges! (Why?) In this case, (5) is always executed since  $p_{ij}$  is *always* 0. As there are  $n$  values of  $i$ ,  $j$  and  $k$ , the total number of executions of (5) is  $n^3$ .

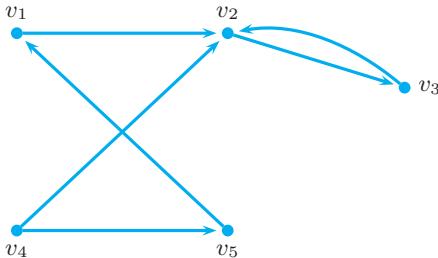
Now suppose that  $G$  is a digraph with  $m$  edges. Then there are  $m$  1's in the adjacency matrix  $A$  and (5) is never executed for these  $m$  pairs of  $i$  and  $j$ . Therefore

$$n(n^2 - m) \tag{6}$$

is at least as great as the number of executions of (5). The bound in expression (6) compares quite favorably with the  $n^4$  estimate of the number of operations in our earlier, brute force algorithm. Actually, we would expect (6) to be a quite conservative (i.e., substantially greater than the actual) upper bound because paths of length greater than 1 will gradually create more nonzero  $p_{ij}$  terms. As you might expect, the average case analysis of WARSHALL is very difficult, indeed.

**EXAMPLE 2**

Apply Algorithm WARSHALL to the following digraph.



**Solution** The digraph is simple, so the modified adjacency matrix is the same as the adjacency matrix:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

For successive values of  $k$ , we obtain for the  $P_k$  matrices:

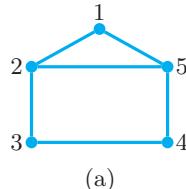
$$k=1: \quad \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}; \quad k=2: \quad \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix};$$

$$k=3, 4: \quad \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}; \quad k=5: \quad \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix} = P.$$

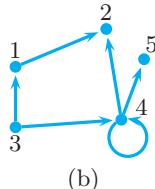
You should be able to verify the correctness of this result by inspecting the preceding figure. For example, there is a 1 in the (5, 2) position when  $k = 1$  because there is a path from  $v_5$  to  $v_2$  which passes through  $v_1$  only. The reason that the matrices for  $k = 3$  and  $k = 4$  are the same is that the addition of  $v_4$  to the set of vertices does not allow any new paths since there are no edges into  $v_4$ . If you count how many times (5) is actually executed in this example, you will see that expression (6) is, indeed, a very conservative bound [33]. ■

## Problems: Section 3.2

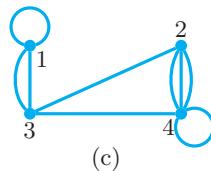
1. (1) Give the adjacency matrix for the following graphs and digraphs.



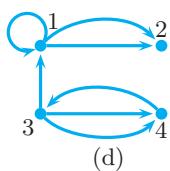
(a)



(b)



(c)



(d)

2. (1)

- a) Draw the graphs or multigraphs with adjacency matrices (i) and (ii).

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & \\ 1 & 0 & 1 & & \\ 1 & 0 & & & \\ 0 & & & & \end{bmatrix}$$

(i)

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 2 & 1 & \\ 1 & 1 & 0 & & \\ 1 & 0 & & & \\ 0 & & & & \end{bmatrix}$$

(ii)

- b) Draw the digraphs with adjacency matrices (i) and (ii).

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

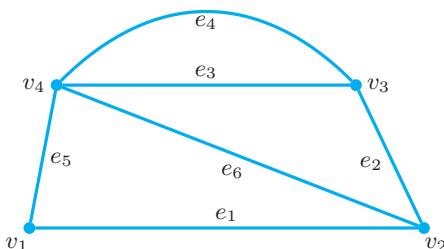
(i)

$$\begin{bmatrix} 0 & 2 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 2 \\ 2 & 0 & 1 & 0 \end{bmatrix}$$

(ii)

3. (3) The **incidence matrix** of a graph  $G(V, E)$  without loops is a Boolean matrix (all entries 0 or 1) with  $|V|$  rows and  $|E|$  columns such that the  $(i, j)$  entry is 1 if vertex  $i$  is at one end of edge  $j$  and 0 otherwise.

- a) Write the incidence matrix for the following graph. Call this matrix  $M$ .



- b) Write the adjacency matrix  $A$  for the same graph.

- c) What can you say about the row sums of an incidence matrix? What about the column sums?

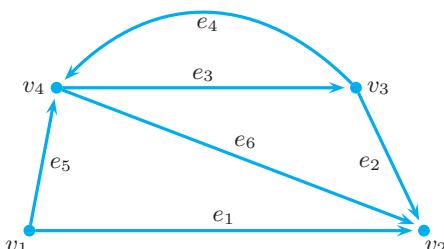
- d) Compute  $MM^T$ , where  $m_{ij}$  in  $M^T$  equals  $m_{ji}$  in  $M$ . How do you interpret this result?

- e) Prove that this interpretation holds for arbitrary graphs without loops.

- f) What goes wrong (or needs to be redefined) if there are loops?

4. (3) The incidence matrix is also defined for directed graphs. The entries are now +1, 0, or -1.

- a) Come up with a definition yourself. Apply it to the following digraph.



- b) Answer [3c] for digraphs.

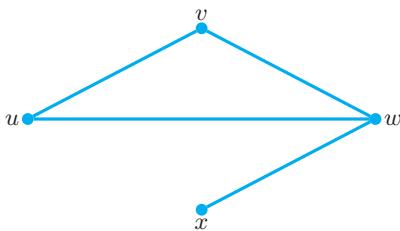
- c) Answer [3d] for the digraph above.

5. (2) Write an algorithm whose input is an adjacency matrix and whose output is a list of edges.

6. (2) Write an algorithm whose input is a list of edges and whose output is an adjacency matrix.

7. (2) For the accompanying graph, let  $A$  be the adjacency matrix. Compute  $A^2$  two ways:

- Directly.
- By counting visually the number of paths of length 2 between each pair of vertices.



8. (2) For the graph in [7]:

- List all paths of length 4 between  $u$  and  $w$ . (The simplest way to record your answer is to denote each path by its vertex sequence. For instance one such path is  $uvwuw$ .)

- Check your answer using the adjacency matrix.

9. (2) Consider the graph with five vertices 1, 2, 3, 4, and 5 and edges  $\{1, 2\}$ ,  $\{2, 3\}$ ,  $\{2, 4\}$ ,  $\{3, 4\}$ ,  $\{3, 5\}$ , and  $\{4, 5\}$ .

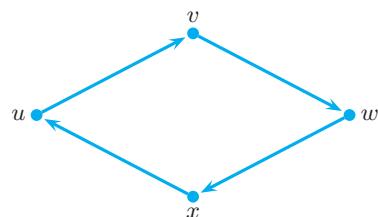
- Write the adjacency matrix.
- Compute the matrix which, for all pairs  $(i, j)$  of vertices, lists the number of paths of length 2 between them.
- Compute the number of paths of length 3 between vertices 2 and 3. (This is a single entry from another matrix. Show how you can avoid computing the entire matrix.)
- Check your answer to c) by listing all the paths of length 3 from vertex 2 to vertex 3. Name each path as a sequence of vertices.

10. (3)

- Explain how to use the adjacency matrix of a simple graph and matrix multiplication to determine the number of triangles incident on any given vertex (i.e., the number of triangles which have the given vertex as one point).
- Explain how to use the adjacency matrix to determine the number of triangles in a graph.

11. (2) State and prove a version of Theorem 1 for
- digraphs.
  - multigraphs.

12. (2) Repeat [7] for the following digraph.



- (2) Repeat [8] for the preceding digraph.
- (2) Repeat [9] in the case where each edge is directed (i.e.,  $(1, 2)$  instead of  $\{1, 2\}$ ).
- (2) Let  $G$  have  $n$  vertices. Explain why, in order to determine whether a pair of vertices is connected, it is sufficient to compute powers of  $A$  only through  $A^n$  (or  $A^{n-1}$  for distinct vertices).
- (2) If a graph has at least one edge, it has infinitely many paths. Explain.
- (2)
  - If we allow null paths, then it makes sense to ask for a matrix which counts the number of paths of length 0. Ideally, Theorem 1 would hold for  $m = 0$  as well as for  $m = 1, 2, \dots$ . Show that it does.
  - In the text we argued that the path matrix is just the matrix
$$\sum_{k=1}^{n-1} A^k,$$

with all nonzero entries replaced by 1. If we allow null paths this statement is still correct if revised slightly. How?
- In WARSHALL we need to know that elements of  $P_{k-1}$  needed during the  $k$ th pass through the loop have not been changed by the  $k$ th pass before they are needed. Why can't this happen? (In general, whenever we overwrite a structure during an algorithm loop, it is important to be assured that we do not lose information needed later.)
- (2) Let  $A'$  be the same as the modified adjacency matrix  $A^{(1)}$  of Algorithm WARSHALL, except that all diagonal entries of  $A'$  are 1. (Thus if  $G$  is a simple graph,  $A' = A + I$ .) Suppose that we replace  $A^{(1)}$  by  $A'$  but otherwise leave WARSHALL unchanged. What happens? Are there any graphs for which the final output is different from that of WARSHALL as shown in the text? If so, is one of the versions wrong, or is it a matter of a slightly

different definition of when vertices are joined by a path?

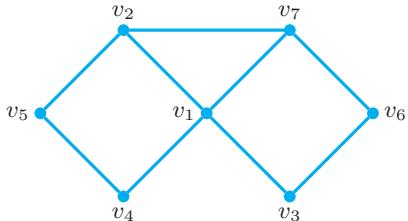
20. (2) Consider the directed graph with five vertices 1, 2, 3, 4, and 5 and edges

$$(1, 4), (2, 1), (2, 4), (3, 2), (4, 5), \text{ and } (5, 3).$$

Determine the path matrix using WARSHALL. Show each of your intermediate matrices. In each matrix circle each 1 which appears where there was a 0 in previous matrices.

21. (2) Consider the undirected graph obtained from the digraph in [20] by removing all the arrows. Use WARSHALL to determine the path matrix for this graph. You should find that you will be able to quit before computing all the matrices. What lets you know that you can quit? What one-word property that we like graphs to have can you attribute to this graph when you are done?

22. (2) For the accompanying graph, carry out the following part of WARSHALL. Start with the adjacency matrix and do three iterations of the matrix update. That is, your final matrix should have a 1 in the  $(i, j)$  entry only when there is a path from vertex  $i$  to vertex  $j$ , using as intermediate vertices at most vertices  $v_1, v_2$ , and  $v_3$ . Show that you are really computing all the entries in the path matrix by explaining how you get the  $(1, 4)$  and  $(5, 6)$  entries of your final matrix. (To do the entire algorithm by hand on this graph is tedious.)



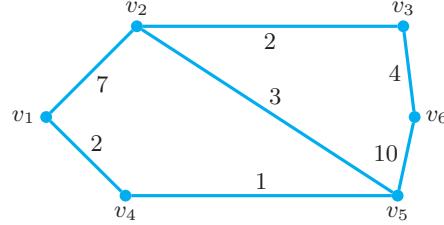
23. (2) Repeat [22], but first make every edge directed from left to right. Now 1's in the matrix should indicate the existence of a directed path.

24. (2) Suppose that the input to Algorithm WARSHALL contains a list of those rows and columns of the adjacency matrix which are all 0. How could you use this information to modify the algorithm to increase its efficiency?

25. (2) Show how the computation of Algorithm WARSHALL can be halved for undirected graphs. Rewrite the algorithm to show what you mean.

26. (3) Suppose that the edges of a simple graph have nonnegative weights associated with them. For such a graph define the length of a path to be the sum of the weights of the edges on it. Use the idea behind Algorithm WARSHALL to devise an algorithm (known as **Floyd's algorithm**) which produces a matrix  $C$  in which the element  $c_{ij}$  is the length of the shortest path between vertices  $i$  and  $j$ . (*Hint:* Start with the matrix which is the adjacency matrix, except that 0 elements are replaced by  $\infty$  and nonzero elements by edge weights. Think recursively.)

27. (2) Find the shortest path length from  $v_1$  to  $v_6$  in the following undirected graph using the shortest-length variant of Algorithm WARSHALL from [26].



28. (2) Extend Algorithm WARSHALL so that, for each pair of connected vertices, it outputs a path between them. Think of it this way: The output now becomes two matrices. The entries of the first are 0's and 1's as before. The  $(i, j)$  entry of the second is empty if there is no  $(i, j)$  path and otherwise is the sequence of vertices in such a path.

29. (3)

- a) [28] involves an excessive amount of work if all you want to do is look at a few paths. For that purpose, you need the algorithm to output only the first internal (or the last internal) vertex of the connecting path. Explain.

- b) Write a postprocessor algorithm which takes this more limited output and, given  $i$  and  $j$ , outputs the entire list of vertices forming a path between them.

30. (3) Carry out the modifications of [28] and [29] on the shortest-length version of Algorithm WARSHALL in [26].

31. (2) Show how to halve the computations in [26] for an undirected graph.

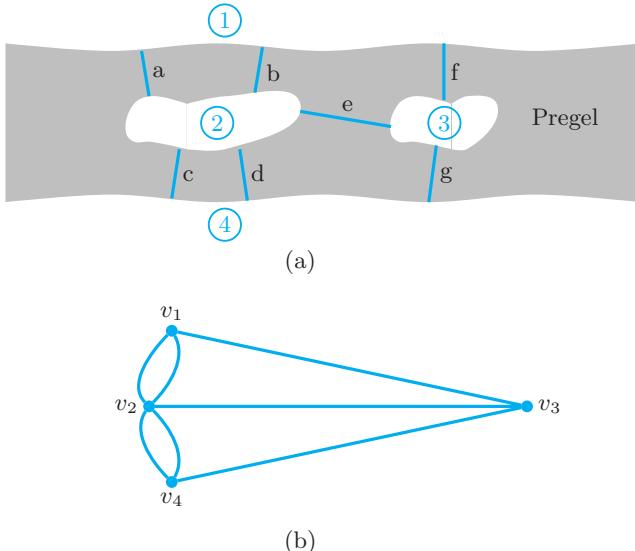
32. {3} The shortest-length version of Algorithm WARSHALL still runs if the graph has negative weights. Does it still succeed in finding the shortest simple paths? If it does, explain why. If it doesn't, give an example of why you might have expected it not to work even before you found a counterexample. (One way to do this is to show

that the recursive thinking which led to Algorithm WARSHALL breaks down; that is, knowing the previous case does not make the current case easy to compute.)

33. {2} Count how many times (5) is executed in the computation of Example 2. Compare this result with expression (6). What do you conclude?

### 3.3 Eulerian and Hamiltonian Paths and Cycles

One of the most famous problems in graph theory is the Königsberg Bridge Problem. Solved by Leonhard Euler in 1736, this problem is usually credited with initiating graph theory as a mathematical subject.



The Königsberg Bridge Problem. (a) The river Pregel at Königsberg, with seven bridges connecting two islands and the two river banks. (b) Graph of the Königsberg Bridge Problem, with land areas as vertices and bridges as edges.

There were seven bridges across the river Pregel at Königsberg, which joined the two banks and two islands in the river as illustrated in Fig. 3.13a. (Königsberg was in East Prussia in Euler's time but is now in Russia and is called Kaliningrad.) The story (legend?) goes that the townspeople of Königsberg asked whether it was possible to take a walk in which each of the seven bridges is crossed once and only once.

If we replace land masses by vertices and the bridges joining them by edges, we get the graph shown in Fig. 3.13b. The problem therefore is equivalent to asking

**FIGURE 3.13**

whether this graph has a path that traverses each edge in  $E$  once and only once. A path in a graph with this property is (unsurprisingly!) called an **Eulerian path**, or sometimes an **Euler path**. If an Eulerian path is a cycle, it is called an **Eulerian cycle**. Since Euler's problem, usually known as the *Königsberg Bridge Problem*, doesn't require that the path across all seven be a cycle, an equivalent question to Euler's is whether the graph has an Eulerian cycle or whether the addition of *one more edge* between any two vertices (which could take us from where we ended our walk back to where we started) would result in a graph with an Eulerian cycle.

Some quite practical problems involve Eulerian paths and cycles. One class of problems is based on the graph formed by the streets of a city or town, where the edges are the individual streets and the vertices are the intersections. Thus, for example, a street cleaning crew would like to travel an Eulerian cycle from the garage and back along the streets which have to be cleaned. Or, if we make our street graph into a multigraph (on which the definition of an Eulerian path or cycle is just the same as for a graph) by having two edges between a pair of vertices whenever there are houses on both sides of the street, then a postal worker delivering letters would like to travel an Eulerian cycle from the post office and back along the sides of streets on which mail has to be delivered. Such a route would be a minimum-length path since no side of a street would be traversed more than once. In such problems we want to know whether there is an Eulerian cycle and, if there is, how to find it. (If there is no Eulerian cycle, then we would like to find the shortest cycle which traverses each edge at least once; this is quite a hard problem, which we won't discuss in this book.)

How might we go about determining whether a graph has an Eulerian cycle? Perhaps by using the path matrix? Well, if there is a 1 in position  $i$  on the diagonal of the path matrix, we know that there is a path from the  $i$ th vertex back to itself. But this may or may not be an Eulerian cycle. Alternatively, if there are  $m$  edges, we might compute  $A^m$ , which tells us whether there are any paths of length  $m$ . But a path of length  $m$  need not be an Eulerian path because it could traverse one edge more than once. In fact, the path matrix just isn't much use in trying to find Eulerian paths and cycles. A more effective approach is to use the theorem below.

First, though, some remarks about the problem of finding Eulerian cycles:

- Surely we may restrict ourselves to connected graphs. The only way a graph that is not connected could have an Eulerian cycle is if the vertices that are not reachable from all the others are isolated (why?). And isolated vertices may be ignored when looking for an Eulerian cycle. Remember that the definition of an Eulerian path (cycle) requires only that all the edges but not necessarily all the vertices be on the path (cycle).
- Nevertheless in proving the theorem below we shall have need to break up the given graph into connected subgraphs. For this purpose it is useful to define a **component** of a graph  $G$  to be a connected subgraph  $G'$  that is not itself a subgraph of any other connected subgraph of  $G$ . That is, a component is a *largest* connected subgraph in the sense that there is no edge incident on any vertex of the component that is not already part of the component. Of course, an isolated vertex is a component by this definition. If we call a component

that is not an isolated vertex a **nontrivial component**, then, if a graph has two or more nontrivial components, it cannot have an Eulerian path. This follows because an Eulerian path traverses all the edges of a graph, but no path can traverse edges in two different nontrivial components (why?).

- Loops can be ignored when searching for an Eulerian cycle since loops can be removed from or added to a graph without affecting the existence of an Eulerian cycle (why?).

Now we're ready for the theorem.

---

**Theorem 1.** A connected, undirected graph (or multigraph) has an Eulerian cycle if and only if it contains no vertices of odd degree.

---

Theorem 1 epitomizes a kind of mathematical theorem called an **existence theorem**. It postulates the existence of an entity (an Eulerian cycle) if a connected graph has a certain property (no vertices of odd degree). In addition, with the “if and only if”, it claims that this property (no vertices of odd degree) is not only *sufficient* for there to be an Eulerian cycle but also *necessary*; there can be no Eulerian cycle in a connected graph that does not have this property (see Section 0.6 for a discussion of this terminology). Broadly speaking, there are two kinds of proofs of existence theorems:

1. **Nonconstructive proofs**, in which you do indeed prove the theorem, but don't, in so doing, produce any useful mechanism which enables you to *construct* the object whose existence is being proved.
2. **Constructive proofs**, in which the existence is proved (a) by exhibiting an object of the desired type or (b) by exhibiting an *algorithm* which can be shown to produce an object of the desired type.

For the past century, nonconstructive existence proofs have been far more common than constructive proofs; most mathematicians have preferred them esthetically. But it won't surprise you that we prefer constructive proofs whenever we can find them. We especially like type (b) constructive proofs; we call such a proof a **proof by algorithm**.

We'll begin our proof of Theorem 1 by proving the necessity part of the theorem, namely that, if there is an Eulerian cycle, then all vertices must be of even degree. There is a simple, straightforward proof of this, but not in proof-by-algorithm form, using the fact that for every time you enter a vertex you also leave it. However, we leave this to a problem [13] because we prefer to present a slightly longer, but similar, argument in proof-by-algorithm form. By giving a relatively simple and informal proof by algorithm first, we hope to smooth the way for a more difficult proof by algorithm for the sufficiency part of the theorem.

Our algorithm is just to traverse the Eulerian cycle  $C$  on a graph  $G$ , starting at any vertex  $v$  on  $C$ , deleting each edge as we traverse it and keeping track of what

happens to the degree of each vertex on the cycle. When we delete the first edge, we reduce the degree of  $v$  by 1 and the degree of the vertex at the other end of this edge by 1. For each subsequent edge, we also reduce the degrees of the vertices at each end by 1. For each vertex after  $v$ , this means that the degree is reduced by 2, since it is reduced by 1 when we arrive and by 1 more when we leave the vertex. As we traverse  $C$ , we may, of course, come back to any vertex (including  $v$ ) more than once, but each time the degree is reduced by 2 as the edges are deleted. Finally, the last edge on  $C$  must be one that returns to  $v$  to complete the Eulerian cycle. When this edge is deleted, it reduces the degree of  $v$  by 1 and, thus, altogether the degree of  $v$  has been reduced by an even number. We are left with all the original vertices of  $G$ , each one now isolated (and, thus, with degree 0) since every edge has been deleted (because our Eulerian cycle  $C$  contains all edges of  $G$  by definition). As the degree of each vertex has been reduced by an even number (one or more 2's), each vertex must originally have been of even degree.

Now we turn to sufficiency, namely, that an Eulerian cycle exists if all vertices are of even degree. We give two proofs, the first a classical nonconstructive one and then a proof by algorithm. The nonconstructive proof, like many existence proofs, is a proof by contradiction (a type of proof we discussed in Sect. 0.6).

**PROOF 1 (nonconstructive)** Suppose that having all vertices of even degree is not enough to ensure the existence of an Eulerian cycle. Then there must be at least one connected graph, all of whose vertices are of even degree, that does not have an Eulerian cycle. Among all graphs with even-degree vertices and *no* Eulerian cycle, let's choose a graph  $G$  with the minimum number of edges. Note that  $G$  must be loopless for otherwise we could just delete any loop, thereby obtaining a graph with fewer edges and still with all edges of even degree but no Eulerian cycle.

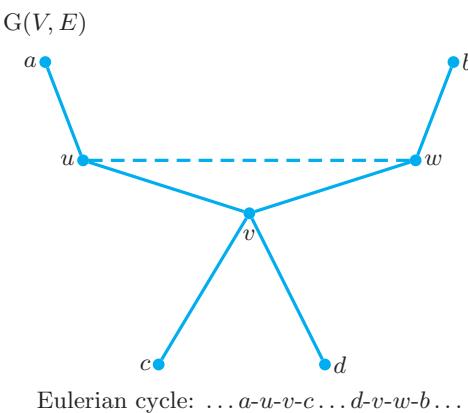
The graph  $G$  must have at least two edges because there is no one-edge, loopless graph whose vertices all have even degrees (why?). Also, since we are only considering connected graphs, there must be two edges of  $G$  that share a vertex (why?). Call these edges  $\{u, v\}$  and  $\{v, w\}$  (see Fig. 3.14). (The vertices  $u$  and  $w$  *could* be the same since we are allowing multigraphs.) Now consider the graph  $G'$  formed from  $G$  by deleting these two edges and replacing them by an edge  $\{u, w\}$  (that would be a loop if  $u$  and  $w$  are the same vertex). By this construction all vertices of  $G'$  also have even degree (why? – [16a]).

Suppose first that  $G'$  is not connected. Then it consists of two components, each of which has fewer edges than  $G$ . So, each component has an Eulerian cycle because of our assumption that all graphs satisfying the hypotheses of the theorem, but which have fewer edges than  $G$ , have Eulerian cycles. (Note that  $v$  could now be isolated in which case it has a (trivial) empty Eulerian cycle.) One component must contain  $\{u, w\}$  and the other  $v$ . This is so because deleting  $\{u, v\}$  and  $\{v, w\}$  can only disconnect  $G$  if there is no longer a path either from  $v$  to  $u$  or from  $v$  to  $w$  (why?).

We can now construct an Eulerian cycle on the original  $G$  (refer to Fig. 3.14) by replacing  $\{u, w\}$  in the Eulerian cycle on its component with  $\{u, v\}$  followed by the Eulerian cycle on the other component followed by  $\{v, w\}$ . Again, by referring

**FIGURE 3.14**

A nonconstructive proof of the Eulerian Cycle Theorem. Note that not all edges are shown;  $a, b, c, d$ , in particular, must have other edges attached to them in order that their degrees be even.



to Fig. 3.14, convince yourself that this would give you an Eulerian cycle on  $G$ . Since we assumed no Eulerian cycle on  $G$ , this result is a contradiction.

Now suppose that  $G'$  is connected. As  $G'$  has one less edge than  $G$ , it must have an Eulerian cycle because of our assumption that  $G$  is a graph with a minimum number of edges, all of even degree, and no Eulerian cycle. Where  $\{u, w\}$  appears in this Eulerian cycle, replace it by  $\{u, v\}$  followed by  $\{v, w\}$ , which then gives an Eulerian cycle on  $G$  (why?). But this result is again a contradiction.

Thus whether  $G'$  has one or two components, we have obtained a contradiction of our assumption that  $G$  did not have an Eulerian cycle. This completes our first proof of the sufficiency part of Theorem 1 and, thus, with the proof of the necessity part above, we have a complete proof of the theorem. But, as promised, we'll give another proof of the sufficiency part below. ■

Did you notice the induction-like flavor of this proof? We used an assumption about graphs with a smaller number of edges than the given graph to arrive at our contradiction.

Do you also see that our proof is nonconstructive in that it does not of itself construct an Eulerian cycle on a graph that satisfies the conditions of the theorem? You *could* use the technique in the proof recursively to divide  $G$  into successively smaller components until you had ones on which Eulerian cycles could easily be found. Then you could piece these cycles together as the theorem does for two components. But this would be terribly tedious to do by hand, particularly if you were given a large graph  $G$  as only a list of edges. And it wouldn't be easy to program for a computer.

**PROOF 2 (constructive)** We want to construct an algorithm which finds an Eulerian cycle on a connected graph, all of whose vertices have even degree. We first develop the algorithm and explain it with a lengthy example. Then we finish off with an inductive proof that the algorithm is correct.

Such an algorithm will certainly need some mechanism for “growing” paths which don’t traverse any edge more than once. We begin therefore with procedure

Pathgrow whose purpose is to grow paths. In this procedure  $E$  is the edge set of a graph,  $v$  is any vertex in the vertex set,  $V$ , and  $I(w)$  is the set of edges incident on vertex  $w$ . The output  $P$  is the path, which could be a cycle, that is “grown” by the procedure. (Actually,  $P$  is the list of edges on that path, in traversal order. Recall that, for simple graphs at least, we defined a path to be a list of vertices.)

### Procedure 3.3

#### Pathgrow

```

procedure Pathgrow(in  $E, v$ ; out  $P$ )
     $P \leftarrow \emptyset$                                 [ $P$  is sequence of edges on path]
     $U \leftarrow E$                                  [ $U$  is set of edges not yet on path]
     $w \leftarrow v$                                [Initialize  $w$ , the current end of  $P$ ]
    repeat until  $I(w) \cap U = \emptyset$       [Until no untraversed edges from  $w$ ]
        Pick  $e \in I(w) \cap U$                   [Choose an edge]
         $U \leftarrow U - \{e\}$                    [Delete  $e$  from  $U$ ]
        Add  $e$  to sequence of edges on  $P$ 
         $w \leftarrow$  the other end of  $e$           [Reset  $w$ ]
    endrepeat
endpro
```

At each step procedure Pathgrow picks an arbitrary edge, which has not been previously traversed, from among those incident on the current vertex and adds it to the path  $P$ . The procedure continues until it can go no further because there aren’t any untraversed edges from the current vertex. Procedure Pathgrow is an example of a “follow-your-nose” algorithm in that it proceeds from one edge to the next by going in some available direction.

Do you see, in particular, that if the set of edges  $E$  defines a graph all of whose vertices are of even degree, then the result of Pathgrow is not just a path but a *cycle* (although perhaps not an *Eulerian cycle*)? In any event, we shall prove this fact shortly.

Our aim, however, is not just to “grow” cycles but to use Pathgrow to grow an Eulerian cycle on a connected graph with all vertices of even degree. Our vehicle for doing this is Algorithm 3.4 ECYCLE. We’ll use the graphs in Fig. 3.15 to illustrate the steps of the algorithm.

First, however, you may doubt that ECYCLE really is a constructive algorithm. How, for example, would you accomplish the tasks in the five lines following **then**? We’ll leave answers to these questions to the problems [19].

Let’s walk through Algorithm ECYCLE with the example in Fig. 3.15. In the main algorithm we call Procedure Euler with **in** arguments  $V, E$  and  $v$  and **out** argument  $C$  so that the values of  $V, E$  and  $v$  in ECYCLE are assigned to  $V', E'$  and  $v'$  in the definition of Euler, and  $C$  is created in ECYCLE and made synonymous with  $C'$  in procedure Euler. Because our discussion below focuses on what happens in procedure Euler, we shall refer to the cycle we are building as  $C'$  and only revert to calling it  $C$  when we consider what is returned to Algorithm ECYCLE.

## Algorithm 3.4

### Ecycle

|                        |                                                    |
|------------------------|----------------------------------------------------|
| <b>Input</b> $G(V, E)$ | [As a list of vertices (of even degree) and edges] |
| <b>Output</b> $C$      | [Eulerian cycle for $G(V, E)$ ]                    |

**Algorithm ECYCLE**

```

procedure Euler(in  $V', E', v'$ ; out  $C'$ )      [ $C'$  becomes an Eulerian
   cycle on  $G(V', E')$ ]
    Pathgrow( $E', v'; C'$ )          [Finds some cycle  $C'$  on  $G(V', E')$ ]
    if  $C'$  is not an Eulerian cycle on  $G(V', E')$ 
        then Delete from  $E'$  the edges on the cycle  $C'$  found
               and denote the nontrivial components of
               the subgraph which is left by  $G_1(V_1, E_1)$ ,
                $G_2(V_2, E_2), \dots, G_j(V_j, E_j)$ ; let  $v_i, i = 1, \dots, j$ ,
               be some vertex in  $V_i$  on  $C'$ .
        for  $i = 1$  to  $j$ 
            Euler( $V_i, E_i, v_i; C_i$ )      [Returns an Eulerian
   cycle on  $G_i$  named  $C_i$ ]
            Attach  $C_i$  to  $C'$  at  $v_i$ 
        endfor
    endif
endpro

```

$v \leftarrow$  any vertex in  $V$  [Main algorithm]

Euler( $V, E, v; C$ )

Euler now calls Procedure Pathgrow, which starts traversing a path from  $v'$ . Suppose that on the graph in Fig. 3.15a with  $v' = v_1$ , we traverse the cycle

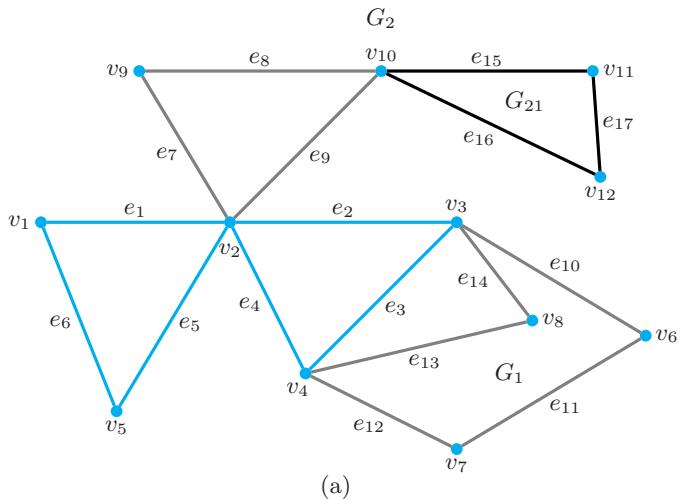
$$P = \underline{v_1, v_2, v_3, v_4, v_2, v_5, v_1}$$

Cycle in color in Fig. 3.15a.

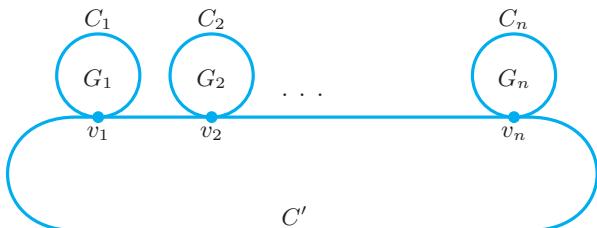
At this point we can't continue because there is no edge incident on  $v_1$  that we have not traversed. It's important to realize that the only place we can get stuck is at the starting vertex  $v_1$ . This is tantamount to claiming that Pathgrow must create a cycle when all vertices have even degree. The reason is that, when we reach any other vertex on the path, we will have traversed an *odd* number of edges *at that vertex* and so, since all vertices are of even degree, there must be an unused edge on which to leave.

Of course, the first time Euler is called, Pathgrow might find an Eulerian cycle  $C'$  on  $G$ , in which case the condition after **if** would be false. Then there would be an immediate return to the main algorithm, which then terminates.

But if, as in our example above,  $C'$  is not an Eulerian cycle, we continue with the **then** portion of the procedure and, for the graph of Fig. 3.15a, delete edges  $e_1$  through  $e_6$  from  $E'$ . This leaves a graph with two nontrivial components,  $G_1$  and  $G_2$ . The important thing to see here is that each of these components contains –



(a)



(b)

An Eulerian Cycle on a connected graph with all vertices of even degree. (a) A connected graph with 12 vertices and 17 edges; the initial cycle  $(v_1, v_2, v_3, v_4, v_2, v_5, v_1)$  is shown in color; a complete Eulerian cycle is  $v_1, v_2, v_3, v_6, v_7, v_4, v_8, v_3, v_4, v_2, v_9, v_{10}, v_{11}, v_{12}, v_{10}, v_2, v_5, v_1$ ; (b) the general case.

**FIGURE 3.15**

and must contain — a vertex on the cycle already found; otherwise, the original graph would not have been connected [16b]. Also, each vertex in the resulting components still has even degree because the edges deleted each contributed an even number to the degree of each vertex on the cycle  $C'$ .

Then Euler is called recursively for each component. In each recursive call the result of the call of Pathgrow *could* be an Eulerian cycle on the component  $G_i$  in which case the cycle  $C_i$  is attached to the cycle  $C'$  found on the previous call of Pathgrow at a vertex they have in common. Otherwise, when Pathgrow finds a cycle on  $G_i$  that is not an Eulerian cycle, Euler is again called recursively. In Fig. 3.15a, the call of Euler for component  $G_1$  would certainly find an Eulerian cycle  $C_1$  on this component (why?), which would then be attached at, say  $v_3$ , to the cycle  $C'$  found on the original call of Pathgrow. The cycle  $C_1$  on  $G_1$  might be

$$v_3, v_6, v_7, v_4, v_8, v_3,$$

or it might be

$$v_3, v_8, v_4, v_7, v_6, v_3,$$

depending on the choice made in the first line of the repeat-loop in Pathgrow. Let's assume the former, so that  $C'$  becomes

$$\begin{aligned} C' = & v_1, v_2, \underline{v_3, v_6, v_7, v_4, v_8, v_3}, v_4, v_2, v_5, v_1, \\ & \text{Eulerian cycle on } G_1 \end{aligned} \quad (1)$$

as indicated in color and gray in Fig. 3.15a. However, for component  $G_2$ , Pathgrow might just find the cycle

$$C_2 = v_2, v_9, v_{10}, v_2, \quad (2)$$

which is not an Eulerian cycle on this component. The result would be another recursive call of Euler for the component labeled  $G_{21}$  in Fig. 3.15a. This call would immediately find an Eulerian cycle  $C_{21}$  on  $G_{21}$ , which would then be attached to  $C_2$  in Eq. (2) at vertex  $v_{10}$  to give

$$\begin{aligned} & \text{Eulerian cycle on } G_{21} \\ C_2 = & \underline{v_2, v_9, \overline{v_{10}, v_{11}, v_{12}, v_{10}, v_2}}, \\ & \text{Eulerian cycle on } G_2 \end{aligned}$$

This would now be an Eulerian cycle on  $G_2$  and would result in the end of the call to Euler for  $G_2$ . Control would then return to the original call and the Eulerian cycle found on  $G_2$  would be attached to  $C'$  in Eq. (1) at, say, the second occurrence of vertex  $v_2$  obtaining finally an Eulerian cycle on the entire graph:

$$C = v_1, v_2, v_3, v_6, v_7, v_4, v_8, v_3, v_4, v_2, v_9, v_{10}, v_{11}, v_{12}, v_{10}, v_2, v_5, v_1.$$

Or we could have attached the cycle found on  $G_2$  at the first occurrence of  $v_2$  in Eq. (1).

Note that if a component  $G_i$  contains more than one vertex of the cycle from which it was called, as is the case with  $G_1$  in Fig. 3.15a, the new cycle can be attached at any of the common vertices on the calling cycle. Thus, we could attach the Eulerian cycle found on  $G_1$  at  $v_3$  as we did but it could also have been attached at  $v_4$  as

$$v_4, v_8, v_3, v_6, v_7, v_4.$$

(Remember: Vertices can be repeated on an Eulerian cycle.)

In the general case shown in Fig. 3.15b, we have assumed that there are  $n$  components  $G_1, G_2, \dots, G_n$  with Eulerian cycles  $C_1, C_2, \dots, C_n$  that are attached at vertices  $v_1, v_2, \dots, v_n$ , respectively, to the cycle  $C'$  found by the first call of Pathgrow. The complete Eulerian cycle  $C$  is then given by

$$C = v_1, C_1, v_1 \dots v_2, C_2, v_2 \dots, v_n, C_n, v_n \dots, v_1,$$

with the ellipses representing the portions of  $C$  between  $v_i$  and  $v_{i+1}$ ,  $i = 1, \dots, n-1$ , and between  $v_n$  and  $v_1$ .

But how do we know that the algorithm works? That is, could the recursion just get deeper and deeper without ever completing the initial call? Or even if it does terminate, might it fail to put the pieces just right to get an Eulerian cycle for the original graph? The answer is that ECYCLE does work, and the strong induction principle provides a simple proof. Let  $P(n)$  be the proposition that procedure Euler terminates and outputs an Eulerian cycle when fed a graph with  $n$  edges. Since Algorithm ECYCLE consists essentially of one call to Euler, this induction will prove that ECYCLE works.

The basis case,  $P(1)$ , requires the consideration of the graph with a single vertex and a loop at that vertex (why?). For this graph the call to Euler results in Pathgrow finding the Eulerian cycle consisting of that loop; the long if-clause is then skipped and Euler terminates with this loop as the output. This proves the basis case. Now suppose that procedure Euler works for all graphs (and multigraphs) with  $n - 1$  or fewer edges. For a graph with  $n$  edges, after the first call of Pathgrow, an Eulerian cycle may be found in which case the procedure terminates immediately with that cycle as output. Otherwise, a non-Eulerian cycle  $C'$  will be found and its edges will be deleted from  $E'$ , after which all the remaining nontrivial components will have fewer than  $n$  edges. Therefore by the induction hypothesis, each subcall to Euler in the for-loop will terminate after providing an Euler cycle for its component. These cycles are each attached to  $C'$  at an appropriate place, creating an Euler cycle for the original  $n$ -edge graph. The main call of Euler now returns this Euler cycle and terminates. ■

This completes our constructive proof of the sufficiency part of Theorem 1. It was somewhat longer than the nonconstructive one, and you may have found it more difficult to follow. But the important point is that, when you've found a constructive proof, you have not just a proof but a way of constructing — via an algorithm in this case — the object whose existence the theorem asserts.

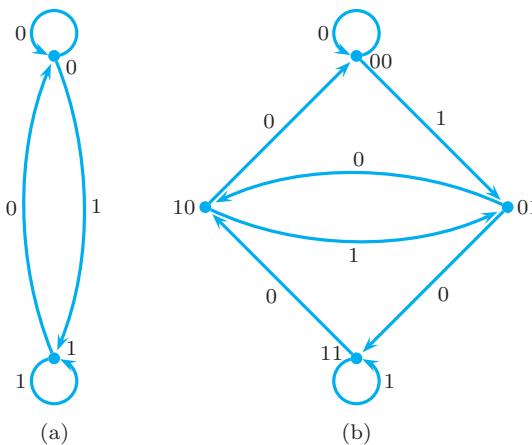
Theorem 1 provides an immediate solution of the Königsberg Bridge Problem. As neither the graph in Fig. 3.13b nor that graph augmented by a single edge has all vertices of even degree, there is no Eulerian path or cycle over the bridges of the river Pregel.

By a similar argument to that of Theorem 1 you should be able to prove [21]:

**Theorem 2.** A connected digraph with no loops has an Eulerian cycle if and only if every vertex has the same indegree as outdegree.

We may apply Theorem 2 to the de Bruijn cycle problem introduced in Example 2, Section 3.1. Figure 3.16 displays two examples of **de Bruijn graphs**, which are digraphs depending on a parameter  $n$  with the following properties:

- a) A de Bruijn graph has  $2^n$  vertices, each of which is labeled with one of the  $2^n$  distinct sequences of  $n$  0's and 1's.



**FIGURE 3.16**

De Bruijn Graphs  
for (a)  $n = 1$  and  
(b)  $n = 2$ . (For  
 $n = 3$  see Fig. 3.2.)

b) Let a given vertex have the label

$$b_1 b_2 \dots b_n, \quad b_i = 0 \text{ or } 1. \quad (3)$$

Then from this vertex there are two outgoing edges labeled, respectively, 0 and 1 to the vertices with labels

$$b_2 b_3 \dots b_n 0 \quad \text{and} \quad b_2 b_3 \dots b_n 1. \quad (4)$$

From property b) and expression (4) it follows that each vertex (3) also has two incoming edges from the vertices with labels

$$0b_1 b_2 \dots b_{n-1} \quad \text{and} \quad 1b_1 b_2 \dots b_{n-1}. \quad (5)$$

The reason is that, whatever the value of  $b_n$ , each of the nodes with labels given by expression (5) will have one outgoing edge to the node with label (3).

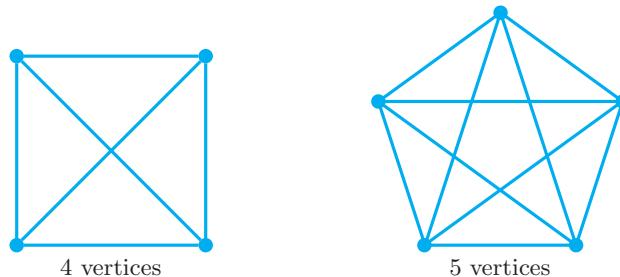
Can we apply Theorem 2 to de Bruijn graphs? Clearly, every vertex has indegree and outdegree 2. But is a de Bruijn graph a connected graph? Yes, it is, but we leave a proof of this to [30]. So, from Theorem 2 we know that there is at least one Eulerian cycle on any de Bruijn graph (see Fig. 3.2 for the case  $n = 3$ ). The length of this cycle (i.e., the number of edges on it) is  $2^{n+1}$ , as each of the  $2^n$  vertices has 2 edges emanating from it. If we traverse any such Eulerian cycle, writing down the label on each edge as we do so, the resulting sequence includes all possible subsequences of length  $n + 1$  if we allow wraparound from the end to the beginning of the sequence [8]. For example, starting from the node labeled 11 in Fig. 3.16b and proceeding successively to nodes 10, 00, 00, 01, 10, 01, 11, and finally back to 11 again, we obtain the sequence

$$00010111, \quad (6)$$

which contains all possible subsequences of length 3 (find them!).

We conclude this section with one theorem about **Hamiltonian paths**, which are paths that pass through every vertex of a graph once and only once. Thus, naturally, a **Hamiltonian cycle** is a Hamiltonian path that is also a cycle.

To state this theorem, we first need to define a **complete graph**, which is a simple undirected graph (thus no loops) containing every possible edge  $\{u, v\}$ . Figure 3.17 shows complete graphs with 4 and 5 vertices.



**FIGURE 3.17**

Two complete graphs.

**Theorem 3.** Every complete graph contains a Hamiltonian path.

**PROOF** This is an existence theorem, so we naturally prove it by using an algorithm. As with Pathgrow, Algorithm 3.5 is a follow-your-nose algorithm. Not much of an algorithm, you may think. But it does the trick. Suppose you got stuck somewhere and couldn't proceed without returning to a vertex already on the path. But you can't be stuck at a vertex  $u$  if there is a vertex  $w$  not already on the path because, by the assumption of completeness, there is an edge  $\{u, w\}$  in the graph and so we can go from  $u$  to  $w$ . Thus the process can be stopped only when all the vertices are on the path. ■

**Algorithm 3.5**  
**HamPath**

**Input**  $G(V, E)$  [A complete graph]

**Output** A Hamiltonian path

**Algorithm** HAMPATH

Choose an arbitrary vertex  $v$  and traverse a path which never returns to the same vertex twice.

A corollary whose proof you should be able to supply [31] is

**Corollary 4.** Every complete graph contains a Hamiltonian cycle.

A famous problem related to this corollary is the Traveling Salesman Problem, which we shall discuss in Section 4.9.

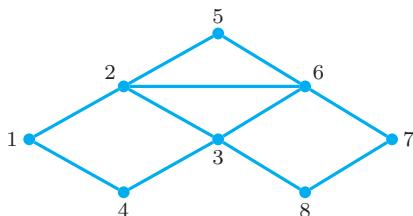
One final point. If you label the vertices of a graph  $1, 2, \dots, n$ , and then write these labels as you traverse a Hamiltonian graph, the result is a **permutation** of the integers  $1, 2, \dots, n$ . In Section 4.9, we will discuss ways of generating such permutations.

## Problems: Section 3.3

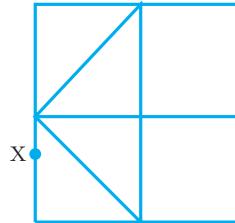
1. (1) What's the least number of new bridges which would have to be built in Königsberg for a cyclical walk without retracing one's steps? Using Fig. 3.15b, is there more than one choice for where to build the bridge(s)?

2. (2) Carry out Algorithm ECYCLE for the following graph twice, each time starting at 1.

- a) First, whenever there is a choice of which edge to choose next, choose the permissible edge which goes to the lowest-numbered vertex. Whenever (in a recursive call) there is a choice of which vertex to start at, choose the lowest-numbered vertex. Show pictures of your partial results at enough stages so that a grader can be certain about what you have done and in what order.
- b) Use the highest-numbered vertex throughout instead of the lowest-numbered.



3. (2) The Highway Department of Smallville is instructed to repaint the white lines which go down the middle of all roads in town. On the map of Smallville, the department's garage is marked with an  $X$ .



Is it possible for the Department to send out its paint truck so that it can paint all the lines without going over any stretch of road twice (and end up back at the garage)? Explain. If it is not possible, what is the smallest number of blocks (edges on the map) which the truck must travel twice?

4. (2) Consider the de Bruijn graph  $D(3, 2)$ , in which the 3 means that the vertex labels are *triples* and the 2 means that each of the three is a *binary* digit.
- a) Draw this graph. It's already in the text, but draw it from scratch to show that you understand how to construct it.
- b) Using our correspondence between paths in de Bruijn graphs and sequences of bits, find and highlight in  $D(3, 2)$  the cycle corresponding to the sequence 0101100 (*Hint:* At what vertex must the cycle end?)
- c) Using a new copy of  $D(3, 2)$ , pick a cycle of your choosing with six edges (none of which is a loop), highlight it, and then write the sequence  $S$  corresponding to it.
5. (2) Consider the de Bruijn graph  $D(1, 3)$ , in which the vertex labels are single "trits" (ternary digits, i.e., 0, 1, and 2).
- a) Draw it.

- b) Highlight the cycle corresponding to the sequence 01210211 starting at vertex 1. Note that this path overlaps itself. What caused that?
- c) By eyeballing (or using our algorithm informally — you don't have to explain what you did), indicate a directed Eulerian cycle in  $D(1, 3)$ . Indicate what your cycle is by labeling the edges 1, 2, 3, ... (in circles, so as not to confuse them with other numbers) in the order of traversal on the cycle.
- d) Write a ternary “lock-picking” sequence which will open doors at a hotel whose room numbers are two trits.
6. ⟨3⟩ Referring to Example 2 in Section 3.1, in an actual hotel, each of the four digits can be anything from 0 to 9. Is there still a de Bruijn cycle for the burglar to use? (Obviously you aren't going to find out by drawing the graph!)
7. ⟨3⟩ Generalize further. An  $n$ -long,  $k$ -ary sequence is a sequence of  $n$  symbols, each of which is chosen from a fixed set of  $k$  symbols. Thus the hotel problem is about 4-long, 10-ary sequences. For which  $n$  and  $k$  is there a de Bruijn cycle?
8. ⟨3⟩ Show that, if we traverse any Euler cycle in the de Bruijn graph  $D(n, 2)$ , writing down the label on each edge as we go, the resulting sequence includes all possible subsequences of length  $n + 1$  if we allow wraparound.
9. ⟨3⟩ In the game of dominoes, two dominoes can be put end to end if the ends have the same number of dots on them. A standard set of dominoes has one piece for each (unordered) pair of distinct integers from 0 to 6 inclusive — for example (0, 3) and (4, 6) — and one piece of the form  $(i, i)$  for each  $i$  from 0 to 6.
- a) Is it possible to arrange all the dominoes into one big circle?
- b) If all the pieces with no dots on one or both ends are excluded, is it possible to arrange the remaining pieces in a circle?
10. ⟨3⟩ The following is a simplified version of situations that arise in DNA research. The graph theory that applies here also applies to actual problems.
- Suppose a chromosome is made up of building blocks A, B, and C, which are used repeatedly in some sequence. Suppose biochemists believe that the chromosome is a single closed loop; that is, they believe it is some one circular sequence of A's, B's, and C's. By a chemical technique, it is possible to break many copies of the chromosome randomly into pieces exactly two blocks long and then count the relative frequency of different pieces. For instance, suppose the chromosome was very simple:
- 
- Then the following pieces would show up with equal frequency, and no other pieces would be found:
- AB, BC, CD, DA.
- However, the chemical technique does not preserve the orientation of the blocks around the circle, which is unknown to the researchers. For example, they can't tell DA and AD apart. So they would probably record the data as equal frequency for
- AB, AD, BC, CD.
- All right, suppose this technique is carried out on the mystery chromosome, and the pieces and their relative frequencies are recorded as:
- |       |       |
|-------|-------|
| AA 3, | BB 0, |
| AB 2, | BC 4, |
| AC 2, | CC 1. |
- Is this data consistent with the circular hypothesis?
11. ⟨3⟩ Now suppose that the chemical technique is more precise than in [10]. We get to pick a starting point on the chromosome and chop the whole thing from that point on into subsequences two blocks long; furthermore, we get an exact count of the number of each type of piece. Then we get to take another copy of the chromosome, shift over by one block from our previous starting point, and chop again. The first group of relative frequencies is
- |       |       |
|-------|-------|
| AA 2, | BB 0, |
| AB 1, | BC 2, |
| AC 0, | CC 1, |
- and the second is

|       |       |
|-------|-------|
| AA 1, | BB 0, |
| AB 1, | BC 2, |
| AC 2, | CC 0. |

(Again AB and BA, say, are indistinguishable as types.) Is this data still consistent with the hypothesis that the chromosome is a single loop?

12. ⟨2⟩ How many components are there in the divisibility graph of [33] in Section 3.1?
13. ⟨2⟩ Instead of the proof-by-algorithm given for the necessity part of Theorem 1, just give a normal deductive proof based on the definition of an Eulerian cycle.
14. ⟨2⟩ Argue that the steps in the non-constructive proof of the sufficiency part of Theorem 1 do not at any point change the *parity* (evenness or oddness) of the degree of any of the vertices in  $V$ .
15. ⟨2⟩ Write an algorithm which, given as input an undirected graph in adjacency matrix form, determines whether all the vertices have even degree. It should print Yes or No; if No, it should name a vertex with odd degree.
16. ⟨2⟩ For the components that result after the first call of procedure Euler show that
  - a) Each vertex on any component must have even degree.
  - b) Each component must have a vertex on the cycle found on the original call of Euler.
17. ⟨2⟩ How do you know that, if all the vertices of a graph have even degree, then the output of Path-grow is always a cycle?
18. ⟨2⟩ Given a graph as a list of edges (i.e., as pairs of vertices), and a given vertex,  $v$ , write an algorithm to find  $I(v)$ , the set of edges incident on  $v$ .
19. ⟨3⟩ Algorithm ECYCLE blithely assumes that all the items following **then** can be accomplished in a constructive context. But can they? Suppose you are given a graph  $G$  as a set of edges  $E$  (such that  $E$  is a list of pairs of vertices) and a cycle  $C'$  on that graph, also as a list of edges.
  - a) Write an algorithm to find  $E'$ , the subset of  $E$  that contains no edges on  $C'$ .
  - b) Now sketch — don't actually write the algorithm — how you would use  $E'$  and  $C'$  to find the nontrivial components of the graph defined by  $E'$ .

c) Your answer to b) should imply why each non-trivial component of the graph defined by  $E'$  contains at least one vertex of  $C'$ . Does it? Explain why.

20. ⟨2⟩ In the proof of correctness of ECYCLE we used the case of a graph with one edge as the basis case.
    - a) Instead use as the basis case the graph with zero edges.
    - b) Now use the case  $n = 2$  as the basis case.
  21. ⟨2⟩ Prove Theorem 2, the directed version of the Eulerian cycle theorem.
  22. ⟨3⟩ The Eulerian cycle theorem (Theorem 1) has a path version: A connected, undirected multigraph has an Eulerian path if and only if there are either zero or two vertices of odd degree. (If there are zero vertices of odd degree, every Eulerian path is a cycle.) Prove this version three ways:
    - a) Give an existence proof similar to Proof 1 of Theorem 1 in the text.
    - b) Give a proof by algorithm similar to Proof 2 of Theorem 1 in the text. (*Suggestion:* You need only consider the case of two odd-degree vertices. Why? Where should you start growing your path?)
    - c) Both a) and b) take as much work as the original Proofs 1 and 2, yet are very similar to them. Is there a way to avoid this repetition? Yes: Instead of slightly changing the proof, change the graph. If  $G$  has two vertices of odd degree, temporarily add a new edge between them. Call the resulting graph  $G'$ . Now apply Theorem 1 to  $G'$ . What does this tell you about  $G$ ? Why?
- Part c) is a standard example of Reduce to the Previous Case. It is also sometimes called the **method of ideal elements**. The ideal element in this case is the new edge, which is not really in the graph. That is, it is ideal, not real.
23. ⟨2⟩ Modify the algorithm [15] so that it prints Yes if either all vertices are of even degree or exactly two are of odd degree (and still prints No otherwise). When there are two vertices of odd degree, the algorithm should print the indices of those vertices.
  24. ⟨3⟩ The directed Eulerian cycle theorem has a path version. Below, for variety, we state the result to cover only paths which are not also cycles. (Compare with [22].)

A connected digraph has a directed Eulerian path (beginning and ending at different vertices) if and only if all vertices except two have indegree equal to outdegree, and one of those two has outdegree 1 greater than indegree while the other has indegree 1 greater than outdegree.

- a) Prove this theorem by an existence argument.
- b) Prove it by analyzing an algorithm.
- c) Prove it by the method of ideal elements (see [22]).

**25.** ⟨3⟩ Discover and prove a necessary and sufficient condition for a connected graph to be *traceable* (i.e., each edge appears once and only once on a path) using exactly two paths. (The two paths can't be cycles or have an end in common.)

**26.** ⟨1⟩ Suppose that you are given a connected digraph in which the indegree and outdegree are the same at every vertex. Obviously, if you strip off the arrows, you're left with a connected graph with even degree everywhere. It's not so obvious how to do the reverse procedure — given a connected graph with even degree at every vertex, direct the edges so that  $\text{indegree} = \text{outdegree}$  at each vertex — because putting on an orientation which “balances” is more difficult than stripping it off. Nonetheless, the reverse is always possible, and its proof by algorithm is easy. Explain. (*Hint:* Take a trip around the graph.)

**27.** ⟨1⟩ Prove that the result of [26] is still true even in the unconnected case.

**28.** ⟨3⟩ Prove: It is possible to direct the edges of any graph  $G$  so that, at each vertex  $v$ , the outdegree is either  $\lfloor \deg(v)/2 \rfloor$  or  $\lceil \deg(v)/2 \rceil$ . (Note that this is a natural generalization of [26].)

**29.** ⟨2⟩ Can the result of [28] be strengthened further? Is it always possible to choose in advance which vertices will have outdegree  $\lfloor \deg(v)/2 \rfloor$ ?

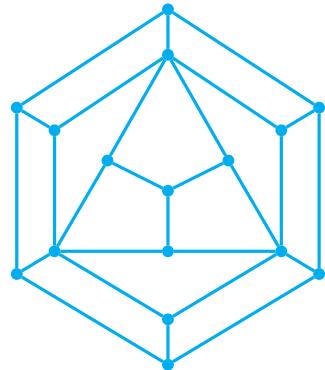
**30.** ⟨3⟩ In order to prove that a de Bruijn graph is connected, you need to show that there is a path from any vertex to any other. Do so by using the definition of a de Bruijn graph in Eqs. (3), (4), and (5) to find a sequence of vertex labels that defines the desired path.

**31.** ⟨2⟩ Prove Corollary 4: Every complete graph has a Hamiltonian cycle.

**32.** ⟨3⟩ How many distinct Hamiltonian cycles does the complete graph with  $n$  vertices have? You need to specify what you count as distinct.

**33.** ⟨2⟩ Theorem: Every tournament contains a directed Hamiltonian path. We have already proved this theorem. Where?

**34.** ⟨3⟩ Show that the following graph has no Hamiltonian path. (*Hint:* Label the vertices with A's and B's so that A vertices are adjacent only to B vertices and vice versa. What would happen if there were a Hamiltonian path?)



**35.** ⟨4⟩ Suppose that you want to list all three-digit binary numbers in such an order that each number differs from the previous one by just one digit.

- a) Turn this into a path problem about some graph.
  - b) Answer the question by drawing the graph and eyeballing a solution. (*Hint:* Try to draw the graph in a pleasing, symmetric way; the fact that each vertex has three digits may help.)
  - c) Is there still a solution if the first and last numbers must also differ by just one digit?
  - d) Can you generalize your answers to b) and c) to  $n$ -digit binary numbers?
  - e) Can you generalize to  $n$ -digit,  $k$ -ary numbers?
- 36.** ⟨2⟩ Derive a formula for the number of edges of a complete graph with  $n$  vertices.
- 37.** ⟨2⟩ Let  $G$  be the complete graph on four vertices. Give
  - a) a connected subgraph with three edges.
  - b) a disconnected subgraph with three edges.
  - c) a spanning subgraph with two edges and no isolated vertices.

38. (3) A complete subgraph of a graph is called a *clique*. The *clique number* of a graph  $G$  is the maximum number of vertices in a clique of  $G$ . (See also Section 3.6.) What is the clique number of the graph defined by

$$V = \{1, 2, \dots, 25\}, \quad E = \{\{i, j\} \mid |i - j| > 4\}$$

39. (3) A **bipartite graph**  $G(V, E)$  is one in which the vertex set  $V$  can be partitioned into two disjoint sets,  $V_1$  and  $V_2$  such that every edge of  $E$  joins a vertex in  $V_1$  to a vertex in  $V_2$  (see also

Section 3.6). A **complete bipartite graph** is a bipartite graph in which there is an edge between *every* pair of vertices in the sets  $V_1$  and  $V_2$ . The complete bipartite graph  $K_{mn}$  has  $m$  vertices in  $V_1$  and  $n$  vertices in  $V_2$ .

- a) Draw  $K_{14}$  and  $K_{23}$ . The former is an example of a *star*. Why do we use this name?
- b) In general, how many edges does  $K_{mn}$  have?
- c) What is the size of the largest clique in any bipartite graph?

## 3.4 Shortest Paths

Finding the shortest path between vertices of a graph in which each edge has a nonnegative **length** (or **weight**) is a common problem. Consider, for example, a road map of the United States such as the abbreviated one shown in Fig. 3.18. If you were going to drive from New York to Los Angeles and were constrained to follow a route using only the links in Fig. 3.18, you would want the shortest path between the two (unless, perhaps, you wanted to visit your grandparents in Minneapolis on the way). By **shortest path** we naturally mean the least distance, that is, the path whose edge lengths have minimum sum.

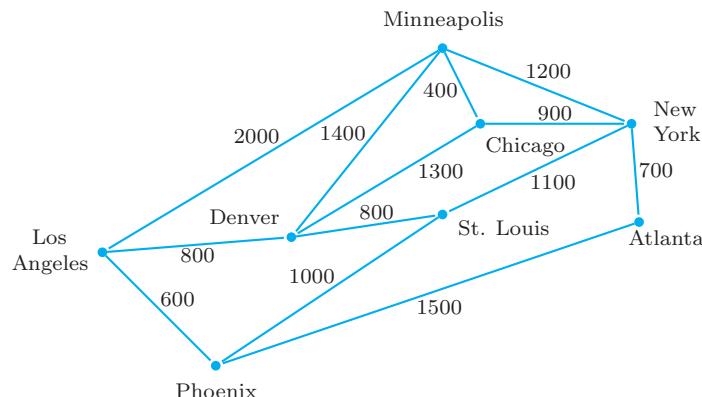


FIGURE 3.18

Portion of a mileage map.

Many other problems may be interpreted as shortest path problems on graphs, even when the edge weights are not distances and paths cannot be interpreted as trips. For example, we could reinterpret Fig. 3.18 as a chart in which the numbers represent the costs of sending messages between each pair of cities. In that case we could find the minimum-cost communication path from New York to Los Angeles. As another example, suppose that there were weights on the edges of the wolf-cabbage-goat-farmer graph (Fig. 3.1) which might represent the costs of trips with particular passengers. (It would be reasonable to charge more to transport a heavy

wolf than a light — even if large — cabbage.) Then we might ask for the cheapest way to transport the wolf, the cabbage, and the goat across the river.

We will assume that shortest path problems are given as digraphs because, in fact, they often are. But even when they are not, as in the transportation and communications examples, we can convert the given graph into a digraph by replacing each edge by two edges, one in each direction, with equal weights.

Consider, then, the digraph shown in Fig. 3.20 on p. 270. Our object is to find the shortest path from  $S(\text{tart})$  to  $F(\text{inish})$  and, in so doing, to develop a general shortest path algorithm for any digraph with nonnegative edge weights. (We limit ourselves to nonnegative weights not only because this is natural for distance and many other problems, but also because the method of this section simply fails otherwise [16].)

As with so many problems in discrete mathematics, the choice of a good approach can make a great difference in efficiency. One approach might be to compute first the number of paths from one vertex to the other by the technique already described in Section 3.3. Then we could search for each of these paths and calculate the length of each. Although possible, this approach would be quite slow because large digraphs will typically have many such paths. By contrast the method we are about to describe is not only quite elegant but also efficient.

So for any digraph  $G(V, E)$  such as that in Fig. 3.20, we wish to find a shortest path from some vertex  $S$  to some vertex  $F$ . We have seen (in the Prologue) that sometimes the best way to solve a problem is to generalize it. This is such a case. Our generalization is to develop a method that can be used to find the shortest paths from  $S$  to *all* the other vertices. We shall do this by finding the vertices in order of their distance (i.e., their *shortest* distance) from  $S$  until we have found the distance to  $F$ .

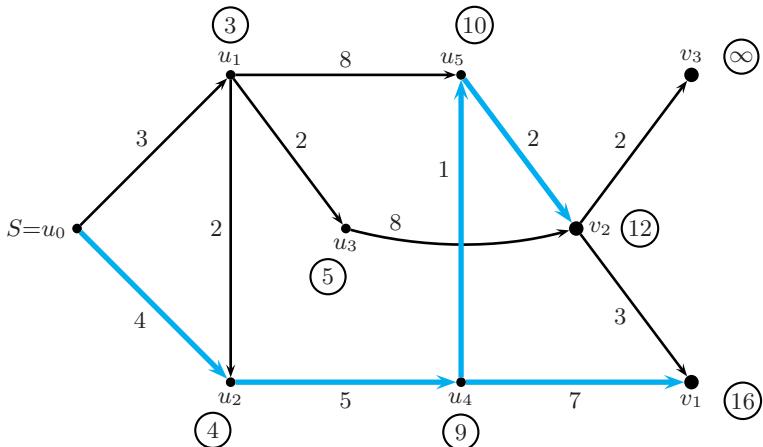
Specifically, for each  $k = 0, 1, 2, \dots$ , let  $U_k$  consist of  $S$  and the  $k$  other vertices least distant from  $S$ . (Don't worry about ties here or later in this section; if two or more vertices are equidistant from  $S$ , we can choose them in any order.) Henceforth, let the vertices of  $U_k$  be called  $u_0, u_1, \dots, u_k$ , where  $u_0 = S$ ,  $u_1$  is the closest to  $S$ ,  $u_2$  the next closest, and so on. For each edge  $(u, v)$ , denote its weight by  $w(u, v)$ .

Now use the recursive paradigm! Suppose we have just found  $u_k$  so that we know  $u_0, u_1, \dots, u_k$ . How do we find  $u_{k+1}$ ?

It turns out we can find  $u_{k+1}$  efficiently if we compute partial information about the distances to *all* vertices each time we find a new  $u_k$ . For each  $k = 0, 1, 2, \dots$ , let us call a (directed) path from  $u_0$  to any vertex  $v \in V$  a  **$U_k$ -path** if all its vertices are in  $U_k$ , except perhaps for the end vertex  $v$ . Let us call the length of the shortest  $U_k$ -path to  $v$ , if any  $U_k$ -path exists, the **current distance** and denote it by  $d(v)$ . (If there is no such path, set  $d(v) = \infty$ . When won't there be a  $U_k$ -path to a vertex  $v$  [20]?) As necessary, we'll denote  $d(v)$  by  $d_k(v)$ , since it depends on  $U_k$ , but mostly we shall not need the subscript. We shall show just below that  $d(v)$  is the true least distance when  $v$  enters  $U_k$ . However, for  $v \notin U_k$ ,  $d(v)$  may or may not be the true least distance to  $v$  since the true shortest path to  $v$  may not be a  $U_k$ -path. As the algorithm progresses [21],  $d(v)$  is updated until it becomes the true least distance at some stage before  $v$  enters  $U_k$ .

So, suppose again that we have already found  $U_k$  ( $u_0 = S$  and its  $k$  closest vertices) and that for all  $v$  we have found  $d(v) = d_k(v)$ . To find  $u_{k+1}$ , first note that the shortest path to the closest vertex to  $S$  not in  $U_k$  must be a  $U_k$ -path. For if  $v$  were the closest vertex, yet there was some other vertex  $w \notin U_k$  on the shortest path to  $v$ , then  $w$  would be closer to  $S$  than  $v$ . Thus, for that  $v \notin U_k$  for which  $d(v)$  is least,  $d(v)$  is the shortest distance, and we should set  $u_{k+1} = v$ .

Fig. 3.19 illustrates this situation.



Finding a shortest  $U_k$ -path with  $k = 5$ . Small dots are vertices in  $U_k$  and large dots are vertices  $\notin U_k$ . The circled numbers are the current distances  $d(v)$  to each vertex. There are  $U_5$ -paths to vertices  $v_1$  and  $v_2$  but not to vertex  $v_3$ . The colored edges are the shortest  $U_5$ -paths to  $v_1$  and  $v_2$ . The vertex to be added to  $U_5$  so that it becomes  $U_6$  is  $v_2$ . Note that after  $v_2$  is added to  $U_5$ ,  $d(v_3)$  becomes 14 because there is a  $U_6$ -path to  $v_3$ . Also note that there is a shorter  $U_6$ -path to  $v_1$  than the shortest  $U_5$ -path, so  $d(v_1)$  becomes 15. Then  $v_3$  will be the vertex added to  $U_6$  to make  $U_7$ .

In short, to find  $u_{k+1}$ , we merely find the  $v \notin U_k$  for which  $d(v)$  is minimum. An algorithm fragment to do this is

```

mindist ← ∞
for v ∈ U_k
    if d(v) < mindist then u_{k+1} ← v; mindist ← d(v)
endfor

```

(1)

But this isn't all we need to do before we can use this fragment again (with  $k$  replaced by  $k + 1$ ) to find  $u_{k+2}$ . First, we must update  $d(v)$  for  $v \notin U_{k+1}$  to give the shortest length of a  $U_{k+1}$ -path to  $v$ .

What could cause  $d_{k+1}(v)$  to differ from  $d_k(v)$  for  $v \notin U_{k+1}$ ? This could happen only if there is a  $U_{k+1}$ -path to  $v$  that is not also a  $U_k$ -path (i.e., which actually has  $u_{k+1}$  on it — why?). We claim that such a path  $P$  to  $v$  could be shorter than the

shortest  $U_k$ -path to  $v$  only if there is an edge  $e = (u_{k+1}, v)$  and  $P$  ends with  $e$ . For if instead  $P$  ended with  $(u_i, v)$ , where  $i < k+1$ , then, for the segment from  $S$  to  $u_{k+1}$  to  $u_i$ , we could just substitute the shortest path to  $u_i$ , resulting in a  $U_k$ -path to  $v$  shorter than the path with  $u_{k+1}$  on it [8].

There are two possibilities:

1. If  $d_k(v) = \infty$ , the path consisting of  $e$  appended to the shortest path to  $u_{k+1}$  is a  $U_{k+1}$ -path to  $v$  of finite length and thus shorter than  $d_k(v)$ .
2. If  $d_k(v) \neq \infty$ , then it *might* be the case that the  $U_{k+1}$ -path consisting of the shortest  $U_k$ -path to  $u_{k+1}$  with  $e$  appended to it has a length less than  $d_k(v)$ .

In fact, both these possibilities can be included in a single statement in the following algorithm fragment for updating  $d(v)$  for  $U_{k+1}$ -paths.

```

for  $v \notin U_{k+1}$  for which  $(u_{k+1}, v) \in E$ 
     $d(v) \leftarrow \min\{d(v), d(u_{k+1}) + w(u_{k+1}, v)\}$ 
endfor

```

(2)

Now we can give a sketch of the whole shortest path algorithm, into which the fragments above will be substituted later to give the complete algorithm. (Actually, as often happens, in the complete algorithm none of the subscripts are necessary.)

```

Initialize  $U$  to  $\{S\}$ 
Initialize  $d(v)$  to 0 for  $v = S$ , to  $w(S, v)$  if  $(S, v) \in E$ , to  $\infty$  otherwise
repeat
    Find a vertex  $u \notin U$  for which  $d(u)$  is minimum using fragment (1)
    if  $u = F$  then exit
     $U \leftarrow U \cup \{u\}$ 
    Update  $d(v)$  for all  $v \notin U$  using fragment (2)
endrepeat

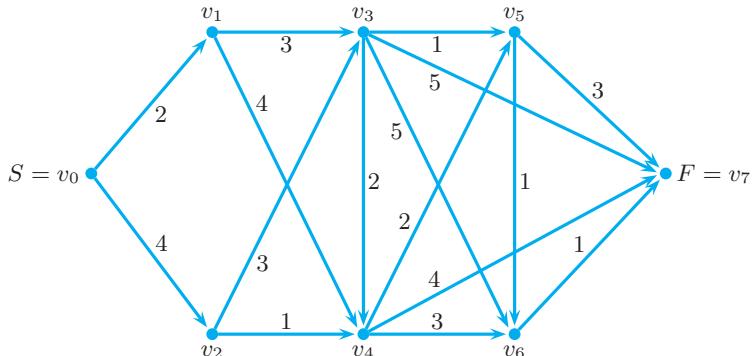
```

Before giving a more detailed statement of this algorithm (Algorithm DIJKSTRA on p. 272), we'll show how it works with the digraph of Fig. 3.20.

## EXAMPLE 1

Find the shortest path from  $S$  to all other vertices in Fig. 3.20.

**Solution** The following table contains the complete answer. The values in row  $k$  are those at the end of the  $k$ th iteration of the repeat-loop in the algorithm sketch immediately above. In particular, the row with  $k = 0$  represents the state just before the first entry into the repeat-loop. (*Note.* The numbers in the Vertex Added columns are the names of vertices; e.g., “2” is the vertex labeled  $v_2$  in Fig. 3.20.) The numbers in all the  $d(v_i)$  columns are current distances.)



**FIGURE 3.20**

The Shortest Path Problem. Object:  
Find the shortest path from  $S$  to  $F$ .

| k | Vertex<br>Added<br>to $U$ | Vertex<br>Added |        |          |          |          |          |          |  |
|---|---------------------------|-----------------|--------|----------|----------|----------|----------|----------|--|
|   |                           | $d(1)$          | $d(2)$ | $d(3)$   | $d(4)$   | $d(5)$   | $d(6)$   | $d(F)$   |  |
| 0 | $S$                       | 2               | 4      | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |  |
| 1 | 1                         | 2               | 4      | 5        | 6        | $\infty$ | $\infty$ | $\infty$ |  |
| 2 | 2                         | 2               | 4      | 5        | 5        | $\infty$ | $\infty$ | $\infty$ |  |
| 3 | 3                         | 2               | 4      | 5        | 5        | 6        | 10       | 10       |  |
| 4 | 4                         | 2               | 4      | 5        | 5        | 6        | 8        | 9        |  |
| 5 | 5                         | 2               | 4      | 5        | 5        | 6        | 7        | 9        |  |
| 6 | 6                         | 2               | 4      | 5        | 5        | 6        | 7        | 8        |  |
| 7 | $F$                       |                 |        |          |          |          |          |          |  |

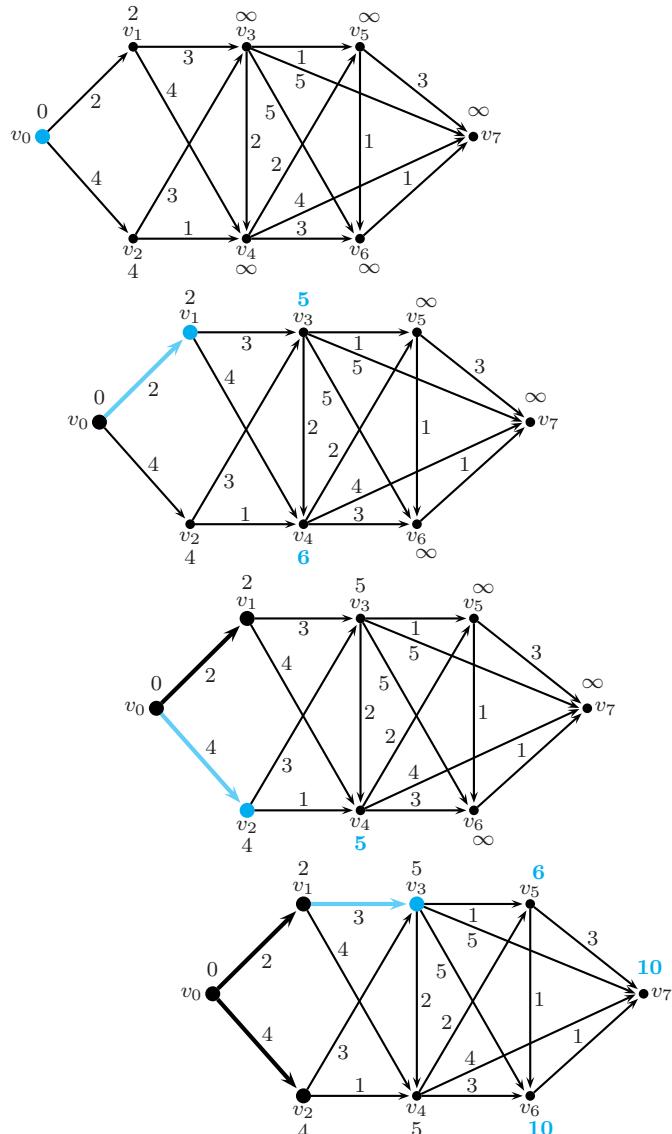
*Remarks:*

1. The vertex  $k$  enters the set  $U$  on the  $k$ th step only because of the way we numbered the vertices in Fig. 3.20.
2. After iteration 2,  $d(3) = d(4)$ , so we arbitrarily chose  $u_3$  to be 3 [17].
3. Focusing on vertex 6, note that after iteration 3 the shortest  $U_3$ -path  $(S, 1, 3, 6)$  has length 10 (another  $U_3$ -path  $(S, 2, 3, 6)$  has length 12). But after iteration 4, the  $U_4$ -path  $(S, 2, 4, 6)$  with length 8 is now the shortest, and then after iteration 5 the  $U_5$ -path  $(S, 1, 3, 5, 6)$  with length 7 becomes the shortest.
4. Our method doesn't actually give the shortest path to  $F$  but only the *length* of the shortest path, namely, 8. We'll discuss the details of recording the shortest path itself at the end of this section.

Alternatively to the table method just completed, humans can with small graphs carry out this algorithm right on one or more copies of the graph. Fig. 3.21 shows how, by showing the initial conditions and the results of the first 3 iterations for the same graph as before. Vertices that have joined  $U_k$  have large dots. The number above or below each  $v_i$  is the current  $d(v_i)$ . The vertex that joins  $U_k$  in each iteration is colored, as are the values of  $d(v_i)$  that change in that iteration. The coloring is just for highlighting. In fact, all the essential steps of this graphical

method can be carried out on a single copy of the graph, if it doesn't get too cluttered, by crossing out old  $d(v)$  values as new ones arise.

One thing done in this figure that is not done in the algorithm is to highlight with color and thickness the edge to each new  $u \in U$  from the prior vertex as discussed above. Notice that we have kept these edges thick as we move on to later iterations. If you continue this graphical solution until the end, you will notice something interesting about the thick edges [3]. ■



**FIGURE 3.21**

Part of Algorithm DIJKSTRA for Fig. 3.20 carried out graphically. The algorithm is represented just before it enters the repeat-loop and after each of the first 3 iterations.

Our more complete version of the shortest path algorithm, Algorithm 3.6, is named after Edsger Dijkstra, who first published a version of it in 1956.

## Algorithm 3.6 Dijkstra

|               |                         |                                                            |
|---------------|-------------------------|------------------------------------------------------------|
| <b>Input</b>  | $G(V, E)$               | [Digraph and the weight $w(u, v)$ for each edge $(u, v)$ ] |
|               | $S, F$                  | [Start and Finish vertices]                                |
| <b>Output</b> | $\text{Distance } d(F)$ |                                                            |

**Algorithm** DIJKSTRA

```
     $U \leftarrow \{S\}$                                 [Initialize  $U$ ]
     $u \leftarrow S$                                [ $u$  is the most recent vertex to enter  $U$ ]
     $d(S) \leftarrow 0$                            [Initialize distance function]
    for  $v \in V - \{S\}$ 
        if  $(S, v) \in E$  then  $d(v) \leftarrow w(S, v)$ 
        else  $d(v) \leftarrow \infty$ 
    endfor
    repeat   [Main algorithm]
         $mindist \leftarrow \infty$                       [Find new  $u$ ]
        for  $v \notin U$ 
            if  $d(v) < mindist$  then  $u \leftarrow v$ ;  $mindist \leftarrow d(v)$ 
        endfor
        if  $u = F$  then exit                         [Distance to  $F$  found]
         $U \leftarrow U \cup \{u\}$                           [Update  $U$ ]
        for  $v \notin U$  and  $(u, v) \in E$            [Update  $d(v)$ ]
             $d(v) \leftarrow \min\{d(v), d(u) + w(u, v)\}$ 
        endfor
    endrepeat
```

*Remarks:*

- This algorithm works equally well for undirected graphs so long as the input includes for each edge  $(u, v)$  another edge  $(v, u)$  with the same weight as  $(u, v)$ .
- If all the lengths are 1, this algorithm will find the path with the least number of vertices on it, and in this case becomes an example of Breadth First Search, discussed in Section 3.5 (see Algorithm PATHLENGTH there).
- The algorithm computes the distance to all vertices whose distance is found before  $d(F)$ . This makes it easy to adapt the algorithm to find the distance to all vertices if that is desired [6].

**Theorem 1.** Algorithm DIJKSTRA is correct; that is, given a digraph with nonnegative weights on the edges, and start and finish vertices  $S$  and  $F$ , it will find the shortest distance from  $S$  to  $F$ .

**PROOF** As usual, our tool is induction using a loop invariant, but now the invariant is quite complex. For  $k = 0, 1, 2, \dots$ , let  $P(k)$  be the assertion that, at the *end* of the  $k$ th pass through the repeat-loop of the main algorithm, these three conditions hold:

1.  $U$  consists of  $S$  and the  $k$  other vertices closest to  $S$ .
2. For each  $u \in U$ ,  $d(u)$  is the length of the shortest path in  $G$  from  $S$  to  $u$ .
3. For each  $v \notin U$ ,  $d(v)$  is the length of the shortest  $U_k$ -path to  $v$ , where  $d(v) = \infty$  if there is no such path. (Recall that a  $U_k$ -path to  $v$  starts at  $S$  and has intermediate vertices in  $U_k$  only.)

*Note.* Previously we insisted that loop invariants need to be correct on *entry* to each pass through the loop. So why did we make an assertion above about  $P(k)$  at the *end* of the pass? Answer: To avoid clumsiness. If our claim about  $P(k)$  had referred to the entry, then everywhere below where we write  $k$ , we would instead have had to write  $k - 1$ .

The basis case ( $k = 0$ ) claims that the loop invariant is true at the “end of the 0th pass”, i.e., before we enter the loop the first time. This claim is true because of the initialization lines in the algorithm; specifically:

- Condition 1 is true because  $U$  is set to  $\{S\}$ , so  $U$  certainly contains  $S$  and the zero other vertices closest to  $S$ ;
- Condition 2 is true because  $d(S)$  is set to 0, which is certainly the shortest distance from  $S$  to itself; and
- Condition 3 is true because the only  $U_0$ -paths to  $v$  at this point are edges from  $S$ , and for all such we set  $d(v) = w(S, v)$  when  $(S, v) \in E$ . For all other  $v$  the condition is correct since we set  $d(v) = \infty$  for these.

As for the inductive step, assume  $P(k)$ . To prove  $P(k+1)$ , we have already argued that the next closest vertex,  $u_{k+1}$ , will have as its distance from  $S$  in  $G$  the length of the shortest  $U_k$  path to  $u_{k+1}$ , and by Condition 3 for  $P(k)$ , these lengths are exactly the values of  $d(v)$  at the end of the  $k$ th pass. Therefore,  $u_{k+1}$  is correctly identified at the *start* of pass  $k+1$  as the vertex  $v \notin U_k$  for which  $d(v)$  is least. Thus, when  $u_{k+1}$  is added to  $U$  in this pass, Conditions 1 and 2 remain true.

If  $u_{k+1} = F$ , then by Condition 2 the theorem is true and we do not need to finish the inductive step. If  $u_{k+1} \neq F$ , then, to finish the inductive step, we must also show that Condition 3 becomes true for the enlarged  $U$  by the updates to  $d$  during the  $(k+1)$ st pass. This too we have already argued, when we noted that the only way  $d(v)$  could decrease is if the shortest path from  $S$  to  $u_{k+1}$ , followed by edge  $(u_{k+1}, v)$ , if it exists, gives a shorter length than the old  $d(v)$ . Therefore, the final lines of the main repeat loop update  $d(v)$  correctly. ■

*Complexity.* Let’s now briefly do a complexity analysis of DIJKSTRA. Assume  $G$  has  $n$  vertices and that  $G$  is a simple digraph. Therefore it has up to  $n(n-1)$  edges, one for each ordered pair of vertices. The initializations will take  $\text{Ord}(n)$

steps, as there is an assignment for each vertex. Then the algorithm has two for-loops inside a repeat-loop. The repeat-loop is executed at most  $n - 1$  times. The first for-loop is executed at most  $n$  times, once for each vertex  $v$  to check if  $v \notin U$ , with a fixed amount of work if it is not. The second for-loop is executed  $n$  times if carried out in the obvious fashion — test each vertex  $v$  to see if it is adjacent to  $u$  and not in  $U$ . For  $v$  that meet the tests, there is a fixed amount of work within the for-loop. Thus, the main algorithm is  $\text{Ord}(n^2)$ . Conclusion: The whole algorithm is  $\text{Ord}(n^2)$  in the worst case when  $F$  is the farthest vertex from  $S$ .

**Extensions.** So far we only wanted to know the distance between a single pair of vertices,  $S$  and  $F$ . We leave it to you to change DIJKSTRA so that it always finds the shortest distance to all the vertices [6].

As a further extension, we could wish to solve the **all pairs** shortest path problem. You should be able to extend Dijkstra's algorithm to handle this [28]. Alternatively, you could use Warshall's algorithm in the form described in [26, Section 3.2]. How do these two algorithms compare in efficiency [29]?

*Finding the path.* To find the shortest path itself to each  $v$ , not just its length, at each iteration we keep track of the previous vertex in the shortest path to  $v$  found so far. This is easy to do, because whenever  $d(v)$  is reduced, the  $u$  that caused the reduction (in the last for-loop of DIJKSTRA) is the new previous vertex. Let us call this vertex  $\text{prev}(v)$ . When  $v$  enters  $U$ , we have the true shortest distance and  $\text{prev}(v)$  is the previous vertex on a true shortest path. To find the whole shortest path when  $v = F$ , after the main algorithm is completed we only have to work our way back to  $S$  by following the links backwards:  $v, \text{prev}(v), \text{prev}(\text{prev}(v)),$  etc. The changes needed in DIJKSTRA to compute the array of previous vertices  $\text{prev}(v)$  are minimal [18].

The following table records the values of  $\text{prev}(v)$  at the end of the  $k$ th pass through the repeat-loop for the graph of Fig. 3.20. Thus after iteration 1 the shortest path to 4 has previous vertex 1, but after iteration 2 the shortest path has previous vertex 2.

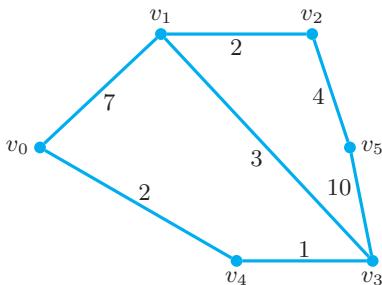
| $k$ | Previous vertex on shortest path so far to |     |   |   |   |   |     |
|-----|--------------------------------------------|-----|---|---|---|---|-----|
|     | 1                                          | 2   | 3 | 4 | 5 | 6 | $F$ |
| 0   | $S$                                        | $S$ |   |   |   |   |     |
| 1   |                                            |     | 1 | 1 |   |   |     |
| 2   |                                            |     |   | 2 |   |   |     |
| 3   |                                            |     |   |   | 3 | 3 | 3   |
| 4   |                                            |     |   |   |   | 4 | 4   |
| 5   |                                            |     |   |   |   | 5 |     |
| 6   |                                            |     |   |   |   |   | 6   |

On the shortest path to  $F$ , the prior vertex is 6; on the shortest path to 6, the prior vertex is 5, etc. Working our way back to  $S$ , we find that the shortest path from  $S$  to  $F$  is

$$(S, 1, 3, 5, 6, F).$$

## Problems: Section 3.4

1. (2) Find the shortest path lengths from  $v_0$  to all vertices in the following undirected graph, using Algorithm DIJKSTRA.



- a) Write up your work in a table of the form shown on p. 270.  
 b) Write up your work on copies of the graph, one for each iteration of the algorithm. Highlight the current  $U$  with color and show the current value of  $d(v)$  over each vertex.

2. (2) Repeat [1], but now let  $v_5$  be the start vertex.  
 3. (2) Repeat [1], but this time find the shortest paths themselves. Observe anything interesting about the union of these shortest paths?

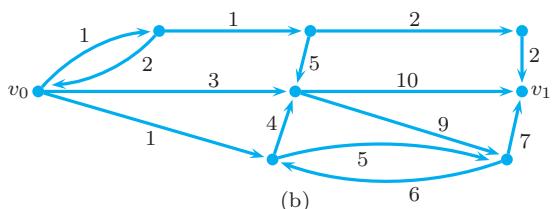
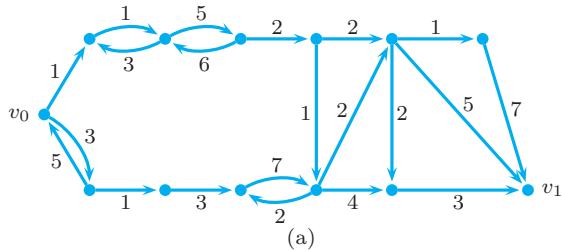
4. (2) Repeat [2] finding the shortest paths.  
 5. (2) Any CAS that does discrete mathematics should have Dijkstra's algorithm built in. For instance, *Mathematica* has a package *Combinatorica* which includes a command `Dijkstra[G, s]`, where  $G$  is a graph and  $s$  is a start vertex. Find out if your favorite CAS has a Dijkstra command and, if so,

- a) Use it to do [1] again.  
 b) Use it to repeat the calculation of Example 1.

6. (2)  
 a) Display a modification of DIJKSTRA that finds the shortest path from  $S$  to all the vertices of  $G$ . Show only your additions and modifications of the algorithm and where they go in the algorithm.  
 b) Apply the result of a) to the graph of [1] with  $S = v_0$ .  
 c) Apply the result of a) to the graph of [1] with  $S = v_5$ .

- d) Apply the result of a) to the graph of Fig. 3.20.

7. (2) Find the shortest path from  $v_0$  to  $v_1$  in the following digraphs. Show your work in abbreviated form on one copy of each graph (otherwise it's too tedious).



8. (2) Often the best way to understand a difficult or tricky part of an algorithm is to draw a diagram to illustrate what is going on. So to show that  $d_{k+1}(v)$  can differ from  $d_k(v)$  only if there is an edge  $e = (u_{k+1}, v)$ , draw a diagram that illustrates that no  $U_{k+1}$ -path without this edge could possibly be the shortest  $U_{k+1}$ -path to  $v$ .  
 9. (3) Consider the following **length matrix**, in which the  $(i, j)$  entry is the length of the directed edge from vertex  $i$  to vertex  $j$ . (If the  $(i, j)$  entry is “–”, there is no edge from  $i$  to  $j$ .) Determine the length of the shortest path from  $v_1$  to  $v_8$  without ever drawing any part of the graph. Also determine which vertices are on this path and in which order, again without drawing any part of the graph.

As humans, we prefer to carry out DIJKSTRA on small graphs with the aid of pictures. This problem asks you to perform DIJKSTRA in a manner much closer to how a computer would do it. Try to make your work easy for the reader to follow and please explain what you are doing!

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| - | - | 5 | - | - | 2 | 3 | - |
| 4 | - | 6 | - | 7 | - | 5 | - |
| - | 3 | - | 9 | 2 | 6 | - | 7 |
| 3 | - | 2 | - | 1 | - | 7 | 6 |
| - | 5 | - | 1 | - | - | 4 | - |
| - | - | 2 | - | 8 | - | 9 | - |
| 1 | 2 | 3 | - | - | 6 | - | - |
| 5 | - | 8 | - | 2 | - | 9 | - |

10. ⟨2⟩ Suppose that you have a weighted **bigraph**: Each edge has two weights on it, one for each direction. (For instance, an edge might represent a stretch of three-lane road, with two northbound lanes and one southbound lane; the weights might represent the average time needed to traverse the road going both north and south.) Again, you need to find the shortest paths from  $v$  to all other vertices, where the length of a path is the sum of its weights in the directions the edges are traversed. Show how to solve this problem using Algorithm DIJKSTRA.

11. ⟨2⟩ In discussing the complexity of DIJKSTRA, we assumed that our graph  $G$  is simple. But suppose  $G$  is a multigraph and suppose also that, when there are one or more edges joining two vertices  $u$  and  $v$ , the Input of the weights is a sequence of one or more values for each such pair  $(u, v)$ .

- a) What should you add to the initialization portion of DIJKSTRA so that the main algorithm in DIJKSTRA need not be changed at all?
- b) How about the complexity of DIJKSTRA? How must it be changed? To answer this, make an assumption about the maximum number of edges joining two vertices.

12. ⟨2⟩ When introducing the shortest path problem, we said that we would assume the graph is directed, because we could always make it directed by replacing each undirected edge with two directed ones. Is this a general procedure for reducing undirected graph problems to digraph problems? Does it work, for instance, in Eulerian path and cycle problems? Explain.

13. ⟨2⟩ Where in the proof that DIJKSTRA works did we use the assumption that edge weights are non-negative?

14. ⟨2⟩ Nowhere in this section have we said that the paths we find must be simple paths. Yet they always are. Explain why.
15. ⟨2⟩ Now assume that edges are allowed to have negative weights. The concept of shortest *simple* path still makes sense, but the concept of shortest path may not. Explain.
16. ⟨3⟩ In light of [13, 15], we ask: what happens when you run Algorithm 3.6 on a graph with some negative weights. Will you get the shortest simple paths? Do some examples and find out.
17. ⟨2⟩ Repeat the computation of Example 1 but in iteration 3 choose  $u_3$  to be  $v_4$ .
18. ⟨2⟩ Modify Algorithm DIJKSTRA so that it also computes  $\text{prev}(v)$  for all  $v$ . (You need only state your modifications and where they go.) Show that the algorithm's complexity does not change with this extension.
19. ⟨3⟩ Write a companion algorithm that takes as input any vertex  $v$  and the function  $\text{prev}()$  from [18] and outputs the shortest path from  $S$  to  $v$ . The path can be output as a list of vertices.
  - a) Make the vertices come out in *backwards* order (from  $v$  to  $S$ ). This is pretty easy.
  - b) Make them come out in *forwards* order (from  $S$  to  $v$ ). This is harder.
20. ⟨2⟩ Describe the situation when there is no  $U_k$ -path to a vertex  $v$ .
21. ⟨2⟩ We have defined  $d(v)$  differently on different sets. When  $v \in U_k$  we defined it to be the shortest distance from  $S$  to  $v$  in the whole graph. When  $v \notin U_k$  we defined it to be the shortest length of a  $U_k$ -path from  $S$  to  $v$ . Explain why the latter definition is sufficient for all  $v \in V$ , that is, for  $v \in U_k$  the two definitions give the same value.
22. ⟨2⟩ At one time the following problem was purportedly sprung on job applicants to Microsoft during interviews; they were given 5 minutes to solve it. We don't ask you to solve it (though we won't stop you); we merely ask you to explain how it can be turned into a graph theory problem that can be solved by the method of this section. We don't claim that the approach of this section is the fastest approach — intuition and special arguments can cut down the size of the problem to the point where you may be able to do it in your head —

but it is important to know that there is a systematic approach to such problems when shortcuts fail you.

The problem: Four people, A, B, C, D, must cross a long narrow bridge without a railing on a moonless night. They must cross with a flashlight and they have only one. At most two people can cross at once, so people must come back with the flashlight so that others can cross. Person A can cross in 1 minute, B in 2, C in 5, and D in 10. When two cross together, they cross at the slower speed. What is the minimum amount of time to get them all across the bridge?

23.  $\langle 3 \rangle$  Let  $P$  be a 15-gon, with vertices labeled cyclically 1 through 15. Let  $G$  be the graph obtained from  $P$  by including the additional edges

$$\{1, 5\}, \{4, 8\}, \{7, 9\}, \{8, 12\}, \{11, 15\}.$$

- a) What is the number of edges in the smallest cycle containing edge  $\{1, 15\}$ ? (*Hint:* Delete edge  $\{1, 15\}$  from the graph.)  
b) List the edges in a smallest cycle containing  $\{1, 15\}$ .  
c) Is the answer to b) unique? Explain how you know.

24.  $\langle 3 \rangle$  Repeat [23], but use  $\{5, 6\}$  instead of  $\{1, 15\}$  in a) and b).

25.  $\langle 2 \rangle$  The **girth** of a graph is the length of its shortest simple cycle. Using DIJKSTRA as a procedure, give an algorithm for finding the girth of a graph. In terms of the number  $n$  of vertices, analyze the complexity of your algorithm.

26.  $\langle 3 \rangle$  Suppose a graph  $G$  represents a network over rugged land. For each edge, let the weight be the elevation of that segment. (Of course, the elevation probably varies over the segment, so actually let the weight be the maximum elevation reached on that segment.) Define the *height* of a path to be the maximum of the weights (elevations) of its edges. Suppose that you desire to find the path from  $v_0$  to  $v_n$  with the minimum height. (This *minimax* objective would be useful if the edges represent sections of pipe in a water distribution network; the work needed to pump a liquid through a pipeline depends on the maximum elevation it will reach.) Modify Algorithm DIJKSTRA to accomplish this task. Verify that your algorithm works.

27.  $\langle 2 \rangle$  Suppose that you have a (di)graph with a weight  $\geq 1$  on each edge, and you want to find the paths from  $S$  to each vertex with the smallest *product* of its weights. Modify Algorithm DIJKSTRA to find these paths. Explain why your modification works. Why is it necessary that all weights be  $\geq 1$ ?

28.  $\langle 2 \rangle$  Extend Algorithm DIJKSTRA so that it finds the shortest distances between *all* pairs of vertices (in both directions for a digraph). You need only show the statements that have to be added to DIJKSTRA. (We are looking for a very simple extension of DIJKSTRA, though there may be others that are more efficient.)

29.  $\langle 2 \rangle$  Compare the complexity of the two algorithms we have discussed for the all pairs shortest path problem: the extended DIJKSTRA of [28] and the weighted Warshall algorithm of [26, Section 3.2].

30.  $\langle 2 \rangle$  One reason Algorithm 3.6 has complicated loop- and if-conditions is because the algorithm must keep checking whether  $v$  is on the end of an edge from somewhere (say,  $u$ ) before processing  $v$ . All these conditions can be simplified if every pair  $(w, x)$  is an edge.

- a) How can this be arranged? How can weights be assigned to every ordered pair of vertices so that only the “real” edges count in computing the paths?  
b) Now rewrite DIJKSTRA with simpler loop- and if-conditions.

31.  $\langle 3 \rangle$  Do the following inductive statements work? Here “work” means that, if you already knew the information in the previous case of the statements, it would be relatively easy to determine the information for the next case. Thus an efficient algorithm could be devised using iteration on this information, just as Algorithm DIJKSTRA is obtained from iteration on knowledge of the shortest path lengths for the  $k$  closest vertices to  $S$ .

- a) The shortest paths are known from  $S$  to  $v_1, v_2, \dots, v_k$  (where these are *not* necessarily the  $k$  closest vertices to  $S$  but just some previous numbering). If you think that this doesn’t work, explain why it *did* work in the Prologue.  
b) The shortest paths to the  $k$  farthest vertices (and which vertices these are) are known.

32. (3) Suppose you want to find the *longest* simple path from  $S$  to other vertices. Can you do this by replacing min by max in various places in

DIJKSTRA, and perhaps other simple changes (like changing 0 to  $\infty$ )? Justify your answer.

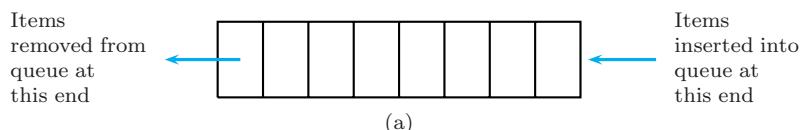
## 3.5 Breadth First Search and Depth First Search

Finding paths between vertices, as we did in Sections 3.2–3.4, is just one aspect of a more general problem: that of *searching* a graph to “visit” all its vertices or just to find a vertex with some particular property or with some specific data associated with it. That searching a graph is an important task shouldn’t be surprising. After all, it is plausible that, for many applications, we will want somehow to process the data associated with each vertex. To do so, we’ll first have to search through the graph to find the data at each vertex and usually we’ll need assurance that we’ve processed the data at each vertex once and only once.

There are two general methods for searching a graph which occur so often in applications that we devote this section to them. They are called **breadth first search (BFS)** and **depth first search (DFS)**.

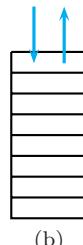
### Breadth First Search

In breadth first search we do the searching by always visiting (i.e., processing the data associated with) the vertices nearest the one we start from before visiting any others. Then we apply this idea recursively, although — as you can see in Algorithm 3.7 BREADTHFIRSTSEARCH — the most convenient implementation of this idea is iterative. This algorithm introduces the idea of a **queue** (Fig. 3.22a), which is a linear structure in which items are inserted at one end and removed from the other. A queue is often contrasted with a **stack** (Fig. 3.22b), in which items are inserted and removed at the same end. You may know from studying computer science that stacks are used in implementing recursive algorithms on computers.



(a)

Items entered and removed from top



(b)

**FIGURE 3.22**

- Queues and stacks.  
(a) A queue is a first-in-first-out (FIFO) structure.  
(b) A stack is a last-in-first-out (LIFO) structure.

### Algorithm 3.7

#### BreadthFirst-Search

**Input**  $G(V, E)$

[A connected graph]

**Output** The results of processing the data at each vertex

#### Algorithm BREADTHFIRSTSEARCH

Choose  $v \in V$

Visit  $v$  [Process its data and mark it visited]

Put  $v$  on  $Q$  [ $Q$  is a queue]

**repeat** while  $Q \neq \emptyset$  [ $\emptyset$  represents an empty queue]

$u \leftarrow \text{head}(Q)$  [Assign first item on  $Q$  to  $u$ ]

**for**  $w \in A(u)$  [ $A(u)$  is set of vertices adjacent to  $u$ ]

**if**  $w$  not visited **then**

    Visit  $w$  [Process and mark visited]

    Put  $w$  on  $Q$  [Put  $w$  on  $Q$ ]

**endif**

**endfor**

Delete  $u$  from  $Q$  [All neighbors of  $u$  visited]

**endrepeat**

If  $G$  is connected, then BREADTHFIRSTSEARCH is supposed to visit all the vertices of  $G$  once and only once. Does it? Yes, as shown by the following theorem.

**Theorem 1.** BREADTHFIRSTSEARCH visits each of the vertices of a connected graph  $G$  once and only once.

**PROOF** To prove that no vertex can be visited more than once is immediate: Since Algorithm 3.7 only visits a vertex if it has not been marked “visited” and since it marks a vertex “visited” when it is visited, no vertex can be visited more than once.

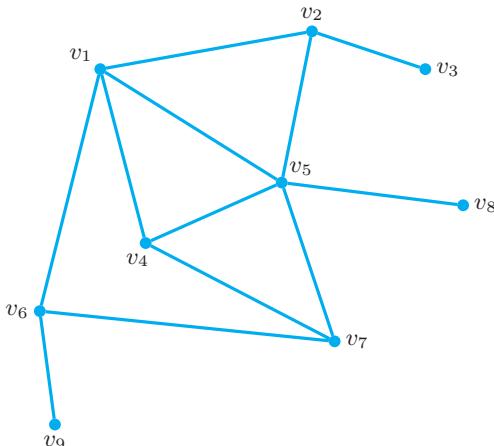
Proving that each vertex must be visited is not quite so simple. We do it by contradiction. Suppose vertex  $v'$  is not visited by BREADTHFIRSTSEARCH. Since  $G$  is connected, there is a path from the starting vertex  $v$  to  $v'$ . Let  $w'$  be the first vertex on this path that the algorithm does not visit, and let  $u'$  be the vertex just before  $w'$  on this path. So  $u'$  is visited by the algorithm. But then  $u'$  is placed on the queue  $Q$  after which, at some later time,  $u'$  is removed from  $Q$  and becomes the vertex  $u$  in the algorithm. But then the algorithm visits  $w$  for each vertex  $w$  in  $A(u)$ . Since  $w'$  is one of the vertices in  $A(u)$  (why?),  $w'$  is visited, contrary to our assumption that it isn’t. ■

Is breadth first search restricted to connected graphs? No, but for a graph which is not connected, we must have some criterion other than an empty queue

for ending the algorithm. What should this criterion be? We leave the answer to this question as well as the problem of designing a breadth first search algorithm which works for unconnected graphs to a problem [1].

BREADTHFIRSTSEARCH does not impose any rule about the order in which vertices adjacent to the current vertex are visited. If desired, such an order could be imposed by, for example, numbering the vertices and entering the vertices adjacent to  $u$  into the queue in numerical order. If we do this for the graph in Fig. 3.23 and choose vertex  $v_1$  as  $v$ , then the order in which the vertices are visited is

$$v_1, v_2, v_4, v_5, v_6, v_3, v_7, v_8, v_9.$$



**FIGURE 3.23**

A graph for use with breadth first search and depth first search examples.

You should not view BREADTHFIRSTSEARCH as a rigid model of how to do this search. Vertices could be visited as they are removed from the queue rather than just before they enter the queue. Or some vertices might not be visited at all, depending on some criterion. But BREADTHFIRSTSEARCH does provide an adaptable model for many specialized algorithms which are, in essence, based on breadth first search.

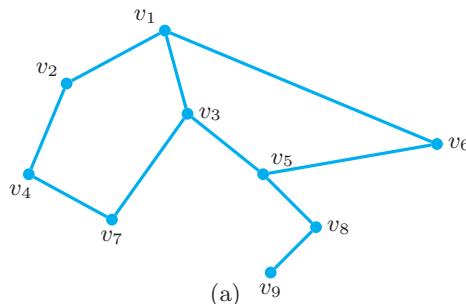
Suppose that we want to know the shortest path length from a vertex  $v$  to all the other vertices of a graph, where the length of a path is the number of edges on it. For example, in a graph representing a grid of streets in a city where all the streets are of similar length, you would wish to find the shortest way to get from one point (say, your home) to any other. This problem is just a special case of the generalization of the shortest path problem considered in [6, Section 3.4]. This simplification (i.e., with all edges having the same weight) makes the problem much easier and makes it possible to solve it using a much simpler algorithm than DIJKSTRA. Algorithm 3.8 PATHLENGTH fits quite closely the BREADTHFIRSTSEARCH paradigm. Marking each vertex with its length from  $v$  is a way of marking it as visited. The relationship between PATHLENGTH and DIJKSTRA is explored in a problem [5].

### EXAMPLE 1

Applying PATHLENGTH to the graph in Fig. 3.24a, we obtain the path lengths from vertex  $v_1$  which are shown in Fig. 3.24b. ■

**Algorithm 3.8****PathLength****Input**  $G(V, E)$  [A connected graph] $v$ [A particular vertex in  $V$ ]**Output** Each vertex marked with its length from  $v$ **Algorithm** PATHLENGTHMark  $v$  with length 0Put  $v$  on  $Q$ [ $Q$  is a queue]**repeat** while  $Q \neq \emptyset$      $u \leftarrow \text{head}(Q)$      $l \leftarrow \text{length marked on } u$ [Length of path from  $v$  to  $u$ ]    **for**  $w \in A(u)$ [ $A(u)$  is set of vertices adjacent to  $u$ ]        **if**  $w$  not marked with a length **then**            Mark  $w$  with  $l + 1$ ; put  $w$  on  $Q$         **endif**    **endfor**Delete  $u$  from  $Q$ **endrepeat**

|       |   |
|-------|---|
| $v_1$ | 0 |
| $v_2$ | 1 |
| $v_3$ | 1 |
| $v_4$ | 2 |
| $v_5$ | 2 |
| $v_6$ | 1 |
| $v_7$ | 2 |
| $v_8$ | 3 |
| $v_9$ | 4 |



(b)

**FIGURE 3.24**

An application of Algorithm PATHLENGTH:  
 (a) graph and  
 (b) path lengths.

PATHLENGTH results in a marking which is the *minimum* distance from  $v$  to each vertex. We leave the proof to [4]. (Remember that there can be more than one path from  $v$  to another vertex and that these paths may have different lengths.)

If the edges used to reach unvisited vertices are also marked, breadth first search (and depth first search) result in the construction of a *spanning tree* for any connected graph  $G$ , that is, a tree (in the undirected, mathematician's sense) which is a subgraph of  $G$  and which contains all the vertices of  $G$ . We shall further consider spanning trees and their relation to BFS and DFS in Section 3.7.

**Depth First Search**

Here the idea is to start at  $v$  and go as far as you can, following your nose, until you can't go any farther (i.e., until there are no unvisited vertices adjacent to the

vertex at which you have arrived). Then back up along the path just traversed until you find a vertex with an unvisited neighbor and start down a new path, again following your nose for as long as you can. Implementing this idea could be done using a sequence of applications of Procedure Pathgrow in Section 3.3. But it's rather messy to keep track of where you have been on each path as you grow it, so that you can back up when you need to (in particular, see [26, Supplement]). The way to avoid this mess is just to let our algorithmic language take care of the bookkeeping by making the algorithm recursive.

### Algorithm 3.9

#### DepthFirst-Search

|                                                     |                                                   |                                                 |
|-----------------------------------------------------|---------------------------------------------------|-------------------------------------------------|
| <b>Input</b>                                        | $G(V, E)$                                         | [A connected graph with all vertices unvisited] |
| <b>Output</b>                                       | The results of processing the data at each vertex |                                                 |
| <b>Algorithm</b> DEPTHFIRSTSEARCH                   |                                                   |                                                 |
| <b>procedure</b> Depth( $u$ )                       |                                                   |                                                 |
|                                                     |                                                   | [ $u$ is a vertex]                              |
| Visit $u$                                           | [Process and mark as visited]                     |                                                 |
| <b>for</b> $w$ in $A(u)$                            | [ $A(u)$ is set of vertices adjacent to $u$ ]     |                                                 |
| <b>if</b> $w$ is unvisited <b>then</b> Depth( $w$ ) | [Recursive call]                                  |                                                 |
| <b>endfor</b>                                       |                                                   |                                                 |
| <b>endpro</b>                                       |                                                   |                                                 |
| Choose an arbitrary $v \in V$                       | [Main algorithm]                                  |                                                 |
| Depth( $v$ )                                        |                                                   |                                                 |

Algorithm 3.9 DEPTHFIRSTSEARCH really just grows a path by, in effect, calling the path grower recursively each time you get to a new vertex. When you get to a point from which you can't continue — because there are no unvisited vertices adjacent to the vertex at which you've arrived — the recursion automatically goes back to a level at which there is a vertex with an unvisited neighbor.

If we apply DEPTHFIRSTSEARCH to Fig. 3.23 starting at vertex  $v_1$  and if, in the for-loop, we choose the vertices to assign to  $w$  in numerical order of their subscripts, the order in which the vertices are visited is

$$v_1, v_2, v_3, v_5, v_4, v_7, v_6, v_9, v_8.$$

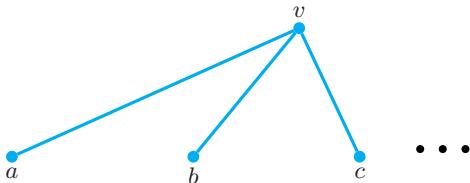
But does DEPTHFIRSTSEARCH surely visit all the vertices of a connected graph? Proving that it does is a nice example of how induction can be used to prove the correctness of recursive algorithms.

---

**Theorem 2.** If  $G$  is connected, then DEPTHFIRSTSEARCH visits all the vertices of  $G$  once and only once and then terminates.

---

**PROOF** We prove that DEPTHFIRSTSEARCH visits each vertex once and only once and then terminates by strong induction on  $n$ , the number of vertices in  $G$ . When  $n = 1$ , the basis case, the result is immediate because the (solitary) vertex  $v$  gets visited when Procedure Depth is called from the main algorithm, and the algorithm terminates after this visit since  $A(v)$  is empty. Now assume that the theorem is true when  $G$  has fewer than  $n$  vertices and consider a connected graph  $G$  with  $n$  vertices. As in the algorithm, choose a vertex  $v$  at which to start and call Depth. The situation is as shown in Fig. 3.25, with  $a, b, c, \dots$  the vertices adjacent to  $v$ . Suppose that, without any loss of generality,  $a$  is chosen as the first vertex adjacent to  $v$ . Let  $G'$  be the graph consisting of  $G$  except for  $v$  and the edges incident on  $v$ . When Depth is called with argument  $a$ , the situation is the same as if the input had originally been the (necessarily connected) component  $C$  of  $G'$  containing  $a$  with  $a$  the vertex chosen in the main algorithm (why?).



**FIGURE 3.25**

Induction proof of Theorem 2.

Since this component has at least one less vertex than  $G$ , our induction hypothesis says that, when Depth is called with argument  $a$ , this call visits all the vertices in  $C$  once and terminates. After doing this, control goes back to the for-statement within Depth( $v$ ), and the algorithm looks for any other vertices adjacent to  $v$  which are still unvisited. (There may be none because  $G'$  *might* be connected so that  $b, c$ , etc., would have been visited as a result of the call of Depth with argument  $a$ .) If there is an unvisited vertex adjacent to  $v$ , then Depth is called with this unvisited vertex as argument, and again by induction, all vertices in another component of  $G'$  are visited once. As there can be no more components of  $G'$  than there are vertices of  $G$  adjacent to  $v$  (why?), successive calls of Depth will visit all the vertices of  $G$ . After the last component is visited, control returns to the for-loop of Depth( $v$ ). This loop may cycle further (there may still be untested  $w \in A(u)$ ) but no action will be taken (these  $w$  have been visited) and then the algorithm terminates. ■

Depth first search embodies the important algorithmic idea of **backtracking**. The essence of any backtracking algorithm is that you search for a solution in a particular direction until either you find it or you reach some kind of dead end. If you reach a dead end, then you back up (“backtrack”) to the closest previous point at which you had some decision to make about the direction in which to go. At this point, if there is a direction to go in and you haven’t tried it before, do so; if not, backtrack further until you find such a direction. Either you will find a solution, if there is one, or you will end up back at the beginning with no untried directions. The crucial attributes of this idea are that you never try a direction more than once and you never skip over any possible solution.

DEPTHFIRSTSEARCH is an instance of a backtracking algorithm because, starting from any vertex, we go down some path until we can go no farther (i.e., there is no unvisited vertex to go to). Then we back up on the path just traversed until a visited vertex is reached which has unvisited neighbors. We then apply this same idea repeatedly, each time going as deeply as possible and then backtracking until a new path is found on which there are unvisited vertices. Backtracking has many other applications, two of which are treated in [9–10].

Theorems 1 and 2 provide a method of determining whether a given graph is connected. Run either algorithm, and at the end check to see if every vertex has been visited [7].

This completes our introduction to BFS and DFS. We will discuss various other applications of these algorithms in the remaining two sections of this chapter.

## Problems: Section 3.5

---

1.  $\langle 2 \rangle$  Show how to modify BREADTHFIRSTSEARCH for graphs which are not connected so that it searches all components of a graph.

2.  $\langle 2 \rangle$  Let  $G(V, E)$  be defined by  $V = \{1, 2, \dots, 15\}$  and let

$$E = \{\{u, v\} \mid u \equiv v \pmod{3} \text{ or } u \equiv v \pmod{5}\}.$$

(“ $\equiv$  is defined in [35, Section 0.2].) Draw a BFS tree for  $G$  starting at vertex 1. Break ties by choosing the vertex with the lowest index. Show your work by drawing a picture of the graph with the final tree darkened and the tree edges labeled by their order of inclusion.

3.  $\langle 1 \rangle$  Apply Algorithm PATHLENGTH to the graph obtained in [2], starting at vertex 1.

4.  $\langle 2 \rangle$  Prove that Algorithm PATHLENGTH marks each vertex with the minimum distance from the starting vertex.

5.  $\langle 2 \rangle$  We said that PATHLENGTH was just another way to solve the shortest path problem when all the weights are 1. But how different is what PATHLENGTH actually does from what DIJKSTRA does when all the weights are 1? Let’s see.

- a) Apply PATHLENGTH to the graph of Fig. 3.23 to find the path length from  $v_8$  to each of the other vertices. At each stage test the vertices in  $A(u)$  in subscript order. Keep a table like that on p. 270 with a new line for each vertex removed from the queue.

- b) Now apply DIJKSTRA to the same graph with all the weights 1 to find the shortest path from  $v_8$  to all the other vertices. (Use the modification of DIJKSTRA developed in [6, Section 3.4].) Again, when you have a choice of vertices, always choose the one with lowest subscript and keep a table like that on p. 270.

- c) Comparing the tables found in a) and b), what do you conclude?

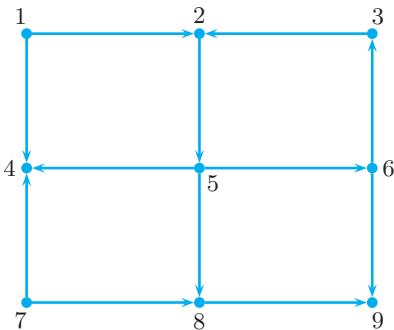
6.  $\langle 2 \rangle$  Consider Algorithm PATHLENGTH again. “Mark  $v$  with length  $l$ ” is not standard mathematical language. One way to “fix” this is with function notation by defining a length function  $l(v)$ . To assure that  $l(v)$  is always defined for each  $v$ , initialize  $l(v)$  with some obviously wrong value, like  $l(v) = \infty$  or  $l(v) = -1$ . Then  $l(v)$  can always be looked up [the algorithm won’t bomb if  $v$  hasn’t been visited yet and you ask about  $l(v)$ ] and you can tell when  $v$  has been visited because  $l(v)$  will have a plausible value.

With these ideas, rewrite PATHLENGTH avoiding all words like “mark”, “visit”, and “length”.

7.  $\langle 2 \rangle$
- a) Write a DFS algorithm to test whether a given graph is connected.
  - b) Repeat for BFS.
8.  $\langle 2 \rangle$  The BFS and DFS algorithms can be modified for directed graphs. At each vertex  $v$ , you may only go to vertices adjacent from it. That is, you

may only travel along edges in the direction of the arrows.

- a) Apply BFS to the following graph, starting at  $v_1$  and always choosing the lowest-indexed vertex when there is a choice.



- b) Same as a) but use DFS.
  - c) In a directed graph, what vertices will BFS or DFS find? (Just state the answer.)

{3} The Eight Queens Problem. The object of this problem is to place eight queens on a chessboard so that no two queens are attacking each other (i.e., no two queens are in the same row, the same column, or on the same diagonal).

  - a) Describe a graph in which each vertex is one square of a chessboard and for which each solution to the eight queens problem is a set of mutually nonadjacent vertices. Don't try to draw the graph!
  - b) Actually, graphs are not a very useful tool in solving the eight queens problem, but backtracking is. Describe a backtracking algorithm

for this problem which begins by putting a queen in the first column of the chessboard, then another which doesn't attack the first in the second column, a third which doesn't attack either of the first two in the third column, etc. When you can't find a place to put a queen in column  $i$ , backtrack to column  $i - 1$  and try again. In each column try the first row first, then the second row, etc., so that, when you have to backtrack, you can then try only rows which you haven't tried before.

- c) Use your algorithm from b) to find one solution to the eight queens problem. It's a little tedious but shouldn't take you more than 10 minutes by hand. It's best to use an actual chessboard.
  - (3) Consider the problem of forming sequences of the digits 1, 2, and 3 with the property that nowhere in the sequence are there two *adjacent* subsequences which are the same. Thus two adjacent digits can never be the same and, for example, 121323213231 does not satisfy the criterion because of the two adjacent subsequences 21323.
  - a) Use backtracking to devise a method for generating such sequences of any length using the symbols 1, 2, and 3. (*Hint:* Each time you add a digit, how far back in the sequence do you have to look to make sure that there are no two identical adjacent subsequences?)
  - b) Display your method using our algorithmic language.
  - c) Use your algorithm to generate a sequence of length 10 having the desired property.

## 3.6 Coloring Problems

A **coloring** of a graph, sometimes called a **proper coloring**, is any assignment of colors to vertices such that no two adjacent vertices have the same color. The idea of coloring graphs is an important one in graph theory because it has many applications. In this section we'll mention some of these applications, discuss some basic algorithms and theorems, and indicate how graph coloring is related to the problem of coloring a map.

A general class of problems to which graph coloring can be applied is scheduling. Suppose that you want to schedule the classes for a university. Because a professor

normally teaches more than one course, you must make sure that each professor's courses are taught at a different time. One way to describe this problem is by a graph in which each vertex represents a class to be scheduled and an edge joining two vertices represents classes taught by the same instructor. If each color is then associated with a particular class time (red, 8–8:50 MWF; blue, 9–9:50 MWF; etc.), a coloring of the graph provides a schedule in which no professor has two classes at the same time. You can extend this idea to take into account other constraints on a class schedule. Suppose that Professor Sleepplate is not a “morning person” and thus should never be scheduled for an 8 A.M. class. Then add to the graph vertices corresponding to possible class times (8 A.M. green; 9 A.M. yellow; etc.) and an edge from the 8 A.M. vertex to each class vertex taught by Professor Sleepplate. Then a coloring of the graph gives a schedule in which no class taught by Professor Sleepplate is scheduled at 8 A.M.

Various aspects of computer scheduling may also be attacked using the graph coloring model. Suppose that we have a large database (e.g., student records at a university), which is accessed by many application programs (e.g., to update addresses, to send out tuition bills, to insert course grades, etc.). Suppose also that certain application programs should never access the database at the same time because a change made by one program might cause the other program to make an error. For example, the program which calculates a student's grade point average should never be accessing the database at the same time as the program which inserts course grades in the database. We can let the vertices in a graph represent the application programs, and an edge would join any two vertices corresponding to programs which should not be active at the same time. Then a coloring of the graph provides a schedule for these application programs in which no two programs which interfere with each other can be active at the same time. Graph coloring can also be applied to other aspects of job scheduling on a computer, as well as to the scheduling of computer resources, such as the assigning of registers by a compiler to hold intermediate results of a computation.

We begin with the essential definition needed for a discussion of graph coloring problems.

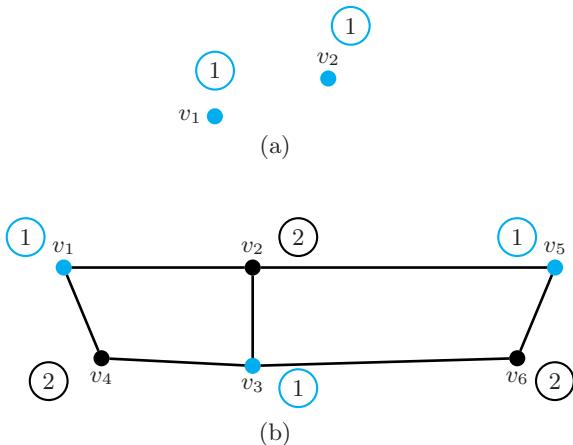
---

**Definition 1.** The **chromatic number**  $\chi(G)$  of a graph  $G$  is the minimum number of colors needed so that no two adjacent vertices have the same color. A graph  $G$  is said to be  **$k$ -colorable** for any  $k \geq \chi(G)$ .

---

Figure 3.26a–c displays a minimal coloring — and thus the chromatic number — of some graphs. For simplicity in what follows, we'll refer to the colors of vertices not as “red” or “green” or “blue” but rather as “1” or “2” or “3”.

Now suppose a graph  $G(V, E)$  is 2-colorable so that each edge joins a vertex with color 1 to one of color 2. If all the vertices with color 1 are put in a subset  $V_1$  of  $V$  and all those with color 2 are put in a subset  $V_2$  (so that  $V_1$  and  $V_2$  are disjoint), then each edge in  $E$  joins a vertex in  $V_1$  to one in  $V_2$ . A graph that is 2-colorable



**FIGURE 3.26**

Examples of chromatic numbers (colors shown in circles):  
 (a)  $\chi(G) = 1$ ;  
 (b)  $\chi(G) = 2$ ;  
 (c)  $\chi(G) = 4$ .

is called **bipartite**. Analogously,  $k$ -colorable graphs are sometimes referred to as  $k$ -partite graphs (see [14]).

An important property of bipartite graphs is expressed by the following theorem.

**Theorem 1.** A graph is 2-colorable (i.e., bipartite) if and only if it has no cycles of odd length.

Theorem 1 is a **structure theorem**; that is, it characterizes 2-colorable graphs in terms of some structure that must or must not (in this case must not) be in the graph. However, the theorem does not say how to find out if this structure is there. Traditionally, the proofs of such theorems are nonconstructive existence proofs that would also not say how to find the structure. Thus neither the theorem statement nor the proof gives you a procedure for determining if a graph is 2-colorable.

We have already expressed our preference for constructive proofs of existence theorems. Now we go a step further: We substitute a different theorem which, because of the way it is stated, *requires* a constructive proof.

---

**Theorem 2.** A connected graph is 2-colorable (i.e., bipartite) if and only if Algorithm 3.10 BIPARTITE produces a coloring (which will always be a proper 2-coloring) instead of terminating early.

---

Proving Theorem 2 will, in effect, also prove Theorem 1. Read on!

Theorem 2 is a **constructive theorem** in that its very statement says that in order to determine if a graph is 2-colorable, you may use an algorithm to produce a 2-coloring, if there is one..

## Algorithm 3.10 Bipartite

**Input**  $G(V, E)$  [A connected graph]  
**Output** A 2-coloring of  $G$ , or a message that  $G$  is not bipartite

**Algorithm** BIPARTITE

Choose an arbitrary  $v \in V$  and color it 1

Put  $v$  on  $Q$  [ $Q$  is a queue]

**repeat while**  $Q \neq \emptyset$

$u \leftarrow \text{head}(Q)$

**for**  $w \in A(u)$  [Vertices adjacent to  $u$ ]

**if**  $w$  has the same color as  $u$

**then print** “Graph not bipartite”; **exit** (\*)

**else** if  $w$  uncolored, color it opposite to  $u$

(i.e., 1 if  $u$  is colored 2 and 2 if  $u$  is colored 1) and, in any case,

put  $w$  on  $Q$

**endif**

**endfor**

Delete  $u$  from  $Q$

**endrepeat**

---

By the way, the only reason that Theorem 2 is restricted to connected graphs is because we wrote Algorithm BIPARTITE for the connected case only, to keep it simpler. If a graph is not connected, you just have to run BIPARTITE on each component and see if BIPARTITE produces a 2-coloring each time.

**PROOF OF THEOREM 2** If line (\*) of the algorithm is never activated (i.e.,  $w$  is never the same color as  $u$ ), then we claim every vertex is colored and the coloring is proper. Every vertex is colored because, without an early stop due to (\*), BIPARTITE is a variant of BFS in which the coloring is the “visit”. As in BREADTH-FIRSTSEARCH, we put the initial vertex  $v$  on a queue  $Q$ , then visit all vertices

adjacent to  $v$ , then visit all vertices adjacent to those adjacent to  $v$ , etc., unless (\*) causes the algorithm to terminate. So, since we are considering the case that (\*) is not activated and since we proved in Section 3.5 that BFS visits every vertex of a connected graph, we conclude that BIPARTITE colors every vertex. The coloring is proper because, when BIPARTITE does not exit at (\*), it colors each still uncolored vertex adjacent to any vertex  $v$  the opposite to the color of  $v$ .

If line (\*) is activated (because  $u$  and  $w$  have the same color), we claim that  $G$  has an odd cycle, and therefore can't be 2-colored. Since both  $u$  and  $w$  are colored, each must have been on a path of colored vertices starting at  $v$  with each vertex on the path to  $u$  being adjacent to a previously colored vertex and similarly for  $w$ . If we follow these paths from  $u$  and  $w$  back toward  $v$ , they first coincide at some vertex  $v'$  (which may be  $v$  itself). Thus, the path  $v' \dots u, w \dots v'$  is a cycle  $C$  and on this cycle the colors alternate except at the edge  $\{u, w\}$ . Therefore, the paths  $v' \dots u$  and  $v' \dots w$  have the same parity (i.e., both have an even number of edges or both have an odd number). Since  $C$  consists of these two paths plus the edge  $\{u, w\}$ ,  $C$  is an odd cycle (i.e., it has an odd number of edges). Lastly, an odd cycle cannot be 2-colored because on a 2-coloring of any cycle, the colors must alternate and thus must come in pairs. Therefore any graph containing an odd cycle cannot be 2-colored.

Finally, for any connected  $G$ , either line (\*) is activated or not. In the first case we have shown that BIPARTITE correctly states that there is no 2-coloring. In the second case it correctly produces one. Thus, it produces one if and only if one exists. ■

**PROOF OF THEOREM 1** In the proof of Theorem 2, we saw that every connected  $G$  either has a 2-coloring (when line (\*) is not activated), or has an odd cycle, but not both. Therefore, a connected graph is 2-colorable if and only if it has no odd cycles. The same is true for a disconnected graph because it is true component by component. ■

Now let's consider computing the chromatic number of a graph. First, we state an almost obvious fact: The chromatic number of a complete graph of  $n$  vertices is  $n$ . This is so because all pairs of vertices of a complete graph are adjacent and so must be colored differently. It is plausible, therefore, that the chromatic number is related to how complete a graph is, whatever that may mean. Well, let's make a definition that attempts to give meaning to this nebulous statement. A **clique** is a subgraph of a graph  $G$  which is itself complete. That is, each vertex of this subgraph is adjacent to every other vertex of the subgraph. The **clique number**  $c(G)$  of a graph  $G$  is the maximum number of vertices in a clique of  $G$ .

What else might plausibly play a role in determining the chromatic number of a graph? Well, if vertices in the graph generally have low degree, then you might expect the chromatic number to be low since a vertex of low degree requires at most a small number of colors for the vertices it is adjacent to. These musings motivate the following theorem that gives us some bounds on the chromatic number.

---

**Theorem 3.** Let  $\chi(G)$  be the chromatic number of a graph  $G(V, E)$ , let  $c(G)$  be the clique number of the graph, and let  $\Delta(G)$  be the maximum degree of any vertex of  $G$ . Then

$$c(G) \leq \chi(G) \leq \Delta(G) + 1.$$


---

**PROOF** The first inequality follows immediately from the discussion above. As there is an edge between any two vertices of a complete graph, each vertex of the clique must have a different color. Therefore

$$c(G) \leq \chi(G).$$

Now let's color  $G$  by choosing the vertices one at a time and coloring each vertex  $v$  with the smallest (i.e., lowest number) color not used for any already colored vertex adjacent to it. Since no vertex has more than  $\Delta(G)$  vertices adjacent to it, it will never be necessary to give  $v$  a color greater than  $\Delta(G) + 1$ . Thus

$$\chi(G) \leq \Delta(G) + 1. \blacksquare$$

Theorem 3 might be termed “half-useful”. It’s easy to determine the maximum degree of the vertices of  $G$ , so an algorithm to compute the chromatic number of a graph can stop whenever it determines that the chromatic number is at least  $\Delta(G) + 1$ . On the other hand, the lower bound  $c(G)$  isn’t very useful because calculating the clique number is generally very difficult.

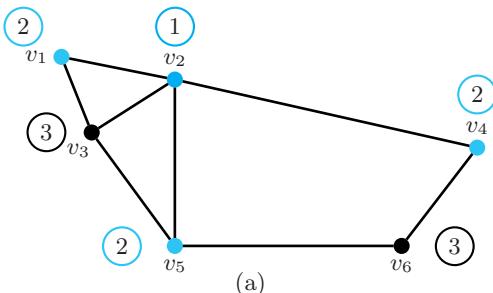
A fairly obvious way to go about computing an upper bound on  $\chi(G)$  is just to use the approach in the proof of Theorem 3, by choosing one vertex after another and giving each the lowest color possible. This approach is illustrated by Algorithm  $K$ -COLORABLE. As Fig. 3.27 illustrates, the output  $k$  of  $K$ -COLORABLE is not generally the chromatic number. As Theorem 3 would lead us to expect, at least  $K$ -COLORABLE outputs a  $k \leq \Delta(G) + 1$ . We leave a proof of this, as well as a proof that  $K$ -COLORABLE gives a proper coloring, to problem [28].

$K$ -COLORABLE is an example of a greedy algorithm that does not generally achieve the best possible result. Indeed, the computation of the chromatic number for general graphs is a very difficult problem, and we’ll not pursue it further here.

Finally in this section, suppose that you want to print a map of, say, the 48 contiguous states of the United States, using the minimum number of colors such that no two states with a common boundary have the same color. (It’s OK if two states which meet at a single point have the same color.) This problem can be directly related to the graph coloring problem if, for any map, you associate a vertex with each region of the map and have an edge between two vertices whenever the corresponding regions have a common boundary, as illustrated in Fig. 3.28. (This is essentially what we did in Section 3.3 when we derived a graph representation of the Königsberg bridge problem.) What is the minimum number of colors needed to print any map? The answer is given by the famous Four Color Theorem whose

**Algorithm 3.11*****K*-Colorable****Input**  $G(V, E)$ 

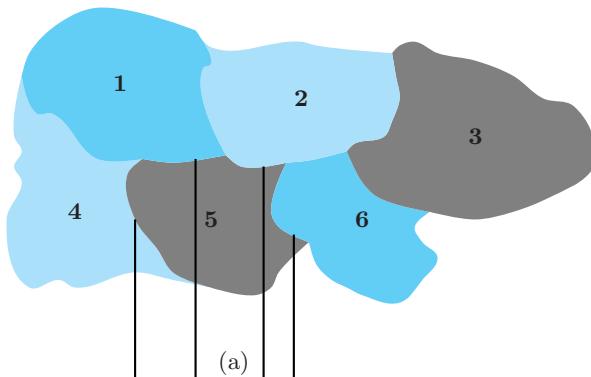
[A connected graph]

**Output** An integer  $k$  and a  $k$ -coloring of  $G$ **Algorithm *K*-COLORABLE**Choose an arbitrary vertex  $v$  in  $V$  and color it 1 $V' \leftarrow V - \{v\}$ [Initialize  $V'$ ]**repeat while**  $V' \neq \emptyset$      $u \leftarrow$  an arbitrary vertex in  $V'$     Color  $u$  the minimum color not yet assigned to a  
    vertex adjacent to  $u$      $V' \leftarrow V' - \{u\}$ **endrepeat** $k \leftarrow$  maximum color assigned to any vertex

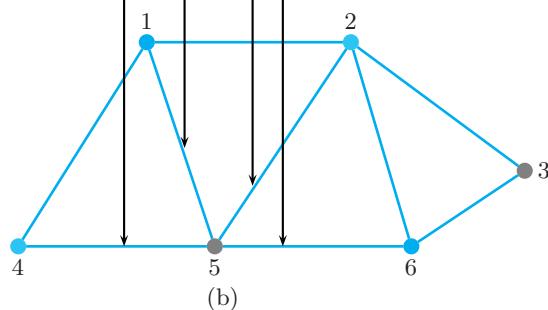
- i) Color  $v_6$  1
- ii) Color  $v_1$  1
- iii) Color  $v_3$  2 (since it is adjacent to  $v_1$ , which is colored 1)
- iv) Color  $v_5$  3 (since it is adjacent to vertices colored 1 and 2)
- v) Color  $v_4$  2 (since it is adjacent to a vertex colored 1)
- vi) Color  $v_2$  4 (since it is adjacent to vertices colored 1, 2, and 3)

(b)

Calculation of  $k$ -colorability. (a) A graph with  $\chi(G) = 3$ . A 3-coloring is denoted by the numbers in circles. (b) Application of Algorithm  $K$ -COLORABLE to the same graph. All we know from using the algorithm is that the graph is 4-colorable. For certain other orders of choosing the vertices, the algorithm would have 3-colored the graph [22].



(a)



**FIGURE 3.28**

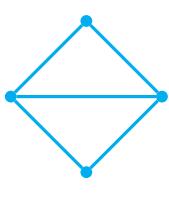
Maps and graphs:  
 (a) a map of six regions;  
 (b) a graph corresponding to this map. The arrows indicate how the boundaries of region 5 become edges from vertex 5 to other vertices.

proof would be even longer than this chapter. We presume that you can guess what the answer is.

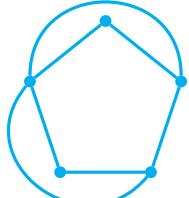
## Problems: Section 3.6

---

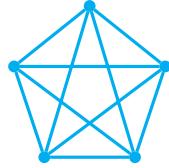
1. ⟨1⟩ What graphs are 1-colorable?
2. ⟨2⟩ What are the chromatic numbers of the following graphs?



(a)

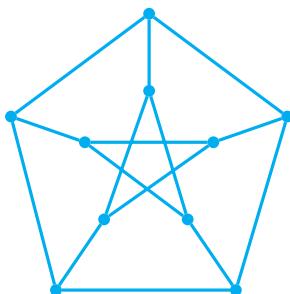


(b)



(c)

3. ⟨2⟩ Determine the chromatic number of the following graph, called the Petersen Graph.



4. ⟨2⟩ With as few colors as possible, color the vertices of a cube so that adjacent vertices have different colors. How do you know that you have the minimum number?

5. ⟨2⟩ With as few colors as possible, color the *faces* of a cube so that faces with an edge in common have different colors. How do you know that you have the minimum number? (Solve by turning this into a vertex coloring problem on an appropriate graph.)
6. ⟨2⟩ Suppose that we have six chemicals and, for each  $i$  from 1 to 4, chemical  $i$  cannot safely be stored in the same room as chemical  $i + 1$  or  $i + 2$ . Determine how many rooms are needed for safe storage by turning this into a graph coloring problem.
7. ⟨2⟩ Use the same conditions as in [6], but now the following additional pairs of chemicals cannot be stored together:  $\{5, 6\}$ ,  $\{5, 1\}$ ,  $\{6, 1\}$ , and  $\{6, 2\}$ .
8. ⟨2⟩ How many colors does it take to color any tree? Why?
9. ⟨2⟩ Suppose that you are given a tree on which two nonadjacent vertices are already colored red. What is the maximum number of colors needed to complete a coloring of the tree? Give a proof by algorithm: Exhibit an algorithm which you can show always uses at most the number of colors you have claimed.
10. ⟨3⟩ Suppose that you are given a tree on which  $k$  mutually nonadjacent vertices have already been colored red. Determine the maximum number of colors needed to complete the coloring. Give a proof by algorithm.
11. ⟨2⟩ A college registrar must try to arrange final exams so that no student is scheduled to take two exams at the same time. Figure out how to turn this into a coloring problem. What do the vertices represent? When is there an edge between two vertices? What do the colors represent? Why might you wish to find a coloring with a minimum number of colors?
12. ⟨2⟩ Use the same conditions as in [11], but now the registrar is also mindful of professors and doesn't want any professor to have to give two exams at the same time. How must the graph be changed so that the coloring still provides a solution?
13. ⟨2⟩ Use the same conditions as in [12], except now there are further constraints. The Physics 1 exam cannot be scheduled for Monday morning, and the PoliSci 3 exam cannot be scheduled for Tuesday evening. Explain how to augment the graph to take care of this.
14. ⟨2⟩ A graph is  **$k$ -partite** if the vertex set  $V$  can be partitioned into  $V_1, V_2, \dots, V_k$  so that no edge has its two ends in the same set. Explain why a graph is  $k$ -colorable if and only if it is  $k$ -partite.
15. ⟨3⟩ Determining whether a graph  $G$  has  $\chi(G) = 3$  is not so tough. First, determine whether  $\chi(G) < 3$ , using Algorithm BIPARTITE. If not, try in turn each three-way partition of the vertices. If you find one for which there are no edges between any pair of vertices in the same set, then  $G$  is 3-partite, thus 3-colorable, and hence  $\chi(G) = 3$  (since we already verified that  $\chi(G) \not< 3$ ). If no three-way partition works, then  $G$  is not 3-partite and hence not 3-colorable. Critique this claim.
16. ⟨3⟩ The **generalized octahedral graphs**  $\{O_n\}$  are defined recursively as follows:
- $O_1$  consists of 2 isolated vertices.
  - $O_{n+1}$  is obtained from  $O_n$  by creating 2 new vertices and connecting each one to all the vertices in  $O_n$ .
- Draw  $O_1$ ,  $O_2$ , and  $O_3$ .
  - Why are these called octahedral graphs? Hint: What is an octahedron?
  - Determine  $\chi(O_1)$ ,  $\chi(O_2)$ , and  $\chi(O_3)$ .
  - Conjecture and prove a formula for  $\chi(O_n)$ .
17. ⟨2⟩ Consider the undirected graph given by the following adjacency matrix:
- $$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- 18.** *(3)* Construct a graph  $G$  as follows. Put vertices  $v_1$  through  $v_9$  evenly spaced along a circle, in increasing order. Now draw an edge from each vertex to its four closest neighbors (two on each side). For instance,  $v_1$  is adjacent to  $v_8, v_9, v_2$ , and  $v_3$ .
- What are the bounds of  $\chi(G)$  according to Theorem 3?
  - What actually is  $\chi(G)$ ? None of our theorems will answer this for you, so you must give a special justification for your answer, based on the specific nature of this graph.
- 19.** *(3)* Repeat [18], except now the circular graph has 10 vertices.
- 20.** *(3)* Repeat [18], except now the circular graph has 11 vertices.
- 21.** *(2)* Algorithm  $K$ -COLORABLE leaves the order in which vertices are colored up to you. What sort of vertices do you think it would be best to color first: high degree, low degree, some other condition? Give your reasons.
- 22.** *(2)* Find an order for choosing the vertices of the graph in Fig. 3.27 so that Algorithm  $K$ -COLORABLE would give a 3-coloring of the graph.
- 23.** *(3)* A set of vertices is **independent** if no two are adjacent. The **independence number** of a graph is the size of the largest set of independent vertices. The independence number is often denoted by  $\alpha(G)$ , or just  $\alpha$ . Prove: For any graph,  $\chi(G) \geq \lceil |V|/\alpha \rceil$ .
- 24.** *(3)* In light of Theorem 3,
  - Find a graph for which  $c(G) = \Delta(G) + 1$  (and thus  $\chi(G)$  is completely determined by the theorem).
  - Find a graph for which  $c(G)$  and  $\Delta(G) + 1$  differ by 1, and for which  $\chi(G) = c(G)$ .
  - Find a graph for which  $c(G)$  and  $\Delta(G) + 1$  differ by 1, and for which  $\chi(G) = \Delta(G) + 1$ .
- 25.** *(3)* Here is a slight improvement in the upper bound of Theorem 3: If  $G$  is connected and has at least one vertex  $v$  with degree  $< \Delta$ , then  $\chi(G) \leq \Delta$ .
  - Prove this. Hint: Algorithm  $K$ -COLORABLE can help if the order of vertices is chosen wisely.
  - Prove that if the word “connected” is deleted, the claim is not true.
- 26.** *(4)* Unfortunately Theorem 3 is often of little help, because  $c(G)$  and  $\Delta(G) + 1$  can be far apart, as we now show. Define  $G_n$  as follows. Start with a 5-cycle  $C_5$ . Replace each vertex with a copy of the complete graph on  $n$  vertices. We will call these the supernodes of  $G_n$ . Two vertices in  $G_n$  are adjacent iff either i) they are in the same supernode, or ii) they are in supernodes that were adjacent in the original  $C_5$ .
- Show that  $c(G_n) = 2n$ .
  - Show that  $\chi(G_n) \geq \lceil 5n/2 \rceil$ . Use [23]. Thus  $c(G_n)$  and  $\chi(G_n)$  differ by at least  $n/2$ .
  - Find  $\chi(G_3)$ .
- In fact, for any  $n$  there exists  $G$  (not shown here) such that  $\chi(G) = n$  yet  $c(G) = 2$  (not even any triangles in  $G$ )!

### Algorithm 3.12 COLOR-BY-VERTICES

```

Input  $G(V, E)$                                  $[V = \{v_1, v_2, \dots, v_n\}]$ 
   $[n = |V|]$ 
Output A coloring of  $G$ 

Algorithm COLOR-BY-VERTICES
  for  $i = 1$  to  $n$ 
    Color  $v_i$  the lowest numbered color not
      already used for adjacent vertices
  endfor

```

### Algorithm 3.13 ONE-COLOR-AT-A-TIME

```

Input Same
Output A coloring of  $G$ 

Algorithm ONE-COLOR-AT-A-TIME
   $c \leftarrow 1$                                       $[\text{Initialize to first color}]$ 
   $U \leftarrow V$ 
  repeat until  $U = \emptyset$ 
     $W \leftarrow \emptyset$ 
    for  $i = 1$  to  $n$ 
      if  $v_i \in U$  and is not adjacent
        to any vertex in  $W$  then
          Color  $v_i$  with color  $c$ 
           $W \leftarrow W \cup \{v_i\}$ 
           $U \leftarrow U - \{v_i\}$ 
    endfor
     $c \leftarrow c + 1$ 
  endrepeat

```

27. (3) Consider Algorithms 3.12 and 3.13 for coloring a graph. The first is a variant of Algorithm *K*-COLORABLE. In both, colors are described by numbers, not names. Thus the “lowest numbered” color is color 1.

- a) Apply both algorithms to the graph obtained in [18].
- b) Apply both algorithms to the graph obtained in [19].
- c) What do you observe about the relative behavior of the two algorithms?
- d) Prove your observations in c).

28. (3)

- a) Prove that the output of K-COLORABLE has a value  $\leq \Delta(G) + 1$ .
- b) Prove that K-COLORABLE always produces a proper coloring.

This problem shows that the upper bound in Theorem 3 can be proved using K-COLORABLE.

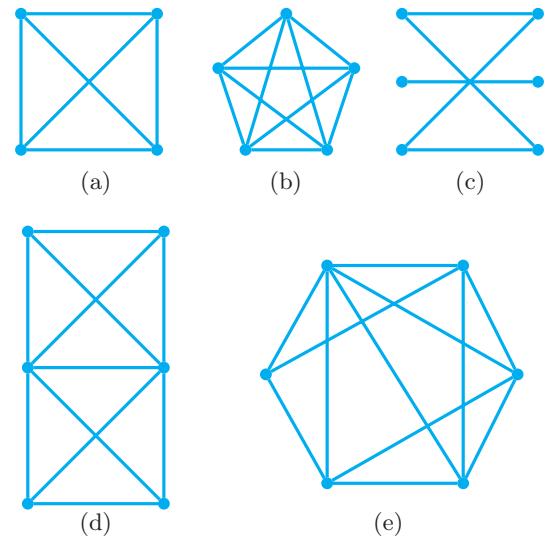
29. (3) *Edge Coloring.* Now let’s color edges instead of vertices. An edge coloring is *proper* if no two edges incident on the same vertex have the same color. The **edge chromatic number** of  $G$ , written  $\chi'(G)$ , is the minimum number of colors needed to edge-color  $G$  properly.

- a) Find  $\chi'$  for the six-cycle (i.e., the graph with a cycle of six vertices and no other edges).
- b) Find  $\chi'$  for the seven-cycle.
- c) Find  $\chi'$  for the complete graphs having five and six vertices.
- d) Generalize from c) and prove your result.
- e) Prove that, for any  $G$ ,  $\chi'(G) \leq 2\Delta - 1$ , where  $\Delta$  is the maximum degree of a vertex in  $G$ . Give a proof by algorithm. (In fact,  $\chi'(G) \leq \Delta + m$ , where  $m$  is the maximum edge multiplicity (number of edges between the same two vertices), but the proof is subtle. This result is known as Vizing’s theorem.)

The remaining problems in this section concern planar graphs.

30. (2) A graph is *planar* if it can be drawn on a plane surface with no edges intersecting except at vertices. Which of the graphs in Fig. 3.29, drawn with crossing edges, are planar (because they could be redrawn without crossing edges)? You needn’t

give a proof when you claim that the graph is not planar.



**FIGURE 3.29**

31. (2) Let  $G(V, E)$  be an acyclic graph with at least one edge. Give an algorithmic proof that there must be at least one vertex of degree 1. Hint: Walk around the graph following your nose; if you are not at a vertex of degree 1, then either you can continue the walk or you have found a cycle.

32. (2) Use the result of [31] to show that a connected, acyclic graph with  $n$  vertices has  $n - 1$  edges. Hint: Any degree-1 vertex and the edge incident on it can be removed from the graph without affecting the relative number of edges and vertices.

33. (3) An area surrounded by the edges of a planar graph and containing no smaller such area within it is called a **region**. The **infinite region** is the area not surrounded by edges of the graph (i.e., the area outside all the edges). Let  $G$  be a connected, planar graph with  $n$  vertices,  $m$  edges, and  $r$  regions where  $r$  includes the infinite region. Derive **Euler’s formula**:

$$n - m + r = 2.$$

Hint: Use induction on the number of regions with the result of the previous problem used for the basis case.

**34.**  $\langle 3 \rangle$  For the complete bipartite graph  $K_{mn}$  (see [39, Section 3.3]),

- a) show that  $K_{22}$  and  $K_{32}$  are planar; and
- b) show that  $K_{33}$  is not planar. *Hint:* By Euler's formula (see [33]), how many regions would a planar representation have? At the least, how many edges would surround each region? Now count the number of edges two ways and get a contradiction.

**35.**  $\langle 3 \rangle$  The  $n$ -cube graph  $Q_n$  is the edge graph of the analog of a cube in  $n$  dimensions. Thus,  $Q_1$  is an edge,  $Q_2$  is a four-cycle, and  $Q_3$  is the usual skeletal image of a cube.

- a) Define  $Q_n$  for all  $n$  recursively.
- b) What is the smallest positive integer  $n$  for which  $Q_n$  is not planar?

**36.**  $\langle 3 \rangle$  Let  $G$  be a simple, connected planar graph with no loops,  $n$  vertices,  $m$  edges, and  $r > 1$  regions.

- a) If each region has at least  $k$  edges bounding it, show that  $kr \leq 2m$ .
- b) Show that  $n \geq (1 - 2/k)m + 2$ .

**37.**  $\langle 3 \rangle$  Suppose that  $G$  is a graph as in [36] in which every region is bounded by at least four edges. Show that  $G$  has a vertex of degree 4 or less.

**38.**  $\langle 3 \rangle$  In the spirit of [37], what can you prove about vertex degrees for a planar graph in which every region has at least five edges?

- 39.**  $\langle 2 \rangle$  Draw maps which require exactly
  - a) two,
  - b) three, and
  - c) fourcolors so that no two regions with a common boundary have the same color.

## 3.7 Trees

---

We devote a section in this chapter to trees because they are by far the most important special case of graphs. Trees are useful in numerous applications, particularly in computer science. Sometimes a tree provides the most appropriate structuring of the data for *thinking* about a problem, and sometimes *using* a tree in an algorithm to solve the problem is most appropriate. In this section we'll discuss spanning trees and binary trees, special cases useful in many applications.

### Spanning Trees

We first encountered the idea of a spanning tree in Example 3, Section 1.1, about building a fiber optics network. Formally, we define a spanning tree as follows.

---

**Definition 1.** A **spanning tree** of a graph  $G(V, E)$  is a connected, acyclic subgraph of  $G$ , which includes all the vertices in  $V$ .

---

A spanning tree, therefore, is a tree in the graph (not the digraph) sense of the Trees subsection of Section 3.1. Since a spanning tree is a subgraph of  $G$ , all its edges must be in the set  $E$ . Recall from Section 3.1 that a subgraph  $G(V', E')$  of  $G(V, E)$  spans  $G$  if  $V' = V$ .

Does every graph have at least one spanning tree? No, because a spanning tree is connected so that an unconnected graph cannot have a spanning tree. Is

it sufficient for a graph to be connected for it to have a spanning tree? Yes, and Algorithm 3.14 SPANTREE generates a spanning tree for any connected graph. This claim deserves a formal proof.

### Algorithm 3.14

#### SpanTree

|                                                                   |                |                                         |
|-------------------------------------------------------------------|----------------|-----------------------------------------|
| <b>Input</b>                                                      | $G(V, E)$      | [A connected graph]                     |
| <b>Output</b>                                                     | $T = (V, E_T)$ | [ $V$ and the edges of a spanning tree] |
| <b>Algorithm</b> SPANTREE                                         |                |                                         |
| Choose some $v \in V$                                             |                |                                         |
| $V_T \leftarrow \{v\}$ [Initialize vertex set of $T$ ]            |                |                                         |
| $E_T \leftarrow \emptyset$ [Initialize edge set]                  |                |                                         |
| <b>repeat</b> until $V_T = V$                                     |                |                                         |
| Pick a $w \in V_T$ and $u \in V - V_T$ such that $\{w, u\} \in E$ |                |                                         |
| $V_T \leftarrow V_T \cup \{u\}$ [Update]                          |                |                                         |
| $E_T \leftarrow E_T \cup \{\{w, u\}\}$                            |                |                                         |
| <b>endrepeat</b>                                                  |                |                                         |

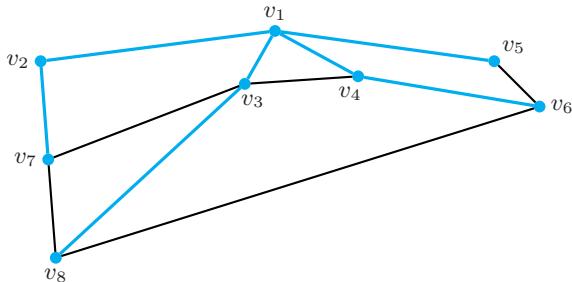
**Theorem 1.** If  $G$  is connected, SPANTREE terminates and the output,  $T = (V, E_T)$ , is a spanning tree of  $G$ .

**PROOF** First, must the algorithm terminate? Yes, because  $V$  is finite and, as the subset  $V_T$  of  $V$  contains one more vertex after each pass through the loop, eventually  $V_T = V$ . But could the algorithm fail before  $V_T = V$ ? That is, at some point before  $V_T = V$  might it be impossible to pick a  $w \in V_T$  and a  $u \in U = V - V_T$  with an edge between them? No, because we have assumed that  $G$  is connected, so there must be a path from any vertex in  $V_T$  to any one in  $U$ . Some edge on this path will have one end in  $V_T$  and one in  $U$ .

But is the output really a spanning tree? We argue it is, using the loop invariant that, after each pass through the loop,  $T$  is a tree that spans a subgraph of  $G$ . This is certainly true before the first entry into the loop, since the single vertex  $v$  in  $V_T$  is (trivially) a tree and a subgraph of  $G$ . Suppose (induction hypothesis) that  $T$  has the desired property after  $k$  passes through the loop. At the subsequent pass, the edge added to  $T$  is in  $G$  and connects the vertex  $u$  to a vertex  $w$  already in  $V_T$ . The induction hypothesis assures us that the old  $T$  was connected, so the edge  $\{w, u\}$  makes the new  $T$  connected. Further, the new  $T$  is still acyclic because there is only one edge from  $u$  to vertices on the previous  $T$  so  $u$  cannot be on a cycle in the new  $T$ . Therefore  $T$  is still a tree and a subgraph of  $G$  at each exit from the loop. Thus at the end, since  $V_T = V$ ,  $T$  is a spanning tree of  $G$ . ■

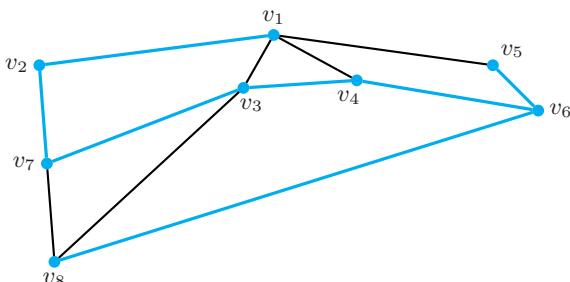
Since SPANTREE does result in a spanning tree, Theorem 1 shows that every connected graph does have a spanning tree. Now suppose that  $G$  is not a connected graph. Then what is the result of SPANTREE? With only a slight modification, it is just a tree which spans that *component* of  $G$  containing the vertex chosen in the first step of the algorithm. Can you explain this [6]?

The notion of spanning trees unifies a variety of different algorithms in graph theory, some of which we have discussed earlier in this chapter. Consider first the algorithms for BFS and DFS. BREADTHFIRSTSEARCH starts with a vertex  $v$  and then “visits” all the vertices adjacent to  $v$ . If we let part of each visit consist of putting the vertex visited and the edge joining it to  $v$  into a set, we would have the start of a spanning tree. If we continue building this set as subsequent vertices are visited we would end up with a spanning tree. Figure 3.30 illustrates this process. Similarly, for Algorithm DEPTHFIRSTSEARCH, if we place the appropriate edge and vertex into  $T = T(V_T, E_T)$  each time we visit a vertex, we get a spanning tree for  $G$ . This is illustrated in Fig. 3.31. This means that our BFS and DFS algorithms are really nothing more than special cases of SPANTREE. In these cases, the rules to pick the vertices  $v$  and  $u$  are specified, so that vertices will be added to the spanning tree in BFS or DFS order. Our argument here doesn’t constitute a proof of this fact, but it can be proved without too much trouble [1].



**FIGURE 3.30**  
A spanning tree by breadth first search.

Spanning Tree:  $v_1; v_2; \{v_1, v_2\}; v_3; \{v_1, v_3\}$   
 $v_4; \{v_1, v_4\}; v_5; \{v_1, v_5\}; v_7; \{v_2, v_7\}$   
 $v_8; \{v_3, v_8\}; v_6; \{v_4, v_6\}$



**FIGURE 3.31**  
A spanning tree by depth first search.

Spanning Tree:  $v_1; v_2; \{v_1, v_2\}; v_7; \{v_2, v_7\}; v_3$   
 $\{v_3, v_7\}; v_4; \{v_3, v_4\}; v_6; \{v_4, v_6\}$   
 $v_5; \{v_5, v_6\}; v_8; \{v_6, v_8\}$

Another algorithm which, in effect, constructs a spanning tree is Algorithm DIJKSTRA in Section 3.4. In this algorithm we add a vertex to the set  $U$  by looking for that vertex closest to the starting vertex  $S$  that hasn't yet been added to the set  $U$ . Suppose each time we add a vertex  $u$  we also add the edge joining  $u$  to the vertex in  $U$  on the shortest path from the initial vertex to  $u$ . If we use the modification of DIJKSTRA in which the shortest path to all vertices is found (see [6, Section 3.4], then we would end up with a spanning tree (really a *directed* spanning tree since we defined this algorithm for digraphs; the starting vertex  $S$  is the root of this tree).

## Minimum Spanning Trees

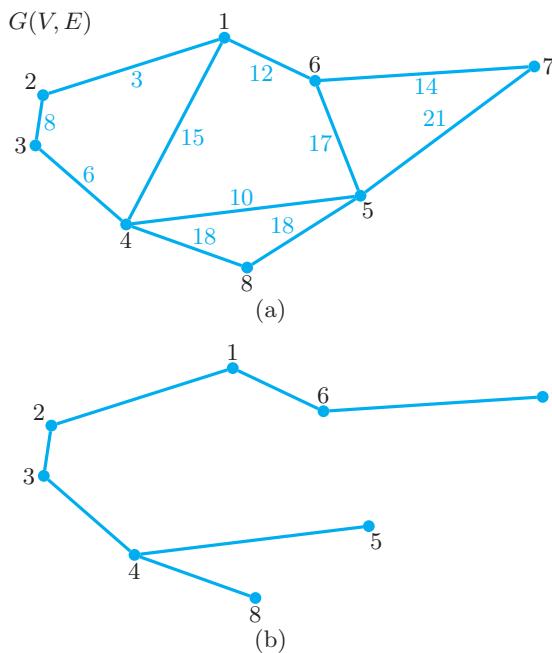
Given a weight for each edge of a graph  $G$ , define the weight of any subgraph  $G'$  to be the sum of the weights on the edges of  $G'$ . For a connected graph  $G$ , a **minimum spanning tree** (or MST) is a spanning tree of  $G$  with the minimum weight among all spanning trees of  $G$ . (We say *a* minimum spanning tree, not *the*, because, conceivably, more than one spanning tree could have the same minimum weight).

Knowing how to find an MST is of great practical importance, as we first encountered in Example 3, Section 1.1, where we considered finding a minimum spanning tree for a fiber optics network. In that example we presented a greedy algorithm and claimed that it finds an MST. Now we'll prove that claim. Algorithm 3.15 MINSPANTREE is a generalization of Algorithm FIBERNET in Section 1.1 and works for any connected graph. An example of how MINSPANTREE works is given in Fig. 3.32.

### Algorithm 3.15

## MinSpanTree

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                          |                          |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|--------------------------|
| <b>Input</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | $G(V, E)$                | [A connected graph]      |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | $w(e)$ for all $e \in E$ | [Weights of all edges]   |
| <b>Output</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | $E_T$                    | [The edge set of an MST] |
| <b>Algorithm</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | MINSPANTREE              |                          |
| <p>Choose any vertex <math>v</math> in <math>V</math></p> $V_T \leftarrow \{v\}; E_T \leftarrow \emptyset$ <p><b>repeat until</b> <math>V_T = V</math></p> <p style="padding-left: 2em;">Find a minimum weight edge <math>\{j, k\}</math> among all edges from vertices <math>j</math> in <math>V_T</math> to vertices <math>k</math> not in <math>V_T</math>;</p> <p style="padding-left: 2em;">if a tie, choose arbitrarily</p> $V_T \leftarrow V_T \cup \{k\}$ $E_T \leftarrow E_T \cup \{\{j, k\}\}$ <p><b>endrepeat</b></p> |                          |                          |



An example of Algorithm MINSPANTREE. (a) The given graph; (b) the spanning tree that results has edges added in the order  $\{1, 2\}$ ,  $\{2, 3\}$ ,  $\{3, 4\}$ ,  $\{4, 5\}$ ,  $\{1, 6\}$ ,  $\{6, 7\}$ ,  $\{4, 8\}$ . Note that at the last stage we chose  $\{4, 8\}$  arbitrarily instead of  $\{5, 8\}$ .

**FIGURE 3.32**

---

**Theorem 2.** MINSPANTREE finds a minimum spanning tree of any connected graph  $G(V, E)$ .

---

To prove the theorem we first need the following lemma.

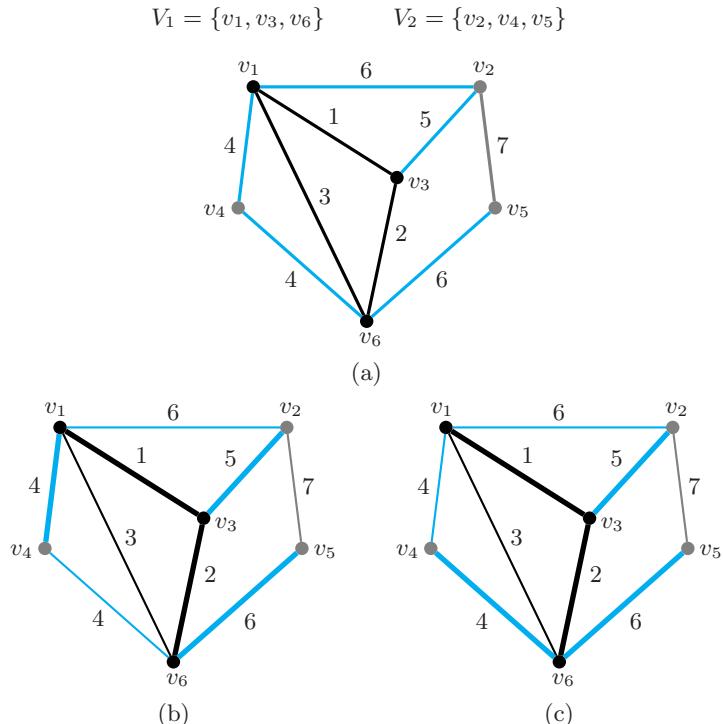
---

**Lemma 3.** Suppose the vertices of a weighted, connected graph  $G$  are divided into any two disjoint, nonempty sets,  $V_1$  and  $V_2$ . Suppose further that  $T$  is any MST of  $G$ . Finally, let  $e$  be any minimum weight edge among all edges in  $G$  linking a vertex in  $V_1$  with a vertex in  $V_2$ . Then either

- a)  $e \in T$ , or
  - b) There is another MST,  $T'$ , containing  $e$ , that differs from  $T$  only by the insertion of  $e$  in place of another edge  $e'$  of equal weight between  $V_1$  and  $V_2$ .
-

**PROOF** If  $e \in T$  we are done. If  $e \notin T$ , then add  $e$  to  $T$  to get  $G'$ , which must contain a cycle (why?). This cycle must have at least one other edge  $e'$  linking a vertex in  $V_1$  to a vertex in  $V_2$  (why?). If we delete  $e'$  from  $T$ , we again have a spanning tree,  $T'$  (why?). If  $e'$  was not also of minimum weight among edges linking  $V_1$  and  $V_2$ , then  $T'$  would have lower weight than  $T$ , a contradiction. So we conclude that  $w(e) = w(e')$  so that  $w(T) = w(T')$ , and, therefore,  $T'$  is an MST containing  $e$ . ■

Figure 3.33 illustrates the key point in the proof of this lemma. With  $T$  the MST in Figure 3.33b and  $V_1$  and  $V_2$  as shown in the figure, one minimum weight edge from a vertex in  $V_1$  to a vertex in  $V_2$  is  $e = \{v_4, v_6\}$ . Edge  $e$  is not in  $T$  but it is in  $T'$  shown in Figure 3.33c in which  $e' = \{v_1, v_4\}$  in  $T$  has been replaced by  $e$ . ( $V_1$  is the set  $V_T$  in Algorithm MINSPANTREE after two iterations of the repeat-loop if  $v_1$  is chosen as the initial  $v$ .)



An illustration of Lemma 3. (a) Graph  $G$ , with the vertices of  $V_1$  and the edges between them in black, the vertices of  $V_2$  and the edges between them in gray, and the edges between  $V_1$  and  $V_2$  in color; (b) one minimum spanning tree  $T$  of  $G$  (thick lines); (c) a second MST  $T'$  of  $G$  (thick lines).

**FIGURE 3.33**

**PROOF OF THEOREM 2.** Let  $E_k$  be the set of  $k$  edges in the tree  $T$  after the  $k$ th pass through the repeat-endrepeat loop in MINSPANTREE (i.e.,  $E_k$  is the set  $E_T$

after  $k$  passes). Our proof is by induction on  $k$ , where  $P(k)$  is the claim that  $E_k$  is contained in some MST of  $G$ . The final assertion, when  $k = |V|$ , is thus that all the edges of  $T$  are in some MST  $W$ . At that point  $T$  is a spanning tree, so  $T = W$ , which will prove the theorem.

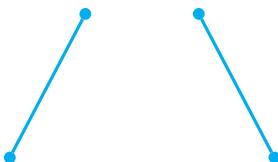
The basis case  $P(0)$  (before entering the loop) asserts that the empty set of edges is on an MST. This is true since the empty set is contained in the set  $E$  of edges of any MST. Now suppose  $P(k)$  is true. To prove  $P(k) \Rightarrow P(k+1)$ , let  $V_1$  be the vertices in  $T$  after the  $k$ th pass and let  $V_2$  be all other vertices. Let  $W$  be an MST that contains  $E_k$  and let  $e$  be the edge chosen by MINSPANTREE on the  $(k+1)$ st pass. If  $e \in W$ , then  $P(k+1)$  is true. But if  $e \notin W$  then by Lemma 3, there is another edge  $e' \notin E_k$  linking a vertex in  $V_1$  to one in  $V_2$  such that  $W' = W + e - e'$  is an MST. That is,  $E_{k+1}$  is in  $W'$ , proving  $P(k+1)$ . This completes the induction and thus proves  $P(|V|)$ . ■

Now that you have had more experience with the algorithmic notation than when we presented Example 3, Section 1.1, you should be able to write MINSPANTREE without resort to English [10]. MINSPANTREE is often called **Prim's algorithm**. A related algorithm, called **Kruskal's algorithm**, is considered in [12–13].

## Binary Trees

A binary tree is a digraph in which:

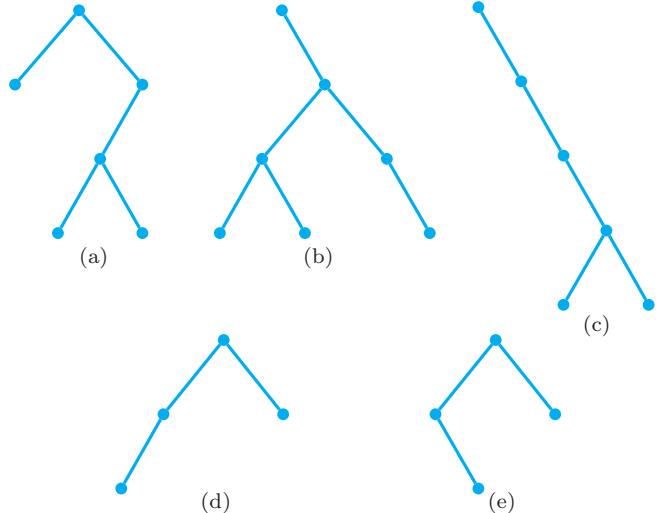
1. Each node  $n$  has at most two offspring. (The **offspring**, or **children**, of a node  $n$  are the nodes at the ends of branches leading from  $n$ .)
2. Any offspring of a node must be designated as a left or a right offspring. Thus the following are distinct binary trees because one is a root with a single left branch and the other is a root with a single right branch.



Some further examples of binary trees are shown in Fig. 3.34.

Thus, a binary tree is a tree in the digraph sense defined in Section 3.1 but with the addition of Condition 2 above.

Recall Example 3, Section 3.1, in which we showed how a list of words can be sorted using a tree. That tree was, in fact, a binary tree. Many algorithms that involve two-way (binary) decisions such as sorting algorithms, as well as many other algorithms and problems related to trees, are naturally expressed in terms of binary trees. Another advantage of binary trees is that, because they are restricted to a maximum of two offspring per node, there are efficient computer representations of binary trees. In this section we consider only the problem of *traversing* (i.e.,

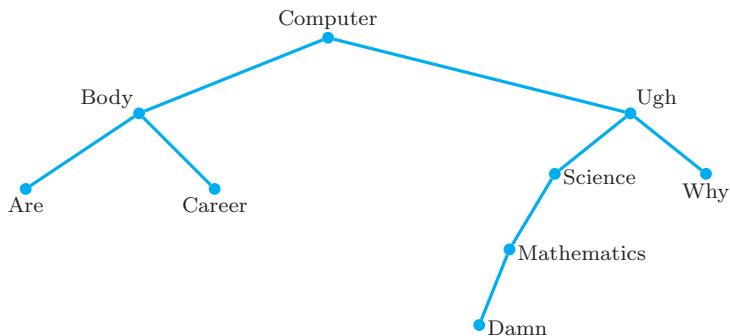


**FIGURE 3.34**

Some examples of binary trees. (Note that as *trees*, (d) and (e) are isomorphic, (as defined in Section 3.1), but not as *binary trees* [25].)

visiting) the nodes of binary trees. Doing this efficiently is important in applications of binary trees, and it also illustrates another application of depth first search.

In Example 3, Section 3.1, we gave a rule for alphabetizing the words stored in the binary tree of Fig. 3.3b (which is reproduced in Fig. 3.35) by just listing them “left to right”. Now we’ll develop a more formal and algorithmically practical method using binary tree traversal.



**FIGURE 3.35**

A tree of words to be sorted.

The general problem is this: We have a binary tree in which certain information is stored in (associated with) each node. This information might be words, as in Fig. 3.35, or much larger *records*, such as an employee’s personnel record (name, address, date of birth, social security number, salary, etc.). We want to process the data associated with each node. Such processing might, for example, consist of

- alphabetizing the data (e.g., by surname) or numerically ordering it (e.g., by social security number);
- finding all data with a common characteristic (e.g., salary greater than \$30,000); or

- printing mailing labels in order to send some document to all employees.

We naturally want some systematic procedure which ensures that the data at each node is processed once and only once. For data stored in a binary tree there are three basic paradigms for doing this:

### **Preorder traversal**

- Process the root.
- Traverse the left subtree (by Preorder).
- Traverse the right subtree (by Preorder).

### **Inorder traversal**

- Traverse the left subtree (by Inorder).
- Process the root.
- Traverse the right subtree (by Inorder).

### **Postorder traversal**

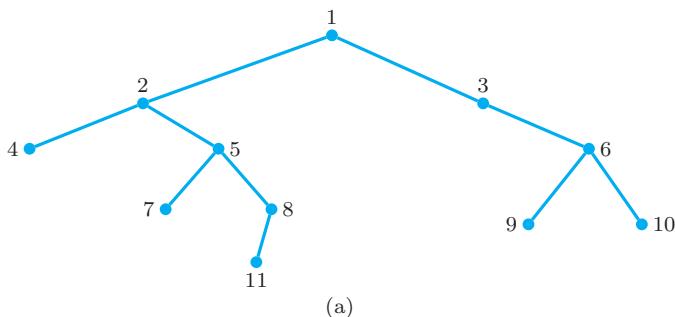
- Traverse the left subtree (by Postorder).
- Traverse the right subtree (by Postorder).
- Process the root.

By this time you should have seen enough recursion and inductive definitions to be comfortable with descriptions like these that define things in terms of themselves. There are, of course, three other traversal methods that we might define by interchanging left and right, but there is no good reason for doing so — except perhaps if your native language is Arabic or Hebrew.

Be sure you understand the meaning of “process the root” in each method of traversal. In order to traverse the tree we will have to look at some nodes more than once, in the sense of determining whether a node has a left or right child. But we’ll only visit (i.e., process) the data associated with each node once.

An example of each of these methods of traversal is shown in Fig. 3.36, where for convenience, we have only numbered the nodes and haven’t shown any data associated with them. Let’s consider the Inorder traversal in detail to make sure that the definitions are really clear:

1. We first traverse the left subtree, that is, the subtree whose root is node 2.
2. Applying Inorder traversal to this subtree, we first traverse its left subtree, which is the single leaf node 4.
3. As node 4 is a leaf, its left and right subtrees are empty, and so we immediately “process the root”, namely, node 4 itself, after which the traversal of the subtree with root 4 is complete.
4. As the traversal of the left subtree of the tree rooted at 2 is now complete, following Inorder we then process the root (i.e., 2).
5. Now we traverse the right subtree of the subtree with root 2, which consists of the subtree with root node 5. Etc.



(a)

| Preorder | Inorder | Postorder |
|----------|---------|-----------|
| 1        | 4       | 4         |
| 2        | 2       | 7         |
| 4        | 7       | 11        |
| 5        | 5       | 8         |
| 7        | 11      | 5         |
| 8        | 8       | 2         |
| 11       | 1       | 9         |
| 3        | 3       | 10        |
| 6        | 9       | 6         |
| 9        | 6       | 3         |
| 10       | 10      | 1         |

(b)

**FIGURE 3.36**

Examples of binary tree traversal:  
 (a) diagram and  
 (b) order of node processing.

Which traversal method should we use to alphabetize the tree in Fig. 3.35? That is, which traversal method will process the nodes in alphabetical order? Inspection of Fig. 3.35 reveals that Inorder traversal has the desired property, which we can easily prove by strong induction on the number of nodes  $n$  of the tree. The basis case ( $n=1$ ) is immediate because there is only the item at the root. Now suppose (the induction hypothesis) that Inorder traversal processes in alphabetical order any tree with  $n$  or fewer nodes constructed as in Example 3, Section 3.1. Consider a tree with  $n+1$  nodes. Inorder traversal first processes the left subtree. This subtree has  $n$  or fewer nodes so, by the induction hypothesis, it processes the left subtree in alphabetical order. Then it processes the root which is next in alphabetical order by the construction of the tree. (Recall that, in the construction, when a new word was added, it went left or right from the root depending on whether it came before or after the root alphabetically.) Finally, Inorder traversal processes the right subtree, all of whose nodes follow the root in alphabetical order. By the induction hypothesis the right subtree is also processed in alphabetical order. Therefore the whole tree is processed in alphabetical order.

Because of their simple recursive form, we can readily express each of the three traversal methods in algorithmic form. Rather than present three separate algorithms, Algorithm 3.16 BINARYTREETRAVERSAL is a three-in-one algorithm for all three traversals with the method of traversal determined by the value of the input variable  $c$ .

By comparing this algorithm with Algorithm DEPTHFIRSTSEARCH in Section 3.5, you can see that BINARYTREETRAVERSAL is just a variation on DFS.

### Algorithm 3.16

#### BinaryTree- Traversal

**Input** A binary tree

$c$  [ $c = 1$  for Preorder,  $c = 2$  for Inorder,  $c = 3$  for Postorder]

**Output** The results of the processing of the data at each vertex

**Algorithm** BINARYTREETRAVERSAL

**procedure** traverse( $u$ )

**if**  $c = 1$  **then** process( $u$ )

[For preorder traversal]

**if**  $u$  has a left child  $lc$  **then**

        traverse( $lc$ )

**if**  $c = 2$  **then** process( $u$ )

[For inorder traversal]

**if**  $u$  has a right child  $rc$  **then**

        traverse( $rc$ )

**if**  $c = 3$  **then** process( $u$ )

[For postorder traversal]

**endpro**

$v \leftarrow$  root of tree

[Main algorithm]

    traverse( $v$ )

In DFS we deal with a general graph. Because binary tree traversal deals with a special kind of digraph — namely, a binary tree — we are able to replace the for-endfor loop in the DFS algorithm with the if statements in BINARYTREETRAVERSAL. From the perspective of depth first search, the only difference between the three traversals is the place in the algorithm at which the processing of vertices (the visits) is performed.

Algorithm BUILDTREE given in Example 3, Section 3.1, for building the binary tree together with BINARYTREETRAVERSAL with  $c = 2$  for inorder traversal, defines a method of *sorting* (e.g., alphabetizing) any set of data which has associated with it a *key* (e.g., social security number, surname). Methods of sorting have led to a large literature in computer science. **Tree sorting**, as presented here, is the basis of some efficient and useful methods of sorting. We'll discuss other methods of sorting in Section 7.7.

## Problems: Section 3.7

1. ⟨2⟩ Prove that both BREADTHFIRSTSEARCH and DEPTHFIRSTSEARCH are just adaptations of SPANTREE in which the vertices are added to the tree in a specified order.

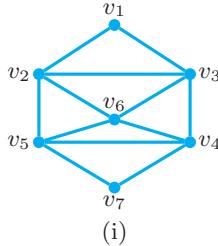
2. ⟨2⟩

- a) Restate the algorithms for BFS and DFS so that the output of each is a spanning tree for

the input graph.

- b) For graphs (i) and (ii) below, apply the algorithm for BFS from a) to find a spanning tree, starting at  $v_1$ . Where there is a choice of which vertex to visit next, visit the one with the lowest index. Show your work by drawing a picture of the graph with the final tree highlighted

and the tree edges labeled by their order of inclusion.



c) Repeat b) for the DFS algorithm found in a).

3. ⟨1⟩

a) Redraw the graphs in [2b] with the trees dangling down from the root, each edge in the tree having the same length, and nontree edges fitting in where they must.

b) Repeat for [2c].

4. ⟨2⟩

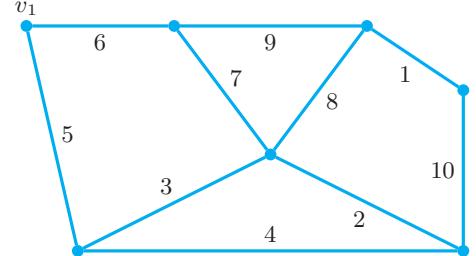
a) Let  $G$  be a connected graph and let  $T$  be any spanning tree obtained by BFS applied to  $G$ . Let a **cross-edge** be any edge of  $G - T$ . Suppose that  $T$  has been drawn with the root  $v_0$  at the top and all other vertices hanging down by levels (i.e., vertices two edges away from  $v_0$  in  $T$  are drawn two units lower, and so on). If the cross-edges from  $G$  are now drawn in, argue that each cross-edge is either horizontal or goes between adjacent levels.

b) What can you say about cross-edges when DFS is applied to  $G$ ?

5. ⟨2⟩ Modify Algorithm DIJKSTRA so that, when it finds the shortest path from  $S$  to all the other vertices (see [6, Section 3.4]), it outputs a spanning tree for the graph.

6. ⟨2⟩ Algorithm SPANTREE doesn't work for graphs that are not connected. What goes wrong? Modify SPANTREE so that for disconnected graphs it outputs a tree which spans that component of  $G$  containing the initially chosen vertex  $v$ .

7. ⟨1⟩ Perform Algorithm MINSPANTREE for the following graph  $G$ . Display your work by highlighting the tree edges; also, to indicate the order in which you add edges to the tree, label those edges 1, 2, etc., in circles (to avoid confusion with the weights, which are uncircled).



8. ⟨2⟩ Using the terminology and notation of Lemma 3, show that, if the minimum weight edge joining a vertex in  $V_1$  to a vertex  $V_2$  is unique, then this edge is on every MST.

9. ⟨2⟩ Prove that if the edge weights of  $G$  are distinct, then the MST is unique and MINSPANTREE finds it.

10. ⟨2⟩ Rewrite Algorithm MINSPANTREE so that all the English language statements are replaced by statements in our algorithmic language.

11. ⟨2⟩ In the repeat-loop of MINSPANTREE you must find the least weight edge among all those from  $V_T$  to its complement. The direct way to do this requires checking all possible pairs of vertices, one in  $V_T$  and one in the complement, and makes MINSPANTREE an  $\text{Ord}(|V|^3)$  algorithm. However, there is a smarter way, very similar to the way the current distance  $d(v)$  was updated in Algorithm 3.6, DIJKSTRA, in Section 3.4. With this approach you can easily reduce MINSPANTREE to  $\text{Ord}(|V|^2)$ . Namely, for each  $v$ , let  $m(v)$  be the minimum weight of any edge from  $v$  to the current  $V_T$ . After vertex  $k$  joins  $V_T$  in another iteration of the repeat-loop, for each  $v \notin V_T$ , there is at most one more edge to look at to update  $m(v)$ .

Redo [10] implementing this smarter approach (unless you already thought of it the first time you did [10]!).

12. ⟨1⟩ For the graph of [7], apply the following variant of Algorithm MINSPANTREE: At each stage add the edge of minimum cost *anywhere* which does not form a cycle with any edges you have already chosen. (Thus the set of edges chosen at any point may be a disconnected acyclic set — a **forest**.) Label the edges with circled numbers in the order you add them.

13. ⟨3⟩ Call the algorithm obtained in [12] MINSPANTREE'. (It is usually called **Kruskal's**

**algorithm.)** Prove that MINSPANTREE', like Algorithm MINSPANTREE, finds the MST. *Hint:* Use Lemma 3. When you add edge  $e$  using MINSPANTREE', let  $v$  be either end node of  $e$  and let  $V_1$  consist of  $v$  and all nodes connected to  $v$  by edges that you have already chosen (there may be none).

14. (2) There is a maximum weight spanning tree version of Prim's algorithm.

a) State it and apply it to the graph in [7].

b) Prove that it is correct.

15. (2) There is also a maximum weight spanning tree version of Kruskal's algorithm (see [12–13]).

a) State it and apply it to the graph in [7].

b) Prove that it is correct.

16. (3) Review the pipeline problem of [26, Section 3.4]. A variant of DIJKSTRA solved it. Prove that Algorithm MINSPANTREE, *unchanged*, also solves it. This is very surprising because MINSPANTREE minimizes a sum, not a minimax, and does so over the whole tree, not over paths.

17. (3) Assertion: Algorithm MINSPANTREE is correct. What's wrong with the following proof of this assertion?

“Proof”: Induction on the number  $n$  of nodes. In the basis case,  $n = 2$ , there is only one edge, so the algorithm is clearly correct. Inductive step. Assume that the algorithm is correct when there are  $n$  nodes. We are given a graph  $G$  with  $n + 1$  nodes. Let  $v$  be the node which MINSPANTREE connects last. Suppose that we had run this algorithm on the subgraph  $G'$  connecting the other  $n$  nodes. Then the final tree from the run on  $G'$  would be the same as the tree obtained after  $n$  steps on  $G$ . Why? Because the subset of edges from which MINSPANTREE picks a minimum at each stage in  $G'$  is a subset from the set from which it picks in  $G$ . However, the subset contains the element which is picked in  $G$ , so the same edge is picked. Therefore we conclude that after  $n$  nodes are joined to the tree, when the algorithm works on  $G$ , it has picked the minimum tree connecting those nodes. Now MINSPANTREE goes on to connect the final node  $v$ . It picks the minimum length edge to  $v$ . As it has picked the minimum set of edges apart from  $v$ , and the minimum edge

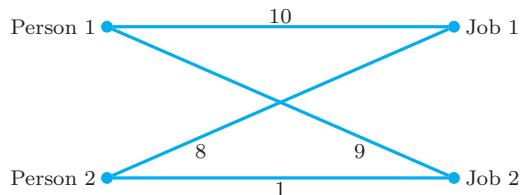
to  $v$ , it has picked the minimum tree connecting all  $n + 1$  nodes. ■

(Note: We don't know of a correct proof of the validity of MINSPANTREE using induction on the number of vertices.)

18. (2) Although greed pays in the MST problem, in other problems it can backfire miserably. Consider the assignment problem:  $n$  people are to be assigned to  $n$  jobs. Each person is given a rating on each job, where higher is better. Assume that the goal is to maximize the sum of the ratings of the assignment (See [7, Supplement, Chapter 1]).

a) Describe a greedy approach to this problem.

b) Carry out your greedy approach for the ratings depicted in the graph below. How good is your answer compared to the optimal solution?



c) Suppose lower ratings are better so that we wish to find the assignment that minimizes the sum of the ratings. Now a greedy approach works for the graph above. Change the numbers so that it does not work, and show the results.

19. (2) Use Algorithm SPANTREE to explain why, if a tree has  $n$  nodes, it has exactly  $n - 1$  edges.

20. (3) Show that any two of the following conditions imply the third and thus that any two may be used in the graph definition of a free tree.

i)  $G$  is connected.

ii)  $G$  is acyclic.

iii) In  $G$ ,  $|E| = |V| - 1$ .

21. (2) In chemistry, an **alkane** is a saturated acyclic hydrocarbon molecule. In graph theory terminology, this means that an alkane is a free tree with two types of nodes: C(carbon) nodes of degree 4 and H(ydrogen) nodes of degree 1.

a) Draw an alkane with three carbons. Draw three different alkanes with five carbons.

b) Chemistry books state without proof that every alkane is of the form  $C_nH_{2n+2}$ ; that is, if

there are  $n$  carbon atoms, there are  $2n + 2$  hydrogen atoms. Prove this assertion, using the result of [20] and Eq. (5), Section 3.1. Start by supposing that the alkane is  $C_nH_m$  and show that  $m = 2n + 2$ .

- 22.** (3) Define a “hydronitron” to be an acyclic molecule made up of hydrogen atoms (one bond) and nitrogen atoms (three bonds). (Actually, very few such molecules exist naturally, but suppose they did.) Determine all mathematically possible formulas  $N_nH_m$  for hydronitrons.

- 23.** (2) Consider a family tree, that is, a tree diagram of some person and all his or her descendants. Describe in terms of this tree what it means for two people to be

- a) siblings.    b) cousins.    c) second cousins.
- d) second cousins once removed (if you don’t know what this means, look it up in a dictionary or encyclopedia).
- e) first cousins twice removed.
- f) Based on these relations, write a general definition of “ith cousin jth removed”.

- 24.** (2) Draw all distinct binary trees with  
a) three nodes.    b) four nodes.

- 25.** (2) Explain why d) and e) of Figure 3.34 are isomorphic as digraph trees but not as binary trees.

- 26.** (2) Find and explain a recurrence for  $B_n$ , the number of binary trees with  $n$  nodes. Use it to compute  $B_3$ ,  $B_4$ , and  $B_5$ . (Compare your results with those of [24].)

- 27.** (2) A **ternary tree** is a tree in which each node has at most three offspring, which are the roots of left and/or center and/or right subtrees. Display all ternary trees with at most three vertices.

- 28.** (2) Derive a recurrence, similar to that in [26], for the number of ternary trees with  $n$  nodes.

- 29.** (3) In a binary tree, a **leaf** is defined to be a vertex on the bottom, that is, a vertex of outdegree 0. (We continue the convention that binary trees are really directed, with all directions down the page.) Any vertex in a binary tree that is not a leaf is called an **internal** node.

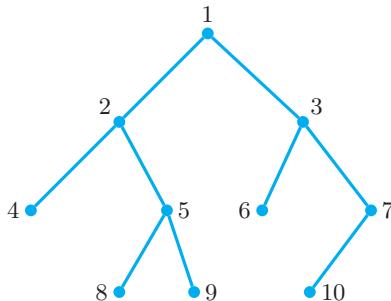
- a) In an undirected graph, a vertex is called a **leaf** if it has degree 1. If a binary tree is viewed as a free tree, it may or may not have the same number of leaves as when viewed as a binary tree. Explain.

- b)** A binary tree is a **2-tree** if every internal node has 2 children. There is a simple equation, true for every 2-tree, that relates the number of internal nodes,  $i$ , to the number of leaves,  $\ell$ . Find and prove this equation.

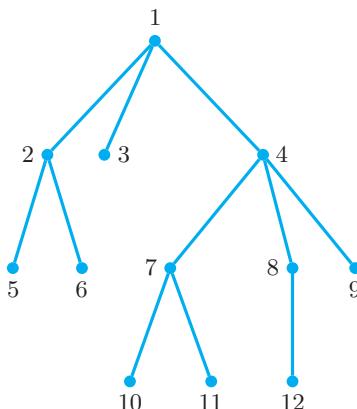
- c)** A  **$d$ -tree** is a generalization of a 2-tree in which every internal vertex has  $d$  children. Generalize the equation in b) to  $d$ -trees.

- 30.** (1) List the vertices of the binary tree shown below in

- a) Inorder.    b) Preorder.    c) Postorder.



- 31.** (2) In an **ordered tree**, each internal vertex may have any finite number of children, but they are ordered, say left to right on the page. Preorder and postorder traversals generalize easily to ordered trees. Write down the preorder and postorder for the vertices of the following ordered tree.



- 32.** (2) Suppose that you’re given a list of words like that in Fig. 3.35 and want to build a binary tree as in Example 3, Section 3.1. Describe the trees for which the nodes will be visited in alphabetical order using

- a) Preorder traversal.    b) Postorder traversal.

33. (3) Can you uniquely reconstruct a binary tree if you are told
- the Preorder and Inorder traversals? Explain.
  - the Preorder and Postorder traversals? Explain.
  - the Inorder and Postorder traversals? Explain.

Problems [34–39] are about game trees. A **game tree** is a tree in which the root represents the position at the start of the game, the nodes at the first level down represent the positions after all possible first moves by player 1, the nodes at the next level down represent all possible responses by player 2 to each of player 1's moves, etc. The leaves of the tree (the nodes with no branches going down from them) represent the final positions of the game and are labeled W, D, or L, depending on whether the final position is a win, draw, or loss for player 1. (See also [49–53, Supplement].)

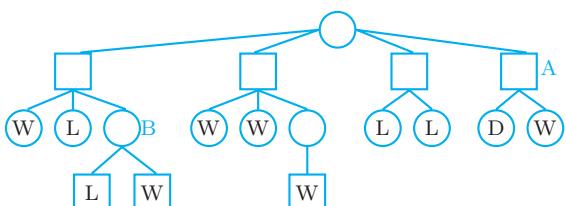
34. (2) Consider the following *minimax* algorithm for labeling the nonleaf nodes of a game tree.

i) Suppose that all the children of a node have labels W, D, or L. If the node represents a position where it is player 1's move, label the node with the *best* result among the labels of its children's nodes. If the node represents a position where it is player 2's move, label the node with the *worst* result among the labels of its children's nodes. Thus in the tree below, the node A would be labeled D (since it is player 2's move) and node B would be labeled W (since it is player 1's move).

ii) Working up from the bottom, label all the nodes. The label of the root is called the *value* of the game.

a) Explain why step i) gives reasonable values to each of the nonleaf nodes and why, therefore, we call the root the value of the game.

b) Find the label of the root of the following tree.



35. (2) Consider  $2 \times 2$  tic-tac-toe — not much of a game, but it serves a purpose. Draw the complete game tree. (For instance, the root has degree 4, with one edge for each corner in which player 1 can put an X.) Analyze the game; that is, for each node determine whether it should be labeled W or L.

36. (2) There is a lot of symmetry in tic-tac-toe. For instance, in [35], each of the four possible moves is into a corner and thus they are all essentially the same. Using all such symmetries, draw and analyze the “reduced” game tree for  $2 \times 2$  tic-tac-toe.

37. (2) Using the symmetry described in [36], draw the reduced game tree for ordinary  $3 \times 3$  tic-tac-toe, down through the point where each player has moved once. (The complete game tree is too much to ask for.)

38. (2) In the preceding problems, the value assigned to a node (W, D, or L) is the value to player 1. There is another approach: Assign to each node the value of the game to the player who moves from that node, should the game reach that point and assuming play proceeds optimally. (This approach makes the analysis of some games easier as the next problem will illustrate.) Modify the minimax algorithm in [34] for evaluating a game so that it works under this alternative approach.

39. (3) In the game of daisy, two players alternate picking petals from a daisy. Each time, each player gets to pick 1 or 2 petals. The person who picks the last petal wins. If the daisy has  $n$  petals, we say the game is  $n$ -petal daisy.

a) Draw and analyze the complete game tree for four-petal daisy.

b) Draw and analyze the complete game tree for five-petal daisy, working up from the bottom, as usual.

c) Determine the winner of five-petal daisy more quickly than in b) by making use of a) and labeling the nodes as described in [38]. That is, by looking at the four-petal tree, or just part of that tree, you can determine immediately who wins from the children nodes of the root of five-petal daisy. Explain, using the word “recursion”.

## Supplementary Problems: Chapter 3

1. ⟨3⟩ Prove that a graph without loops contains an elementary cycle of even length if the degree of each vertex is at least 3. *Hint:* Show that, if there is a cycle of odd length, it must be possible to construct another cycle which intersects it to get a cycle of even length.

2. ⟨2⟩

a) Suppose that, after Algorithm BUILDTREE terminates (for whatever size list), a new word comes along. It's easy to add it to the tree. How?

b) Now, suppose that, after BUILDTREE terminates, you want to delete some word on the list and adjust the tree accordingly. (For example, the list might be names of students and one student may have dropped out.) You can't simply delete the corresponding vertex; the remaining vertices must be reconnected so that the tree looks just as it would if the deleted word had not been present when BUILDTREE built the tree. Of course, you could just run the algorithm again on the shorter list but there's a better way. Devise an appropriate algorithm to take the tree and the word to be deleted as input and produce the desired tree as output.

3. ⟨3⟩ Let  $G$  be a simple, non-complete graph such that every vertex has degree 3.

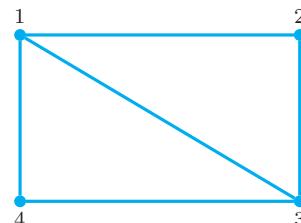
a) Explain why  $G$  has at least six vertices.  
b) There are two nonisomorphic graphs of this sort having exactly six vertices. Find them. Can you explain why there are no others?

4. ⟨2⟩ A **matching** in a graph  $G$  is a subgraph in which each vertex has degree 0 or 1. A **maximum matching** in  $G$  is a matching with as many edges as possible.

a) Find all nonisomorphic maximum matchings in the complete graph having four vertices.  
b) Find all nonisomorphic matchings (maximum or not) in the complete graph having five vertices.

5. ⟨2⟩ Given  $G(V, E)$  and  $U \subset V$ , the **induced** subgraph of  $G$  on  $U$ , denoted by  $\langle U \rangle$ , is defined to have vertex set  $U$  and edge set  $\{\{u, v\} \in E \mid u, v \in U\}$ .

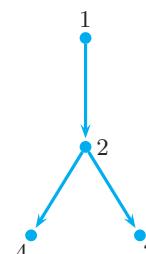
That is,  $\langle U \rangle$  consists of all edges of  $G$  between vertices of  $U$ . Now, let  $G$  be the graph



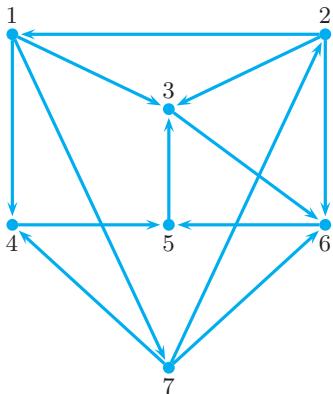
- a) Draw  $\langle \{1, 2, 3\} \rangle$  and  $\langle \{1, 3, 4\} \rangle$ .  
b) Draw a subgraph of  $G$  which is *not* an induced subgraph. Explain why it isn't.  
6. ⟨3⟩ The idea of a component seems straightforward. It's obvious that every vertex is in a largest connected subgraph of the entire graph and that these subgraphs partition the whole graph. However, when looked at more generally, the idea of a component is not so straightforward.

Let's define a  **$P$ -component** of a graph or digraph to be a maximal subgraph with property  $P$ . For instance, if  $P$  is the property of being connected, then a  $P$ -component is just an ordinary component. To say that subgraph  $H$  of graph  $G$  is maximal means that  $H$  is not a proper subgraph of any other subgraph  $J$  of  $G$  which also satisfies property  $P$ .

- a) Let  $P$  be the property of being weakly connected. Consider the digraph shown below. Show that there are two distinct maximal, weakly connected subgraphs containing vertex 2. Show that there is no way to partition the graph into maximal, weakly connected subgraphs.



- b) Now let  $P$  be the property of being strongly connected. For each vertex, find all the strongly connected components it belongs to in the digraph below.



- c) Show that if  $J$  and  $K$  are strongly connected subgraphs of some digraph, and  $V(J) \cap V(K) \neq \emptyset$ , then  $J \cup K$  is also strongly connected.
- d) Show that c) implies that each vertex of a digraph is in a unique maximal, strongly connected component and that these components partition the vertex set. (Note that some edges need not be in any strongly connected component.)
7. (3) Let  $B$  be a Boolean matrix (all entries 0 or 1), and let  $B^{(k)}$  be the **Boolean product** (ordinary matrix product with all nonzero terms replaced by 1) of  $B$  with itself  $k$  times. As special cases, let  $B^{(0)} = I$  and  $B^{(1)} = B$ . Now let  $A''$  be the Boolean adjacency matrix of a graph: the  $(i, j)$  entry is 0 if there are no edges between  $i$  and  $j$  and 1 otherwise.
- a) Prove that  $(A'' + I)^{(k)} = \sum_{i=0}^k (A'')^{(i)}$ , where the summation is a Boolean sum — 0 if the ordinary sum is 0 and 1 otherwise.
- b) Using a) or otherwise, prove that the  $(i, j)$  entry of  $(A'' + I)^{(k)}$  is 1 if and only if there is a path of length at most  $k$  between  $i$  and  $j$ , including null paths.
8. (3) Let  $D$  be a digraph with  $V = \{v_1, v_2, \dots, v_n\}$  and in which all edges point from lower to higher numbered vertices (as in the Prologue). There are no directed cycles in such a graph, so the total

number of directed paths between any pair of vertices is finite. Therefore it makes sense to try to compute  $N = [n_{ij}]$  where  $n_{ij}$  is the total number of directed paths from  $i$  to  $j$ . As usual,  $A$  is the adjacency matrix.

- a) Prove that  $N = (N + I)A$ .
- b) Part a) would seem to be of no help in obtaining  $N$ , because to obtain  $N$  on the left you already need to know it on the right. However,  $A$  is an **upper triangular matrix**, with all entries strictly above the main diagonal. Convince yourself that in order to determine the  $k$ th column of the product  $(N + I)A$ , you need only know the leftmost  $k - 1$  columns of  $N$  (and the  $k$ th column of  $A$ ). Use this observation to develop an iterative algorithm to compute  $N$ .
- c) Suppose that all you want to know is the number of directed paths from  $v_1$  to each other vertex. There is a much simpler algorithm that can do this. Think recursively and figure it out. (In fact, this algorithm amounts to looking only at the first row of both sides of  $N = (N + I)A$  and then using the ideas from part b). It's simpler to think about this problem without referring to matrices.) Where have you seen this problem — and this algorithm — before?
9. (3) Consider the graph with vertices labeled 1 through 50 and with vertex  $i$  adjacent to vertex  $j$  if either
- the sum of the digits of  $i$  is the same as the sum of the digits of  $j$  (e.g., vertices 8, 17, and 35 are pairwise adjacent because  $1+7 = 3+5 = 8$ ); or
  - $i$  is the square of  $j$  or vice versa (e.g., vertex 16 is adjacent to vertex 4).
- Answer the following questions using Algorithm WARSHALL run on a computer.
- a) Is vertex 25 connected by a path to vertex 36?
- b) How many connected components does the graph have?
- c) How many vertices are in the smallest component?
10. (3)
- a) Answer the same questions as in [9], using Algorithm DIJKSTRA. Show your work by drawing the shortest paths which the algorithm produces.

- b) Do the same thing, using Algorithm BREADTH-FIRST-SEARCH.

- c) Do the same thing once more, using Algorithm DEPTHFIRSTSEARCH.

11. (2) The surprising thing about Algorithm WARSHALL, particularly in its shortest path version (see [26, Section 3.2]) is what it does induction on: not the length of the path, nor the indices of the end vertices, but rather the indices of the intermediate vertices. Is such an unnatural choice necessary? To answer this question, determine whether the following seemingly more natural choices of inductive assumptions work to find the shortest path between each pair of vertices. That is, find out whether determining the next case is easy if the previous cases are known, so that an efficient algorithm can be constructed.

- a) An  $n \times n$  matrix  $M^{(k)}$  is known where the  $(i, j)$  entry is the length of the shortest path from  $v_i$  to  $v_j$  with at most  $k$  edges.

- b) A  $k \times k$  matrix  $N^{(k)}$  is known where the  $(i, j)$  entry gives the length of the shortest path from  $v_i$  to  $v_j$  in the induced subgraph on  $v_1, v_2, \dots, v_k$ . (See [5] for the definition of *induced*.)

12. (2) Suppose that digraph  $D$  has some negative edge weights, but that no directed cycle has negative weight.

- a) Explain why there is always a shortest path between any two points and why that path is simple.

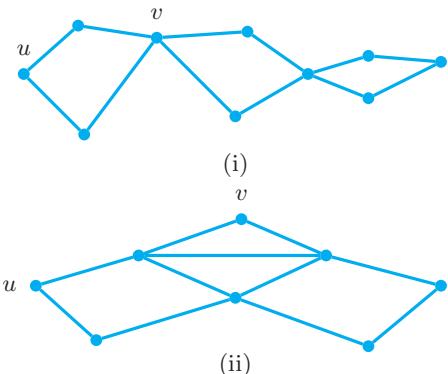
- b) Determine whether the shortest-path variant of WARSHALL works for such a path.

13. (4) Here is a modification of the path-growing procedure which forms the first part of Algorithm ECYCLE. If  $G(V, E)$  is the original graph, at each stage let  $G'$  be the graph with the original vertex set  $V$  and all so-far-unused edges. Instead of picking *any* unused edge from your current location, choose an edge whose removal does not increase the number of components of  $G'$  (where isolated vertices do count as components). If there are no such edges, pick any edge. The path-growing procedure with this modification is called **Fleury's algorithm**.

- a) Apply Fleury's algorithm to graphs (i) and (ii) starting at  $y$ .

- b) Apply it to the same graphs, starting at  $v$ .

- c) There is a theorem that says: If  $G$  has an Eulerian cycle, then Fleury's algorithm finds it. That is, with Fleury, recursion is never necessary; you won't get stuck until every edge has been traversed. Prove this theorem. *Hint:* Show that if  $G$  has an Eulerian cycle, then the only occasion in which there is not an edge from the current vertex  $v$  whose removal does not increase the number of components is when the degree of  $v$  in  $G'$  is 1. This means that, when there are no unused edges incident on the current vertex, there are no unused edges anywhere. Why? Why is this important?



14. (4) Prove that a tournament has a Hamiltonian cycle if and only if it is strongly connected.

15. (3) Prove that a strongly connected tournament on  $n$  vertices has directed cycles of length  $k$  for every integer  $k$  from 3 to  $n$ . Hint: Do induction on  $k$ .

16. (2) In the scheduling problem of the Prologue, the algorithm required the vertices of the directed graph  $D$  to be numbered so that all edges are directed from lower indices to higher. We did not explain there how such a numbering is supposed to be found, but we asserted that such a numbering exists if  $D$  has no directed cycles. It's possible to prove this assertion with a path-growing algorithm.

- a) Explain why the nonexistence of directed cycles is necessary for the numbering to exist.

To prove sufficiency, assume that  $D$  has no such cycles and show how to construct a numbering:

- b) Show that the only candidates for vertices to be numbered 1 are vertices with indegree = 0.

c) Give a proof by algorithm that at least one such vertex exists.

d) Suppose that a vertex with indegree = 0 has already been designated as  $v$ . Think iteratively and thus develop an algorithm to number all the vertices. Any such algorithm is called a **topological sort**.

17. (2) Devise another topological sort (see [16]) which uses as input the adjacency matrix of  $D$  and never grows any paths.

18. (3) [Computer Project] In Algorithm ECYCLE, splicing cycles into other cycles is more complicated than it appears — at least if a computer has to do it. Write a computer program which actually runs ECYCLE. First, think hard about the data structure you want to use to represent paths.

19. (3) Write a backtracking program, perhaps recursively, that determines conclusively whether a given (di)graph  $G$  has a Hamiltonian cycle. (The program will be terribly slow, even for moderate-sized graphs because, in effect, it tries every partial path emanating from your choice of start vertex.)

20. (2) Algorithm DIJKSTRA still runs even if the graph has some negative weights. Does it succeed in finding the shortest simple paths? If it does, explain why. If it doesn't, give an example and also give a reason why you might have expected it not to work even before you found a counterexample. (One way to do this would be to show that the recursive thinking which led to DIJKSTRA breaks down — knowing the previous case does not make the current case easy to compute.)

21. (2) Suppose that digraph  $G$  is connected and has some negative weight edges, but no directed cycle has nonpositive weight.

a) Explain why there always is a shortest path between any two points, and why that path is simple.

b) Determine whether Algorithm DIJKSTRA works for such a graph.

22. (3) Devise an algorithm to find the *second* shortest simple path between two vertices in a graph.

23. (4) The following algorithm is another way to find the shortest distances in a graph from some initial vertex  $v_0$  to all others. It is called **Ford's Algorithm**.

### Algorithm 3.17 FORD

**Input**  $G(V, E)$ ,  $v_0$

$w(E)$  [A weight function on the edges]

**Output**  $d(v)$  for all  $v \in V$

[Minimum distances from  $v_0$ ]

#### Algorithm FORD

$U \leftarrow \{v_0\}$ ;  $d(v_0) \leftarrow 0$

**for**  $v \in V - \{v_0\}$

$d(v) \leftarrow \infty$

**endfor**

**repeat until**  $U = \emptyset$

$v \leftarrow$  vertex in  $U$  with  $d(v)$  minimum

**for**  $u \in A(v)$  [Vertices adjacent to  $v$ ]

**if**  $d(v) + w(\{v, u\}) < d(u)$  **then**

$d(u) \leftarrow d(v) + w(\{v, u\})$

$U \leftarrow U \cup \{u\}$

**endif**

**endfor**

$U \leftarrow U - v$

**endrepeat**

a) Carry out Ford's algorithm for graph (i) in Fig. 3.37.

b) Carry out Ford's algorithm for the directed graph (ii) in Fig. 3.37. Edges without arrows can be traversed either way.

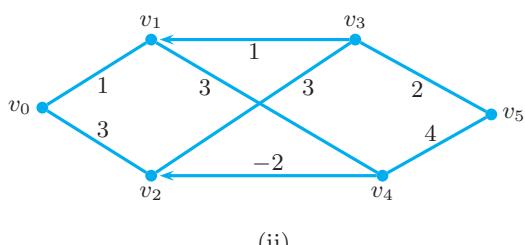
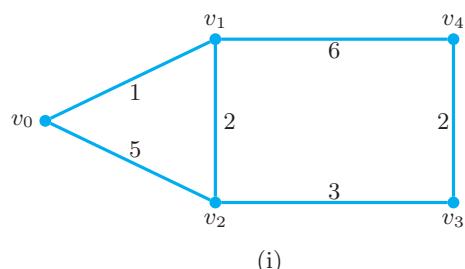


FIGURE 3.37

c) Compare the behavior of Ford's and Dijkstra's algorithms when all edges have positive weights.

d) Prove what you observed in c).

e) Show that Ford's algorithm is valid so long as there are no undirected edges with negative weights and no directed cycles with total weight negative. *Note:* An undirected negative edge amounts to a negative directed 2-cycle — go one way on the edge and then back. Thus the first condition is just a special case of the second.

f) What does Ford's algorithm do if there is a negative weight directed cycle (or a negative undirected edge)?

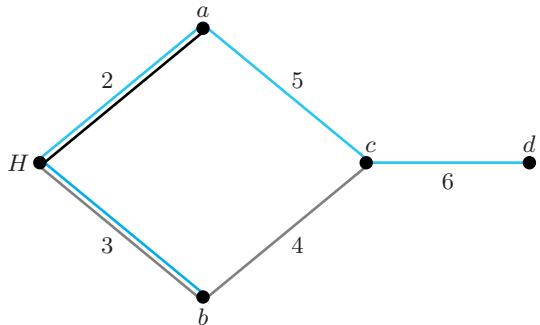
g) Modify Ford's algorithm so that it outputs the shortest path from  $v_0$  as well as the distances.

24. (2) In some circumstances, phone calls cost different amounts depending on the location of the two parties. Suppose there are  $n$  people who need to be informed of some news. Often a phone tree is set up in advance to spread the news when it comes. The person who first gets the news is instructed to phone certain people, those people are instructed to spread the news to certain other people, and so on. If the cost of connection between various of the  $n$  people differ, is there an algorithm we have studied that will find the minimum cost phoning scheme? Explain.

25. (3) We desire to build a network between a headquarters and a number of sites, where for each potential edge  $ij$ , the cost will be a fixed amount  $c_{ij}$  times the number of times the edge is used in paths to the headquarters. For instance, this model makes sense if there must be a separate cable to each site from the headquarters, and you pay per unit length of cable.

The figure below illustrates the situation, where  $H$  is the headquarters;  $a, b, c, d$  are the other sites; and the number over each edge is the cost for that edge. Four paths are shown to headquarters in various shades of black and color, one from each of  $a, b, c, d$ . Two edges are used twice, the rest once, for a total cost of

$$2 \cdot 2 + 2 \cdot 3 + 5 + 4 + 6 = 25.$$



a) Devise an algorithm for determining a minimum cost set of paths and show that it is correct. *Hint:* If you think of the goal in a different way, you will find that we already have an algorithm that we have proved correct.

b) Now assume that for each edge there is a fixed cost for building it plus a cost of the form in part a). This makes sense, say, if you must dig a trench for each segment between vertices and then again buy separate cable to connect each site to headquarters. Now there is no known efficient method to find the minimum cost solution. However, show that there is always a minimum solution where the edges used form a spanning tree.

c) The problem in b) has two special cases. The first is when the fixed cost is 0 on each edge. That case was treated in a). The second is when the variable cost (the  $c_{ij}$ 's) is 0 on each edge. This second case also has an efficient solution, and once again it is by an algorithm we have already studied. Explain.

26. (4) In the late 1800s, Tarry proposed the following method for exploring a labyrinth and getting out alive. (The labyrinth consists of chambers (vertices) and passageways (edges). You are standing at the entrance and have a piece of chalk.) The method is:

■ Step 1: If there is any unmarked passageway leading from the chamber (or entrance) where you are now, mark an  $\times$  at its start and go to the other end, where you make another  $\times$ . Then,

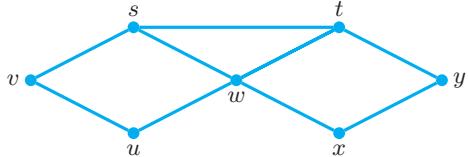
i) if no other passageway into the chamber you just entered is marked, also put an E next to your  $\times$ , but

ii) if there are marks on other passageways into that chamber, immediately go back to the other end of the passageway you just used; and repeat this step.

■ Step 2: If there are no unmarked passageways leading from where you are, then

- iii) if some passageway is marked with an E, move along that passage to its other end and go back to Step 1, but
- iv) if no passageway is marked with an E, stop.

a) Do a Tarry search of the following labyrinth starting at  $v$ . To help the grader, make a list of the edges (with direction traveled) in the order you traverse them. For instance, if you go from  $u$  to  $v$ , write  $(u, v)$ . If later you go from  $v$  to  $u$ , write  $(v, u)$  at that point in your list.



b) Tarry didn't call his method an algorithm because the word didn't exist then. But it is an algorithm. Write it up in our algorithmic language, preferably recursively.

c) It is a fact that, if the labyrinth is connected, Tarry's algorithm will cause you to explore every edge — in fact twice — and finish at the entrance. Prove this assertion.

d) How is Tarry's algorithm related to DFS?

27. ⟨3⟩ What is the least number  $m$  of vertices of degree 3 or more which a graph  $G$  must have before  $\chi(G) = 4$ ? You need to exhibit a  $G$  with  $\chi(G) = 4$  and with the number  $m$  of vertices of degree 3 or more, and you have to explain why any graph  $H$  with fewer vertices of degree  $\geq 3$  has  $\chi(H) \leq 3$ .

28. ⟨3⟩ Generalize your answer in [27] to  $k$ -colorable graphs, and give a proof by algorithm.

29. ⟨2⟩ [Computer project] Rewrite the algorithms of [27, Section 3.6], using arrays so that they can run in some actual computer language.

30. ⟨3⟩ Prove: For any graph  $G$ , there is some way to number the vertices so that Algorithm ONE-COLOR-AT-A-TIME [27, Section 3.6] colors  $G$  with the minimum possible number of colors.

31. ⟨2⟩ Consider the squares of a checkerboard as countries on a map. Then the usual coloring of a checkerboard proves that two colors suffice to color the graph derived from this map. Suppose that you have a small checkerboard (or large dominoes), so that a domino exactly covers two adjacent squares. Use the standard coloring of the board to prove that it is impossible to cover all but two opposing corner squares of a checkerboard using dominoes. (This problem shows that coloring is sometimes useful in contexts which don't seem to have anything to do with coloring.)

32. ⟨2⟩ A graph is called **color-critical** if the removal of any vertex reduces the chromatic number.

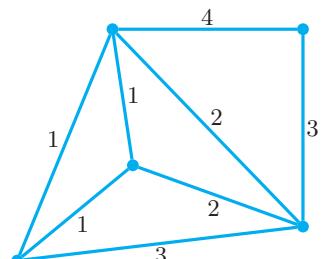
- a) Which cycles are color-critical?
- b) Show that if  $\chi(G) = n$  and  $G$  is color-critical, then every vertex of  $G$  has a degree at least  $n - 1$ .

33. ⟨2⟩ We call a graph **spherical** if we can draw it on the surface of a sphere without crossing edges. Explain why a graph is spherical if and only if it is planar. Hint: To get from spherical to planar, puncture the sphere within one of the faces of the graph.

34. ⟨2⟩ A graph is **toroidal** if it can be drawn on a doughnut without crossing edges. (Mathematicians call the shape of a doughnut a **torus**.) Show by construction that the complete graph with five vertices is toroidal. Note: The largest complete graph which is toroidal has seven vertices.

35. ⟨3⟩ When some edges have the same weight, a graph can have more than one minimum weight spanning tree.

- a) Display all minimum weight spanning trees for the accompanying graph. (Find them by any method you please, including brute force.)



b) What do you observe about the weights on all the minimum weight spanning trees?

c) Prove your observation in b).

36. (3) In cases of graphs with ties for one or more edge weights, Algorithm MINSPANTREE may be reinterpreted to say: When there is a tie for cheapest edge from the set of vertices connected so far to all other vertices, pick any one of the cheapest edges.

a) Prove: Every minimum weight spanning tree in such a graph can be obtained by some set of choices using this algorithm.

b) Prove: Every tree obtainable by making such choices is a minimum weight spanning tree.

37. (2) Devise an algorithm for finding the *second* lowest weight spanning tree in a graph.

Problems [38–44] concern induction. If the problem makes an Assertion and then presents a “Proof”, the induction may be invalid, although the Assertion may be true. For such problems, identify any errors in the proof, and provide a good proof if the assertion is true.

38. (3) A leaf in a free tree is a vertex of degree 1. In [31, Section 3.6], we asked you to prove that an acyclic graph had at least one vertex of degree 1. That proof shows that any free tree also has a node of degree 1, but now we ask you to prove this result by induction. It’s surprisingly tricky. (*Caution:* Don’t do a faulty buildup proof. *Suggestion:* Strengthen the hypothesis.)

39. (3) Fact: In chemistry, all alkanes have atomic formulas of the form  $C_nH_{2n+2}$ . See [21, Section 3.7] for the definition of alkane and a proof of this fact using counting methods.

A chemist once justified this fact to us as follows. Note that this argument (if valid) also shows that at least one alkane exists for each  $n \in N^+$ .

If you have one carbon, then you must have 4 hydrogens, and this fits the formula with  $n = 1$ . Now to get a bigger alkane you must remove some hydrogen atom and replace it with a carbon attached to 3 hydrogens. This gives a net gain of one carbon and two hydrogens, preserving the relationship of  $2n + 2$  hydrogens to  $n$  carbons. Repeat the process until you get whatever size alkane you want.

Is this proof valid? If not, can it be fixed?

40. (2) Assertion: Every connected graph contains a spanning tree.

“Proof”: Let  $P(n)$  be: Every connected graph with  $n$  vertices contains a spanning tree.

Basis step.  $P(1)$  is obvious: The vertex alone, without any edges, reaches all vertices without using any cycles.

Inductive step. Imagine that you are given a connected graph  $G$  on  $n + 1$  vertices. Temporarily remove any one vertex. By  $P(n)$ , the remaining graph has a spanning tree. Now consider vertex  $n + 1$  again. As  $G$  is connected, there must at least one edge from  $n + 1$  to another vertex. Pick any such edge and add it to the  $n$ -tree. It cannot create a cycle because it is the only edge we use which touches vertex  $n + 1$ . Thus we have found a spanning  $(n + 1)$ -tree.

41. (2) Assertion: Every tree is planar.

“Proof”: Let  $P(n)$  be the claim that any tree with  $n$  vertices is planar. Clearly  $P(1)$  and  $P(2)$  are true, for a single dot and a single edge can be drawn in the plane. Now for  $n > 2$ , assume  $P(n-1)$ . You can create an arbitrary tree on  $n$  vertices by taking any tree on  $n - 1$  vertices and putting an extra vertex in the middle of some edge; this doesn’t require redrawing the tree, so if it was planar before, it’s still planar. So  $P(n)$  is true.

42. (3) Assertion: Every tree with  $n$  vertices has  $n - 1$  edges.

“Proof”: Let  $P(n)$  be the assertion in the case  $n$ . The basis,  $P(1)$ , is obviously true. For the inductive step, take an arbitrary  $n$ -tree, create a new vertex and attach it anywhere by one new edge. By assumption the  $n$ -tree had  $n - 1$  edges, so an arbitrary  $(n+1)$ -tree has  $n$  edges.

43. (3) Suppose you want to prove some statement for all binary trees. Suppose you decide to prove it by induction on the number of vertices. In the inductive step, is it OK to show that, if an  $(n-1)$ -vertex binary tree  $T$  satisfies the statement, then so does the  $n$ -vertex binary tree obtained by adding a child to some leaf of  $T$ ? Explain.

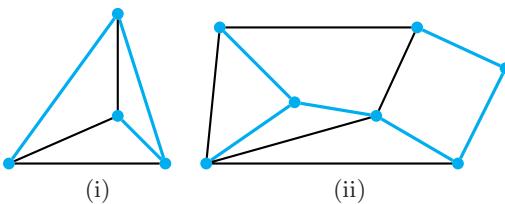
44. (2) Assertion: The number of labeled trees on  $n$  vertices is  $(n-1)!$ . (A **labeled tree** with  $n$  vertices is a free tree with the vertices distinguished by being numbered 1 through  $n$ .)

**“Proof”:** Let  $P(n)$  be the assertion for  $n$ .  $P(1)$  is clearly true:  $0! = 1$  and there is clearly only one labeled tree on one vertex. Now assume the claim is true for  $n$ . To get a labeled tree on  $n+1$  vertices we must take some labeled tree on  $n$  vertices and attach vertex  $n+1$  somewhere. There are  $(n-1)!$  labeled  $n$ -trees to start with, and in each case  $n$  previous vertices at which to attach. That gives  $(n-1)!n = [(n+1)-1]!$  labeled  $(n+1)$ -trees.

45. **(2)** Given a spanning tree  $T$  in a graph  $G$ , and any edge  $e \in G - T$ , there is exactly one cycle in  $T + e$ . The set of cycles obtained in this way, as you consider each edge  $e \in G - T$  in turn, is called the **fundamental system of cycles for  $T$** .

- Find the fundamental system of cycles in graph (i).
- Find the fundamental system of cycles in graph (ii).

The edges in  $T$  are shown in color. Such fundamental systems form a basis in the sense of linear algebra and are important in algebraic graph theory.



46. **(2)** Given a spanning tree  $T$  in a graph  $G$ , the deletion of any edge  $e \in T$  separates the vertices and remaining edges of  $T$  into two components. Let  $V_1$  and  $V_2$  be the vertex sets of these components. Consider the cut-set of  $G$  for  $(V_1, V_2)$ ; see [26, Section 3.1]. The set of cut-sets obtained in this way as you consider each edge  $e \in T$  in turn is called the **fundamental system of cut-sets for  $T$** .

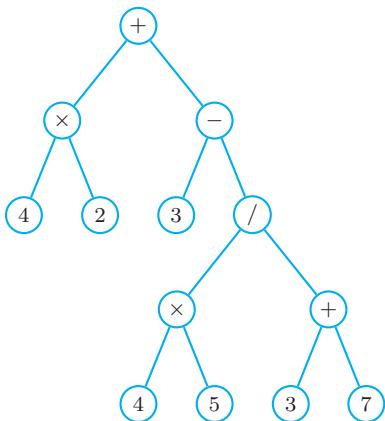
- Find the fundamental system of cut-sets for the spanning tree in graph (i), [45].
- Find the fundamental system of cut-sets for the spanning tree in graph (ii), [45].

47. **(3)** Consider the expression

$$(4 \times 2) + [3 - ((4 \times 5)/(3 + 7))]. \quad (1)$$

We can represent expressions such as (1) by binary trees, since each stage of the calculation consists

of doing a single arithmetic operation on two numbers. Here is the figure for (1):



Note that all the numbers are on leaves and all operations are on internal nodes. The left-to-right order of numbers and operators in (1) is exactly their order in an Inorder traversal of the tree. In this problem and the next we develop other ways to represent (1) by considering Preorder and Post-order traversals of the same tree.

- Evaluate expression (1) using this tree.
- Using the tree, write the numbers and operations in Postorder sequence. This representation is called **Reverse Polish Notation (RPN)**.
- It is a fact that RPN expressions are unambiguous without any parentheses. (RPN is the system available on some Hewlett-Packard calculators.) There is an unambiguous method for evaluating RPN expressions, as follows. Whenever an operator appears as you read from left to right, use that operator to combine the two immediately previous symbols in the current version of the sequence. (If the immediately previous two symbols are not both numbers, or there aren't two previous symbols, the algorithm aborts. But this can happen only if the Postorder order wasn't recorded properly.) As a result of the operation, the two numbers and the operator are now replaced by one symbol. Evaluate your expression from c) by this method, and check to see whether you get the same answer as in b).
- Prove (induction!) the fact stated in c).

48. **(4)** Consider again expression (1) in [47]:

a) Write expression (1) according to Preorder. This representation is called **Polish Prefix Notation (PPN)**. (PPN, like RPN, is unambiguous and parenthesis-free.)

b) Devise a rule for computing expressions in PPN form, assuming that you are allowed to read from right to left. Apply your method to the expression obtained in a).

c) Assume that you must read from left to right. Think of another rule for computing PPN expressions. Apply your method to the expression obtained in a).

d) Prove that your method in b) works, using induction.

e) Prove that your method in c) works, using induction.

f) Why isn't your method in c) of much use on calculators?

The next five problems are about game trees and are a continuation of [34–39, Section 3.7].

49. ⟨2⟩ The daisy game (for however many petals) is symmetric in that, wherever two nodes get label  $n$  (meaning  $n$  petals are left), the subtrees hanging from those nodes are the same. They are the same tree, and their nodes get assigned W and L the same as if we were to use the labeling scheme of [38, Section 3.7]. This approach allows us to condense the game tree into a smaller graph with

one node for each number of remaining petals. In particular, the graph  $G$  for five-petal daisy has six nodes, labeled 5, 4, 3, 2, 1, and 0. Draw this graph. Assign W and L by looking at this graph (*not by copying these labels from the original tree*). Explain briefly what you are doing.

50. ⟨2⟩ With the aid of [49], or otherwise, discover who wins  $n$ -petal daisy. Prove your conclusion by induction.

51. ⟨2⟩ Can you give a simple rule for optimal play of  $n$ -petal daisy?

52. ⟨3⟩ In the “misere” form of any game, the player who makes the last move *loses*.

a) Draw and analyze the game tree for the misere form of four-petal daisy.

b) Discover who wins  $n$ -petal misere daisy. Prove it by induction.

c) Repeat the proof in b) by reference to the result of [51].

d) Give a memoryless algorithm for optimal play in  $n$ -petal misere daisy. (See [51].)

53. ⟨2⟩ There are six sticks. A player must pick up 1, 2, or 3 sticks at each turn. The player who picks up the last stick wins. Determine who wins this game — if both players play optimally — by drawing the game tree and finding the value of the root.

# Fundamental Counting Methods

## 4.1 Introduction

---

You have already had several occasions in this book to count things. Indeed, number is one of humanity’s fundamental concepts, and counting is one of the fundamental intellectual urges. Ever tried to count the stars in the sky — or just the students in your math class? Ever wondered how many matching outfits you could put together from your wardrobe?

We will concentrate in this chapter on *how* to count without worrying too much about *what* we count. We realize that this approach can be taken to an unfortunate extreme. Who cares how many ways a baseball manager can order nine players to form a lineup, or an anagrammist can rearrange the letters of “Mississippi”? The manager would never consider all possible lineups to be equally worthy, and the anagrammist would never need to know how many ways the reordering could take place.

However, there *is* something to say for doing such “toy” examples. First, they are easy to state clearly and quickly, and so we can get on with the main business of learning solution methods. Second, they remind us that, in many situations where there is a lot of choice, the real issue is to find an optimal solution. Often the first step in solving optimization problems is to get an idea of how many choices there are. In an age of computers, if the number is not too large, brute force may be an effective algorithm — find the best choice by trying them all. Even with an ingenious algorithm, we need to know how many steps the algorithm takes in order to know if it is fast enough for practical use. But determining the number of steps is just another counting problem. The subject of the analysis of algorithms is full of counting problems. We mentioned earlier that the hardest part of an analysis is often the average case analysis. Average case analysis uses probability, and for the most part probability computations of this sort consist of doing two counting problems and taking the ratio. See Section 6.8.

Sections 4.2 and 4.3 treat the two fundamental rules on which other counting methods are based: the Sum and Product Rules. Section 4.4 introduces permutations and combinations. Many identities involve permutations and combinations, and Section 4.5 provides methods for deriving them. Section 4.6 is devoted to the single most important combinatorial identity: the Binomial Theorem. Section 4.7 explains how to count ways to place balls in urns and shows that many problems can be reformulated in these terms. Section 4.8 introduces generating functions, a use of polynomial functions to answer many counting problems in a methodical algebraic way. In Section 4.9, Combinatorial Algorithms, we illustrate that counting formulas have interesting algorithmic questions associated with them. You may not need to know how many ways there are to do something but instead may need to list them all or list a random selection. For such tasks you need a combinatorial algorithm. Section 4.10 treats the celebrated Pigeonhole Principle — it's about what happens when there are more pigeons than holes — but with an algorithmic twist.

## 4.2 First Examples: The Sum and Product Rules

---

We introduce the Sum and Product Rules through examples. Try to work the examples yourself before reading our solutions.

### EXAMPLE 1

In a certain computer system, a file name must be a string of letters and digits, 1 to 10 symbols long. The first symbol must be a letter and the computer makes no distinction between uppercase and lowercase letters in any position. Thus L3MAR and L3mar are both the same legitimate name, whereas 3Lmar is not legitimate. How many legitimate file names are there?

### EXAMPLE 2

At one time, auto license plates in New Hampshire consisted of two (uppercase) letters followed by three digits, e.g., RD 357 and SK 092. How many different cars could be registered in New Hampshire in those days?

### EXAMPLE 3

A salesman living in Washington, D.C., sells supplies to state governments. He decides to make a sales trip to all 50 state capitals to sell his wares. Assuming that he goes to one capital after another, visiting them all before returning home, in how many different orders can he make his trip?

### EXAMPLE 4

How many additions and how many multiplications of individual numbers are involved in multiplying two  $n \times n$  matrices? (See Section 0.5 for definitions.)

Here are our solutions.

**Solution to Example 1** A file name either has one symbol, or two symbols, or ..., or 10 symbols. In each case, we have 26 choices for the first symbol (the 26 letters of the alphabet). However, for each later symbol we have 36 choices because the 10 digits are also allowed. So the number of one-symbol names is 26,

and the number of two-symbol names is  $26 \times 36$ , because for each of the 26 possible first choices there are 36 second choices. Likewise, the number of 3-symbol names is  $26 \times 36 \times 36$ , because for each of the  $26 \times 36$  choices of the first two symbols there are 36 choices for the third. Continuing in this fashion, we get that the total number of names is

$$T = \sum_{k=1}^{10} 26 \times 36^{k-1}. \quad (1)$$

Using the formula for summing a geometric series (see [8, Section 2.2]) we find that

$$T = 26(36^{10} - 1)/35 \approx 2.7 \times 10^{15},$$

but Eq. (1) is sufficient for the purposes of this section. ■

**Solution to Example 2** All plates had exactly five symbols, so multiplying the number of allowed symbols for each position we obtain

$$26 \times 26 \times 10 \times 10 \times 10 = 676,000$$

possibilities. New Hampshire changed its system when the number of vehicles approached 676,000. First it switched to four digits after the two letters and later further arrangements of digits and letters were introduced. ■

**Solution to Example 3** Clearly, there are 50 choices for the first capital to visit. For each such choice there are 49 choices for the capital to visit next — any capital but the one already visited. Then there are 48 possibilities remaining for the third capital. This continues until just one choice remains — the last capital. Thus the total number of possible trip orders is

$$N = 50 \times 49 \times \cdots \times 3 \times 2 \times 1 = 50! \quad (2)$$

Now, this is just the sort of problem criticized in the introduction to this chapter. Even if there were such a salesperson, he or she wouldn't care how many possible sales trips there are. The significant question is: Which order is optimal? However, our counting problem is of use to this salesman in a negative sense. A simple mental calculation shows that the number of possible trips is astronomical: The first 41 factors of Eq. (2) are all  $\geq 10$ , so  $N \geq 10^{41}$ . (Or, a calculator shows that  $50! \approx 3 \times 10^{64}$ .) Even if we had a computer that could print out a *billion* trip schedules a second, it would still take over  $10^{32}$  seconds, or more than  $10^{24}$  years, just to print out the list. For comparison, the entire universe is believed to be less than  $10^{10}$  years old. Thus, even on the fastest imaginable sequential computer, or on the fastest and largest parallel processing computer, or even using a DNA computer [17], we are *never* going to solve this problem by brute force. We need a much more ingenious algorithm. In fact, although there *are* ingenious, fast algorithms that give near optimal answers to this problem (see the introduction to Section 4.5 for one approach), there is no known algorithm that runs in a less than astronomical number of steps and that is guaranteed to give *the best* solution. This famous problem is called the **Traveling Salesman Problem** or TSP. (This name

doesn't refer just to the problem of visiting U.S. state capitals, but to the general problem of finding the least distance order in which to visit  $n$  locations, starting and ending at the same one. Thus our state capitals example is an example with  $n = 51$  (50 states plus the District of Columbia). ■

**Solution to Example 4** Recall that if  $A$  and  $B$  are  $n \times n$  matrices, then so is  $C = AB$ , and the  $ij$  entry of  $C$  is defined by

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}.$$

This formula involves  $n$  products and  $n - 1$  additions. (To sum  $n$  numbers you add  $n - 1$  times.) Since all this must be done for each of the  $n^2$   $c_{ij}$ 's, we obtain  $n^3$  multiplications and  $n^2(n - 1)$  additions. ■

Note exactly what we have shown: *If* we evaluate a matrix multiplication directly from the definition, then it takes the number of steps we have claimed. Could there be another, nonobvious way to do the multiplication that takes fewer steps? It was long believed that the answer, for general matrices, was No. However, in 1969 it was shown that the answer is Yes. The improved methods involve Divide and Conquer recurrences of the sort discussed in Section 5.9. No one knows what the optimal algorithm is for multiplying  $n \times n$  matrices, but the best so far takes  $\text{Ord}(n^{2.376})$  steps.

Let us state two general principles used in these examples:

### The Sum Rule

If there are  $n(A)$  ways to do  $A$  and, disjoint from them,  $n(B)$  ways to do  $B$ , then the number of ways to do  $A$  or  $B$  is  $n(A) + n(B)$ . Similarly, one adds three terms when one must do  $A$  or  $B$  or  $C$ ; and so on.

### The Product Rule

If there are  $n(A)$  ways to do  $A$  and  $n(B)$  ways to do  $B$ , then the number of ways to do  $A$  and  $B$  is  $n(A) \times n(B)$ . It is assumed that  $A$  and  $B$  are **independent**; that is, the number of choices for  $B$  is the same regardless of which choice is taken for  $A$ . Similarly, one multiplies three terms if  $A$ ,  $B$ , and  $C$  are independent and one must do  $A$  and  $B$  and  $C$ ; and so on.

The Product Rule is closely related to the fact that if you want to find the number of entries in a rectangular table, you multiply the number of rows by the number of columns. In order to make a chart of all possible pairs of an  $A$  choice and a  $B$  choice, we could make a matrix with one row for each  $A$  choice and one column for each  $B$  choice, as in Fig. 4.1.

## FIGURE 4.1

A matrix showing the 12 possible pairs of choices for 3  $A$  values and 4  $B$  values.

| $A$ | $B$ | 1     | 2     | 3     | 4     |
|-----|-----|-------|-------|-------|-------|
| $p$ |     | $p_1$ | $p_2$ | $p_3$ | $p_4$ |
| $q$ |     | $q_1$ | $q_2$ | $q_3$ | $q_4$ |
| $r$ |     | $r_1$ | $r_2$ | $r_3$ | $r_4$ |

How have we used these rules? In Example 1, our file name was either one symbol *or* two symbols *or* three symbols, . . . , so we added the number of file names in each of these cases. But within each case we had *and* decisions. For instance, for the two-symbol names, we had to pick a first symbol *and* a second symbol, independently. Analyzing still more closely, in choosing the second symbol we had to make an *or* decision: We either picked a letter or a digit, for  $26 + 10$  choices.

In Example 3, *which* choices are available to the traveling salesman for the second city depends on what is chosen first, but the *number* of second choices is independent of the first choice. Furthermore, the salesman has to make a first choice *and* a second choice, and so on. Thus we have used the product rule.

In the matrix multiplication example, we first counted the number of operations required to compute each entry of the product matrix. This is  $n$  multiplications and  $n - 1$  additions. Then since there are  $n^2$  entries in  $C$ , the product rule gives us  $n^3$  multiplications and  $n^2(n-1)$  additions.

We leave a similar analysis of Example 2 to problem [1].

Although the sum and product rules should seem quite straightforward, it is easy to get confused in a problem with a lot of steps within steps. At each stage of analysis, consciously ask yourself if you are “and-ing” or “or-ing”.

## Problems: Section 4.2

1. **(1)** Explain how the Sum and/or Product Rules were used to solve Example 2.

In problems [2]–[9] indicate as part of your solution where you use the Sum and/or Product Rules.

2. **(1)** How many entries are in a  $7 \times 3$  matrix?
3. **(3)** In a Chinese restaurant, you pick a soup, an appetizer, a main dish, and a dessert.
- a) If you have a choice of 5 soups, 6 appetizers, 20 main dishes, and 3 desserts, how many different meals could you arrange?
- b) Actually, you don’t have to have anything except a main dish. Now how many meals can you arrange?

4. **(1)** In another restaurant, for the main dish there are 10 meat, 5 fowl, and 8 fish choices. How many ways are there to pick a main dish?
5. **(1)** In a certain summer school program, each student must take three courses. One must be in History, Philosophy, or Religion. One must be in Math or Science. One must be in Literature, Fine Arts, or Music. Each of these departments offers three courses. How many possible programs are there for a student?
6. **(2)** In each of the following cases, how many 3-letter “words” are there, using lowercase letters, if
- a) any letter can be used in any position;
- b) the letters must be distinct;

- c) the first and last letters must be consonants and the middle letter a vowel (a, e, i, o, u);
- d) the first or last letter must be a consonant and the other two letters must be a double vowel (aa, ee, etc.)?

7. ⟨2⟩ How many 3-letter “words” are there, using letters, if

- a) the first letter may be uppercase or lowercase and the other two must be lowercase;
- b) same as part a), except the letters must be distinct (consider the uppercase and lowercase of the same letter to be distinct).

8. ⟨1⟩ How many batting lineups of 9 baseball players are there,

- a) given exactly 9 players to start with; or instead
- b) given a roster of 20 players to start with?

9. ⟨1⟩ How many integers are there from 1 to 999 inclusive? The answer is obvious, but practice using the Sum and Product Rules by solving as follows. Each integer has either 1, 2, or 3 digits. In each case, how many choices are there for the left-most digit? For later digits? Finish the problem.

10. ⟨1⟩ Do [9] again, devising a shorter method that avoids using the Sum Rule for the number of digits. *Hint:* Allow an integer to have leading zeros.

11. ⟨1⟩ At Washington High School, 35 students play on the football team and 15 play on the basketball team. Can you determine how many ways there are to pick one person from the football and basketball team members? If you can't, explain why the Sum Rule does not apply.

12. ⟨2⟩ How many times does each of the following algorithm fragments “Do Something”? Assume that  $n \geq 3$ .

a) **for**  $i = 1$  **to**  $n$   
    **for**  $j = 1$  **to**  $n$   
        Do Something  
    **endfor**  
**endfor**

b) **for**  $i = 1$  **to**  $n$   
    **for**  $j = 1$  **to**  $n$   
        **if**  $i \neq j$  **then** Do Something  
    **endfor**  
**endfor**

c) **for**  $i = 1$  **to**  $n$   
    **for**  $j = 1$  **to**  $i$   
        Do Something  
    **endfor**  
**endfor**

d) **for**  $i = 1$  **to**  $n$   
    **for**  $j = i+1$  **to**  $n$   
        Do Something  
    **endfor**  
**endfor**

13. ⟨1⟩ In many states auto license plates consist of three capital letters followed by three numerals. Are there any states in which this is probably not enough (even if discarded plate numbers can be reused)? Are there any states in which using both three letters followed by three numerals and three numerals followed by three letters is probably not enough? How many license plates are possible where you live?

14. ⟨2⟩ Before a company puts its computers to work solving an optimization problem (such as the Traveling Salesman Problem), it wants to know whether the cost of obtaining the solution is worth it in terms of the potential savings from implementing the solution. Suppose the only way of solving the Traveling Salesman Problem is brute force. That is, each of the  $n!$  routes must be listed separately, the cost for each one must be computed, and the smallest cost among them must be singled out. Assume further that travel cost is proportional to distance, so that it suffices to find the minimum distance route. Assume that a computer can compute the total distances, and compare each with the minimum distance found thus far, for a million traveling-salesman routes each second; that it costs 1 cent per second to do this, and that a company is prepared to spend \$10,000 to find the optimal route for its salesman. What is the maximum number of cities the salesman can visit on his trip (in addition to his home town) in order for this budget to be sufficient to solve the problem on the computer by brute force?

15. ⟨2⟩ The analysis in [14] is not very refined. How long it takes the computer to find the total distance for a proposed route depends, at the least, on how many links are in the route. Suppose each of the following steps is a basic computer operation: looking up one link distance, adding two numbers, and comparing two numbers to find

which is larger. Suppose that anything else the computer has to do, such as figure out which route to analyze next, takes no time at all. If the computer can do a million basic operations per second, and the financial data of [14] still holds, now what is the maximum number of cities for a brute force solution? (*Note:* Every tour to  $n$  cities in addition to the salesman's home town involves  $n + 1$  links.)

16. ⟨2⟩ It's intuitively clear that, having just visited an East Coast capital, we will not find an optimal Traveling Salesman tour by jumping immediately to a West Coast capital. On the contrary, we would expect to go to the capital of an adjacent state. Thus, after the first capital, we need not really consider 49 choices, but only a few. Assume that after each state is visited, there are really only three other unvisited states that we should consider for the next visit. Do [14, 15] again, using this (plausible but not always correct) assumption.

17. ⟨2⟩ DNA computers are a form of massively parallel computing with very small but slow individual processors — little pieces of DNA. These DNA pieces do the computing by doing their thing, building molecules. The trick (mostly unsolved) is to control the building process and to make certain sorts of molecules correspond to desired solutions.

Suppose a DNA computer has  $10^{15}$  DNA processors (that's roughly the number of cells in a human body) and each computes and compares the length of one Traveling Salesman route each second. Suppose, as in [14], that it costs 1 cent per second to run the computer, and that a company is prepared to spend \$10,000 to find the optimal route for its salesman. What is the maximum number of cities the salesman can visit on his trip (in addition to his home town) in order for this budget to be sufficient?

Telephone numbers have gone through several changes in North America in the last two decades due to the vast increase in the number of phones and other machines using phone numbers and in the number of phone companies vying for blocks of numbers. Whether a proposed change will work (provide enough numbers) reduces to a counting problem. The following problems ask you to do these counts for several of the systems that have been used. (The details have been simplified.) The key thing to understand that explains

the various rules is this: North American phone company equipment interprets dialed digits sequentially, without waiting to determine *how many* digits you dial. So if your first three digits are 741, how does the phone company know if this is an area code or just an exchange (the first three digits of a local number)? Only if 741 is not allowed as an area code, or not allowed to be an exchange, or there is some previous information (say, the number 1) that announces when an area code is coming.

18. ⟨1⟩ In the 1940s, the US and Canada were divided into “area code regions”. Within an area code region, phone numbers are 7 digits, each 0 through 9, except that the first digit cannot be 0 or 1. How many phone numbers are there within an area code region?
19. ⟨1⟩ To call to another area code region, you dial 1, followed by the area code of that region, followed by the number within the region. Area codes are 3 digits. Originally, the first digit of an area code could be 2 through 9, the second digit had to be 0 or 1, and the third digit was again 2 through 9, except it could also be 1 if the middle digit was 0. How many original area codes were there?
20. ⟨2⟩ Problem [18] didn't tell the whole story; from the beginning there were further restrictions on local 7-digit phone numbers. The first 3 digits could not be an area code, nor could they be any number in which both the 2nd and 3rd digits were either 0 or 1.
- a) How many phone numbers were there within an area code region?
  - b) Area codes covered the US and Canada. If every person in this coverage area had just one phone and there were no business phones or fax machines, would there be enough numbers today under the original rules?

Why the additional restrictions on the first 3 digits of local 7-digit numbers? We explained above why they could not be area codes. They could also not be certain other reserved numbers, like 411 for information. Finally, everyone had a “local calling area” (much smaller than an area code region), and every phone call outside that local area had to be preceded by a 1, whether it was to another area code or not. This explains why the first of the seven digits could not be a 1. It could not be a 0 because 0 is for calling the operator.

- 21.** *(1)* In 2004, any call to another area code must be preceded by 1, but any call within an area code region must not be. Any 3 digits may be an area code, so long as they don't start with 0 or 1. Likewise, any 7 digits may be a number within an area code region, so long as they don't start with 0 or 1. How many phone numbers are available under this system?
- 22.** *(2)* Here is a measure of efficiency of a phone number system. Suppose the system uses  $d$  digits and allows for  $N$  different phone numbers. Then the efficiency is  $(\log_{10} N)/d$ .
- Why is this a sensible definition of efficiency?
  - If every 7-digit number from 000–0000 to 999–9999 were allowed, what would be the efficiency of the 7-digit system?
  - Show that, for any decimal phone-numbering system with  $d$  digits, the efficiency is  $\leq 1$ .
  - Suppose that a  $d$ -digit system allowed 8 choices at each position. Show that the efficiency would be  $\log_{10} 8$ .
  - What is the efficiency of the current American system within an area code region (from [21])?
  - What is the efficiency of the 11-digit American system? (11 digits because each area code must be preceded by 1.)

Problems [23–25] concern straightforward arithmetic procedures that involve both single multiplications and single additions. For instance,  $ab+ac$  involves two multiplications and one addition. Or, to multiply  $23 \times 8$  by hand, you do two single-digit multiplications ( $3 \times 8$  then  $2 \times 8$ ) and an addition (to get the 10s place in the product). In these problems you will see that it makes a difference how you do these procedures. However, for simplicity, *ignore the additions*. Just count the multiplications.

- 23.** *(1)* The distributive law says  $\sum_{k=1}^n ab_k = a \sum_{k=1}^n b_k$ . However, the number of multiplications on the two sides are quite different. Count them.
- 24.** *(2)* Consider multiplying an  $m$ -digit number  $M$  by an  $n$ -digit number  $N$  using the traditional paper-and-pencil method.
- Explain why there are exactly  $mn$  single-digit multiplications. (Every product of two digits should be counted as one multiplication, even if it is really simple like  $2 \times 1$  of  $7 \times 0$ .)
  - Explain why  $MN$  has at most  $m+n$  digits and at least  $m+n-1$  digits. (As usual, we don't write leading 0's.)
  - Suppose further that  $P$  is a  $p$ -digit number. At most how many single-digit multiplications are involved in computing  $(MN)P$ ? In computing  $M(NP)$ ? In terms of  $m, n, p$ , when is it more efficient to compute  $(MN)P$  than  $M(NP)$ ?
- 25.** *(2)* Matrix multiplication is associative:  $A(BC) = (AB)C$ . Count the number of multiplications of matrix entries on each side
- When all three matrices are  $n \times n$ .
  - When  $A$  and  $B$  are  $n \times n$  and  $C$  is an  $n \times 1$  column vector.
  - In the general case:  $A$  is  $p \times q$ ,  $B$  is  $q \times r$ , and  $C$  is  $r \times s$ . In terms of  $p, q, r, s$ , determine when the left-hand side takes fewer multiplications.
- 26.** *(3)* Developing new drugs is today part data mining. A single company may have created a million compounds. When they realize that some specific property is needed (say, binding to a specific site on a bloodstream molecule), they test their compounds. If, say, 0.1% of them seem to bind well, they proceed with further rigorous testing of these in hopes that one or two will lead to effective, safe drugs.

Testing can be done by robots, which can test 44 compounds at once in a rectangular “matrix”, in tests lasting, say, 3 hours. However, this is still slow and expensive when there are a million compounds to try.

Here is another approach. Let the robots test a sample, say 10,000 compounds, and then use a computer algorithm that computes chemical properties, comparing this sample to all the compounds to get some idea which other compounds are worth testing. (Neural network algorithms have been used for this purpose.)

Assume that

- Robots cost \$500 per hour to run.
- 0.1% of all compounds in the company's collection would prove “promising” (worth further testing).
- The computer analyzes 100 compounds per second and eventually picks out 1/10 of the remaining compounds for robot testing.
- Computing costs \$100 per hour.

- 0.5% of the compounds that the computer chooses turn out to be promising when tested by the robots.
  - Each promising compound is worth \$10,000 (in the sense that 1 of 1000 of them will result in a drug that earns the company \$10 million over the further costs of testing.)
- a) Which is more cost effective, testing all compounds with robots or using the computer to reduce the number of robot tests? (The computer method saves time and money but also leads to missing many promising compounds.) Is either method profitable for the company?
- b) Same questions if only 0.3% of the computer identified compounds turn out to be promising.

## 4.3 Subtler Examples and Further Rules

You may have concluded from the straightforward examples in the previous section that nothing very interesting could be computed using the Sum and Product Rules. We hope the examples in this section will convince you otherwise. We'll also introduce two further rules, the Division Rule, and a more general Sum Rule.

### EXAMPLE 1

If there are  $n$  Senators on a committee, in how many ways can a subcommittee be formed? Also, in how many ways can the full committee split into two sides on an issue?

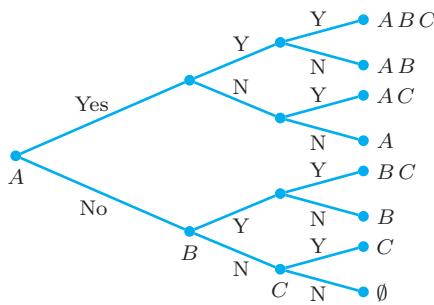
**Solution** The way most people naturally think of answering the first question is to count the number of subcommittees with one member (there are  $n$ ), then the number of subcommittees with two members (it turns out there are  $n(n-1)/2$  of them), then the number with three members (getting harder!) and so on, and then add up all these counts (Sum Rule). It *can* be done this way, and we'll do it this way in Section 4.4, but there is a much simpler way — using just the Product Rule. Don't first decide the size of the subcommittee; instead consider each Senator in turn and decide whether he or she will be on the subcommittee. Thus there are two possibilities, Yes or No, decided independently for each of the  $n$  Senators. This approach gives a product of 2 times itself  $n$  times, that is,  $2^n$  subcommittees.

Figure 4.2 illustrates this approach when the full committee has three members. Yet another use for trees! Here they are called **decision trees**. Each path from the root to a leaf represents a sequence of possible Yes/No membership decisions for all the members of the full committee. The corresponding subcommittee is recorded at the leaf. Note that at each level (which corresponds to the membership decision for a particular full-committee member) the number of nodes doubles, just as the Product Rule says it should.

We have to be a little careful, though, about our simple solution. When we considered subcommittee size first, we started counting at size 1. However, the product method includes in its count one subcommittee with zero members (see Fig. 4.2). It also includes one subcommittee with all  $n$  members. Do we really want to include the “null” subcommittee and the subcommittee of the whole? This is a matter of definition. For instance, if Congress refuses to allow null subcommittees, but does allow committees of the whole, the correct answer to the first question in Example 1 is  $2^n - 1$ .

**FIGURE 4.2**

All possible subcommittees of a three-member committee.



*Note 1.* Null committees do occur in the real world. If, say, Congress wants the President to set up a new Commission (i.e., Committee) to oversee something and s/he doesn't want to, s/he might "forget" to appoint the committee. That is, s/he appoints a null committee.

*Note 2.* If we allow both null and whole subcommittees, Example 1 is simply asking: How many subsets does an  $n$ -set have? So this is not the first time we have discussed the problem; see Section 0.1 and [28, Section 2.2]. However, the Product Rule solution given here is new.

The second question in Example 1, about splitting on an issue, is even more ambiguous. For instance, suppose the issue is a motion and the full committee consists of Senators  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $E$ . One way for them to split is to have  $A$ ,  $B$ , and  $C$  vote Yes, and  $D$  and  $E$  vote No. Another way is to have  $D$  and  $E$  vote Yes and  $A$ ,  $B$ , and  $C$  vote No. Are these different splits? The same people are in agreement and the same ones are in opposition in both cases. Indeed, the first split becomes the second if the motion is reworded in the negative.

Again, this is a matter of definition. If you are interested in who voted Yes, the two splits above are different; if you are interested only in the pattern of opposition, they are the same. Since we have two interpretations, let's answer both. Interpretation one: Who votes Yes makes a difference. Then each Senator either votes Yes or No, giving  $n$  2-way choices and a product of  $2^n$  splits.

With this interpretation, the answer is the same as the number of subcommittees. This doesn't sound like happenstance, does it? Indeed, for each Yes/No split, think of those who voted Yes as forming a subcommittee (the Yes-Subcommittee) and just don't think about the others. (*Note:* Since everybody could vote Yes, or nobody could do so, we definitely must allow the full subcommittee and the null subcommittee in this context.) Conversely, for each subcommittee, imagine that all the subcommittee members have voted Yes on some issue, and all the others on the full committee have voted No. Thus we have a **one-to-one correspondence** between the objects counted in the first problem (subcommittees) and the objects counted in the second (Yes/No splits). Looking for such correspondences is a very powerful method in combinatorics. If you find a one-to-one correspondence between the counting problem before you and some other problem you have already solved, then you have the answer to the new problem without a single additional

computation. Even if the second problem is not one you have already solved, it may be one you can solve more easily than the first.

But what is the number of split votes for the second interpretation, where all you are interested in is the pattern of opposition? For each split in which the actual vote doesn't matter, there are two splits in which it does matter — one side votes Yes and the other No, or vice versa. Moreover, every Yes/No split corresponds to a vote-doesn't-matter split in this two-to-one fashion. Therefore the number of doesn't-matter splits is  $2^n/2 = 2^{n-1}$ . This is our first instance of the **Division Rule**. We'll do another example shortly before we try to state the rule more formally. ■

*Dealing with ambiguity.* Example 1 brings out an important point about counting problems. It is very hard in a brief description to make such a problem completely unambiguous. So what do you do when the meaning of a problem (on a test, say) is unclear to you? Our suggestion:

- state that you think the problem is ambiguous;
- state unambiguously what you think is the most likely interpretation; and
- then answer that interpretation correctly.

*Caution 1:* If the interpretation you choose makes the problem trivial to answer, it is probably not the interpretation the professor intended, and you probably won't get much credit for solving it. *Caution 2:* Not all professors are willing to believe that their wonderfully worded questions could be ambiguous. If the question uses only terms that the professor has taken great pains to define precisely in class, the professor may be right. But if you are honestly confused, we still think our approach is the best one. Even professors (other ones!) make mistakes.

## EXAMPLE 2

Find a lower bound on the number of digits the burglar in Example 2, Section 3.1, must punch sequentially in order to try all 4-digit combinations.

**Solution** The thief tries a sequence of digits. Each digit tried becomes the final digit of at most one attempted combination, the digit just tried and its three predecessors if they exist.. But there are  $10^4$  possible combinations (Product Rule), so the thief must try at least  $10^4$  digits in a row.

Again we have used the idea of a correspondence, but here we have an inequality, not an equality. We said the final digit of *at most* one possible combination. After the  $k$ th attempted digit, for  $k \geq 4$ ,  $k - 3$  four digit sequences have been tried. Therefore in order to cover all  $10^4$  combinations, the thief must try at least  $10^4 + 3$  digits. In fact, this number of digits is enough; see [6, Section 3.3]. Therefore  $10^4 + 3$  is the optimal solution. ■

## EXAMPLE 3

Find an upper bound on the number of steps in Towers of Hanoi.

**Solution** Forget for the moment the main problem of how to get from one configuration of Towers of Hanoi to another. Just ask: How many legitimate configurations

of rings on poles are there? If it turns out that there are, say, 37 configurations, then at most 36 steps are needed to solve the puzzle (if it can be solved at all). In other words, the worst that could happen is that you have to pass through every configuration to get to the desired end, since you certainly never have to pass through the same configuration more than once. (Why?)

To see this, you might think in terms of a state graph as described in Section 3.1. The different legitimate configurations are the vertices, and you seek a path between two specific vertices, the start configuration vertex and the end configuration vertex. The worst that could happen would be for the graph to be a single, simple path with those two vertices at the ends.

So, how many configurations are there? First, you only have to worry about which rings are on which poles; the order of the rings on each pole is uniquely determined as largest to smallest. So you merely have to divide the  $n$  rings up among the poles. That is, each ring merely has to “vote”, independently, for pole 1, 2, or 3. This gives  $3^n$  configurations and an upper bound of  $3^n - 1$  moves.

Not very impressive, you say, since the actual answer,  $2^n - 1$ , is so much less than this bound. But this bound applies to *any* version of TOH in which

- only one piece can move at a time;
- smaller pieces must always be on top of larger pieces; and
- any further restriction on moves depends solely on the current configuration, not on past moves. (Where have we used this last condition?)

Indeed, there is a natural modification of TOH in which the bound we have obtained is exact — Straightline TOH from one endpole to the other [6, Section 2.4]. ■

## EXAMPLE 4

You have 13 identical-looking coins and a balance scale. You are told that one of the coins is fake and has a different weight than the others. You are asked to find which coin is fake, and whether it is light or heavy, using only the balance scale. Find a good lower bound on how many weighings this will take. (A weighing is one use of the scale, e.g., weighing coins 1 and 2 against 3 and 13 and finding which pair, if either, is heavier.)

**Solution** If you have tackled this coin problem before, you know that determining how to find the fake coin in a small number of steps is hard. But here we asked only for a lower bound on the number of steps, and a very good lower bound can be obtained by a simple Product Rule calculation.

The hard part of the complete problem is to devise a successful *strategy*. You must decide which coins to weigh first, and then, for each possible result (i.e., balance, right sinks, left sinks), you must decide in advance which coins you will weigh next depending on the outcome of the previous weighing. Such a strategy involves another decision tree. Fig. 4.3 shows a successful strategy for three coins.

Define the “situation” to mean the (initially unknown) information about which coin is fake and whether it is heavier or lighter.

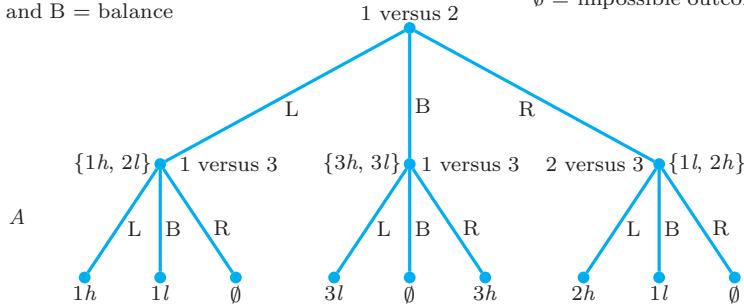
For any given situation, each weighing will cause you to go down one more edge on a particular path in your strategy tree. Suppose that two different situations

A versus B means A on left pan  
and B on right pan.  
L = left sinks, R = right sinks,  
and B = balance

$ih$  = coin  $i$  heavy  
 $il$  = coin  $i$  light  
 $\emptyset$  = impossible outcome

**FIGURE 4.3**

A decision tree for finding one fake coin from three with a pan balance. Each node is marked with the set of situations leading to it.



would lead you down the same path. Then you won't be able to distinguish between them; they give you exactly the same result. In other words, a strategy is successful iff each leaf corresponds to at most one possible situation. A simple *necessary* condition, therefore, for a successful strategy is that there be at least as many leaves as situations. Also, the maximum number of edges leading from the root to a leaf in a successful strategy tree is the number of weighings that strategy requires in the worst case.

In Fig. 4.3, we have marked each node by the set of situations that lead to it. Since each leaf has at most one situation, the strategy is successful. However, if we lopped off the tree after one weighing, the strategy would not be successful. Also note that every leaf is at the same level; thus the worst case and the best case are the same.

So where's the Product Rule? Figuring out your strategy is hard, but the *shape* of the strategy tree is always the same: It's a **ternary** tree. Each weighing has three possible outcomes: The left pan sinks, the right pan sinks, or they balance. Thus a strategy with one weighing has 3 leaves, a strategy with at most two weighings has at most 9 leaves, and, in general, with  $n$  weighings there can be  $3^n$  leaves.

In our specific case with 13 coins, there are  $13 \times 2$  situations to distinguish; each coin may be heavy or light. Thus a successful strategy tree must have at least 26 leaves. Since  $3^2 = 9$  and  $3^3 = 27$ , you should conclude that the very best strategy will need at least three weighings in the worst case. This is the bound you sought. But it doesn't necessarily imply that there is a successful strategy with three weighings.

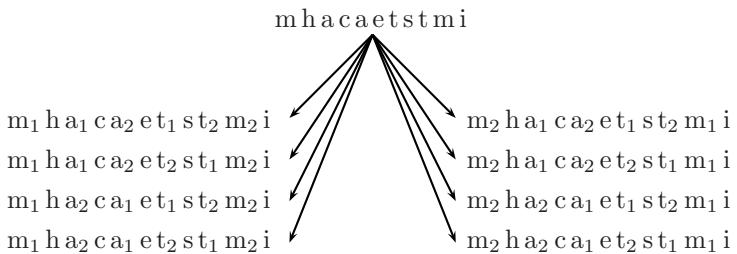
Indeed, with more refined counting, you can show that at least four weighings are necessary and, by actually devising strategies, you can then show that four are sufficient. See [15–18]. ■

## EXAMPLE 5

In how many distinguishable ways can the letters of “computer” be arranged? In how many distinguishable ways can the letters of “mathematics” be arranged?

**Solution** There are 8 letters in “computer”, all distinct. Any one of them can go first, then any of the remaining 7, and so on. So the answer is  $8! = 40,320$ .

The letters of “mathematics”, on the other hand, are not all distinct. There are two m’s, two a’s and two t’s. But let us try to reduce this problem to the previous case. Suppose all the letters *were* different — say, the m’s were labeled  $m_1, m_2$ , and so on. Then the answer would be  $11!$ . Clearly we have overcounted, but how much? For each arrangement with copies of the same letter indistinguishable, there are  $8 (= 2 \times 2 \times 2)$  arrangements with the copies distinguished: there are  $2 = 2 \times 1$  ways to put the subscripted m’s in the two m slots, independently 2 ways to order the subscripted a’s, and the same for the t’s. See Fig. 4.4. The correct answer is  $11!/8 = 4,989,600$ . ■



For each arrangement of the letters in “mathematics” with copies of the same letter indistinguishable, there are eight arrangements with distinguishable copies.

This then is the

---

### Division Rule

If there is a *k*-to-1 correspondence of objects of type *A* with objects of type *B*, and there are  $n(A)$  objects of type *A*, then the number of objects of type *B* is  $n(A)/k$ .

---

A ***k*-to-1 correspondence** from *A*-objects to *B*-objects is an onto mapping in which every *B*-object is the image of exactly *k* *A*-objects. See Fig. 4.5. This is a natural generalization of the concept of 1-to-1 correspondence discussed in Example 1.

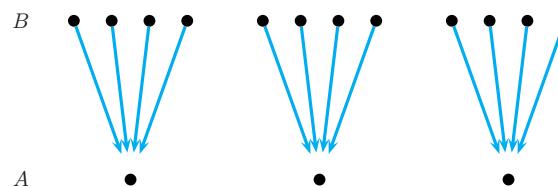
The Division Rule is commonly used for counting ordered arrangements when some of the objects are indistinguishable (e.g., the m’s in Example 5). It is also commonly used when order doesn’t matter; often it’s easiest to first suppose order does matter and then divide out [31–32].

*Caution:* The Division Rule works only if every object you want to count corresponds to *exactly k* objects of the comparison set. It’s easy to think that you have an exact *k*-to-1 correspondence when you don’t, as the next example shows.

**FIGURE 4.4**

## FIGURE 4.5

A 4-to-1 correspondence between  $A$ -objects and  $B$ -objects.



## EXAMPLE 6

Count the number of 0–1 sequences of length  $n$  when a sequence and its reverse are considered to be the same. For instance, if  $n = 4$ , then 1010 and 0101 should together be counted just once.

This problem is not so silly. Think about the Universal Product Code made of bar lines. Electronic scanners can read this code backwards or forwards, so a code word and its reverse cannot represent different things.

**Solution Start** It seems pretty clear that we can take the total number of binary sequences of length  $n$  and divide by 2, for an answer of  $2^n/2 = 2^{n-1}$ . However, this answer isn't right! To take a very simple example, when  $n = 2$  this formula gives 2, yet the right answer is 3:

$$00, \quad 01, \quad 11.$$

What went wrong? What's the right formula? [23–24] ■

In general, counting problems that involve a variable many-to-one correspondence are quite tricky. There is a general result called Polya's Theorem that helps when there is a lot of symmetry. This theorem is treated in a more advanced combinatorics course.

## Sums with Overlaps (Inclusion-Exclusion)

## EXAMPLE 7

At Washington High School, 35 students play in the band, and 15 students are on the math team. How many students are in at least one of these activities?

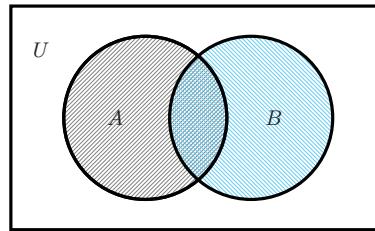
**Solution** This would be a candidate for the Sum Rule, except that the Sum Rule rule requires the two things being counted to be disjoint. However, a student could certainly be both in the band and on the math team. This situation is illustrated by Fig. 4.6. Think of the set of students at the school as represented by points within the box, and the set of students in the band or on the math team as represented by points within the appropriate circle. This is a Venn diagram, a type of figure introduced in Section 0.1.

Clearly, if we simply add  $35 + 15$ , we double-count the students who do both. If, say, 4 students do both, then the correct answer is  $35 + 15 - 4 = 46$ . ■

The principle here is easily formalized using set notation. As in the figure, let  $A$  be the set of band members and  $B$  the set of math team members. With  $|A|$

## FIGURE 4.6

A Venn diagram of the Washington High School student body:  
 $U$  = students;  
 $A$  = band;  
 $B$  = math team.



representing the cardinality of  $A$  as in Section 0.1, we seek  $|A \cup B|$ , and

$$|A \cup B| = |A| + |B| - |A \cap B|. \quad (1)$$

Note the difference from the notation in the Sum Rule on p. 324. There  $A$  was an activity, so we used  $n(A)$  for the number of ways to do  $A$ . Now  $A$  is a set, so we can use the standard notation  $|A|$  for the number of things in  $A$ .

### EXAMPLE 8

How many integers from 1 to 100 are divisible by 2 or 5?

**Solution** This is like Example 7 except that now  $A \cap B$  is determined by the information in the problem. Specifically, we seek  $|A \cup B|$  where

$$\begin{aligned} U &= \{1, 2, \dots, 100\}, \\ A &= \{n \in U \mid 2 \text{ divides } n\}, \\ B &= \{n \in U \mid 5 \text{ divides } n\}. \end{aligned}$$

Since 2 divides a number iff the number is a multiple of 2, then  $|A| = 100/2$ . Likewise,  $|B| = 100/5$ . Since  $A \cap B$  is the set of integers  $\leq 100$  that are divisible by both 2 and 5, i.e., divisible by 10, it follows that  $|A \cap B| = 100/10$ . Thus by Eq. (1), the answer is

$$50 + 20 - 10 = 60. \blacksquare$$

### EXAMPLE 9

Two integers are **relatively prime** if they have no factors in common, other than 1. How many positive integers  $\leq 100$  are relatively prime to 100?

**Solution** An integer is relatively prime to  $100 = 2^2 5^2$  if it doesn't have 2 or 5 as a factor. Therefore, continuing with the notation of Example 8, the answer is the number of integers in  $U$  that were *not* just counted, in other words,

$$|U| - (|A| + |B| - |A \cap B|) = 100 - 60 = 40.$$

Note that the expression on the left in this display is general: whenever we want the number of elements in a universe  $U$  that are not in  $A \cup B$ , the LHS provides the answer. ■

### EXAMPLE 10

How many integers from 1 to 100 are divisible by 2 or 3 or 5?

**Solution** Now the Venn diagram is Fig. 4.7, where  $C = \{n \in U \mid 3 \text{ divides } n\}$ . We seek the total number of integers within the three circles, that is,  $|A \cup B \cup C|$ . Let's begin by adding  $|A|$ ,  $|B|$  and  $|C|$ . This counts the integers divisible by more than one of 2, 3, 5 twice or more. (Which numbers are counted exactly twice? Which more?) So let's subtract  $|A \cap B|$ ,  $|A \cap C|$ , and  $|B \cap C|$  once each. Now the integers divisible by exactly two of these primes are also counted just once, but the integers that are divisible by all three primes have been counted  $3 - 3 = 0$  times. (Why?) So let's add back  $|A \cap B \cap C|$ . We get

$$\begin{aligned} |A \cup B \cup C| &= |A| + |B| + |C| - |A \cap B| \\ &\quad - |A \cap C| - |B \cap C| + |A \cap B \cap C|. \end{aligned} \tag{2}$$

We already computed  $|A \cap B|$  in Example 8. We compute the other terms as follows:

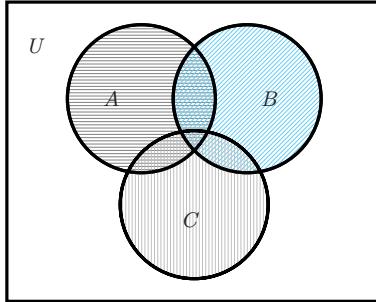
$$\begin{aligned} |C| &= \lfloor \frac{100}{3} \rfloor = 33, \\ |A \cap C| &= |\{n \in U \mid 6 \text{ divides } n\}| = \lfloor \frac{100}{6} \rfloor = 16, \\ |B \cap C| &= |\{n \in U \mid 15 \text{ divides } n\}| = \lfloor \frac{100}{15} \rfloor = 6, \\ |A \cap B \cap C| &= |\{n \in U \mid 30 \text{ divides } n\}| = \lfloor \frac{100}{30} \rfloor = 3. \end{aligned}$$

Therefore, from (2) we conclude that the answer is

$$50 + 20 + 33 - 10 - 16 - 6 + 3 = 74. \blacksquare$$

**FIGURE 4.7**

Venn diagram with 3 sets:  $U$  = all positive integers;  $A$  = integers divisible by 2;  $B$  = integers divisible by 5;  $C$  = integers divisible by 3.



We may state the principle used in these examples as follows:

### The Sum Rule with Overlaps

The number of elements in at least one of sets  $A$  or  $B$  is

$$|A \cup B| = |A| + |B| - |A \cap B|. \tag{3}$$

The number of elements in at least one of sets  $A$ ,  $B$  or  $C$  is

$$\begin{aligned} |A \cup B \cup C| &= |A| + |B| + |C| - |A \cap B| \\ &\quad - |A \cap C| - |B \cap C| + |A \cap B \cap C|. \end{aligned} \tag{4}$$

Clearly there is some pattern here that works for  $n$  sets, but just coming up with a good way to express the pattern succinctly takes some work. The whole subject, which includes many generalizations, is called **Inclusion-Exclusion**. It is a standard topic in a second course in discrete (or combinatorial) mathematics. The name arose because you include some counts (add them) and exclude others (subtract them). For a relatively quick derivation of the basic formula, see [15–16, Supplement].

## Problems: Section 4.3

---

Try to solve [1–8] using the Sum and Product Rules alone.

1.
  1. How many ways can a committee of  $n$  members form two disjoint subcommittees, if not everybody has to serve on either subcommittee?
  2. How many ways can a committee of  $n$  members form two disjoint, nonempty subcommittees, if not everybody has to serve on either subcommittee?
3.
  1. How many ways can two subcommittees in [1] be formed if they need not be disjoint and, again, not everybody has to serve?
  2. In the game of Twenty Questions, one person (the answerer) thinks of something and the other person (the asker) tries to find out what it is by asking Yes/No questions. The answerer must reply truthfully and the asker is limited to 20 questions and a final claim (“I claim the thing is ...”). Suppose that the something must be an English word, of which there are about half a million. Can the asker guarantee a win? (Assume that the asker has a complete dictionary.) What is an optimal strategy?
5.
  1. Suppose that we change Towers of Hanoi to have four poles in a row, but otherwise legitimate configurations are unchanged. Give an upper bound on the number of steps to solve 4-pole TOH, a bound as general as the 3-pole bound we devised in Example 3. For instance, the bound should hold for the linear 4-pole version where the poles are in a row and rings must move to adjacent poles.
  2. Returning to TOH with three poles, some sets are sold commercially with the three poles in a row and with the rings alternately colored red and blue from smallest to largest. Also, the floor under each pole is colored. The largest ring and the floor under the middle pole are the same color, and the floor under the left and right poles is the other color. The instructions say: Never put a ring directly on top of another ring of the same color or on a floor of the same color. With these additional requirements, how many legitimate configurations are there?
7.
  3. How many possible voting splits are there in a committee of  $n$  people if there is three-way voting: Yes, No, Abstain? Give several different interpretations of this question and therefore several different answers.
  2. Given a committee of  $n$  people, how many ways are there to pick a subcommittee and a chairman for that subcommittee?
  2. How many ways can  $n$  people be split into 2 committees with different functions. Solve this by a 1-to-1 correspondence with Example 1. How many ways are there if both subcommittees must be nonempty?
10.
  1. How many ways are there to order the letters of
    - a) English?
    - b) Economics?
    - c) Mississippi?
  11.
    1. How many nine-digit integers can be written using two 1's, four 2's and three 5's?
    1. In a hypnosis experiment, a psychologist inflicts a sequence of flashing lights on a subject. The psychologist has a red light, a blue light, and a green light. How many different ways are there to inflict nine flashes, if two flashes are to be red, four blue and three green?
      - a) Answer this question directly.
      - b) Answer by comparing with [11].

13.  $\langle 2 \rangle$  A family of five is to be seated at a round table. How many ways are there to do this? How many ways are there if only the relative positions of the people count? (The latter question provides an example where the Division Rule applies, but not because some individual objects are indistinguishable.)
14.  $\langle 2 \rangle$  Answer the same two questions as in [13], but now assume there are two extra place settings.
15.  $\langle 3 \rangle$  Show that no strategy for the coin problem of Example 4 is guaranteed to find the fake coin in at most three weighings. *Hint:* Suppose some strategy  $S$  did require only three weighings. Since two weighings can distinguish at most 9 situations, and there are 26 situations to be distinguished, the first weighing of  $S$  would have to divide the situations into three sets of exactly 9, 9 and 8 situations each. Does weighing one coin against one other in the first weighing accomplish this? No, because if the left pan sinks, you have 2 situations left (which ones?); if the right pan sinks, again you have 2 situations; if the pans balance, you have 22 left. What about two coins versus two, and so on?
16.  $\langle 3 \rangle$  Find a strategy for the coin problem that takes at most four weighings. Show that you are correct by drawing a tree diagram for your strategy, using the format of Fig. 4.3. Each leaf must correspond to at most one possibility.
17.  $\langle 3 \rangle$  Suppose that, in addition to the 13 coins, you have an additional “test coin” that you know is the right weight. Now there *is* a strategy that needs at most three weighings. Find it. Prove it correct.
18.  $\langle 5 \rangle$  Generalize the results of [15–17] to  $n$  coins.
19.  $\langle 2 \rangle$  We are given  $n$  coins, where  $n \geq 2$ . All but one of the coins are the same weight and the other is heavier. We have a balance scale.
- Assertion: One weighing suffices to discover which coin is heavier.
- “Proof”: By induction. When  $n = 2$  the result is clear. Suppose we have proved the result for  $k$  coins. We are now given  $k+1$  coins. We proceed as follows. Set one coin aside. Apply the procedure for  $k$  coins to the remaining  $k$  coins. If we find the heavy coin then we are finished. If not, then the heavy coin is the one we set aside. Thus we have a procedure for  $k+1$  coins.
- This can't be right. Where's the error?
20.  $\langle 2 \rangle$  You have a pan-balance scale and  $n$  coins, *each* of which independently may be light, heavy or the correct weight. You also have a test coin known to be the correct weight. You need to determine, for all the coins, if they are light, heavy or good. There is a straightforward way to do this: weigh each coin in turn against the test coin. Prove or disprove that this is a best method. That is, this method takes  $n$  tests to obtain the required information. It is best unless there is some other method using a pan balance which always gets the required information in fewer steps.
21.  $\langle 2 \rangle$  Again you are given 13 coins and one is fake. But now you are told the common weight of the good coins and the different weight of the fake one. Also, you are given a single-pan electronic scale that tells you the weight of what you put on it. Give a lower bound on the number of steps necessary to find the fake coin. Show that this lower bound is exact.
22.  $\langle 2 \rangle$  Generalize the result of [21] to  $n$  coins.
23.  $\langle 2 \rangle$  Define a sequence of digits of length  $n$  to be symmetric if the digit in position  $k$  is the same as the digit in position  $n-k+1$  for all  $k$ . For the problem in Example 6, show that there is a 2-to-1 correspondence only after we (temporarily) exclude the symmetric binary sequences. Then solve the problem.
24.  $\langle 2 \rangle$  Solve the problem in Example 6 again for decimal sequences of length  $n$ .
25.  $\langle 1 \rangle$  A computer operating system requires file names to be exactly four letters long, with uppercase and lowercase letters distinct.
- How many file names are there?
  - The operating system is changed so that uppercase and lowercase are no longer distinguished. Figure out the new number of file names two ways: applying the Division Rule to (a) and directly.
26.  $\langle 2 \rangle$  Repeat [25] with “exactly” replaced by “up to”. Can the Division Rule still be used?
27.  $\langle 3 \rangle$  Consider the de Bruijn graph (see Section 3.3) for the situation in Example 2 of the current section.
- How many nodes does this graph have?
  - Suppose the sequence of digits punched by the thief contains a repeated 4-digit sequence.

What can you say about the path on the de Bruijn graph corresponding to this sequence?

28. ⟨3⟩ A firm produces four-digit house-number signs. The first digit cannot be 0. The company need not produce every number from 1000 to 9999 because some signs can be gotten from others by rotating them 180 degrees; e.g., 6611 is just 1199 turned around. How many different signs must the company make?

29. ⟨2⟩ If  $n$  players compete in a (single-elimination) tennis tournament, how many matches are there?

*Hint:* If you draw the usual binary tree diagram for a tournament, it would seem that the answer is simplest to find when  $n$  is a power of 2. For other  $n$ , there have to be some byes (a player is advanced to the next round without having a match), and you might think this would make the answer dependent on how the byes are distributed. Actually, the answer is not dependent on that, nor even on the usual binary tree set-up. This is shown by using a one-to-one correspondence. Think: What is the real purpose of each match?

30. ⟨2⟩ If you multiply an  $n$ th-degree polynomial  $P(x)$  by an  $m$ th-degree polynomial  $Q(x)$  without combining any like terms as you go along, in general how many terms can you then eliminate by combining? For instance, multiplying  $3x+2$  by  $5x-7$ , without combining, gives

$$15x^2 - 21x + 10x - 14.$$

Then you can eliminate one term to obtain the final answer:  $15x^2 - 11x - 14$ . Why do we say “in general”?

31. ⟨1⟩ How many ways are there to distribute four distinct balls evenly between two distinct boxes?

*Hint:* Choose a first ball to put in the first box, then a second ball, then a first ball for the second box, then a second ball. But the order in which we put the balls in doesn’t matter. What should we divide by?

32. ⟨1⟩ Repeat [31], except now the boxes are indistinguishable. (See Section 4.7 for more on this sort of problem.)

33. ⟨2⟩ At Jefferson High School, 40 students play field hockey, 30 run track, and 1335 of the 1400 enrolled students do neither. How many do both? How many play field hockey but don’t run track? How many only run track?

34. ⟨2⟩ For [33], let  $F$  and  $T$  be the set of students on the field hockey and track teams, respectively. Let  $U$  be the entire student body.

- a) Translate the information given in [33] into set notation. It may be helpful to use complements. Recall from Section 0.1 that  $\overline{F}$  is everything in the universe  $U$  (here the set of all students at Jefferson) that is not in  $F$ .

- b) Take all the formulas you used or devised to answer all three questions in [33] and translate them into set notation too. For instance, for the second question you need a formula for  $|F \cap \overline{T}|$ .

35. ⟨3⟩ Jefferson High School also has a swim team with 50 members. One student is on both the swim and field hockey teams, five are on both the swim and track teams, and none is on all three teams.

- a) How many students play none of these sports?  
b) How many play at least field hockey but don’t swim?  
c) How many only run track?  
d) How many are on at most the track and swim teams?

36. ⟨1⟩ How many of the integers from 1 to 1000 are divisible by at least one of 3, 5, and 7?

37. ⟨1⟩ How many positive integers less than 105 are relatively prime to 105?

38. ⟨2⟩ How many positive integers  $\leq 100$  are

- a) either perfect squares or perfect cubes?  
b) neither perfect squares nor perfect cubes.

## 4.4 Permutations and Combinations

Permutations and combinations are the basic building blocks of combinatorial mathematics.

---

**Definition 1.** A **permutation of  $n$  things taken  $r$  at a time** is any arrangement in a row of  $r$  things from a set of  $n$  *distinct* things. Order in the row makes a difference: The same  $r$  things ordered differently are different permutations.  $P(n, r)$  is the *number* of permutations of  $n$  things taken  $r$  at a time.

---

One also speaks of a “permutation of  $r$  things from  $n$ ” or an “ $(n, r)$  permutation”. One can also speak of permutations of objects that are not distinct but in this case it is easier to consider the number of distinct items and repetitions of these, as we will at the end of this section.

---

**Definition 2.** A **combination of  $n$  things taken  $r$  at a time** is any subset of  $r$  things from a set of  $n$  distinct things. Order makes no difference. The *number* of combinations of  $n$  things taken  $r$  at a time is denoted by  $C(n, r)$  or

$$\binom{n}{r}$$

and read “ $n$  choose  $r$ ”.

---

One also speaks of a “combination of  $r$  things from  $n$ ” or an “ $(n, r)$  combination”. As with permutations, we will consider combinations where objects can be repeated at the end of this section. We will tend to use the  $C(n, k)$  notation within paragraphs and the  $\binom{n}{k}$  notation within displays—the latter doesn’t look so good in paragraphs.

For example, the permutations of the three integers 1, 2, and 3 taken two at a time are

$$(1, 2) \quad (1, 3) \quad (2, 1) \quad (2, 3) \quad (3, 1) \quad (3, 2),$$

and the combinations of two things from these three integers are

$$\{1, 2\} \quad \{1, 3\} \quad \{2, 3\}.$$

Recall that parentheses indicate an ordered pair and braces indicate a set, that is, an unordered pair.

The main difference between permutations and combinations is that, in a combination, order makes no difference. In particular, in computing  $P(n, r)$ , which  $r$  elements are chosen and *how* they are arranged both make a difference. In computing  $C(n, r)$ , only which elements are chosen makes a difference.

It is often said that combinations count sets, while permutations count sequences (or equivalently, lists or arrangements). This is correct, but it doesn't make the choice a no-brainer. Suppose we want to know how many ways to field a team of  $k$  players from a roster of  $n$ . A team is a set of people, right? Well, sometimes. If it is a team where there are no "positions" (as on many quiz shows), then Yes. But if it is a football team, where you not only pick the team but what positions they play, then No, you want a sequence. Imagine the football positions being listed down the page. The order in which you place names of players next to the positions matters as much as which names you pick. A football team is a permutation.

*Computations.* Let us find a formula for  $P(n, r)$ . We find it by counting the number of ways we can pick a permutation. Since order makes a difference, we organize our picking around position. We can begin by choosing the first entry in the permutation (say on the left), then the second (from the left), and so on. There are  $n$  choices for the first entry,  $n-1$  for the second,  $n-2$  for the third, and in general,  $n-(k-1) = n-k+1$  for the  $k$ th. Thus by the Product Rule,

$$P(n, r) = \prod_{k=1}^r (n-k+1). \quad (1)$$

For instance,

$$P(5, 2) = 5 \times 4 = 20 \quad \text{and} \quad P(6, 4) = 6 \times 5 \times 4 \times 3 = 360.$$

There are two shorter ways to write the general formula (1). First, you can use the falling factorial notation from Eq. (2), p. 22:

$$P(n, r) = n_{(r)}.$$

Or second, you can use a better known formula involving factorials. Consider these numerical examples first:

$$P(5, 2) = 5 \cdot 4 = 5 \cdot 4 \cdot \frac{3 \cdot 2 \cdot 1}{3 \cdot 2 \cdot 1} = \frac{5!}{3!}.$$

Similarly,

$$P(6, 4) = \frac{6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{2 \cdot 1} = \frac{6!}{2!}.$$

In general, multiplying the right-hand side of Eq. (1) by  $(n-r)!/(n-r)! = 1$ , we obtain

$$P(n, r) = \frac{n!}{(n-r)!}. \quad (2)$$

Note that Eq. (2) makes sense even when  $r = 0$ ; it says that  $P(n, 0) = 1$ . That is, the number of ways to line up no things from  $n$  is 1: There's just one way to do nothing! Since this interpretation is reasonable, and extends the domain of correctness of a simple formula, we adopt it.

For instance, according to Eq. (2), the number of permutations of the integers 1, 2, 3 taken two at a time should be

$$P(3, 2) = \frac{3!}{1!} = 6,$$

just the number we got earlier.

Since we developed our formula for  $P(n, r)$  through a stepwise analysis, it is quite easy to express the answer with an algorithm. Algorithm PERMVALUE, computes  $P(n, r)$  using Eq. (1). Note that this algorithm works even when  $r = 0$ .

## Algorithm 4.1

### PermValue

|                                                                                                                                                                                    |                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| <b>Input</b> $n, r$<br><b>Algorithm</b> PERMVALUE<br>$P \leftarrow 1$<br><b>for</b> $k = 1$ <b>to</b> $r$<br>$P \leftarrow P \times (n+1-k)$<br><b>endfor</b><br><b>Output</b> $P$ | [Nonnegative integers] |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|

Any combinatorial quantity can be described by giving an algorithm to compute it. Most of the quantities we develop in this chapter have simple formulas, so it would seem there is little point in providing a lengthier algorithmic description as well, except for subalgorithms in other algorithms. However, sometimes formulas are not as useful as they first appear [16], and sometimes there aren't any good formulas.

To find a formula for  $C(n, r)$ , we reduce to the previous problem. Every combination can be made into a permutation by imposing an order. Indeed, given a combination of  $r$  things from  $n$ , there are  $r!$  different orders that can be imposed (Product Rule). Moreover, every permutation of  $r$  things from  $n$  is obtained by imposing an order on some combination. Therefore we have an  $r!$ -to-1 correspondence between the set of all  $(n, r)$  permutations and all  $(n, r)$  combinations. Thus by the Division Rule,

$$\binom{n}{r} = \frac{P(n, r)}{r!} = \frac{n!}{r!(n-r)!}. \quad (3)$$

For instance,

$$C(3, 2) = \frac{3!}{2! 1!} = 3,$$

and

$$\binom{6}{4} = \frac{6!}{4! 2!} = \frac{6 \cdot 5}{2 \cdot 1} = 15.$$

Notice that Eq. (3) makes sense even when  $r = 0$ , since  $0! = 1$ . Namely,  $C(n, 0) = n!/(0!n!) = 1$ . This value makes sense because there is one empty subset of any  $n$ -set.

Using Eq. (1) for  $P(n, r)$ , we may express  $C(n, r)$  another way:

$$\binom{n}{r} = \frac{P(n, r)}{r!} = \frac{\prod_{k=1}^r (n-k+1)}{r!}. \quad (4)$$

The right-hand side above makes numerical sense even if  $n$  is not an integer, and will be useful in Section 4.6.

## EXAMPLE 1

How many different full houses are there in poker? A full house is a hand of 5 cards in which 3 of them are from one denomination and 2 from another. In a pack of cards there are 13 *denominations* (2, 3, ..., queen, king, ace) and 4 cards of each, one from each *suit* (spades, hearts, diamonds, clubs). The order in which cards are dealt makes no difference in counting different hands.

**Solution 1** Since order makes no difference, we can specify a convenient order. Let us imagine the 3-of-a-kind are picked first, followed by the 2-of-a-kind. Then there are 52 choices for the first card, followed by 3 for the second (because it must be one of the remaining cards of the same denomination as the first), and 2 for the third. Then there are 48 choices for the fourth card (anything of some other denomination) and 3 for the fifth. We have thought in terms of permutations, albeit of a restricted sort, to eliminate differences in order. Have we completely succeeded? No! We have forced the 3-of-a-kind to be considered first, but we haven't eliminated the effects of order among them. Each 3-of-a-kind has therefore been counted  $3!$  times. Likewise, each 2-of-a-kind has been counted  $2!$  times. Therefore we have counted each hand  $6 \times 2$  times. So, using the Division Rule, the correct answer is

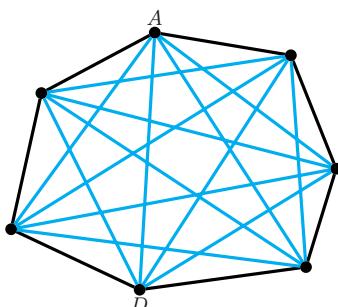
$$\frac{52 \cdot 3 \cdot 2 \cdot 48 \cdot 3}{12} = 3744.$$

**Solution 2** Since order makes no difference, we can think of this as a problem in choosing sets, i.e., combinations. First, we must pick a denomination for the 3-set and then a denomination for the 2-set. There are  $13 \times 12$  ways to do this. Having chosen the denomination for the 3-set, there are  $C(4, 3) = 4$  ways to pick the 3-set. Similarly, there are  $C(4, 2) = 6$  ways to choose the 2-set. Therefore the answer is

$$13 \times 12 \times 4 \times 6 = 3744. \blacksquare$$

## EXAMPLE 2

Diagonal chords are drawn between each pair of nonadjacent vertices of a convex polygon with  $n$  sides (see Fig. 4.8). How many chords are there?



**FIGURE 4.8**

The chords of a convex polygon.

**Solution** From each vertex,  $n-3$  chords emanate. (Why?) Since there are  $n$  vertices, we get  $n(n-3)$  chords total. However, we have counted every chord twice, once for each end. Put another way, we have counted directed chords, and now we must use the division rule to get undirected chords. The right answer is  $n(n-3)/2$ .

**Alternative Solution** A chord corresponds to a 2-subset of the vertices, of which there are  $C(n, 2)$ . However, not every 2-subset gives a chord. The  $n$  pairs of adjacent vertices give sides. Thus the answer is

$$C(n, 2) - n.$$

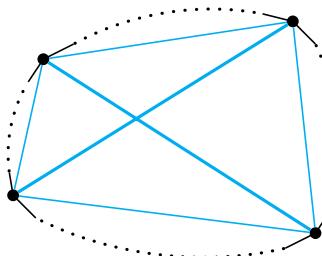
Please check that these answers are the same [12]. ■

### EXAMPLE 3

Considering the same  $n$ -gon, in how many interior points do its chords intersect? Assume that the  $n$ -gon is irregular, so that no three chords intersect at the same interior point.

**Solution** This is a rather hard problem. Most people first consider the number of intersection points on each chord. From Fig. 4.8, you can see that the number varies. However, you can try to figure out how many chords have a given number of intersections and proceed from there. This approach can work but it's tricky [5, Supplement]. Fortunately, there is an entirely different way to answer the question.

Namely, think as follows. What causes an intersection point in the first place? Answer: two criss-cross chords. What causes there to be two chords? Four vertices. Couldn't these two chords come from three vertices by sharing a vertex? Yes, but then they would intersect at one of the vertices and not in the interior. Can any four vertices produce two intersecting chords (i.e., what happened to the observation that only nonadjacent vertices produce a chord)? Yes, by connecting the four vertices alternately to form a criss-cross of chords. Can the same four vertices produce two chords in more than one way? Yes, but only the criss-cross configuration already discussed results in two chords that intersect internally. See Fig. 4.9.



**FIGURE 4.9**

Any four vertices of an  $n$ -gon create just one pair of intersecting chords.

What have we shown? That there is a one-to-one correspondence between interior intersection points and combinations of the vertices taken four at a time. Therefore the number of intersections is simply  $C(n, 4)$ . ■

### EXAMPLE 4

Using the same  $n$ -gon as before, into how many line segments are the chords divided by their various intersections? By segments we mean nonoverlapping pieces. For instance, in Fig. 4.8, chord  $AD$  is divided into seven segments.

**Partial Solution** From the previous two examples, it should be clear that we will save ourselves much grief if we attack this problem using combinations and correspondences. *Hint:* For any given chord, how is the number of segments into which it is divided related to the number of intersection points on it? If we sum up this relation over all chords, we get a statement about the total number of segments. Finish the problem! [13] ■

## Permutations and Combinations with Repetitions

Suppose that we are allowed to use the same elements over again in making our permutations and combinations. (In probability and statistics, this is called sampling with replacement.) For instance, the collection of permutations taken two at a time from the set  $\{1, 2, 3\}$  now becomes

$$(1, 1) \quad (1, 2) \quad (1, 3) \quad (2, 1) \quad (2, 2) \quad (2, 3) \quad (3, 1) \quad (3, 2) \quad (3, 3).$$

The collection of combinations, again two at a time, becomes

$$\{1, 1\} \quad \{1, 2\} \quad \{1, 3\} \quad \{2, 2\} \quad \{2, 3\} \quad \{3, 3\}.$$

Let us call the number of these things  $P^*(n, k)$  and  $C^*(n, k)$ . Are there formulas for these quantities?

(Unlike  $P(n, k)$  and  $C(n, k)$ , which are rather standard notation,  $P^*(n, k)$  and  $C^*(n, k)$  is just temporary notation we have made up. There aren't universally accepted notations for these two quantities.)

Yes, there are formulas. For permutations with replacement the formula is very simple. At each of the  $k$  locations in the sequence, you get to choose any one of the  $n$  objects, whether or not it has been picked before. Therefore  $P^*(n, k) = n^k$ . For example, the number of ordered pairs of three things is  $P^*(3, 2) = 3^2 = 9$ , in agreement with the list of ordered pairs displayed above.

The formula for  $C^*(n, k)$  is harder to derive, though not much more complicated to write down. We will derive it one way in Section 4.7 and another way in Section 5.3.

*Note.* In talking about combinations with repetitions, we have abused the definition of set. For instance, we listed  $\{2, 2\}$  as a set of size two from  $\{1, 2, 3\}$ . We did indeed mean to list  $\{2, 2\}$  and we did mean it to have two elements. The abuse is in calling it a set. As we noted in Section 0.1, the thing we are talking about here is properly called a *multiset*.

## Problems: Section 4.4

1.
  1. Evaluate  $P(100, 97)$ ?
    - a)  $P(5, 1)$ ,  $P(5, 2)$  and  $P(5, 3)$ ,
    - b)  $C(5, 1)$ ,  $C(5, 2)$  and  $C(5, 3)$ .
  2.
    1. Which is more work to compute,  $C(100, 3)$  or  $C(100, 97)$ ?
    2.
      1. When the expression

$$(A+a)(B+b)(C+c)(D+d)(E+e)$$

is multiplied out, how many terms will have three uppercase letters?

5.  $\langle 2 \rangle$  Show that the number of ways to pick a subset of one or two things from  $n$  distinct things is  $n(n+1)/2$ .

6.  $\langle 1 \rangle$  By algebra, show that

$$P(n, k) \times P(n-k, j) = P(n, k+j).$$

Can you also explain this equality without using algebra? (We discuss such non-computational proofs at length in Section 4.5.)

7.  $\langle 2 \rangle$  How many ways are there to pick a combination of  $k$  things from  $\{1, 2, \dots, n\}$  if the elements 1 and 2 cannot both be picked?

8.  $\langle 2 \rangle$  Three distinct balls are to be distributed among seven different urns so that each urn gets at most one ball.

- a) How many ways can this be done?  
b) What if the balls are indistinguishable?

9.  $\langle 2 \rangle$  How many ways are there to put eight rooks on a chessboard so that no one rook can capture another? (This means that no two are in the same row or same column.) How many ways are there to put five noncapturing rooks on a chessboard?

10.  $\langle 1 \rangle$  How many triangles can be formed from the edges and diagonals of an  $n$ -gon, if the vertices of each triangle must be vertices of the  $n$ -gon?

11.  $\langle 3 \rangle$  Same as [10], but now all three sides of the triangle must be diagonals of the  $n$ -gon. Be careful!

12.  $\langle 1 \rangle$  Check that the two solutions to Example 2 are the same.

13.  $\langle 2 \rangle$  Let  $\mathcal{C}$  be a collection of line segments, and let  $I$  be the set of their intersection points. Assume that no endpoint of any segment is in  $I$  and that no three lines intersect at the same point. Let  $i_L$  be the number of intersection points on line  $L$  in  $\mathcal{C}$ .

- a) Explain why  $L$  is divided into  $i_L+1$  segments.  
b) Explain why  $\sum_{L \in \mathcal{C}} 1 = |\mathcal{C}|$ .  
c) Explain why  $\sum_{L \in \mathcal{C}} i_L = 2|I|$ .  
d) Finish Example 4.

14.  $\langle 3 \rangle$  When  $n$  points are picked in general position on a circle, and all chords between them are drawn, these chords divide the inside of the circle into several regions. For instance, when  $n = 3$ , there are 4 regions, as shown in Fig. 2.19 for [25, Section 2.6], of which this problem is a completion. Let  $A_n$  be the number of regions (areas).

- a) Argue that  $A_n = A_{n-1} + c_n + i_n$ , where  $c_n$  is the number of new chords created to the  $n$ th point and  $i_n$  is the number of new intersection points created on these new chords.  
b) Using the notation of the previous problems, show, therefore, that  $A_n = |I| + |\mathcal{C}| + 1$ .  
c) Give a formula for  $A_n$  in terms of combinations.

15.  $\langle 2 \rangle$

- a) How many permutations of length  $m+n$  are there of  $m$  identical a's and  $n$  identical b's?  
b) How many ways are there to choose a set of  $n$  objects from  $m+n$  distinct objects?  
c) Why are the answers to a) and b), the first about permutations, the second about combinations, the same? Give a one-to-one correspondence.

16.  $\langle 2 \rangle$  Algorithm 4.2, PERMVALUE2, computes  $P(n, r)$ , based on the formula  $n!/(n-r)!$ . Although correct, PERMVALUE2 is not as good as PERMVALUE in the text. Why? If you're not sure, compute  $P(25, 3)$  with a hand calculator using both algorithms.

#### Algorithm 4.2. PERMVALUE2

---

```

Input  $n, r$ 
Algorithm PERMVALUE2
   $P \leftarrow 1$ 
  for  $i = 2$  to  $n$ 
     $P \leftarrow P * i$ 
  endfor
  for  $i = 2$  to  $n-r$ 
     $P \leftarrow P/i$ 
  endfor
Output  $P$ 

```

---

17.  $\langle 2 \rangle$  Devise at least two algorithms to compute  $C(n, k)$ , of varying goodness in the sense of [16].

18.  $\langle 1 \rangle$  How many edges can a simple graph on  $n$  vertices have? Recall that a simple graph is one with-

out multiple edges between any pair of vertices and without loops.

19.  $\langle 2 \rangle$  How many different simple, labeled graphs are there with  $n$  vertices? In a labeled graph (see [32, Section 3.1]), the vertices are distinguished, say, by numbering them. A labeled graph with  $n$  vertices and exactly one edge, with the edge between vertices 1 and 2, is considered to be different from the labeled graph with  $n$  vertices and exactly one edge, with the edge between vertices 3 and 4.
20.  $\langle 1 \rangle$  How many edges can a simple digraph on  $n$  vertices have? For each ordered pair  $(u, v)$  of ver-

tices, a simple digraph has at most one edge *from*  $u$  to  $v$ . Thus it may have two edges *between*  $u$  and  $v$ , one from  $u$  and one from  $v$ .

21.  $\langle 2 \rangle$  How many labeled simple digraphs are there on  $n$  vertices?
22.  $\langle 2 \rangle$  Recall that a tournament is a digraph such that, for each pair of vertices  $\{u, v\}$ , exactly one of  $(u, v)$  and  $(v, u)$  is an edge. A **partial tournament** is a tournament from which some of the edges may have been removed. How many labeled tournaments are there on  $n$  vertices? How many labeled partial tournaments?

## 4.5 Combinatorial Identities and Combinatorial Arguments

---

The numbers  $C(n, r)$  are miraculous. They satisfy an immense number of identities, many very beautiful, many very surprising. We will introduce a few of the most fundamental ones. We will use their introduction as an opportunity to discuss several methods of proof in combinatorics — by algebra, computer algebra, induction, and a conceptual method often called **combinatorial argument**.

---

**Theorem 1.** For all nonnegative integers  $k \leq n$ ,

$$\binom{n}{k} = \binom{n}{n-k}. \quad (1)$$

---

**PROOF 1** The formula for  $C(n, r)$  is Eq. (2) of Section 4.4. Let's write it down for  $r = k$  and  $r = n - k$  and see if we can make them look the same.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!},$$

$$\binom{n}{n-k} = \frac{n!}{(n-k)!(n-(n-k))!} = \frac{n!}{(n-k)!k!} = \frac{n!}{k!(n-k)!}.$$

**PROOF 2** Let us think about what  $C(n, k)$  and  $C(n, n-k)$  count and look for a one-to-one correspondence between them.  $C(n, k)$  counts the number of  $k$ -element subsets of an  $n$ -set.  $C(n, n-k)$  counts the number of  $(n-k)$ -element subsets. But in picking a  $k$ -element subset we inescapably pick an  $(n-k)$ -element subset as well, namely, all the elements we left out of the  $k$ -subset. In short, within the  $n$ -set, complementation (see Section 0.1) provides a one-to-one correspondence between  $k$ -subsets and  $(n-k)$ -subsets. ■

**Theorem 2.** For all positive integers  $k < n$ ,

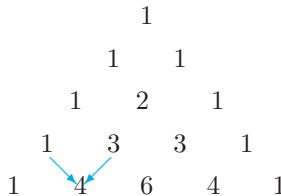
$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}. \quad (2)$$

The recurrence (2) will play an important role in this chapter since, if you know the combinations of  $n-1$  things, you then need only a single addition to find  $C(n, k)$ . Eq. (2), therefore, leads to a simple algorithm for computing combinations [12].

Traditionally, Theorem 2 and its algorithm are depicted graphically, as in Fig. 4.10. This is a row-by-row chart of the combinations  $C(n, k)$ , where  $n$  indexes the row and  $k$  varies across the row. The top row is row 0 and the left-most entry in any row is the 0th entry. With the rows centered as shown, Theorem 2 says that each entry is the sum of the two entries immediately above it. For instance,

$$\binom{4}{1} = 4 = 1 + 3 = \binom{3}{0} + \binom{3}{1},$$

as highlighted by the colored arrows. You should be able to fill in the next row by sight [11].



**FIGURE 4.10**

Pascal's Triangle.

This famous figure is called **Pascal's Triangle**, after the French mathematician and philosopher Blaise Pascal (1623–1662). Note that the entries are symmetric around the center, which is exactly what Theorem 1 says.

**PROOF 1 (ALGEBRAIC)** We begin with the RHS of Eq. (2), since it is more complicated. We substitute in the factorial formula for combinations, Eq. (3) of Section 4.4, and massage the expression until it looks like the LHS:

$$\begin{aligned}\binom{n-1}{k} + \binom{n-1}{k-1} &= \frac{(n-1)!}{k!(n-1-k)!} + \frac{(n-1)!}{(k-1)!((n-1)-(k-1))!} \\&= \frac{(n-1)!(n-k)}{k!(n-1-k)!(n-k)} + \frac{k(n-1)!}{k(k-1)!(n-k)!} \\&= \frac{(n-1)!(n-k)}{k!(n-k)!} + \frac{(n-1)!k}{k!(n-k)!} \\&= \frac{(n-1)!((n-k)+k)}{k!(n-k)!} = \frac{n!}{k!(n-k)!} = \binom{n}{k}. \quad [\text{Whew!}]\end{aligned}$$

**PROOF 2** We look for interpretations of the two sides of Eq. (2) that make it clear they are counting the same thing. The LHS is about picking  $k$ -subsets from an  $n$ -set. The RHS is about picking  $k$ -subsets and  $(k-1)$ -subsets from an  $(n-1)$ -set. How can choosing from an  $(n-1)$ -set be about the same thing as choosing from an  $n$ -set? Perhaps by singling out one element first, making our decision about that, and then dealing with the rest. That's it! Both sides are about, say, choosing a set of  $k$  players from a roster of  $n$  players. On the LHS, we simply choose a set of  $k$  players all at once. On the RHS, we have singled out a certain player, call him Joe, and decide about him first. Either we don't choose Joe, in which case we have to pick a full set of  $k$  players from the remaining  $n-1$  players on the roster, or else we do choose Joe, in which case we must pick only  $k-1$  players from the remaining  $n-1$ . ■

For both Theorems 1 and 2, the first proof was an algebraic argument and the second proof was a combinatorial argument. In the second there wasn't any computation. Instead, we showed that both sides of the equation count the same thing. In one case we showed this by exhibiting a one-to-one correspondence. In the other, we made up a story about two different approaches to accomplishing the same choice. Finding these stories is like playing the game where you are given the answer and must find the question. Actually, you are given two answers (the two sides of the identity) and must find a question they both answer.

Combinatorial arguments are harder to think up than algebraic proofs — there is no formula for finding an appropriate correspondence or story, whereas in the algebraic proof you just grind it out (assuming your algebra is good enough!), or get a computer to grind it out. Though combinatorial arguments involve hard thinking, with practice you can get good at finding them. Why bother, when machines do more and more? See the final subsection of this section, Combinatorial Arguments vs. Computer Algebra.

*Note.* We have not defined  $C(n, k)$  for either  $k < 0$  or  $k > n$ . Because the previous theorem involves formulas where  $n$  and  $k$  change by 1, it would be very convenient to let  $k$  “slip over” the 0 and  $n$  bounds. Precisely because  $C(n, k)$  hasn't been defined beyond these bounds, we can define it any way we wish. Mathematicians always strive to extend definitions so that theorems which already hold for the restricted definitions will continue to hold more generally. It turns out in this case that the “right” extension is

**Definition 1.** Let  $n$  and  $k$  be integers with  $n \geq 0$ . Then  $C(n, k) = 0$  for  $k < 0$  and  $k > n$ .

For instance, with this definition Theorems 1 and 2 are true for any integer  $k$  [16].

We are now ready for a more substantial result:

---

**Theorem 3.** For  $n$  any nonnegative integer,

$$\sum_{k=0}^n \binom{n}{k} = 2^n. \quad (3)$$

---

**PROOF 1** Actually, we've already proved this equation, and by a combinatorial argument! We just didn't have the notation then to write the result down. To wit, we noted in Section 4.3 two different ways to go about choosing a subcommittee from a committee of  $n$ . Either you first choose the size of the subcommittee and then choose a subset with that size, or else you go through the full committee one by one and decide for each member whether that member goes on the subcommittee. The first method gives the LHS (Sum Rule); the second, the RHS (Product Rule).

**PROOF 2** We can't give an algebraic proof of the sort we gave for the previous two theorems because Eq. (3) involves a variable number of terms, depending on  $n$ . This sort of algebraic situation is ripe for mathematical induction, especially since we already have a result, Theorem 2, that ties combinations of  $n$  things to combinations of  $n-1$ . Let  $P(n)$  be the assertion that Eq. (3) is true for the particular value  $n$ .

$P(0)$ . The RHS is  $2^0 = 1$ ; the LHS is  $0!/0!0! = 1$  also.

$P(n-1) \Rightarrow P(n)$ . Take the LHS of Eq. (3), break it down using Theorem 2, and substitute into Eq. (3) itself in the case  $n-1$  (the inductive hypothesis). Let's hope this gives us the RHS of Eq. (3). Indeed, here are the computations:

$$\begin{aligned} \sum_{k=0}^n \binom{n}{k} &= \sum_{k=0}^n \left[ \binom{n-1}{k} + \binom{n-1}{k-1} \right] && [\text{Thm. 2 and Def. 1}] \\ &= \sum_{k=0}^{n-1} \binom{n-1}{k} + \sum_{k=1}^n \binom{n-1}{k-1} && [\text{Def. 1}] \\ &= \sum_{k=0}^{n-1} \binom{n-1}{k} + \sum_{j=0}^{n-1} \binom{n-1}{j} && [j = k-1 \text{ in second sum}] \\ &= 2^{n-1} + \sum_{j=0}^{n-1} \binom{n-1}{j} && [P(n-1)] \\ &= 2^{n-1} + 2^{n-1} = 2^n. && [P(n-1)] \end{aligned}$$

Definition 1 makes the right side of the first line correct when  $k = 0$  and  $k = n$  and enables the changes in the limits of summation in the second line. ■

Our final theorem of this section is not anywhere near as important as the previous ones, but it is a good challenge for you to see if you can now anticipate the combinatorial argument we give to prove it.

---

**Theorem 4.** For any nonnegative integer  $n$ ,

$$\sum_{k=0}^n k \binom{n}{k} = n2^{n-1}. \quad (4)$$

---

Let's think it through. The LHS tells us that we can think in terms of subcommittees (i.e., combinations) again, where we decide on subcommittee size first. But what is this extra factor  $k$ ? Since it multiplies the number of size  $k$  subcommittees, it must be an extra choice for each of these subcommittees. But  $k$  is the number of members of such a subcommittee, so we must be singling one of them out for special attention — the subcommittee chair! So the appropriate story is about how to pick a subcommittee and a leader for it. The RHS must give another way to accomplish the same task. All those 2's say that once again, we are making Yes/No subcommittee appointment decisions for committee members. But this time we are doing this for only  $n-1$  members. Before that there is one  $n$ -way choice. Ah, picking the chair for the subcommittee before making other choices. Now we are ready to write down a more formal argument.

**PROOF** Both sides of Eq. (4) count the number of subsets of an  $n$ -set when each subset also has a distinguished element. We can either first decide the subset size  $k$ , then pick the subset, then decide which of the  $k$  elements in it is distinguished and then add together all the results for different subset sizes (LHS); or we can first decide which of the original  $n$  elements is to be the distinguished subset element and then, for each of the remaining  $n-1$  elements, decide whether or not it is in the subset too (RHS). ■

**Combinatorial Arguments vs. Computer Algebra.** Algebraic proofs are being automated more and more. If the point of combinatorial arguments was to avoid hard algebra, are they still needed?

Yes, for two reasons. First, while CASs are getting better, factorials still can cause them problems. For instance, using the 2003 version of a standard CAS, although we could get Theorems 3 and 4 easily, we only got Theorem 1 with some effort, and we couldn't get Theorem 2 at all. For Theorems 3 and 4 we just typed in the left-hand sides of Eqs. (3) and (4), hit enter, and the right-hand sides popped up. For Theorem 1 we had to be a little trickier. Call the RHS of Eq. (1)  $A$  and the LHS  $B$ . When  $B$  is not much simpler than  $A$ , there is no point in typing  $A$  into your CAS and hoping that  $B$  will come back; how is the CAS supposed to know that  $B$  is the form you want? A good technique in this case is to type in  $A - B$  and hope that 0 comes back, or type in  $A/B$  and hope for 1. Or try to force these results by asking for simplification. This approach worked on the third try for Theorem 1: typing in  $A - B$  and asking for “FullSimplify” resulted in an output of 0. However, for Theorem 2 even these tricks failed us. If you have a CAS, you might want to try these experiments yourself. Maybe you can do better.

Another objection to a CAS solution is: How do you know it's correct? And indeed, CASs sometimes give wrong answers. Just to give the most common situation, a CAS might give an answer which is correct when all variables are nonzero, but which is not right otherwise. Or, there might simply be a programming error.

So the first reason for the answer Yes is that CASs are not perfect.

However, this is not enough of an answer. CASs are getting better, and the probability they are wrong is probably not greater (maybe a lot less) than the probability that your own derivation is wrong. Most of the algorithms they use are provably correct, at least in principle. Finally, a CAS is a general purpose algebra system. Combinatorial identities are a special subdomain. Research in the last 25 years has led to special algorithms (and proofs of correctness) not only for verifying combinatorial identities but even for discovering many of them. See Section 5.9.

So is there some other reason why we still need combinatorial arguments? Yes! Combinatorial arguments provide understanding and insight, and they make the results easy to remember, in a way algebra or computers do not. For instance, Eq. (1) makes perfect sense once you understand the combinatorial argument that every way to *include*  $k$  things is a way to *exclude*  $n - k$  things. Once you grasp this symmetry between including and excluding, both Eq (1) and a proof for it are unforgettable.

Mathematicians have always loved combinatorial arguments, and with good reason they still do.

## Problems: Section 4.5

For each of problems 1–7 solve it two ways, by algebra and by a combinatorial argument. You might also experiment to see which ones your favorite computer algebra system can verify.

1.  $\langle 2 \rangle$  Show that:

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

2.  $\langle 2 \rangle$  Verify that

$$\binom{n+1}{k+1} = \frac{n+1}{k+1}\binom{n}{k};$$

To give a combinatorial argument, first multiply both sides by  $k+1$  to avoid division.

3.  $\langle 2 \rangle$  Verify

- a)  $C(n, k)C(n-k, j) = C(n, j)C(n-j, k),$
- b)  $C(n, k)C(n-k, j) = C(n, k+j)C(k+j, k).$

Then verify, directly from a) and by a combinatorial argument, that

- c)  $C(n, k)P(n-k, j) = P(n, j)C(n-j, k).$

4.  $\langle 3 \rangle$  Show that

$$C(n, k) = C(n-2, k-2) + \\ 2C(n-2, k-1) + C(n-2, k).$$

5.  $\langle 1 \rangle$  Show that

$$P(n, k) = n P(n-1, k-1).$$

6.  $\langle 2 \rangle$  Show that

$$P(n, k) = P(n-1, k) + k P(n-1, k-1).$$

7.  $\langle 1 \rangle$  Verify that

$$P(n, k)P(n-k, j) = P(n, j)P(n-j, k).$$

8.  $\langle 2 \rangle$  In [31, Section 2.6], you probably discovered that every triangulation of an  $n$ -gon has  $n - 2$  triangles and uses  $n - 3$  diagonals (chords). Here we will prove these assertions by combinatorial arguments.

- a) Prove that in every triangulation of an  $n$ -gon,  $t = d+1$ , where  $t$  is the number of triangles and  $d$  is the number of diagonals. Hint: Imagine

the diagonals put in one by one. Each time, how many new polygons are formed?

- b) Come up with an equality relating  $t$ ,  $d$  and  $n$  as follows. For each triangle in the triangulation, put a tick mark on each side of the triangle. Now count the ticks two ways. (This is similar to the argument that proved Theorem 1 in Section 3.1.)

- c) Solve the two equations in a) and b) for  $t$  and  $d$  in terms of  $n$ .

9. ⟨2⟩ A **quadrangulation** of an  $n$ -gon is a division by nonintersecting diagonals into quadrilaterals. Use the method of [8] to find formulas for the number of diagonals and quadrilaterals. (Not every  $n$ -gon can be quadrangulated, but your answer to this problem should suggest which ones can.)

10. ⟨3⟩ Verify both by induction and by a combinatorial argument:

$$\sum_{k=m}^n \binom{k}{m} = \binom{n+1}{m+1}.$$

What does this equality say about Pascal's Triangle?

11. ⟨1⟩ Fill in the next two rows of Fig. 4.10.

12. ⟨1⟩ Use Theorem 2 to write an iterative algorithm that computes Pascal's Triangle down through the  $n$ th row.

13. ⟨4⟩ Theorem 2 can also be used to write a recursive algorithm which, given  $(n, k)$ , outputs  $C(n, k)$ . The recursive algorithm doesn't compute every number in the  $n$ th row to get  $C(n, k)$ . It doesn't compute all entries in earlier rows either. However, it computes certain previous entries many times (unless a lot of “flags” are included to keep it from doing computations more than once).

- a) Write this recursive algorithm (don't worry about flags).

- b) What previous combinations does your algorithm compute to get  $C(n, k)$ ? Answer by shading in part of Pascal's Triangle.

- c) Conjecture how many times your algorithm computes various entries when invoked to compute  $C(n, k)$ . Prove it.

- d) For various pairs  $(n, k)$  compute the total number of recursive calls your algorithm makes (including the main call) to compute  $C(n, k)$ . Conjecture a formula for this number. Which algorithm, the one for this problem or the one in the previous problem, appears to be more efficient?

14. ⟨2⟩ If  $p$  is a prime, show that all numbers in row  $p$  of Pascal's Triangle are divisible by  $p$  (except the 1's at the ends).

15. ⟨4⟩ Show the converse of [14]: If every entry in row  $n$  of Pascal's Triangle (except the ends) is divisible by  $n$ , then  $n$  is a prime.

16. ⟨1⟩ Check that Theorems 1 and 2 are true for all nonnegative integers  $n$  and all integers  $k$ , if  $C(n, k)$  is defined to be 0 for  $k < 0$  and  $k > n$ .

17. ⟨2⟩ Use the second equality in [1] to give an algebraic proof of Theorem 4.

18. ⟨3⟩ Give an algebraic, inductive proof of Theorem 4.

19. ⟨4⟩ Prove two ways that

$$\sum_{i=0}^k \binom{n}{i} \binom{m}{k-i} = \binom{n+m}{k}.$$

This equality is sometimes called the **Vandermonde identity**.

20. ⟨3⟩ For each positive integer  $r$  show that

$$\sum_{k=0}^N \binom{k-1}{r-1} \binom{N-k}{n-r} = \sum_{k=r}^{N-n+r} \binom{k-1}{r-1} \binom{N-k}{n-r} = \binom{N}{r}.$$

*Hint:* Fix  $r$  and count the number of ways to pick  $n$  integers from  $1, 2, \dots, N$  by considering each possible value of the  $r$ th smallest of the  $n$  chosen integers.

21. ⟨2⟩ For  $n$  odd, show that

$$\sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{k} = \sum_{k=\lceil n/2 \rceil}^n \binom{n}{k}.$$

Therefore, what are both sums equal to?

22. ⟨2⟩ Do [21] again for  $n$  even.

## 4.6 The Binomial Theorem

---

How fast can you multiply out  $(3x+4y)^2$ ? Speed in such hand calculations is no longer important per se. However, some products, such as this one, are easily expanded by hand, or even mentally, and it is often quicker to do one of these than to use a calculator. The most important result about expansions is the Binomial Theorem, which gives the expansion of  $(x + y)^n$  and, by a simple extension, the expansions of the  $n$ th power of binomials like  $3x+4y$  above. It also has many applications to counting.

In this section, we use the inductive paradigm to guess the statement of the Binomial Theorem. Then we give some examples. Then we prove it and state some generalizations.

To guess the theorem we start by expanding the first few positive integer powers of  $x + y$ . The results are

$$\begin{aligned}(x+y)^1 &= x+y, \\(x+y)^2 &= x^2 + 2xy + y^2, \\(x+y)^3 &= x^3 + 3x^2y + 3xy^2 + y^3, \\(x+y)^4 &= x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4.\end{aligned}$$

The powers of  $x$  and  $y$  on the RHS form an easy pattern: it's always  $x^i y^j$ , where  $i$  goes down by 1 for each new term and  $j$  goes up by 1. The sum of  $i$  and  $j$  is always  $n$ , the power on the LHS.

What about the coefficients? These should look familiar, especially if we rewrite  $1x$  for  $x$  and  $1y$  for  $y$ : They are just the entries of Pascal's Triangle. We are missing the top of the triangle, but that's easily fixed. Since  $(x+y)^0$  is defined to be 1, we get the 1 we need at the top.

Therefore, with considerable confidence we assert:

---

**Theorem 1, The Binomial Theorem.** For all nonnegative integers  $n$ ,

$$(x+y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k. \quad (1)$$

---

Because of this theorem, the numbers  $C(n, k)$  are called **binomial coefficients**.

Do you see [14] that we could also write the RHS as

$$\sum_{k=0}^n \binom{n}{k} x^k y^{n-k} = y^n + \binom{n}{1} xy^{n-1} + \binom{n}{2} x^2 y^{n-2} + \cdots + \binom{n}{n-1} x^{n-1} y + x^n?$$

Sometimes it is convenient to write a polynomial with descending powers of  $x$ , as in Eq. (1), and other times with ascending powers, as in this alternative version.

**EXAMPLE 1**

Expand  $(x + 2)^4$ .

**Solution** Setting  $y = 2$  and  $n = 4$  in the Binomial Theorem, we have

$$\begin{aligned}(x + 2)^4 &= 1x^42^0 + 4x^32^1 + 6x^22^2 + 4x^12^3 + x^02^4 \\&= x^4 + 8x^3 + 24x^2 + 32x + 16.\end{aligned}\blacksquare$$

**EXAMPLE 2**

Set  $x = y = 1$  in the Binomial Theorem and see what happens.

**Solution** The LHS of the Binomial Theorem is easy:  $(1 + 1)^n = 2^n$ . The RHS simplifies nicely too: All the powers of  $x$  and  $y$  go away. Therefore we obtain

$$2^n = \sum_{k=0}^n \binom{n}{k}.$$

We got this same result two other ways in Theorem 3 in Section 4.5.  $\blacksquare$

**EXAMPLE 3**

Derive a general expansion for  $(x - y)^n$  and then apply the special case  $x = y = 1$  to it.

**Solution** Since  $x - y = x + (-y)$ , we can reduce to the previous case by substituting  $-y$  for  $y$  in the Binomial Theorem. This substitution is perfectly legitimate:  $y$  represented any number, and so does  $-y$ . We get

$$\begin{aligned}(x - y)^n &= \sum_{k=0}^n \binom{n}{k} x^{n-k}(-y)^k \\&= \sum_{k=0}^n (-1)^k \binom{n}{k} x^{n-k} y^k.\end{aligned}$$

Now, when  $x = y = 1$ , the LHS is even simpler than before: It's 0, at least when  $n \geq 1$ . So we get

$$0 = \sum_{k=0}^n (-1)^k \binom{n}{k}. \quad \blacksquare \tag{2}$$

Can you think of a nice way to put in words what this last identity says? Can you think of a combinatorial argument for it? [7]

**EXAMPLE 4**

Compute  $(1.02)^3$  to four decimal places.

**Solution** Let's evaluate this expression using the Binomial Theorem, and see what we get. Plugging into the theorem with  $x = 1$ ,  $y = .02$ , and  $n = 3$  we get

$$(1.02)^3 = 1 + 3(.02) + 3(.0004) + 1(.000008).$$

Generalizing, we see that whenever  $y$  is close to zero, the terms involving higher and higher powers of  $y$  will contribute less and less to the answer, and so if we only want a certain number of decimal places, we can often stop computing terms at

some point. In particular, to compute 4 decimal places of  $(1.02)^4$ , it suffices to quit after the  $y^2$  term:

$$(1.02)^3 \approx 1 + .06 + .0012 = 1.0612. \blacksquare$$

### EXAMPLE 5

Compute  $\sqrt{1.02}$ .

**Solution**  $\sqrt{1.02} = (1.02)^{1/2}$ , so set  $x = 1$ ,  $y = .02$ , and  $n = \frac{1}{2}$  and plug in:

$$\sqrt{1.02} = \sum_{k=0}^{1/2} \binom{\frac{1}{2}}{k} 1^{\frac{1}{2}-k} (.02)^k. \quad (3)$$

The factor  $1^{\left(\frac{1}{2}-k\right)}$  equals 1 for any  $k$ . The factor  $C\left(\frac{1}{2}, k\right)$  is a bit of a problem — you can't choose  $k$  things from  $\frac{1}{2}$  thing! But recall from Eq. (4), Section 4.4, that one formula for  $C(\alpha, k)$  is

$$\binom{\alpha}{k} = \frac{1}{k!} \prod_{j=1}^k (\alpha+1-j) = \frac{\alpha(\alpha-1)(\alpha-2)\cdots(\alpha-k+1)}{k!}, \quad (4)$$

which makes sense for any *real number*  $\alpha$ . (We have switched from  $n$  to  $\alpha$  to emphasize this fact; traditionally,  $n$  refers to integers only.) *Note:* Since there are  $k$  factors in the product, in both numerator and denominator,  $k$  must still be a nonnegative integer. ( $k = 0$  is all right because empty products are defined to be 1, as discussed in Section 2.7. See also [23].) Thus

$$\binom{\frac{1}{2}}{0} = 1, \quad \binom{\frac{1}{2}}{1} = \frac{1}{2}, \quad \binom{\frac{1}{2}}{2} = \frac{\left(\frac{1}{2}\right)\left(-\frac{1}{2}\right)}{2} = -\frac{1}{8}, \quad \binom{\frac{1}{2}}{3} = \frac{\left(\frac{1}{2}\right)\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)}{6} = \frac{1}{16},$$

and so on.

There's one more novelty in Eq. (3): a noninteger on top of the summation sign. Recall that  $\sum_{k=a}^b$  means to start the sum at  $k = a$  and go up by 1 until you hit  $b$ . But starting at 0,  $k$  will never hit  $\frac{1}{2}$ . So let's just keep going:

$$\begin{aligned} \sqrt{1.02} &= 1 + \left(\frac{1}{2}\right)(.02) + \left(-\frac{1}{8}\right)(.0004) + \left(\frac{1}{16}\right)(.000008) + \cdots \\ &\approx 1.00995005. \blacksquare \end{aligned}$$

Is this a joke? We have taken the Binomial Theorem, which we have asserted for nonnegative integers  $n$ , and blithely assumed that it is also true for all real  $\alpha$ . This is pretty outrageous, considering that the only evidence we have for the theorem are the calculations we made at the beginning of the section for  $n = 1, 2, 3, 4$ , and the only general proof technique we have is induction, which only works for integers. Yet we have just charged ahead, giving  $C(\alpha, k)$  and  $\sum_{k=0}^{\alpha}$  the first plausible interpretation that occurs to us. This is the sort of unthinking plug-in that math teachers attribute to their most mechanically minded students!

Yes, it is a joke. But like all good jokes, there's something serious underneath. The method works! 1.00995005 is extremely close to  $\sqrt{1.02}$ . In fact, it is correct

to seven decimal places. Furthermore, if we added all those other terms for  $k = 4, 5, \dots$ , it turns out we would get  $\sqrt{1.02}$  exactly.

The point is, it's a rather marvelous fact about mathematics that results which are proved in limited domains often hold much more generally, if we can only figure out how to state them. Therefore, sometimes it's worth our while to let our minds wander expansively and test if our formulas still work.

The infinite sum binomial theorem we have hit upon is called the Binomial Series Theorem. We are in no position to prove it; it is usually proved in advanced calculus or real analysis. (You can't actually do an infinite number of additions, so infinite sum has to be defined in terms of "limits" of finite sums.) However, we will *state* it precisely in a moment. First, let us prove the ordinary (finite sum) Binomial Theorem. We give two proofs: the first, an algebraic proof by induction; the second, a combinatorial argument.

**FIRST PROOF OF THEOREM 1** The case  $n = 1$  is an easy check. Assume that the formula is true for  $n$  and now extend it to  $n + 1$ . Since

$$(x + y)^{n+1} = (x + y)^n(x + y),$$

we may assume that

$$(x + y)^{n+1} = \left( \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k \right) (x + y). \quad (5)$$

Now all we have to do is multiply out the two factors on the right, combine like terms, and see if it's what we want. Since the  $y$ 's have simpler exponents than the  $x$ 's, let's concentrate on combining terms with the same power of  $y$ . Let  $j = 0, 1, \dots, n, n+1$  (why  $n+1$ ?) be any such power. It is helpful to restate the RHS of Eq. (5) as

$$\left( \dots + \binom{n}{j-1} x^{n-(j-1)} y^{j-1} + \binom{n}{j} x^{n-j} y^j + \dots \right) (x + y).$$

The only ways to get  $y^j$  when we multiply out is to multiply  $\binom{n}{j} x^{n-j} y^j$  from the first factor by  $x$  from the second, and to multiply  $\binom{n}{j-1} x^{n-(j-1)} y^{j-1}$  from the first factor by  $y$  from the second. Combining the two products, we get

$$\left[ \binom{n}{j-1} + \binom{n}{j} \right] x^{n+1-j} y^j.$$

But, by Theorem 2, Section 4.5, this is just

$$\binom{n+1}{j} x^{n+1-j} y^j.$$

We conclude that the RHS of Eq. (5) is

$$\sum_{k=0}^{n+1} \binom{n+1}{k} x^{n+1-k} y^k.$$

But this is just what the case  $n+1$  of the Binomial Theorem states. The inductive step is complete. ■

**SECOND PROOF OF THEOREM 1** We are looking for a formula for the expansion of

$$(x+y)(x+y) \cdots (x+y)$$

when there are  $n$  factors. Terms in the expansion arise by (and only by) our selecting exactly one of  $x$  and  $y$  from each of the  $n$  factors and multiplying our selections together. For instance, if we pick  $x$  from the second and fourth factors and  $y$  from all the others, we get a contribution of  $x^2y^{n-2}$  to the expansion. The expansion can be simplified by combining like terms. For instance, choosing  $x$  from the first and second factors and  $y$  from the others also leads to  $x^2y^{n-2}$ . So we merely need to decide what sorts of terms of the form  $x^i y^j$  can occur and how many of them do occur.

Since we get one power of either  $x$  or  $y$  from each factor  $(x+y)$ , and there are  $n$  factors, we conclude that  $i+j = n$  and both  $i$  and  $j$  are nonnegative. That is, all terms are of the form  $x^{n-k}y^k$ , where  $k$  goes from 0 to  $n$ . Now, how many ways can we pick exactly  $k$   $y$ 's from the  $n$  factors  $(x+y)$ ? We are simply picking a subset of  $k$   $y$ 's from a set of  $n$   $y$ 's. The number of ways to do this is  $C(n, k)$ . Thus the expansion is  $\sum_{k=0}^n C(n, k)x^{n-k}y^k$ . ■

We hope you liked the second proof better — it reduces the Binomial Theorem to recognizing the definition of  $C(n, k)$ . Surprisingly, it doesn't seem to involve induction at all. However, that's a misleading perception. When you find two proofs of something, one clearly using induction and the other seemingly not, what has usually happened in the second proof is that the induction has been pushed back to something so well accepted that you don't immediately recognize it as an induction. Where is the induction in our second proof? In the claim that to multiply a string of binomial factors, you add all the terms you can create by taking one summand from each of the factors. For instance, for two factors you get

$$(a_1 + a_2)(b_1 + b_2) = a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2.$$

This fact may be very familiar, but a careful proof for arbitrarily many factors requires induction and a firm command of algebra. To get from  $n$  factors to  $n+1$ , you use the distributive law. See [29].

We said at the beginning of this section that the Binomial Theorem applies easily to binomials of the form  $(ax+by)^n$  and we ask you to show this in a problem [30].

We promised a statement of the Binomial Series Theorem.

**Theorem 2, Binomial Series Theorem.** If  $\alpha$  is any *real* number, and  $|y| < |x|$ , then

$$(x+y)^\alpha = \sum_{k=0}^{\infty} \binom{\alpha}{k} x^{\alpha-k} y^k, \quad (6)$$

where  $\binom{\alpha}{k}$  is defined as in Eq. (4).

Actually, what one usually sees is the special case

$$(1+x)^\alpha = \sum_{k=0}^{\infty} \binom{\alpha}{k} x^k \quad \text{when } |x| < 1. \quad (7)$$

(The number 1 has been substituted for  $x$  in Eq. (6) and then  $y$  has been renamed  $x$ .) This case isn't really any less general, because for any  $x$  and  $y$  we may write

$$(x+y)^\alpha = x^\alpha \left(1 + \frac{y}{x}\right)^\alpha.$$

Thus instead of using  $x$  and  $y$  in Eq. (6), we may use Eq. (7) with  $y/x$  in place of  $x$ , and then multiply the result by  $x^\alpha$ . Also, while the condition  $|y| < |x|$  in the general theorem may seem mysterious, the equivalent condition  $|x| < 1$  in the special case is quite plausible: If  $|x| < 1$ , then the factors  $x^k$  go to 0 as  $k$  increases, so it is credible that the sum effectively stops growing after some number of terms. Conversely, if  $|x| \geq 1$ , then  $x^n$  does not go to 0, so it is credible that there is no limiting value for the sum as we add more terms. But as we've said, we are in no position to give careful proofs of any of these claims.

## The Multinomial Theorem

Why stop at a formula for  $(x+y)^n$ ? How about  $(x+y+z)^n$ , or even  $(\sum_{k=1}^m x_k)^n$ ? The second proof of the Binomial Theorem suggests how to proceed. For instance, in the case of  $(x+y+z)^n$ , every term in the expansion should be of the form  $C(n; i, j, k)x^i y^j z^k$ , where  $i$ ,  $j$ , and  $k$  are nonnegative integers summing to  $n$ , and  $C(n; i, j, k)$  is the number of ways to pick  $x$  from  $i$  of the  $n$  factors,  $y$  from  $j$  of them, and  $z$  from the remaining  $k$  of them. The only remaining step is to find a formula for  $C(n; i, j, k)$ , which is called a trinomial coefficient. Let's begin with a general definition.

---

**Definition 1.** The **trinomial coefficient**  $C(n; i, j, k)$  is the number of ways to divide  $n$  distinct objects into a sequence of three sets, the first set with  $i$  elements, the second with  $j$ , and the third with the remaining  $k = n - i - j$ . It is often written

$$\binom{n}{i, j, k}.$$

Similarly, the **multinomial coefficient**  $C(n; k_1, k_2, \dots, k_m)$ , often written

$$\binom{n}{k_1, k_2, \dots, k_m},$$

is the number of ways to divide  $n$  distinct objects into a sequence of  $m$  sets, with  $k_i$  elements in the  $i$ th set.

---

*Note.* The division of the four distinct objects  $a, b, c, d$  into

$$\{a\}, \quad \{b\}, \quad \{c, d\} \tag{8}$$

is different than the division into

$$\{b\}, \quad \{a\}, \quad \{c, d\} \tag{9}$$

because in (8)  $\{a\}$  is the first set, whereas in (9)  $\{a\}$  is the second set. Thus the number  $C(4; 1, 1, 2)$  is at least 2. Similarly, while it turns out that the numbers  $C(4; 1, 1, 2)$  and  $C(4; 1, 2, 1)$  are the same, they count different divisions of 4 objects: every division counted by  $C(4; 1, 1, 2)$  has one object in the second set (for instance, (8) above), while every division counted by  $C(4; 1, 2, 1)$  has two objects in the second set (for instance,  $\{a\}, \{c, d\}, \{b\}$ ).

---

**Theorem 3.** For all nonnegative integers  $n$ ,

$$(x+y+z)^n = \sum_{i+j+k=n} \binom{n}{i, j, k} x^i y^j z^k, \tag{10}$$

where the sum is understood to range over all triples of nonnegative (but not necessarily distinct) integers  $i, j, k$  that add to  $n$ . More generally,

$$(x_1+x_2+\cdots+x_m)^n = \sum_{k_1+k_2+\cdots+k_m=n} \binom{n}{k_1, k_2, \dots, k_m} x_1^{k_1} x_2^{k_2} \cdots x_m^{k_m}, \tag{11}$$

where each  $k_i$  must be a nonnegative integer.

---

Eq. (11) is called the **Multinomial Theorem**. The special case, Eq. (10), is sometimes called the **Trinomial Theorem**.

**PROOF** It suffices to prove the general case, Eq. (11). When we multiply  $n$  sums together, as on the LHS of Eq. (11), the result is the sum of all terms obtained by multiplying together one term from each sum. For instance,

$$\begin{aligned} (x_1+x_2+x_3)(x_1+x_2+x_3)(x_1+x_2+x_3) \\ = x_1 x_1 x_1 + x_1 x_1 x_2 + x_1 x_1 x_3 + x_1 x_2 x_1 + x_1 x_2 x_2 + x_1 x_2 x_3 + \\ \cdots + x_3 x_3 x_1 + x_3 x_3 x_2 + x_3 x_3 x_3. \end{aligned} \tag{12}$$

The RHS of Eq. (12) contains like terms that can be combined. For example,  $x_1^2 x_2$  appears three times, once as  $x_1 x_1 x_2$ , once as  $x_1 x_2 x_1$  and once as  $x_2 x_1 x_1$ , the first two shown and the last included in the ellipsis. Similarly each term on the RHS of (11) is the product of a certain number of  $x_1$ 's,  $x_2$ 's,  $\dots$ ,  $x_m$ 's, with  $n$  of these factors altogether (counting multiplicities). Thus the RHS will contain the term  $x_1^{k_1} x_2^{k_2} \cdots x_m^{k_m}$  as many times as we can pick  $x_1$  from  $k_1$  of the copies of  $S = x_1 + x_2 + \cdots + x_m$ , and pick  $x_2$  from  $k_2$  other copies of  $S$ , and so on. But this is just the number of ways to divide the set of  $n$  copies of  $S$  into a sequence of  $m$

sets, of sizes  $k_1, k_2, \dots, k_m$ , respectively. By definition, this is  $\binom{n}{k_1, k_2, \dots, k_m}$ . Thus, after combining like terms, we obtain the RHS of Eq. (11). ■

*Note.* Continuing the Note after Definition 1, in expanding  $(x+y+z)^4$  we have that  $C(4; 1, 1, 2)$  is the coefficient of  $xyz^2$ , whereas  $C(4; 1, 2, 1)$  is the coefficient of  $xy^2z$ . As the proof above shows,  $C(4; 1, 1, 2)$  is the number of divisions of the four factors  $(x+y+z)$  into three sets where the first set contains those factors from which we choose  $x$ , and the second contains those from which we choose  $y$ . In  $C(4; 1, 1, 2)$  we are choosing  $y$  once; in  $C(4; 1, 2, 1)$  we are choosing  $y$  twice, so we are indeed counting different things.

Theorem 3 isn't of much help unless we can evaluate multinomial coefficients. The next theorem shows how.

**Theorem 4.**

$$\binom{n}{i, j, k} = \frac{n!}{i!j!k!}. \quad (13)$$

More generally,

$$\binom{n}{k_1, k_2, \dots, k_m} = \frac{n!}{k_1!k_2!\cdots k_m!}. \quad (14)$$

**PROOF** We outline an inductive proof. For a one-to-one correspondence, which gets the multinomial case all at once, see [32]. To verify Eq. (13), we pick the three subsets sequentially. For the first subset, we must pick  $i$  of the  $n$  elements. There are  $C(n, i)$  ways to do this (an ordinary binomial coefficient). Now, whatever  $n-i$  elements are left, there are  $C(n-i, j)$  ways to pick the second subset. At this point, no choices are left: The third subset must consist of all the remaining elements. By the Product Rule, we conclude that

$$\binom{n}{i, j, k} = \binom{n}{i} \binom{n-i}{j} = \frac{n!}{i!(n-i)!} \frac{(n-i)!}{j!(n-i-j)!} = \frac{n!}{i!j!(n-i-j)!}.$$

Since  $k = n-i-j$ , we have Eq. (13). Then we may prove Eq. (14) by induction with Eq. (13) as the basis case. The key idea of the inductive step is: Dividing  $n$  into  $m+1$  sets can be accomplished by first picking the set of size  $k_1$ , and then dividing the remaining  $n-k_1$  elements into  $m$  sets [31]. ■

## Problems: Section 4.6

The problems in this set are meant to be done by hand, or with brain power, so that you learn how the binomial theorems work. In practice, complicated evaluations are done more and more by machine, and you

may want to see how much is doable with whatever CAS is available to you.

- 1. **a)**  $(x+y)^5$       **b)**  $(x-1)^4$       **c)**  $(2x-1)^3$

2.  $\langle 1 \rangle$  Approximate to four decimal places  
 a)  $(1.01)^4$       b)  $(.98)^3$       c)  $(2.01)^3$

3.  $\langle 1 \rangle$  Show that  $\sum_{k=0}^n C(n, k)2^k = 3^n$ .

4.  $\langle 2 \rangle$  Simplify

- a)  $\sum_{k=0}^n (-1)^k C(n, k)2^k$
- b)  $\sum_{k=0}^n (-1)^k C(n, k)4^k$
- c)  $\sum_{k=0}^n (-1)^k C(n, k)2^{n-k}$
- d)  $\sum_{k=0}^n (-1)^k C(n, k)2^{n-k}3^k$
- e)  $\sum_{k=0}^n (-1)^k C(n, k)2^n3^k$

5.  $\langle 1 \rangle$  What is the effective rate of interest per year if the “nominal” rate is 12% and compounding is monthly? (If a nominal rate of  $r$  is compounded  $k$  times a year, that means the principal is multiplied by  $(1+\frac{r}{k})$  a total of  $k$  times. Thus a nominal rate of 20% compounded semiannually means that the principal is multiplied by  $(1+.1)^2 = 1.21$ , so the effective (or “annualized”) interest rate is 21%).

6.  $\langle 2 \rangle$  Show that  $(a + \epsilon)^n \approx a^n + n\epsilon a^{n-1}$ . Here  $\epsilon$  stands for a number small enough relative to  $a$  that  $\epsilon^2$  and higher powers of  $\epsilon$  are negligible. How is this problem relevant to the previous problem?

7.  $\langle 2 \rangle$

- a) Eq. (2) can be put nicely into words as a statement about the number of odd and even subsets. Make such a statement.
- b) What does Eq. (2) tell you about the size of  $\sum_{k \text{ even}}^n C(n, k)$ ?

8.  $\langle 1 \rangle$  Use the Binomial Theorem to prove that

$$(1+x)^n \geq 1+nx$$

for any  $x \geq 0$  and  $n$  nonnegative. This was proved directly by induction in Example 2, Section 2.2.

9.  $\langle 1 \rangle$  Use the Binomial Theorem to prove that

$$(1+x)^n > 1+nx$$

for any  $x > 0$  and  $n > 1$ .

10.  $\langle 2 \rangle$  Approximate  $\sqrt{1.5}$ . Let  $x = 1$  and  $y = .5$  in the Binomial Series and evaluate terms until it seems there is no further change in the first three decimal places. Compare with the answer from a calculator.

11.  $\langle 2 \rangle$  If  $\epsilon$  is much smaller than  $a$ , a good approximation of  $\sqrt{a^2 + \epsilon}$  is

$$S = a + \frac{\epsilon}{2a}.$$

Justify this claim by simply squaring  $S$ . Now explain how you might *discover* this fact by using the Binomial Series Theorem.

12.  $\langle 2 \rangle$  Approximate  $\sqrt[3]{9}$  using the Binomial Series Theorem. Compute four terms. Let  $x = 8$  and  $y = 1$ , or equivalently, note that  $\sqrt[3]{9} = 2\sqrt[3]{1 + \frac{1}{8}}$  and use the special case of the Binomial Series Theorem. Compare your answer with that from a calculator.

13.  $\langle 2 \rangle$  In the first proof of Theorem 1, we hid some things under the rug. We said our  $j$  could be any integer from 0 to  $n+1$  but, in fact, if  $j = 0$  or  $j = n+1$ , then some of the things we did need more explanation. What things? What’s the explanation?

14.  $\langle 2 \rangle$  Prove the version of the Binomial Theorem displayed right after Eq. (1) — the version with ascending powers of  $x$  — four ways:

- a) Substitute  $y$  for  $x$  and  $x$  for  $y$  in Eq. (1).
- b) Substitute  $n-k$  for  $k$  in Eq. (1).
- c) Do an induction similar to the first proof of the Binomial Theorem in the text.
- d) Give a combinatorial argument parallel to the second proof in the text.

15.  $\langle 1 \rangle$  Compute

$$\binom{5}{2, 2, 1} \quad \text{and} \quad \binom{6}{1, 2, 3}.$$

16.  $\langle 1 \rangle$  Are  $C(6; 1, 2, 3)$  and  $C(6; 3, 2, 1)$  the same number? Do they count the same thing?

17.  $\langle 1 \rangle$  Answer [11, Section 4.3] again, using multinomial coefficients.

18.  $\langle 2 \rangle$  Simplify

- a)  $\sum_{i+j+k=n} C(n; i, j, k)$ ,
- b)  $\sum_{i+j+k=n} (-1)^k C(n; i, j, k)2^j / 3^{i+j}$

19.  $\langle 2 \rangle$  Express  $C(n, k)C(n-k, j)$  as a trinomial. Do the same for  $C(n, k)C(k, j)$ .

20.  $\langle 1 \rangle$  Find a simple formula for  $C(-1, k)$ .

21.  $\langle 2 \rangle$  Verify that

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

22. ⟨3⟩

- a) Verify that

$$\binom{-1/2}{k} = \frac{(-1)^k (2k)!}{2^{2k} (k!)^2}.$$

- b) Find a similar formula for  $C(1/2, k)$ .

23. ⟨2⟩  $C(n, k)$  was initially defined for integers  $0 \leq k \leq n$  (Def. 2, Section 4.4). We've extended that definition twice, in Def. 1, Section 4.5 and in Eq. (4). Are there any pairs  $(n, k)$  to which the two extensions assign conflicting values?

24. ⟨3⟩

- a) When  $|x| < 1$ , the infinite geometric series  $\sum_{k=0}^{\infty} x^k$  sums to  $1/(1-x)$ . Show that this fact is a special case of the Binomial Series Theorem.

- b) Find an infinite series which sums to  $1/(1-x)^2$  when  $|x| < 1$ .

- c) Generalize.

25. ⟨1⟩ Since  $x^{\alpha-k}y^k = x^{\alpha}(y/x)^k$ , the Binomial Series Theorem may be rewritten as: If  $|y| < |x|$  then

$$(x+y)^{\alpha} = x^{\alpha} \sum_{k=0}^{\infty} \binom{\alpha}{k} \left(\frac{y}{x}\right)^k.$$

This form of the theorem makes the condition  $|y| < |x|$  less mysterious. Why?

26. ⟨1⟩ Implicit in our introduction of multinomial coefficients is a new meaning for binomial coefficients. Specifically, there ought to be a binomial coefficient  $C(n; i, j)$ , where  $j = n-i$ , and it ought to be the number of ways to partition an  $n$ -set into two subsets of size  $i$  and  $j$ . But the binomial coefficient we talked of previously, and wrote as  $C(n, i)$ , was defined as the number of ways to pick out *one* subset of size  $i$  from an  $n$ -set. Is there a contradiction between the two uses of “binomial”?

27. ⟨3⟩ Prove the Trinomial Theorem by noting that

$$(x+y+z)^n = [x+(y+z)]^n$$

and applying the Binomial Theorem twice.

28. ⟨3⟩ Think up and prove analogs of Theorems 1 and 2, Section 4.5, for trinomial coefficients. Ever heard of Pascal's pyramid?

29. ⟨4⟩ In the second proof of the Binomial Theorem, we hid the induction in the claim that to multiply a string of binomial factors, you add all the terms you can create by taking one summand from each of the factors. Make the induction explicit by proving this claim inductively. Start by stating the claim as an equation; the LHS should be  $\prod_{k=1}^n (a_k + b_k)$ .

30. ⟨2⟩ How does the statement of the Binomial Theorem change when what you wish to expand is  $(ax+by)^n$ ?

31. ⟨2⟩ Complete the proof of Theorem 4.

32. ⟨2⟩ We claim there is a one-to-one correspondence between the number of ways to partition an  $n$ -set of distinct elements into  $m$  distinct subsets of sizes  $k_1, \dots, k_m$  and the number of ways to permute the letters of a word of length  $n$ , when there are only  $m$  distinct letters in the word and the  $i$ th letter occurs  $k_i$  times. Justify this claim and use it to prove Eq. (14).

The following problems are for students who know calculus.

33. Derive Theorem 4, Section 4.5, again as follows. Take the Binomial Theorem, regard  $x$  as a constant, and differentiate with respect to  $y$ . Now make an appropriate substitution for  $x$  and  $y$ .

34. ⟨3⟩ Derive a new theorem, similar to Theorem 4, Section 4.5, by differentiating the Binomial Theorem with respect to  $x$ . Can you also prove your new theorem directly from the Binomial Theorem without calculus?

35. ⟨3⟩ Evaluate

a)  $\sum_{k=0}^n k(k-1) \binom{n}{k}$       b)  $\sum_{k=0}^n k^2 \binom{n}{k}$

c)  $\sum_{k=0}^n \binom{n}{k} / (k+1)$ . (Careful!)

36. ⟨3⟩ Assuming that  $(fg)' = f'g + fg'$ , prove Leibniz' formula for the  $n$ th derivative of a product:

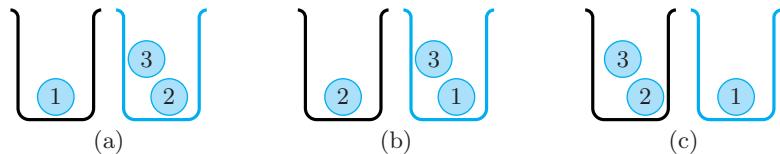
$$(fg)^{(n)} = \sum_{k=1}^n \binom{n}{k} f^{(k)} g^{(n-k)}.$$

## 4.7 Four Common Problems with Balls and Urns

How can *any* problems with balls and urns be common? The answer is that they form paradigms. Many problems that don't on the face of it have anything to do with balls or urns are equivalent to problems of putting balls in urns. So if we can count the number of ways to place balls in urns, we can solve a lot of other counting problems.

We assume throughout this section that there are  $b$  balls and  $u$  urns. We also assume that each urn can hold any number of balls and that there is no order to the balls within an urn. These assumptions leave us with four situations: The balls can be distinguishable or indistinguishable, and, independently, the urns can be distinguishable or indistinguishable.

“Distinguishable” and “indistinguishable” are long words; sometimes “labeled” and “unlabeled” are used instead. In any event, Fig. 4.11 indicates what these words mean. The balls are numbered (distinguishing them), and the urns have different colors (distinguishing them). Thus Fig. 4.11(a), (b), and (c) represent different ways to put three balls into two urns. However, if we ignore the numbers on the balls (making them *indistinguishable*), then (a) and (b) are the same, but (c) is still different. If instead we ignore the colors (making the *urns* indistinguishable), then (a) and (c) are the same and (b) is different. If we ignore both numbers and colors, then all three configurations are the same.



Three ways to put 3 balls into 2 urns. How many of them are different depends on what is distinguishable.

There are, to be sure, further possibilities. For instance, some of the balls may be distinguishable and others not. Some urns may hold only a limited number of balls. The urns might instead be stacks, which means the balls are necessarily ordered within them, and this order might make a difference. The urns might be in a row, in which case *their* order might make a difference. But the four basic problems above will be enough to keep us busy for now.

Before we try to find counting formulas for these four paradigms, let's consider several problems that reduce to them. For now we only do the reductions. Later in the section we will solve most of the problems.

### EXAMPLE 1

How many different configurations are there in Towers of Hanoi with  $n$  rings?

Putting a ring on a pole is like putting a ball in an urn. The rings are distinguishable by size, so we must use distinguishable balls. The poles are distinguishable

too — it's not enough, for instance, to get all the rings on *some* pole; it's got to be the target pole. True, the rings are ordered on the pole, whereas the balls are unordered in the urn. However, the rules of TOH result in a unique ordering. That is, given any specific set of rings to put on a given pole, there is just one way to do it, just as there is only one way to put balls as an unordered set in an urn. In conclusion, we wish to count the number of ways to put  $n$  distinguishable balls in three distinguishable urns.

### EXAMPLE 2

How many solutions are there in nonnegative integers to the equation

$$x + y + z + w = 73?$$

Think of building up a solution unit by unit, that is, by starting with 73 1's and assigning them one at a time to the variables. So the 1's are like balls and the variables are like urns. What's distinguishable? In the end, if three 1's get assigned to  $x$ , it makes no difference if they are the first three 1's or the last three. But it does make a difference if they are assigned to  $x$  or to  $y$ . For example, (3,40,20,10) and (40,20,10,3) are considered different solutions. Thus we are asking: How many ways can 73 indistinguishable balls be put in 4 distinguishable urns?

### EXAMPLE 3

In chemistry, the electrons in a given ring of an atom can each be in one of several energy states. How many ways can six electrons be assigned to five energy states?

It's easy enough to think of electrons as balls and the states as urns. The energy states are distinguishable, because they correspond to different amounts of energy, and we can measure that. The harder question is: Do electrons act distinguishably? We think of them as distinct entities, so it is natural to suppose they will act distinguishably. However, it is an empirical fact that they don't. The appropriate model is putting *indistinguishable* balls in distinguishable urns, as in Example 2.

### EXAMPLE 4

How many ways can  $4n$  bridge players be split into  $2n$  teams for a round-robin tournament?

Putting a person on a team is like putting a ball in an urn. People are certainly distinguishable, so we need distinguishable balls. What about the teams (urns)? We can, and probably do, give them different names, but does it really make a difference if Jones and Smith are called Team *A* and White and Roberts are called Team *B* — instead of vice versa? No. Therefore we are putting  $4n$  distinguishable balls into  $2n$  indistinguishable urns. Since a bridge team always has two members, we must further insist that every urn gets exactly two balls. (We will not include such restrictions when we try to solve the balls and urns problems later, but in fact, for distinguishable balls in indistinguishable urns, such restrictions make the problem easier. [18])

### EXAMPLE 5

A **partition** of a positive integer  $n$  is any set of positive integers which sum to  $n$ . Thus  $\{5, 1, 1\}$  and  $\{4, 3\}$  are partitions of 7. So is  $\{3, 4\}$ , but it's not a different partition from  $\{4, 3\}$ . How many partitions are there of  $n$ ?

This is like Example 2, except that

- we are not restricted to a fixed number of summands;
- the summands are not distinguished, e.g.,  $3 + 4$  is not different from  $4 + 3$ ; and
- each summand must be positive.

At most there are  $n$  summands (since the longest way to partition  $n$  into positive integers is to divide it into  $n$  1's), but there might be only one ( $n$  itself). For instance, there are 7 partitions of 5:

$$\begin{array}{c} 5 \\ 4 \ 1 \\ 3 \ 2 \\ 3 \ 1 \ 1 \\ 2 \ 2 \ 1 \\ 2 \ 1 \ 1 \ 1 \\ 1 \ 1 \ 1 \ 1 \ 1 \end{array}$$

(We have written the partitions in the traditional form: in descending order of size of the parts. Partitions are unordered, but writing them down imposes an order, so the best way to avoid unintended repetitions is to use a standard order.)

We can make Example 5 look much more like Example 2 by insisting that there be exactly  $n$  integers in the partition, but allowing 0's. Thus the list above becomes

$$\begin{array}{cccccc} 5 & 0 & 0 & 0 & 0 \\ 4 & 1 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 \\ 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{array}$$

Now Example 5 is the same as Example 2, except that the variables here are indistinguishable. In short, we are asking for the number of ways to put  $n$  indistinguishable balls (1's) into  $n$  indistinguishable urns (the variables).

What we have called partitions are sometimes called **unordered partitions** to make clear the distinction from the situation in Example 2. However, the study of unordered partitions is such an old and venerable part of mathematics (it's considered part of number theory) that usually they are just called partitions.

## EXAMPLE 6

How many ways are there to pick a combination of  $k$  things from a set of  $n$  distinct things if repetitions are allowed? (This is just the number  $C^*(n, k)$  of multisets discussed at the end of Section 4.4.)

We turn this problem into balls and urns as follows. The urns are the  $n$  distinct types of objects. The balls are the  $k$  individual selections of objects. In other

words, each time we put a ball into urn  $j$ , that means we select distinct object  $j$  another time. Since we are counting combinations, not permutations, the order of selection makes no difference. For instance, if two balls go in urn  $j$ , we don't care if they are the first two or the last two. Thus the balls are indistinguishable. In summary,  $C^*(n, k)$  is just the number of ways to put  $k$  indistinguishable balls into  $n$  distinguishable urns.

Alternatively, it suffices to note that  $C^*(n, k)$  equals the number of nonnegative integer solutions to

$$x_1 + x_2 + \cdots + x_n = k. \quad (1)$$

The point is: To uniquely determine a  $k$ -multiset from  $n$  distinct objects, we merely need to determine how many objects we will choose of each type. Let  $x_j$  be the number we choose of type  $j$ . Then our decision amounts to a solution to Eq. (1). Conversely, every solution to Eq. (1) defines a multiset. Finally, from Example 2, we already know that the number of solutions is the number of ways to put  $k$  indistinguishable balls in  $n$  distinguishable urns.

These correspondences are illustrated in Fig. 4.12 for a few of the possible arrangements with  $b = 6$  and  $u = 3$ .

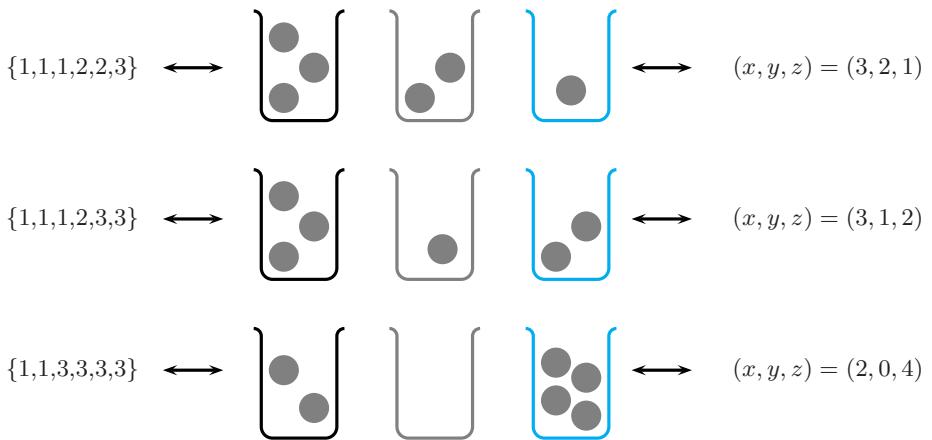


Illustration of the one-to-one correspondence from 6-multisets with 3 distinct objects, to ways to put 6 indistinguishable balls into 3 distinguishable urns, to the positive integer solutions to  $x + y + z = 6$ .

Enough examples. Let's now stick to balls and urns and try to find formulas for the counts in each of our four situations. For each situation, we will state our goal as a question, answer it as best we can, and then state the solution to each example above that is solved by the answer to that question. At the end of the section we summarize the results.

**Question 1:** In how many ways can  $b$  distinguishable balls be put in  $u$  distinguishable urns?

**FIGURE 4.12**

Since the balls are distinguishable, we can talk of a first ball, a second ball, and so on; furthermore, we can decide independently for each ball which urn to put it in. Since the urns are distinguishable, there are  $u$  choices per ball. By the product rule, the total number of choices is  $u^b$ .

**Solution to Example 1** The number of TOH configurations is  $3^n$ . ■

**Question 2:** In how many ways can  $b$  indistinguishable balls be put in  $u$  distinguishable urns?

We are going to solve this by an ingenious one-to-one correspondence. In Section 4.8, Example 3, and then in Section 5.3, Example 5, we will show two more systematic ways to discover this formula. Also see [19] for another ingenious method. As always, the value of developing more advanced techniques is that more problems become solvable by standard applications of those techniques rather than requiring a flash of insight. But the insights are still pretty.

Since the urns are distinguishable, we can imagine them numbered from 1 to  $u$  and lined up from left to right by number. Drawn schematically, with sides of consecutive urns flush with each other, they look like



Now let's imagine an arrangement of the balls in the urns. As the balls are indistinguishable, we might just as well line up the balls in each urn from left to right. Our picture becomes, for example,



We can simplify this picture still further to



What do we see? A collection of balls and *walls*. We must begin and end with walls, we must have  $b$  balls, and we must have  $u - 1$  internal walls. (Why?) But the order of the  $b$  balls and  $u - 1$  internal walls is arbitrary. Every sequence of balls and internal walls is a schematic picture of a way of putting indistinguishable balls into distinguishable urns (distinguished by their order from left to right). Conversely, each way of putting indistinguishable balls into distinguishable urns reduces to a balls and walls picture by the process of pictorial abstraction that we've just gone through. So we merely have to count the number of ways to line up  $b$  balls and  $u - 1$  internal walls. There are  $b+u-1$  positions in the lineup, and  $b$  of them must be balls. Therefore the answer to Question 2 is  $C(b+u-1, b)$ . No calculations at all!

**Solution to Example 2** The number of solutions is

$$\binom{73 + 4 - 1}{73} = \frac{76!}{73!3!} = \frac{76 \cdot 75 \cdot 74}{3 \cdot 2 \cdot 1} = 70,300. \blacksquare$$

**Solution to Example 6**  $C^*(n, k) = C(n+k-1, k)$ . Combinations both with and without repetitions are counted by binomial coefficients. ■

**Question 3:** In how many ways can  $b$  distinguishable balls be put in  $u$  indistinguishable urns?

Watch closely. We will set up a many-to-one correspondence between this problem and the problem in Question 1. Namely, for each way to put the balls in indistinguishable urns, we can distinguish these urns in  $u!$  ways (just number the urns every possible way). Moreover, every arrangement of  $b$  distinguishable balls in  $u$  indistinguishable urns is associated in this way with some arrangement of  $b$  distinguishable balls in  $u$  indistinguishable urns — just take the numbers off the urns. Thus we have a  $u!$ -to-1 correspondence. So the answer to Question 3 is the answer to Question 1 divided by  $u!$ , or  $u^b/u!$ .

Convinced? We've warned you to be careful with the Division Rule (end of Section 4.3). Is our answer plausible? Every natural number up to  $u$  is a factor of its denominator, but only factors of  $u$  appear in the numerator. For instance, with  $u = 3$  and  $b = 2$ ,

$$\frac{u^b}{u!} = \frac{3 \cdot 3}{3 \cdot 2 \cdot 1} = \frac{3}{2}.$$

This isn't an integer. It can't be right!

The error is: Our correspondence is not always  $u!$ -to-1. Some configurations with indistinguishable urns match up with fewer than  $u!$  configurations with distinguishable urns. For instance, suppose that there are four balls, 1, 2, 3, 4, and three indistinguishable urns. One arrangement is



If we now label the urns by their positions, then arrangement (2) certainly does correspond to  $3! = 6$  different arrangements with distinguishable urns: Just consider each permutation of the urns. For instance, switching the last two urns, we obtain



This is a different arrangement of labeled urns than arrangement (2), because now balls 2 and 4 are in the third urn.

But suppose that our initial arrangement, before we give the urns labels by position, is



Now if we label the urns by position, and then switch the last two urns, we are left with exactly the same labeled arrangement: four balls in urn one and no balls in either urn two or urn three. Arrangement (3), viewed as having indistinguishable urns, corresponds to only three arrangements with distinguishable urns — there are only three choices for where the set of balls  $\{1, 2, 3, 4\}$  can be put.

Our  $u!$ -to-1 correspondence breaks down only when two or more urns are empty. So for the moment, let's just worry about arrangements in which *no* urn is empty. Specifically, define

$T(b, u)$  = the number of ways to put  $b$  distinguishable balls into  $u$  distinguishable urns, with no empty urns.

Also define

$S(b, u)$  = the number of ways to put  $b$  distinguishable balls into  $u$  indistinguishable urns, with no empty urns.

Then we really do have a  $u!$ -to-1 correspondence between the things counted by  $T(b, u)$  and  $S(b, u)$ , so

$$S(b, u) = \frac{1}{u!} T(b, u). \quad (4)$$

Next, let

$B(b, u)$  = the number of ways to put  $b$  distinguishable balls into  $u$  indistinguishable urns, with empty urns allowed.

Thus  $B(b, u)$  is just a name for the answer to Question 3. Because the urns are indistinguishable, if  $k$  of them are empty, it doesn't matter which  $k$ . It is as if we only had  $u-k$  urns to start with. Thus

$$B(b, u) = \sum_{j=0}^u S(b, j) = \sum_{j=0}^u \frac{T(b, j)}{j!}. \quad (5)$$

So what? We still haven't solved the problem. But we have reduced it to another problem — putting balls in urns with required ‘occupancies’ — a problem worth tackling for itself. (It shows up frequently in probability.) Fortunately, it can be solved, using Inclusion-Exclusion, a technique we introduced in Section 4.3 but which we leave mostly to a second course in discrete math. The answer is

$$T(b, j) = \sum_{k=0}^j (-1)^k \binom{j}{k} (j-k)^b, \quad (6)$$

and thus the answer to Question 3 is

$$B(b, u) = \sum_{j=0}^u \sum_{k=0}^j \frac{(-1)^k}{j!} \binom{j}{k} (j-k)^b.$$

(For a derivation of (6), see [17, Supplement].)

In the preceding equations,  $S$  stands for the mathematician James Stirling (1692–1770), and  $S(b, u)$  is known as a **Stirling number of the second kind**. (Stirling numbers of the first kind are mentioned in the analysis of Algorithm MAXNUMBER in Section 6.8.) The  $B$  above stands for E. T. Bell, a 20th-century mathematician.

**Question 4:** In how many ways can  $b$  indistinguishable balls be put in  $u$  indistinguishable urns?

Let us call the answer  $p(b, u)$ . Note that the answer to Example 5 is  $p(n, n)$ . Thus  $p(b, u)$  is called a partition function.

There are many beautiful theorems about such partition functions. There are algorithms that allow you to list all the partitions of a given size (if you have the time and paper) [25] and recurrences which allow you to compute any particular values of  $p(b, u)$  [26]. There are also *asymptotic results*; these describe the rate of growth of partition functions very precisely. Many of these results are very deep, and they depend on both continuous and discrete methods. But there are very few closed-form formulas in partition theory. In particular, there is no nice, closed-form answer to Question 4. An attempt to find a formula by taking the answer to Question 2 and dividing by  $u!$  breaks down even more than the analysis right after the statement of Question 3 [32].

We summarize the results about balls and urns in Fig. 4.13.

|                   |  | Balls           |                                                                    |                        |
|-------------------|--|-----------------|--------------------------------------------------------------------|------------------------|
|                   |  | Urns            | Distinguishable                                                    | Indistinguishable      |
|                   |  | Distinguishable | $u^b$                                                              | $\binom{b+u-1}{b}$     |
| Indistinguishable |  |                 | $\sum_{j=0}^u \sum_{k=0}^j \frac{(-1)^k}{j!} \binom{j}{k} (j-k)^b$ | No closed-form formula |

**FIGURE 4.13**

Summary of solutions for  $b$  balls in  $u$  urns.

## Problems: Section 4.7

1.  $\langle 1 \rangle$  Finish Example 3.
2.  $\langle 1 \rangle$  How many legitimate configurations are there for TOH with  $n$  rings and four poles?
3.  $\langle 2 \rangle$  How many possible 4-letter “words” are there using the lower case roman alphabet? What type of ball and urn problem is this?
4.  $\langle 1 \rangle$  Biochemists have found that, in cell replication, all amino acids are coded for using just four nucleotide bases, known as U, C, A and G. Each string of three nucleotide bases codes for a specific amino acid. At most how many amino acids are there? (In actuality there are fewer, because several base triples code for the same acid. Also, some triples code for “end of chain.”)
5.  $\langle 2 \rangle$  For each of our four models, in how many

ways can four balls be put in three urns? (For the two models with indistinguishable urns, answer by actually listing the possibilities. Be systematic!)

6.  $\langle 3 \rangle$  Do [5] again for six balls and four urns. The case of distinguishable balls in indistinguishable urns will now have too many cases to list out. So analyze indistinguishable balls and indistinguishable urns first and, for each possibility, figure out how many cases this would split into if the balls were distinguishable.
7.  $\langle 1 \rangle$  How many ways can you put  $b = b_1 + b_2 + \dots + b_u$  indistinguishable balls into  $u$  distinguishable urns if each urn  $i$  must be given  $b_i$  balls?
8.  $\langle 1 \rangle$  How many ways are there to put  $b$  indistinguishable balls into  $u$  distinguishable urns if each urn may hold at most one ball? *Hint:* This is

actually much easier than Question 2, where urn capacity is unrestricted.

9. ⟨2⟩ How many different “code words” of length  $D + d$  can be made using  $D$  dots and  $d$  dashes? Answer this question by first interpreting it as a problem about putting balls in urns. (This is a hard way to answer a simple question, but we want you to practice changing problems from one representation to another.)

10. ⟨3⟩ How many solutions are there to  $x+y+z+w = 10$  if

- a) the unknowns must be nonnegative integers?
- b) the unknowns must be positive integers?
- c) the same as a), except that  $x + y$  must also equal 6? Hint: Temporarily think of  $x + y$  as a single variable.

11. ⟨2⟩ Make up a ball and urn problem for which  $\binom{n}{i,j,k}$  is the answer. Be sure to state what is distinguishable. Generalize to arbitrary multinomials.

12. ⟨2⟩ Do permutations with repetitions correspond to one of our models? Explain.

13. ⟨1⟩ Make a chart illustrating the one-to-one correspondence between indistinguishable balls in distinguishable urns and “balls and walls” in the case  $b = 3, u = 3$ . That is, draw pictures of all 10 arrangements for each problem and indicate by arrows which pairs correspond.

14. ⟨1⟩ In Example 5, there was a pattern to the order in which we listed the partitions. List the partitions of the number 4 following the same pattern. Do the same for the partitions of the number 6.

15. ⟨2⟩ Rubik’s Cube is a  $3 \times 3 \times 3$  cube constructed from 27 unit cubes so that each face of 9 unit squares can rotate freely. Initially, the cube is painted with a different color on each face, i.e., the 9 unit squares on each face get the same color. But then, as you rotate different faces, things get all mixed up! Getting it back in order can be quite a challenge, especially if you want to do it fast. This puzzle was a world-wide mania in the early 1980s.

- a) Considering just one face, how many different color arrangements are possible? That is, one arrangement has green in the top left, blue to its right, and so on. Don’t worry about

whether the arrangement can actually be obtained by rotating the faces.

- b) Considering just one face, how many different color frequencies are possible? For instance, one frequency is to have five red squares, four blue ones and no other colors.
- c) What do a) and b) have to do with balls and urns?

16. ⟨2⟩ How many ways are there to take four distinguishable balls and put two in one distinguishable urn and two in another if

- a) the order of balls within each urn makes a difference?
- b) the order does not make a difference?

17. ⟨2⟩ Repeat [16] for indistinguishable urns.

18. ⟨3⟩ Answer Example 4.

19. ⟨3⟩ Here is another way to find the number of combinations with repetitions. As shown on the left side of Fig. 4.12, such a combination may be depicted as a sequence that is increasing but not necessarily *strictly* increasing. The set of all  $(n, k)$ -combinations with repetitions is thus the set of all  $k$ -long increasing sequences whose elements are taken from  $1, 2, \dots, n$ .

We claim there is a one-to-one correspondence between the set of such sequences and the set of all  $k$ -long *strictly* increasing sequences whose elements are taken from  $1, 2, \dots, n, n+1, \dots, n+k-1$ . Namely, for each sequence of the first set, and for each position  $i$  from 1 to  $k$  in that sequence, add  $i-1$  to the number in the  $i$ th position. The result is the corresponding sequence in the second set. For instance, for the sequences in Fig. 4.12, the correspondence gives

$$\begin{aligned} 1, 1, 1, 2, 2, 3 &\longleftrightarrow 1, 2, 3, 5, 6, 8 \\ 1, 1, 1, 2, 3, 3 &\longleftrightarrow 1, 2, 3, 5, 7, 8 \\ 1, 1, 3, 3, 3, 3 &\longleftrightarrow 1, 2, 5, 6, 7, 8 \end{aligned}$$

Conversely, for each strictly increasing sequence of length  $k$  using 1 through  $n+k-1$ , subtracting  $i-1$  from the  $i$ th term gives a merely increasing sequence with elements taken from 1 through  $n$ . The number of such strictly increasing sequences is easy to count. How many are there? Why?

20. ⟨3⟩ Combinations with gaps. Count the number of  $k$ -subsets of  $\{1, 2, \dots, n\}$  such that no two consecutive integers occur in the set. For example,  $\{1, 3, 9\}$  is such a 3-subset (assuming  $n \geq 9$ ) but

$\{1, 8, 9\}$  is not. Hint: Reverse the idea presented in [19].

21. ⟨3⟩ In many lotteries, 6 different numbers are chosen at random from  $N$ , where  $N$  is typically 40 or 50. Your ticket is winning depending on how many of the chosen numbers you have. Many people have noticed that consecutive numbers are often chosen. What is the lowest value of  $N$  for which fewer than half of the possible 6-tuples contain at least one consecutive pair? See [20].

22. ⟨2⟩ Suppose that balls and urns are distinguishable, that urns are stacks (so the order of balls within them matters) and that even the order of the urns makes a difference. Again we ask: In how many ways can  $b$  balls be put in  $u$  urns? Try to solve this problem by a method similar to the one we used for counting indistinguishable balls in distinguishable (unstacked) urns.

23. ⟨2⟩ Consider the question: How many ways are there to put  $b$  different flags on  $u$  different flagpoles? Answer it using the following steps to get started.

- a) How many ways are there to put the first flag on some pole?
- b) How many ways are there to put up the second flag on some pole? Hint: The pole with the first flag on it has been divided by that flag into two parts, the part above that flag and the part below.
- c) How many ways are there to put up the third flag?
- d) Now answer the original question. The correct expression is often called the **rising factorial**.
- e) Reframe the original question as one about putting balls into urns. How is it related to [22]?

24. ⟨2⟩ How many ways can  $b$  indistinguishable balls be placed in  $u$  distinguishable urns if each urn must contain at least  $k$  balls?

25. ⟨4⟩ Write an algorithm for the procedure you followed in solving [14].

26. ⟨3⟩ Define  $p^*(b, u)$  to be the number of ways to put  $b$  indistinguishable balls into  $u$  indistinguishable urns so that no urn is empty. Recall from the discussion after Question 4 in the text that  $p(b, u)$ , with no star, is the number of ways to put  $b$  indistinguishable balls into  $u$  indistinguishable urns with no restrictions.

a) For what  $i$  and  $j$  is  $p^*(b, u) = p(i, j)$ ?

b) Express  $p(i, j)$  as a sum of  $p^*$  terms.

c) Deduce a recurrence for  $p^*(b, u)$ .

d) Deduce a recurrence for  $p(i, j)$ .

There can be more than one right answer to some of these parts.

27. ⟨2⟩ Let  $p(n)$  be the number of partitions of  $n$ , i.e., the quantity sought in Example 5. Use the results of [26d] to help you compute  $p(1)$  through  $p(6)$ . Check your answers against Example 5 and [14].

28. ⟨3⟩ Write an algorithm to automate your method from the previous problem.

29. ⟨4⟩ With  $p(b, u)$  as in Question 4 of the text, find closed-form formulas for

a)  $p(b, 1)$       b)  $p(b, 2)$       c)  $p(b, 3)$

That is, the formulas should be in terms of  $b$ , not in terms of other values of  $p(b, u)$ .

30. ⟨3⟩ Let  $p'(b, 2)$  be the number of (unordered) partitions of  $b$  into any number of parts, none of which is greater than 2. Show that  $p'(b, 2) = p(b, 2)$ . Does this relation generalize?

31. ⟨2⟩ Here is an argument that  $T(b, u) = P(b, u)u^{b-u}$ , where  $P(n, k)$  is the number of  $(n, k)$ -permutations and  $T(b, u)$  is defined as in this section. Since no urn can be empty, each urn must have at least one ball. So let's pick a sequence of  $u$  balls to be the initial balls in urns  $1, 2, \dots, u$ , respectively. That is, we want a permutation of  $u$  of the  $b$  balls, of which there are  $P(b, u)$ . Now, the remaining balls can be put in the urns without restriction. Since the balls and urns are distinguishable, there are  $u^{b-u}$  ways to do so.

Is this argument right? If not, where does it break down?

32. ⟨2⟩ In the specific case  $b = 5$ ,  $u = 3$ , write out the correspondence between the ways to put indistinguishable balls in distinguishable urns and the ways to put indistinguishable balls in indistinguishable urns. Verify the statement in the text that this correspondence fails to be an exact many-to-one correspondence on even more occasions than the correspondence discussed for Question 3.

33. ⟨2⟩ In Eq. (5), why did we start  $j$  at 0? Isn't this unnecessary because  $S(b, 0)$  and  $T(b, 0)$  will always be 0? Aren't there always no ways to put  $b$  balls into 0 urns?

## 4.8 Generating Functions

Generating functions are a way to use polynomial algebra to count. We introduce them in this section, and use them occasionally as an alternative method later in the book, for instance, on p. 446. In this section we first give some examples and then, in the second half, we will formalize what we have done.

In a nutshell, a generating function is a polynomial

$$a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

(or we allow the terms to go on forever) where the coefficients  $a_0, \dots, a_n$  count something of interest. Performing suitable algebraic operations on such polynomials creates other polynomials that turn out to correspond to more complicated counting problems. Using these other polynomials is sometimes an easier way to get a handle on those more complicated problems.

You've already seen the key idea about polynomial multiplication that makes this approach work, in the second proof of the Binomial Theorem (p. 359). The Binomial Theorem says that when you expand the product  $(x+y)^n$ , the coefficients have a combinatorial interpretation, that is, they count something. To show this, we analyzed where each term  $x^k y^{n-k}$  came from. Each such term arose by picking the  $x$  term from  $k$  of the factors  $(x+y)$  and the  $y$  term from the remaining factors. So when all the terms  $x^k y^{n-k}$  were combined, the coefficient could be interpreted in terms of how many ways one can pick things from the  $n$  different factors.

### EXAMPLE 1

Expand

$$(1 + x + x^2)^2(1 + x + x^2 + x^4) \tag{1}$$

and interpret the coefficients as counting something.

**Solution** Expanding the polynomial is no big deal, especially with a CAS:

$$1 + 3x + 6x^2 + 7x^3 + 7x^4 + 5x^5 + 4x^6 + 2x^7 + x^8.$$

But what, say, is the 4 in  $4x^6$  counting? Well, to expand we take either 0, 1, or 2 powers of  $x$  from each of the factors  $(1+x+x^2)$  in (1), and 0, 1, 2, or 4 from the last factor. Each way that we can do this to get a sixth power of  $x$  gives a term  $x^6$ . The  $4x^6$  term means that there must be 4 ways to pick a set of 6 things if 0, 1, or 2 come from a first source, and 0, 1, or 2 come from a second source, and 0, 1, 2, or 4 from a third source. In the language of Section 4.7, there are 4 ways to put 6 indistinguishable balls (i.e., the 6 factors of  $x$  in  $x^6$ ) into 3 distinguishable urns. (The urns are the three factors in (1); the two  $1+x+x^2$  factors are distinguishable in the sense that we must separately choose how many balls, i.e., the power of  $x$ , to take from each.) Similarly, there are 7 ways to put 3 indistinguishable balls in the urns with the same restrictions, because the expansion has a term  $7x^3$ .

This problem is small enough that we can check the results by brute force. For instance, for 6 balls the 4 configurations are

$$0, 2, 4 \quad 1, 1, 4 \quad 2, 0, 4 \quad 2, 2, 2$$

where 0, 2, 4 means 0 balls in the first urn, 2 in the second, 4 in the third. ■

## EXAMPLE 2

How many ways are there to make 50¢ change with 10 nickels, 5 dimes and 5 quarters? Assume nickels are indistinguishable, and likewise dimes and quarters.

**Solution** First, we may just as well assume there are only 2 quarters, since we cannot choose more than 2 to make up 50¢. The answer is the coefficient of  $x^{50}$  in

$$(1+x^5+x^{10}+\cdots+x^{45}+x^{50})(1+x^{10}+x^{20}+x^{30}+x^{40}+x^{50})(1+x^{25}+x^{50}). \quad (2)$$

The first factor represents the number of cents we can make using nickels alone. For instance, the term  $x^{10}$  is there because there is 1 way to make 10¢ with nickels, and there is no  $x^9$  term because there is no way to make 9¢ with nickels. Similarly, the second factor represents dimes alone, the third quarters alone. We want 50¢, so the terms we pick from the different factors must have their exponents add to 50. Multiplying out (2) completely is tedious by hand, but a CAS eats it up: the coefficient of  $x^{50}$  is 10. Actually, getting this one term by hand is not so hard by listing cases, but even one term would be too tedious by hand if we wanted 100¢, or 1000¢, or allowed pennies too. In such cases a CAS is indispensable. ■

Did you see the generating functions in these examples? The polynomials in parentheses in displays (1) and (2) are generating functions. In display (2), the first factor is the generating function for nickels (more precisely, for the number of ways to make various amounts of money with nickels alone). The second factor is the generating function for dimes, the third for quarters. Their product is the generating function for nickels, dimes, and quarters together. This is typically what happens: the generating function for a complicated situation is the product of generating functions for simpler situations. Theorem 2 below will make this precise.

You may object that the solutions in Examples 1 and 2 are brute force. The algebra just organizes the traditional case-by-case listing; indeed, if it weren't for CASs it might be more work to do this problem with the algebraic symbolism than without it. To this objection we have several answers. First, it's nifty that, when we cloak our counting problem in polynomials, it reduces to a *standard* algebraic operation — multiplication — so that off-the-shelf software can do it. Second, as we will see in the next example, generating functions don't always require expanding polynomials or other algebra that might be tedious by hand.

## EXAMPLE 3

Using generating functions that don't need CASs to expand them, count the number of ways to put 5 indistinguishable balls in 3 distinguishable urns. Generalize, i.e., answer Question 3 from Section 4.7 again.

**Solution** By the argument for Example 1, the first answer is the coefficient of  $x^5$  in

$$P^3, \text{ where } P = 1 + x + x^2 + x^3 + x^4 + x^5.$$

However, just as there was no harm in Example 2 in deleting quarters beyond the 2 we could use, here there is no harm in including powers of  $x$  higher than  $x^5$ , because

they won't effect the coefficient of  $x^5$  in the expansion. So we might as well go all the way and find the  $x^5$  term in

$$S^3, \text{ where } S = 1 + x + x^2 + x^3 + x^4 + x^5 + \dots.$$

An infinite polynomial such as  $S$  is called a **power series** (because it is a series of powers of the variable). You might think cubing a power series would be more work than cubing a polynomial, but not here, because  $S$  is an infinite geometric series and has a closed form:

$$S = \frac{1}{1-x}. \quad (3)$$

(If you don't recall a derivation of Eq. (3), see [6] or [7] or the discussion around Eq. (10) below.) So now the question is: Is there a simple way to find the coefficients of the expansion of

$$S^3 = \frac{1}{(1-x)^3} ? \quad (4)$$

Yes. Apply the Binomial Series Theorem (p. 359) with  $n = -3$ :

$$(1-x)^{-3} = \sum_{k=0}^{\infty} \binom{-3}{k} 1^{-3-k} (-x)^k,$$

so the term in  $x^5$  is  $\binom{-3}{5}(1)^{-8}(-x)^5 = \binom{-3}{5}(-1)^5x^5$  and the coefficient of  $x^5$  is

$$\binom{-3}{5}(-1)^5 = \frac{(-3) \cdot (-4) \cdot (-5) \cdot (-6) \cdot (-7)}{5!} (-1)^5 = \frac{3 \cdot 4 \cdot 5 \cdot 6 \cdot 7}{5!} = 21.$$

Now the generalization is easy. The number of ways to put  $b$  indistinguishable balls in  $u$  distinguishable urns is the coefficient of  $x^b$  in  $(1-x)^{-u}$ . By the Binomial Series Theorem, this number is

$$\binom{-u}{b}(-1)^b = \frac{u \cdot u+1 \cdots (u+b-1)}{b!} = \binom{u+b-1}{b}. \quad (5)$$

This isn't a full proof, since we didn't prove the Binomial Series Theorem, but the right-hand side of Eq. (5) is the same answer we did prove in Section 4.7 (see the solution for Question 2 on p. 369). If we hadn't already proved this formula, we could regard the right-hand side of (5) as a guess and prove it by induction. (See p. 429, where this induction is carried out after the same guess is obtained another way.) ■

## EXAMPLE 4

Do Example 2 again, with power series generating functions.

**Solution** We replace the polynomial generating functions in (2) with the following power series, thereby making the coefficients of the expansion correct regardless of how many cents we want.

$$G(x) = \left( \sum_{i=1}^{\infty} x^{5i} \right) \left( \sum_{j=1}^{\infty} x^{10j} \right) \left( \sum_{k=1}^{\infty} x^{25k} \right).$$

Each of the factor series is again geometric, with ratios  $x^5$ ,  $x^{10}$  and  $x^{25}$ . Thus

$$G(x) = \frac{1}{1-x^5} \cdot \frac{1}{1-x^{10}} \cdot \frac{1}{1-x^{25}}.$$

Unfortunately, there is no simple way to expand this by hand, but any good CAS can do it — and there is less to type than if you used `display(2)`. Because a CAS can only print a finite amount, you should ask it to expand  $G(x)$  through a specific term, say the  $x^{50}$  term, or just ask for the specific term you want.

## A Formal Approach

The essential idea is very simple: Let  $\{a_k\}$  be any sequence. With this sequence we associate a power series

$$G(s) = \sum_{k=0}^{\infty} a_k s^k. \tag{6}$$

The series in Eq. (6) is called the **generating function** of the sequence  $\{a_k\}$ . If  $\{a_k\}$  is finite with last term  $a_n$ , we understand  $a_k$  to be 0 in (6) for  $k > n$ .

Note we have used  $s$  in Eq. (6) whereas we used  $x$  previously. Of course, the *name* of the variable in an expression like Eq. (6) is really irrelevant except as it may have suggestive value. In this case  $x$  suggests a variable, whereas  $s$  does not; it is just a *symbol* which serves as a *placeholder* for the elements of the sequence  $\{a_k\}$ : each  $a_k$  is held in a different place by a different power of  $s$ . And we have used  $G(s)$  instead of  $P(s)$  because this is the standard notation for a single generating function (although we'll use other letters later when dealing with more than one generating function at a time).

By using generating functions, we turn each infinite sequence into a single unit, namely, its generating function. As we saw in Example 3, these units are easier to manipulate than the sequences, especially if we have closed forms for them. But since it is the sequences we are really interested in, the real question is: When we are done manipulating, can we **expand**, that is, get back to the sequence from the generating function?

Typically we want a formula for some sequence  $\{a_k\}$ , where the terms are derived in some complicated way from terms of simpler sequences. For instance (generalizing Example 2), suppose  $a_k$  is the number of ways to make  $k$  cents change with nickels, dimes, and quarters together. We already know the sequence of counts for making change with nickels alone, dimes alone, and quarters alone. So we turn these simpler known sequences into generating functions. Then, in the “generating function world”, the right thing to do is to multiply the functions. Finally, to find a formula for  $a_k$ , expand the resulting, more complicated generating function.

Before we can do any of this, we shall need to know the generating functions for some simple sequences that can then be used as the building blocks for generating functions of more complicated sequences. The next example considers some useful building blocks.

### EXAMPLE 5

What are the generating functions for the following sequences  $\{a_k\}$ ?

- i)  $a_k = 1$  for  $k = 0, 1, 2, \dots, n$ ;  $a_k = 0$  otherwise.

- ii)  $a_k = 1$ , for all  $k$ .
- iii)  $a_k = 1$ , when  $k$  is a multiple of  $m > 0$  and 0 otherwise.
- iv)  $a_k = a^k$ .

**Solution** The generating function for the first sequence is the finite geometric series

$$1 + s + s^2 + \cdots + s^n$$

for which the well-known closed form is

$$\frac{1 - s^{n+1}}{1 - s}.$$

The generating functions for the other three sequences are infinite geometric series. For (ii), the series is

$$G(s) = \sum_{k=0}^{\infty} s^k = \frac{1}{1 - s}. \quad (7)$$

For (iii), the series is

$$G(s) = \sum_{k=0}^{\infty} (s^m)^k = \frac{1}{1 - s^m}. \quad (8)$$

For (iv), the series is

$$G(s) = \sum_{k=0}^{\infty} a^k s^k = \sum_{k=0}^{\infty} (as)^k = \frac{1}{1 - as}. \blacksquare \quad (9)$$

But wait a minute. Doesn't the convergence of a geometric series depend on the ratio of successive terms of the series being less than 1 in absolute value? Whether this happens or not depends on  $s$ . Yet, we haven't said anything about values of  $s$  so far in this section. Indeed, we mentioned above that we don't think of  $s$  as a numerical variable at all. So how can we say  $1/(1 - s)$  is the answer to (ii)? And if we do say things like that, are we correct that the various manipulations we do to the resulting closed-form generating functions are OK?

Yes, we are correct, but all we'll do here is give a brief idea of a justification using the theory of "formal" power series, where one merely shows that a closed form is *algebraically* equivalent to a series. We say that, algebraically, power series  $G(s)$  equals  $1/f(s)$  if  $f(s)G(s) = 1$ . Specifically, to show that  $\sum_{k=0}^{\infty} s^k = 1/(1 - s)$ , we show that, if we multiply  $(1 - s)$  and  $\sum_{k=0}^{\infty} s^k$  by the usual rules of polynomial algebra, we get 1. Sure enough,

$$(1 - s)(1 + s + s^2 + s^3 + \cdots) = (1 + s + s^2 + \cdots) - (s + s^2 + s^3 + \cdots), \quad (10)$$

so everything cancels on the right except for 1. Similarly we can justify the other portions of Example 5 [22].

Our next step in discussing generating functions is to introduce the notion of **convolution** and show how it relates to the multiplication of generating functions.

---

**Definition 1.** Let  $\{a_k\}$  and  $\{b_k\}$  be two sequences (with  $k = 0, 1, 2, \dots$ ). Then the **convolution sequence**  $\{c_k\}$  is defined by

$$c_k = \sum_{j=0}^k a_j b_{k-j}. \quad (11)$$

---

### EXAMPLE 6

Let  $\{a_k\} = \{b_k\} = \{1\}$ , the sequence of all 1's. Then the convolution sequence is given by  $c_k = k + 1$  because the sum in Eq. (11) has  $k + 1$  terms, each of which is  $1 \cdot 1 = 1$ . ■

### EXAMPLE 7

Let  $\{a_k\} = \{\binom{k+2}{2}\}$  and  $\{b_k\} = \{1\}$ . Then from Eq. (11),

$$c_k = \sum_{j=0}^k \binom{j+2}{2} = \binom{k+3}{3},$$

where the second equality follows from [10, Section 4.5]. ■

Our first theorem about convolutions should help you understand why this is a useful concept.

---

**Theorem 1.** Let  $\{c_k\}$  be the convolution of  $\{a_k\}$  and  $\{b_k\}$ . Let  $a_k$  and  $b_k$  represent, respectively, the number of distinguishable ways to pick a subset of  $k$  objects from set  $A$  and from set  $B$ . Let  $A$  and  $B$  be disjoint and let anything in  $A$  be distinguishable from anything in  $B$ . Then  $c_k$  is the number of distinguishable ways to pick a subset of  $k$  objects from  $A \cup B$ .

---

**PROOF** To pick a set of  $k$  objects from  $A \cup B$ , we must pick  $j$  objects from  $A$  and  $k - j$  objects from  $B$ , where  $j$  is in turn  $0, 1, 2, \dots, k$ . For each  $j$ , the number of ways to pick  $j$  objects from  $A$  and  $k - j$  objects from  $B$  is  $a_j b_{k-j}$  (why?).

So summing  $a_j b_{k-j}$  over  $j$  gives the number of distinguishable ways to form a subset of  $k$  objects from  $A \cup B$ . But this sum is precisely Eq. (11). ■

Do you see how Theorem 1 relates to Examples 6 and 7? In Example 6, let  $A$  be an infinite collection of indistinguishable red balls and  $B$  be an infinite collection of indistinguishable blue balls. (Of course, every red is distinguishable from every blue.) Thus the sequence  $\{1\}$  represents the number of distinguishable ways to choose  $k$  balls from either  $A$  alone or  $B$  alone — for each  $k$  there is only 1 distinguishable way to pick  $k$  red balls because no two sets of  $k$  red balls are distinguishable. The convolution sequence represents the fact that there are  $k + 1$  ways to choose  $k$  balls, some red ones from  $A$  and the rest blue ones from  $B$ .

For Example 7, let  $A$  contain an infinite number of red, green, and yellow balls and let  $B$  be as before. Then choosing  $j$  balls from  $A$  is equivalent to putting  $j$  indistinguishable balls in 3 distinguishable urns labeled red, green, and yellow. (Why are these two ways of handling  $j$  balls equivalent?) We solved this sort of ball-urn problem in Section 4.7; the answer is  $C(j+2, j) = C(j+2, 2)$ . Again the convolution sequence represents the number of ways to choose  $k$  balls from  $A \cup B$ .

Our main result about convolution sequences is the following.

**Theorem 2.** Let  $\{a_k\}$  and  $\{b_k\}$  be two sequences with convolution  $\{c_k\}$  and let  $A(s)$ ,  $B(s)$ , and  $C(s)$  be the generating functions of the three sequences. Then

$$C(s) = A(s)B(s).$$

In other words, generating functions turn the complicated process of convolution into the simple process of multiplication.

**PROOF** To calculate the product of two infinite series  $A(s)$  and  $B(s)$  we recall Eq. (15), Section 0.4, where we displayed a formula for the product of two polynomials. The idea there, as in Examples 1 and 2, was to gather together all terms with the same power of the variable. Doing the same thing here, if we calculate the coefficient of  $s^k$  in

$$C(s) = A(s)B(s) = (a_0 + a_1 s + a_2 s^2 + \cdots)(b_0 + b_1 + b_2 s^2 + \cdots)$$

the result is

$$a_0 b_k + a_1 b_{k-1} + \cdots + a_k b_0. \quad (12)$$

But this is just  $c_k$  from Eq. (11). ■

Theorems 1 and 2 together say: if  $A(s)$  is the generating function for picking distinguishable subsets from a set  $A$ , and  $B(s)$  is the generating function for picking from set  $B$ , then  $A(s)B(s)$  is the generating function for distinguishable ways to pick from  $A \cup B$ . By induction [25], one may justify that  $\prod_{i=1}^n A_i(s)$  is the generating function for picking distinguishable subsets from  $\bigcup_{i=1}^n A_i$ .

For example, let us recapitulate the generalization in Example 3. Let  $A_i$  be an infinite set of indistinguishable balls of color  $i$ . Thus the generating function for picking balls when there are  $u$  colors (or equivalently,  $u$  urns) is  $1/(1-s)^u$ , but we know from Example 3 that

$$\frac{1}{(1-s)^u} = \sum_{k=0}^{\infty} \binom{u+k-1}{k} s^k. \quad (13)$$

As special cases,

$$\frac{1}{(1-s)^2} = \sum_{k=0}^{\infty} \binom{k+1}{k} s^k = \sum_{k=0}^{\infty} (k+1) s^k \quad (14)$$

and

$$\frac{1}{(1-s)^3} = \sum_{k=0}^{\infty} \binom{k+2}{k} s^k = \sum_{k=0}^{\infty} \frac{1}{2}(k+2)(k+1) s^k. \quad (15)$$

In fact, one can show that for any sequence  $\{p_k\}$  with  $p_k$  a polynomial in  $k$ , the generating function  $\sum p_k s^k$  has a closed form obtained by combining copies of Eq. (13) for various positive integers  $u$ . For instance, replacing  $k$  by  $k+1-1$  in order to use (14), we have

$$\sum_{k=0}^{\infty} ks^k = \sum_{k=0}^{\infty} (k+1)s^k - \sum_{k=0}^{\infty} s^k = \frac{1}{(1-s)^2} - \frac{1}{1-s} = \frac{s}{(1-s)^2}.$$

Similarly,

$$\begin{aligned} \sum_{k=0}^{\infty} k(k-1)s^k &= \sum_{j=-2}^{\infty} (j+2)(j+1)s^{j+2} && [j = k-2] \\ &= 2s^2 \sum_{j=0}^{\infty} \frac{1}{2}(j+2)(j+1)s^j && [\text{First 2 terms are 0}] \\ &= \frac{2s^2}{(1-s)^3}. && [(15)] \end{aligned}$$

See [2] for more examples.

## EXAMPLE 8

### Combinations with Limited Repetitions

Determine with generating functions the number of combinations of  $n$  distinct kinds of objects chosen  $k$  at a time, where up to  $r$  copies of each kind of object is allowed, and objects of the same kind are indistinguishable. Specifically, find the numerical answer when  $n = k = 3$  and  $r = 2$ .

**Solution** For one kind of object, there is exactly one way to choose it  $j$  times, for  $j = 0, 1, \dots, r$ . Therefore the generating function for any one kind of object is

$$1 + s + s^2 + \cdots + s^r = \frac{1 - s^{r+1}}{1 - s}. \quad (16)$$

Thus the generating function for  $n$  kinds of objects is

$$\left(\frac{1 - s^{r+1}}{1 - s}\right)^n = \frac{(1 - s^{r+1})^n}{(1 - s)^n}. \quad (17)$$

The number of ways to choose  $k$  objects is the coefficient of  $s^k$ . The numerator of the right side of (17) expands by the Binomial Theorem, while the denominator expands by Eq. (13). In specific cases, very few terms of the expansion are needed and the answer can be found by hand. For instance, for  $n = 3, r = 2$ , we get

$$\frac{(1 - s^{r+1})^n}{(1 - s)^n} = \frac{(1 - s^3)^3}{(1 - s)^3} = (1 - 3s^3 + \cdots)(1 + 3s + 6s^2 + 10s^3 + \cdots).$$

We don't need any more terms because we seek the coefficient of  $s^k = s^3$ . In fact, from the second factor, we need only  $10s^3$  (to multiply against 1 from the

first factor) and 1 (to multiply against  $-3s^3$ ), so we already did too much work! Combining the products  $-3s^3$  and  $10s^3$  gives a coefficient of 7 for the numerical answer sought. You can check this by actually listing all the combinations [16]. ■

## Problems: Section 4.8

1. ⟨1⟩ Give a closed form expression for each of the following generating functions.

- $-1 + 2s - 4s^2 + 8s^3 - 16s^4 + \dots$
- $s^2 + s^4 + s^6 + s^8 + s^{10} + \dots$
- $3s + 6s^2 + 12s^3 + 24s^4 + \dots$

2. ⟨2⟩ Express in closed form the generating functions for the following sequences, whose domain is  $N$ .

- $\{2^n\}$
- $\{1, 0, 1, 0, 1, 0, \dots\}$
- $\{0, 1, 0, 1, 0, 1, \dots\}$
- $\{1, 0, 0, -1, 0, 0, 1, 0, 0, -1, \dots\}$
- $\{k(k+1)\}$  Hint: Binomial Series Theorem.
- $\{k^2\}$  Hint: Write  $k^2$  as the difference of two expressions for which the Binomial Series Theorem tells you the generating function.

3. ⟨2⟩ Generating functions make perfectly good sense for finite sequences as well as infinite sequences. We have talked about infinite sequences only simply because those are the ones for which generating functions usually offer the most advantage over other methods. For  $n$  fixed, what is the generating function for the sequence

- $a_k = \binom{n}{k}, \quad k = 0, 1, 2, \dots, n?$
- $a_k = \binom{n}{k} c^k, \quad k = 0, 1, 2, \dots, n?$
- $a_k = \binom{n}{k} c^{n-k}, \quad k = 0, 1, 2, \dots, n?$

4. ⟨2⟩ Give an expression for  $a_k$  when the generating function for the sequence  $\{a_k\}$  is:

- $1/(1+s)$
- $1/(1-2s)$
- $s/(1-s^2)$
- $1/(1-s)^2$
- $1/(1+s)^3$ .

5. ⟨2⟩ Find the sequence  $\{a_k\}$  corresponding to the generating function

- $2/(1-s) + 3/(1-2s)$

- b)  $1/[(1-s)(2-s)]$ . Hint: Like all rational functions, this one has a partial fraction decomposition. In this case the decomposition is

$$\frac{c}{1-s} + \frac{d}{1-(s/2)}.$$

First find the constants  $c$  and  $d$ .

6. ⟨1⟩ In [7, Section 2.2], the formula for the sum of a *finite* geometric series was derived: for  $r \neq 1$ ,

$$\sum_{k=0}^n ar^k = \frac{a(r^{n+1}-1)}{r-1}.$$

Justify formula (3) by taking the limit as  $n \rightarrow \infty$  and concluding that when  $|x| < 1$ ,

$$\sum_{k=0}^{\infty} x^k = \frac{-1}{x-1} = \frac{1}{1-x}.$$

- ⟨2⟩ Show that formula (3) for the sum of an infinite geometric series is the special case of the Binomial Series Theorem (p. 359) when  $n = -1$ .
- ⟨1⟩ Let  $A$  be a set of 10 distinguishable balls. Let  $a_n$  be the number of distinguishable sets of  $n$  balls that can be chosen from  $A$ . Determine the closed form of the generating function for  $\{a_n\}$ .
- ⟨1⟩ Same as [8] except that the balls are indistinguishable.
- ⟨2⟩ Same as [8] except that  $A$  is infinite and consists of one black ball and otherwise indistinguishable red balls.
- ⟨2⟩ Use generating functions to find the number of ways to make 20 pence with any number of 1, 2 and 5 pence coins.
- ⟨2⟩
  - Find a closed form for the generating function for selecting at least one ball from an unlimited supply of identical balls.
  - Let  $c_{b,u}$  be the number of ways to put  $b$  indistinguishable balls into  $u$  indistinguishable urns if

each urn must contain at least one ball. Determine  $c_{b,u}$  two ways: using generating functions and by a combinatorial argument.

13. (3) For each pair of sequences, find a closed form expression for the general term of their convolution.

- a)  $\{0, 1, 0, 0, 0, 0, \dots\}, \{1, 1, 1, 1, \dots\}$
- b)  $\{1, 1, 0, 0, 0, 0, \dots\}, \{0, 1, 2, 3, 4, \dots\}$
- c)  $\{1, 1, 0, 0, 0, 0, \dots\}, \{1, 1, 0, 0, 0, 0, 0, \dots\}$
- d)  $\{0, 0, 1, 0, 0, 1, 0, 0, 1, \dots\}, \{1, 0, 0, 1, 0, 0, 1, 0, 0, \dots\}$

14. (3) Find the sequences that correspond to the following generating functions. Convolution is one way to proceed, but you may find others.

- a)  $G(s) = (1 + s + s^2)/(1-s)$ .
- b)  $G(s) = 1/[(1-s)(1-2s)]$

15. (1) Show by long division that

$$\frac{1 + 2s - 3s^2}{1 + s} = 1 + s - \frac{4s^2}{1 + s}.$$

Consequently, determine easily the sequence associated with the generating function on the left.

16. (2) Check by brute force that 7 is the correct answer to the numerical part of Example 8. Let the three objects be the letters a, b, c.

17. (2) Use the technique of Example 8 to calculate:

- a) The number of combinations of 4 objects, 3 at a time, with up to 2 repetitions of each.
- b) The number of combinations of 5 objects, 2 at a time, with up to 2 repetitions of each.

18. (2) Do [17] again, but this time don't work the generating functions into closed form. Just multiply out the original polynomials with a CAS. That is, work with the left-hand side of Eq. (16), not the right-hand.

19. (2) How many ways are there to pick 9 things from 3 types if there can be up to 4 things of type A, and 4 things of type B, but as many as we want of type C?

20. (2) Consider the Question: How many ways can we pick 5 balls from 3 distinguishable red balls and 4 distinguishable blue balls?

- a) The generating function for picking red balls alone is  $\sum_{k=0}^3 \binom{3}{k} x^k = (1+x)^3$ . Why?
- b) What is the generating function for picking blue balls alone?

- c) Why is the answer to the Question the coefficient of the product of these two generating functions?

- d) Expand the product generating function to obtain the answer.

- e) Answer the Question without generating functions.

21. (2) Let  $G(s) = \sum_k a_k s^k$  be the generating function for the number of ways to pick a combination of  $k$  things from  $n$  with repetitions allowed. (Note: We are regarding  $n$  as arbitrary but fixed and  $k$  as variable.) Find a closed form for  $G(s)$ . Hint: This counting problem is the same as another that we have already answered in the text.

22. (2) After Example 5 we discussed how to show *algebraically* that a closed form expression equals a power series, and we illustrated with such a proof for (7).

- a) Prove (8) by showing with algebra that

$$(1 - s^m) \sum_{k=0}^{\infty} (s^m)^k = 1.$$

- b) Prove (9) by algebra.

- c) In part i) of Example 5 we asserted that

$$1 + s + s^2 + \cdots + s^n = \frac{1 - s^{n+1}}{1 - s}.$$

What two *polynomials* should be multiplied together, and what should be their product, to verify this identity by similar algebra? Do it.

23. (3)

- a) Show that  $\binom{n+k}{k}$  is a *polynomial* of degree  $n$  in  $k$ . That is, if  $n$  is regarded as a constant and  $k$  is regarded as a variable, then  $p(k) = \binom{n+k}{k}$  is a polynomial. (The issue is the factors involving  $k$  in the denominator when  $\binom{n+k}{k}$  is expanded;  $p(k)$  is not a polynomial unless they go away.)

- b) Show that *any* polynomial  $p(k)$  of degree  $n$  may be written in the form

$$\sum_{j=0}^n c_j \binom{j+k}{k},$$

where the  $c_j$  are constants determined from  $p(k)$ . In fact, for each  $p(k)$  the  $c_j$  are unique. (In linear algebra terms, one says that

$$\binom{k}{k}, \binom{k+1}{k}, \dots, \binom{n+k}{k}$$

form a basis of the space of polynomials of degree at most  $n$ .) *Hint:* First find  $c_n$  so that  $p(k) - c_n \binom{n+k}{k}$  is a polynomial of degree  $n-1$ .

- c) Use part b) to show that every generating function of the form  $\sum_{k=0}^{\infty} p(k)s^k$ , where  $p(k)$  is a polynomial, has a closed form. *Hint:* Use (13).

24. ⟨3⟩ (Requires calculus) We know that

$$\frac{1}{1-s} = \sum_{k=0}^{\infty} s^k.$$

- a) By differentiating, show that

$$\frac{1}{(1-s)^2} = \sum_{k=0}^{\infty} (k+1)s^k.$$

- b) Continue differentiating and thereby show without the Binomial Series Theorem that

$$\frac{1}{(1-s)^n} = \sum_{k=0}^{\infty} \binom{n+k-1}{k} s^k$$

for all positive integers  $n$ .

25. ⟨3⟩ Let  $\{a_{1k}\}, \{a_{2k}\}, \dots, \{a_{mk}\}$  be sequences ( $k$  is the sequence variable each time) with corresponding generating functions  $A_1(s), \dots, A_m(s)$ . For all  $k$  define

$$c_{1k} = a_{1k}, \\ c_{ik} = \sum_{j=0}^k a_{ij} c_{i-1,k-j}, \quad i = 2, \dots, m.$$

- a) Generalize Theorem 1 by induction to show that if  $a_{ik}$  is the number of distinct ways to pick  $k$  items from set  $S_i$ , then  $c_{mk}$  is the number of distinct ways to pick  $k$  items from  $\bigcup_{i=1}^m S_i$ .

- b) Generalize Theorem 2 to show that the generating function of  $\{c_{mk}\}$  is  $\prod_{i=1}^m A_i(s)$ .

- c) Suppose that, for each  $i$ ,  $a_{ik}$  represents the number of ways of choosing  $k$  objects from an infinite set  $S_i$  of identical objects, but each set is distinguishable from every other set (Example 3). Use the results in a) and b) to show that the generating function for choosing  $k$  objects from  $m$  distinct objects with unlimited repetitions is  $\sum_{k=0}^{\infty} c_{mk}s^k = 1/(1-s)^m$ .

- d) Show how a similar argument to that in c) leads to Eq. (17).

26. ⟨2⟩ Give a generating function proof of

$$\sum_{i=0}^k \binom{n}{i} \binom{m}{k-i} = \binom{n+m}{k}.$$

The same identity was proved two other ways in [19, Section 4.5].

27. ⟨2⟩ Let  $p_n$  be the number of unordered partitions from Example 5, Section 4.7.

- a) Show that the generating function of  $\{p_n\}$  is

$$\prod_{k=1}^{\infty} \frac{1}{1-x^k}. \tag{18}$$

- b) Show that the terms of the simpler function

$$\prod_{k=1}^n \frac{1}{1-x^k}. \tag{19}$$

agree with those of (18) up through  $p_n x^n$ .

- c) Use a CAS to compute  $p_5$  and  $p_{10}$ . Your CAS may not be able to evaluate (19) without substituting specific integers for  $n$ , and may not be able to evaluate (18) at all. Try and see.

## 4.9 Combinatorial Algorithms

We've discussed permutations and combinations, and you've learned simple formulas for counting them. But suppose that you're asked to produce an actual permutation of some six things or to produce all permutations of these six things. How would you do that? These are intrinsically algorithmic questions, whose answers are very useful. Consider again the Traveling Salesman Problem of Example 3, Section 4.2. The only solution method we proposed was to compute the length

for each possible tour. If the cities (other than the salesman's home city) are labeled 1 through  $n$ , then there is a one-to-one correspondence between tours and permutations of 1 to  $n$ . For instance, if  $n = 5$ , the tour

home, city 2, city 4, city 3, city 5, city 1, home

corresponds to the permutation 24351. So the first step in the brute force approach of Section 4.2 is to generate all the permutations.

We pointed out that, for  $n = 50$ , this approach is well beyond the capacity of any computer. So what do you do? One alternative is to look for algorithms which run quickly and find near-optimal solutions. One such algorithm is to choose a *random* selection of tours and find the shortest among them. (Random means that each possible tour is equally likely to be chosen.) This shortest tour among those chosen probably won't be the best of all, but in general it will be close to best. To see this, first suppose you found just one random tour. On average, in the ranking of all tours from best to worst it would be right in the middle. (For instance, if there were just three tours and you picked one at random, 1/3 of the time you would pick the best tour, 1/3 of the time you would pick the second best, and 1/3 of the time you would pick the worst, so on average you would pick the second best.) If you found exactly two tours at random, it should be plausible that the better of them would be, on average, 1/3 of the way down the ranking from best to worst. Indeed, if you choose  $n$  things at random from a collection of  $N$  things, the average rank of your best choice is approximately  $1/(n+1)$  of the way down the ranking from rank 1 (best) to rank  $N$  (worst). (We say approximately because the exact average rank depends, among other things, on whether you allow repetitions in your random selection.) Specifically, if you take 99 random tours, only about 1/100 of the remaining  $N - 99$  tours are likely to be shorter than the shortest of your 99. This is true regardless of the size of  $N$ . The larger  $N$  is, the more work you have saved by only computing 99 tours.

The point is, it is very valuable to know how to produce a collection of random permutations. It suffices to know how to produce one random permutation, because as soon as you know how to produce one, you just repeat the process as many times as you want (say, 99 times).

Furthermore, you don't need to take such complicated problems as the Travelling Salesman Problem to see the usefulness of random combinatorial arrangements. For example, suppose that 100 students have signed up for a course and there is room for only 50. One fair way to choose the class is to make a random selection. This means a random combination of 50 things from the total of 100. So you need an algorithm to produce such random combinations. Note that once you have an algorithm to produce a random *permutation* of  $r$  things from a set of  $n$  things, you also have an algorithm to produce a random combination of  $n$  things,  $r$  at a time (why? [13]).

So much for motivation. The rest of this section is devoted to algorithms themselves. To keep things simple, we discuss algorithms for permutations only and only permutations of  $n$  things taken  $n$  at a time ( $n$ -permutations). (Why is it uninteresting to discuss algorithms for generating combinations of  $n$  things,  $n$  at a

time?) The problem set lets you try your hand at algorithms for other combinatorial objects and suggest a few more applications.

We break this section into two subsections. In the first we devise three algorithms for generating a single random permutation. We try to motivate how to devise such algorithms and show how to analyze them. In the second subsection we discuss an algorithm for generating all  $n$ -permutations. We also show that such generating algorithms have applications to seemingly nonalgorithmic questions.

To talk about random objects involves talking about probability, but we don't develop this subject carefully until Chapter 6. All the statements we'll make about probabilities and "average values" in this section are correct, but we provide plausibility arguments only. Do not hesitate to make similar "leaps of intuition" in doing the problems for this section. You may be wrong occasionally, but mathematics would not get very far if people never tentatively proposed solutions they couldn't completely justify.

In order to generate random permutations (or other random objects), we need some sort of randomizing ability. Since permutations are made out of numbers, it would be nice to have a **random number generator**. We will assume that our algorithmic language contains a function  $\text{RAND}[m, n]$  which, on each call, picks an integer between  $m$  and  $n$ , inclusive, with each choice equally likely and independent of all previous choices.

(How can such a function be implemented on a calculator or computer, both deterministic devices? It can't. What many calculators and just about all computers actually have are **pseudorandom number generators**, i.e., programs that produce numbers which look random for all essential statistical purposes but which aren't truly random. The mathematics that explains how this is done is very interesting but beyond the scope of this book.)

## Random Permutations

All right, let's start finding algorithms to produce random permutations. For each algorithm we will ask two questions:

1. Is it really a solution? That is, does it really produce a permutation of  $n$  things and is each of the  $n!$  permutations equally likely?
2. Is it an efficient solution? That is, can we think of another approach that takes fewer steps and/or is easier to state in algorithmic language?

Since we want to produce the numbers from 1 to  $n$  in random order, it seems reasonable to invoke  $\text{RAND}[1, n]$ . Can we just invoke it  $n$  times?

No. Since each choice is independent of previous choices, we might repeat one number and leave out another.

The simplest way around this problem is to keep a record of which numbers have already been picked, and each time we get a repetition, throw it away. Then keep going until all numbers have been picked once. This approach is easy to illustrate by listing the random numbers selected from left to right and putting slashes through numbers which get discarded. For instance, with  $n = 5$  the sequence

gives the permutation 31452.

Let's write this method up as Algorithm 4.3, PERMUTE-1. In the algorithm,

$$\text{Perm} = [\text{Perm}(1), \text{Perm}(2), \dots, \text{Perm}(n)]$$

will be a sequence that will hold the permutation when the algorithm terminates.  $\text{Flag}(i) = 0$  means that, so far, the integer  $i$  has not been placed in the permutation;  $\text{Flag}(i) = 1$  means that it has.

### Algorithm 4.3

#### Permute-1

|                                            |                |                                                 |
|--------------------------------------------|----------------|-------------------------------------------------|
| <b>Input</b>                               | $n$            | [Length of permutation]                         |
| <b>Output</b>                              | $\text{Perm}$  | [A random permutation given as a sequence]      |
| <b>Algorithm</b> PERMUTE-1                 |                |                                                 |
| $\text{Flag} \leftarrow 0$                 |                | [Set <i>all</i> components of Flag to 0]        |
| <b>for</b>                                 | $i = 1$ to $n$ | [ $i$ th pass will determine $\text{Perm}(i)$ ] |
| <b>repeat</b>                              |                |                                                 |
| $r \leftarrow \text{RAND}[1, n]$           |                | [Pick a random number                           |
| <b>endrepeat when</b> $\text{Flag}(r) = 0$ |                | until an unused one found]                      |
| $\text{Perm}(i) \leftarrow r$              |                |                                                 |
| $\text{Flag}(r) \leftarrow 1$              |                | [ $r$ has been used]                            |
| <b>endfor</b>                              |                |                                                 |

It should be clear that when this algorithm terminates, the output is a permutation: The loop invariant for the outer loop is that, for each  $i$ , the numbers  $\text{Perm}(1)$  through  $\text{Perm}(i)$  are distinct integers in the interval  $[1, n]$ . Since the algorithm terminates after  $n$  iterations of the outer loop, the final result is a permutation.

It may not be so obvious that each of the  $n!$  permutations is equally likely to be chosen. But we can argue informally as follows. After  $\text{Perm}(1)$  through  $\text{Perm}(k)$  have been chosen, only the  $n - k$  unchosen numbers are eligible, and the algorithm plays no favorites among them. Thus any one of them will be picked with probability  $1/(n-k)$ . Thus the probability that any given number will be the first number picked (when  $k = 0$ ) is  $1/n$ ; the probability that any given number among the remaining ones will be the second number picked is  $1/(n-1)$ , and so on. Multiplying, the probability that we will get any particular permutation is  $1/n!$ , just what we want.

So PERMUTE-1 is correct. But it's not efficient. Towards the end it spends a lot of time picking things that get discarded. How much time?

Let us find the expected number of RAND calls in each traverse of the outer loop. When  $k$  values of  $\text{Perm}$  have already been chosen (i.e., when the for-loop variable  $i$  has just become  $k+1$ ) the next call of RAND has probability  $(n-k)/n$  of giving an unchosen number. In general, if an action has probability  $p$  of success each time it is tried, we hope it seems reasonable that, on average, it will take  $1/p$

repetitions to achieve the first success (see Section 6.8). Thus the expected number of RAND calls for the entire algorithm is

$$\sum_{k=0}^{n-1} \frac{n}{n-k} = n \sum_{j=1}^n \frac{1}{j}. \quad (1)$$

The harmonic series  $\sum 1/j$  is one that you have met several times before (e.g., Example 1 of Section 1.5). It turns out that  $H(n) = \sum_{j=1}^n 1/j$  grows as  $\log n$ . Therefore we conclude from Eq. (1) that Algorithm PERMUTE-1 requires  $\text{Ord}(n \log n)$  calls of RAND to generate one permutation.

Could we hope for better? Sure, we could hope for just  $n$  calls of RAND, with none wasted. Here's one way to do that. After  $k$  numbers have already been chosen, we have only  $n - k$  to choose from. So let's invoke RAND[1,  $n - k$ ] instead of RAND[1,  $n$ ]. If our random choice is  $r$ , we pick the  $r$ th of the unpicked numbers (in increasing order).

Table 4.1 shows an example with  $n = 5$ . The RAND results are shown going down the left column. Each row on the right shows previously chosen numbers with slashes and the current choice with a circle. For instance, in the fourth row the value of RAND is 2. Since the numbers 1, 3, and 5 have already been picked, the second number still available is 4. The net result of all the RAND choices in the table is the permutation 35142.

---

**TABLE 4.1**  
**Typical Output of Algorithm PERMUTE-2 for  $n = 5$**

---

|                |   |   |   |   |   |
|----------------|---|---|---|---|---|
| RAND[1, 5] = 3 | 1 | 2 | 3 | 4 | 5 |
| RAND[1, 4] = 4 | 1 | 2 | 3 | 4 | 5 |
| RAND[1, 3] = 1 | 1 | 2 | 3 | 4 | 5 |
| RAND[1, 2] = 2 | 1 | 2 | 3 | 4 | 5 |
| RAND[1, 1] = 1 | 1 | 2 | 3 | 4 | 5 |

---

This approach gives us Algorithm 4.4, PERMUTE-2. The variables  $c$  and  $j$  are used to determine the  $r$ th unchosen number. The algorithm marches up from 0 (by incrementing  $j$ ), and  $c$  counts how many of the numbers have not been picked previously. When  $c$  equals  $r$ , the current  $j$  is the  $r$ th unchosen number.

The proof that PERMUTE-2 gives a random permutation is similar to the proof for PERMUTE-1. Each number from 1 to  $n$  is chosen once, and after  $r$  numbers have been chosen, each of the yet unchosen numbers has probability  $1/(n-r)$  of being chosen.

## Algorithm 4.4

### Permute-2

**Input**  $n$   
**Output**  $\text{Perm}$   
**Algorithm** PERMUTE-2

```
Flag ← 0
for  $i = 1$  to  $n$                                 [Pick  $i$ th entry in  $\text{Perm}$ ]
     $r \leftarrow \text{RAND}[1, n-i+1]$ 
     $c \leftarrow 0$                                [Number of unchosen values passed over]
     $j \leftarrow 0$                                [Initializes number considered for  $\text{Perm}(i)$ ]
    repeat while  $c < r$ 
         $j \leftarrow j + 1$ 
        if  $\text{Flag}(j) = 0$  then  $c \leftarrow c + 1$           [Another unchosen number found]
    endrepeat
     $\text{Perm}(i) \leftarrow j$ 
     $\text{Flag}(j) \leftarrow 1$ 
endfor
```

What about efficiency?  $\text{RAND}$  is invoked just  $n$  times, but this is not the right measure for PERMUTE-2 because now  $\text{Flag}$  is checked much more often.  $\text{Flag}$  is checked once in the *inner* loop for each value of  $j$  up through the value that gets assigned to  $\text{Perm}(i)$ . Since the previously assigned numbers are randomly distributed, and the next assigned number is randomly distributed among the unassigned, it seems reasonable that the next assigned number is randomly distributed from 1 to  $n$ . Therefore it should seem reasonable that its average position is  $(n+1)/2$ . (Why is  $n/2$  wrong?) Therefore, on average,  $(n+1)/2$  checks of  $\text{Flag}$  are made in the inner loop for *each* value of  $i$ . Thus the average number of checks in the main loop is  $n(n+1)/2$ . In short, even though Algorithm PERMUTE-2 saves on  $\text{RAND}$  calls, it is *worse* than Algorithm PERMUTE-1 because  $\text{Ord}(n(n+1)/2) = \text{Ord}(n^2) > \text{Ord}(n \log n)$ .

The reason PERMUTE-2 didn't work well is that the yet unchosen numbers got spread out, and so we had to keep checking all the numbers to see whether they had been chosen. The next algorithm, PERMUTE-3, avoids this problem by keeping all the yet unchosen numbers together.

Here's how. Suppose, as in Table 4.1, that we choose  $\text{Perm}(1) = 3$ . In that table we kept 3 in its place after it was chosen. This time, we *exchange* the positions of 1 and 3. See Table 4.2. This exchange will keep the unchosen numbers  $\{1, 2, 4, 5\}$  contiguous. Now, to determine  $\text{Perm}(2)$ , we need merely call  $\text{RAND}[2, 5]$ , and whatever number is in the *position* that  $\text{RAND}$  picks becomes  $\text{Perm}(2)$ . Say  $\text{RAND}[2, 5] = 5$ . Then we exchange the numbers in the fifth and second positions. The unchosen numbers are again together, in positions  $\{3, 4, 5\}$ . Table 4.2 gives a

complete example. In the column on the left we show the values for RAND. In the columns on the right we show in each row the ordering of 1 to  $n = 5$  just *before* making the exchange called for by RAND in that row. The two numbers with asterisks are the ones about to be switched. (If a number is switched with itself, it gets two asterisks.) The numbers underlined are the ones which are already known to be in their Perm positions. Note that this method has the added benefit that the last row *is* the permutation.

---

**TABLE 4.2**  
**Typical Output of Algorithm PERMUTE-3 for  $n = 5$**

---

|                |          |          |          |          |     |
|----------------|----------|----------|----------|----------|-----|
| RAND[1, 5] = 3 | 1*       | 2        | 3*       | 4        | 5   |
| RAND[2, 5] = 5 | <u>3</u> | 2*       | 1        | 4        | 5*  |
| RAND[3, 5] = 3 | <u>3</u> | <u>5</u> | 1**      | 4        | 2   |
| RAND[4, 5] = 5 | <u>3</u> | <u>5</u> | <u>1</u> | 4*       | 2*  |
| RAND[5, 5] = 5 | <u>3</u> | <u>5</u> | <u>1</u> | <u>2</u> | 4** |
|                | 3        | 5        | 1        | 2        | 4   |

---

## Algorithm 4.5

### Permute-3

```

Input n
Output Perm
Algorithm PERMUTE-3
  for  $i = 1$  to  $n$ 
    Perm( $i$ )  $\leftarrow i$                                 [Initialize Perm]
  endfor
  for  $i = 1$  to  $n$ 
     $r \leftarrow \text{RAND}[i, n]$ 
    Perm( $i$ )  $\leftrightarrow \text{Perm}(r)$                   [Exchange values]
  endfor

```

The proof that Algorithm PERMUTE-3 gives a permutation is easy. Perm *starts* as a permutation, and switching two elements can never destroy that. In short, the loop invariant is that Perm is a permutation. The proof that the permutation is random is the same as before.

How efficient is Algorithm PERMUTE-3? As in PERMUTE-2, RAND gets called  $n$  times. So let's see if anything else gets exercised more. There isn't any Flag. The only thing left to look at is assignments of entries of Perm. We find that there are  $n$  initially, plus one exchange for each of the  $n$  times through the second loop. If we

think of an exchange as composed of ordinary one-way assignments, each exchange requires 3 assignments. (Why?) Thus there are  $4n$  assignments altogether. In short, PERMUTE-3 is  $\text{Ord}(n)$ , that is, *linear* in the size of the problem. It is the best of the three algorithms. Note that it is also the shortest in algorithmic language.

## All Permutations

Now we turn to the problem of listing all permutations. In order to be correct and efficient, we must make sure to list each  $n$ -permutation once — but only once. A list inescapably comes out in some order, and we can turn this fact to our advantage. That is, one way to develop a listing algorithm is to start by deciding what order you would like and then attempting to devise an algorithm that achieves this order.

A very natural order for permutations is “odometer” order: We view each  $n$ -permutation as an  $n$ -digit number (we’ve already been doing this), and then we list them in the order they would appear on an odometer. For instance, if  $n = 3$ , the order would be

|     |
|-----|
| 123 |
| 132 |
| 213 |
| 231 |
| 312 |
| 321 |

The more general name for this sort of order is **lexical** or **lexicographic** order. Lexical order is odometer (i.e., numerical) order when the individual symbols are digits and alphabetical order when the symbols are letters. Even if  $n \geq 10$ , for the purpose of lexical order we may think of the numbers  $1, 2, \dots, n$  as “digits”.

There is a simple recursive way to describe an algorithm which produces permutations in lexical order. Notice that all the permutations with first symbol 1 precede all those with first symbol 2, and so on. Furthermore, if we take all the permutations that start with 1 and strip off the 1, we are left with the lexical ordering of all  $(n-1)$ -permutations on the symbols 2 through  $n$ . Similarly, if we take all those  $n$ -permutations that start with 2 and strip off the 2, we are left with the lexical ordering of all  $(n-1)$ -permutations on the symbols 1, 3, 4,  $\dots, n$ . And so on. Thus we can describe the algorithm as  $n$  recursive calls to itself, one for each possible starting symbol.

Although this recursive approach is simple in principle, like most recursions it is hard for humans to carry out. Therefore it is worth our while to seek an iterative algorithm for producing lexical order, even if the iterative algorithm is a bit complicated.

An iterative algorithm has other advantages here, too. First, it forces us to understand carefully which digits get changed going from one permutation to the next. Second, when an iterative and a recursive algorithm produce the same output, the iterative one is generally much faster since there is less computer overhead.

The key to developing an iterative algorithm for lexical order is to determine how to generate the next permutation from any given permutation. Then we just

feed in the starter permutation  $123\dots n$  and let things run until we reach  $n\dots 321$ . So suppose we have a typical permutation

$$[\text{Perm}(1), \text{Perm}(2), \dots, \text{Perm}(n)]. \quad (2)$$

To find the next permutation we reason as follows.

1. If we start from the *right* in permutation (2) and move to the left as long as the digits are *increasing*, then we have marked off a sequence of digits that is the largest possible formed from the digits in it. (Example: If  $n = 7$  and permutation (2) is  $[4,2,5,7,6,3,1]$ , then we stop at 7 and 7631 is the largest possible number that can be formed from 1, 3, 6, and 7.) No change in just these digits can result in a larger permutation.
2. The digit immediately to the left of the left-most digit considered in the previous paragraph is the first one (from the right) which can be exchanged with something to its right to get a permutation later in odometer order. We exchange it with the smallest digit to its right that is greater than it, since this gives the minimum possible increase in this digit. (In the example, we exchange 5 and 6, obtaining  $[4,2,6,7,5,3,1]$ .)
3. After this exchange, we still have a sequence increasing to the left where we had one before. (Now we have 7531 instead of 7631.) To get the smallest permutation greater than the previous one, we must arrange the digits in this sequence in their minimum odometer order. To do this we need only to reverse them. (In our example, the new permutation is  $[4,2,6,1,3,5,7]$ .)

This reasoning is embodied in Algorithm 4.6, ALLPERM. As an example, when  $n = 4$ , ALLPERM generates the permutations shown in Table 4.3. Read down the left half, then down the right half of the table.

The operations in ALLPERM are comparisons of Perm entries, switches of entries, and reassignments of counters. In each passage through the main loop, the number of such steps varies from permutation to permutation. In fact, the analysis of ALLPERM is fairly difficult and we won't say more about it.

## The Rank Function

We've said all we want to say on combinatorial algorithms per se. We introduced these algorithms because counting combinatorial objects is not always enough. However, it is an intriguing fact that such algorithms can shed light back on counting problems and other "structural" combinatorial problems that don't appear to be related to algorithms. Perhaps we can pique your interest with an example of this idea.

We know that there are  $n!$  permutations of the symbols 1 to  $n$ . When we devised Algorithm ALLPERM to generate them all without repetitions, we were in effect constructing a one-to-one correspondence between the set  $[n!] = \{1, 2, \dots, n!\}$  and the set  $\mathcal{P}$  of all  $n$ -permutations. A different algorithm to generate all permutations (there are many, and ALLPERM isn't necessarily the best [32]) would involve a different one-to-one correspondence.

## Algorithm 4.6

### AllPerm

**Input**  $n$

**Output** All  $n!$  permutations of  $1, 2, \dots, n$

**Algorithm** ALLPERM

```
for i = 1 to n                                [Generate first permutation]
    Perm(i) ← i
endfor
repeat   [Main permutation loop]
    print Perm
    b ← n-1      [b will be position of left-most digit to be changed]
    repeat until b = 0 or Perm(b) < Perm(b+1)
        b ← b-1
    endrepeat
    if b = 0 then stop                         [All permutations found]
    c ← n                                     [c will be position to be exchanged with b]
    repeat until Perm(c) > Perm(b)
        c ← c-1
    endrepeat
    Perm(b) ↔ Perm(c)                         [Exchange digits]
    d ← b + 1; f ← n                         [Initialize for reversal]
    repeat until d ≥ f
        Perm(d) ↔ Perm(f)                     [Reverse by exchanging]
        d ← d+1; f ← f-1
    endrepeat
endrepeat
```

If we view the correspondence as going from  $\mathcal{P}$  to  $[n!]$ , we have what is known as the **rank function**. For instance, in Table 4.3,  $\text{Rank}(1324) = 3$ , because 1324 is the third permutation in lexical order. If we view the correspondence as going from  $[n!]$  to  $\mathcal{P}$ , we have what is known as the **unrank function**. For instance, for lexical order  $\text{Unrank}(3) = 1324$ .

We don't have to look at a table to determine the rank function. This is where the recursive analysis of lexical order is quite handy. Given a permutation  $\text{Perm}$ , look at  $\text{Perm}(1)$ . Suppose  $\text{Perm}(1) = k$ . For each  $j$  from 1 to  $k-1$ , all  $(n-1)!$  permutations beginning with  $j$  come before  $\text{Perm}$  in lexical order. Then, if we strip off the first digit, we can apply the same analysis recursively to determine how many permutations beginning with  $k$  also come before  $\text{Perm}$ .

**TABLE 4.3**  
**Output for Algorithm ALLPERM when  $n = 4$**

| Permutation | $b$ | $c$ | Permutation | $b$ | $c$ |
|-------------|-----|-----|-------------|-----|-----|
| 1 2 3 4     | 3   | 4   | 3 1 2 4     | 3   | 4   |
| 1 2 4 3     | 2   | 4   | 3 1 4 2     | 2   | 4   |
| 1 3 2 4     | 3   | 4   | 3 2 1 4     | 3   | 4   |
| 1 3 4 2     | 2   | 3   | 3 2 4 1     | 2   | 3   |
| 1 4 2 3     | 3   | 4   | 3 4 1 2     | 3   | 4   |
| 1 4 3 2     | 1   | 4   | 3 4 2 1     | 1   | 2   |
| 2 1 3 4     | 3   | 4   | 4 1 2 3     | 3   | 4   |
| 2 1 4 3     | 2   | 4   | 4 1 3 2     | 2   | 4   |
| 2 3 1 4     | 3   | 4   | 4 2 1 3     | 3   | 4   |
| 2 3 4 1     | 2   | 4   | 4 2 3 1     | 2   | 3   |
| 2 4 1 3     | 3   | 4   | 4 3 1 2     | 3   | 4   |
| 2 4 3 1     | 1   | 3   | 4 3 2 1     | 0   |     |

For example, consider  $n = 4$  and let  $\text{Perm} = 3241$ . Then all the permutations that begin with 1 or 2 come before  $\text{Perm}$ ; there are  $2 \times 3! = 12$  of these. Now strip off the 3 from  $\text{Perm}$  and consider 241 as a permutation on the set  $\{1, 2, 4\}$ . The  $1 \times 2! = 2$  permutations on this set that begin with 1 all come before 241. Now strip off the 2 and consider 41 as a permutation on  $\{1, 4\}$ . Then  $1 \times 1! = 1$  permutations on this set start with a smaller number than 4 (namely, the permutation 14). So we conclude that

$$(2 \times 3!) + (1 \times 2!) + (1 \times 1!) = 15$$

permutations come *before* 3241 in lexical order. Therefore,  $\text{Rank}(3241) = 16$ . (Some authors define the rank to be the number of things which come before, in which case 15 itself is the answer.)

So what? Well, there was nothing special about 3241. For any  $n$ -permutation  $\text{Perm}$ , let  $a_k, k = 1, 2, \dots, n-1$  be the number of digits to the right of the  $k$ th digit of  $\text{Perm}$  that are numerically smaller than the  $k$ th digit. Then we have illustrated how (one less than) the rank can be computed in the form

$$\sum_{k=1}^{n-1} a_k(n-k)! , \quad 0 \leq a_k \leq n-k. \tag{3}$$

For instance, when  $\text{Perm} = 3241$  we have obtained  $a_1 = 2$ ,  $a_2 = 1$  and  $a_3 = 1$ .

Furthermore, every sequence  $a_1, a_2, \dots, a_{n-1}$  satisfying  $0 \leq a_k \leq n - k$  comes from some  $n$ -permutation, and if  $\text{Perm}$  and  $\text{Perm}'$  are two different  $n$ -permutations, then the sequences  $a_1, a_2, \dots, a_{n-1}$  and  $a'_1, a'_2, \dots, a'_{n-1}$  of coefficients they generate must be different. (Why?) Therefore we have the following result: Every number from 0 to  $n! - 1$  is *uniquely* representable in the form of Eq. (3). This result is just an illustration of a whole class of results. Suppose you are studying doodads, and there are  $D$  of them. Suppose you come up with an algorithm for generating all doodads. Implicit in that algorithm is almost surely a special representation, perhaps new, of all the numbers from 1 to  $D$  (or 0 to  $D - 1$ ). For instance, see [32] for another representation arising from another way to list all permutations.

## Problems: Section 4.9

---

1. ⟨1⟩ It is a misnomer to call RAND a function. Why? (Nonetheless, it is common practice to do so.)
2. ⟨1⟩ Actually, Algorithm PERMUTE-1 isn't an algorithm at all, according to the precise definition in Chapter 1. It is logically possible for it to go on running forever. How? Yet we claim it is “essentially” an algorithm. Explain.
3. ⟨1⟩ Let  $n = 5$ .
  - a) Given

2 4 3 4 1 2 4 3 5 4 2 ...

as a stream of outputs from RAND[1,5], what permutation would Algorithm PERMUTE-1 produce?

- b) What permutation does PERMUTE-2 produce, given the outputs

RAND[1, 5] = 2

RAND[1, 2] = 1

RAND[1, 4] = 4

RAND[1, 1] = 1

RAND[1, 3] = 2

- c) What permutation does PERMUTE-3 produce, given the outputs

RAND[1, 5] = 2

RAND[4, 5] = 4

RAND[2, 5] = 4

RAND[5, 5] = 5

RAND[3, 5] = 4

4. ⟨2⟩ There is a variant of Algorithm PERMUTE-3 that calls

RAND[1,  $n$ ], RAND[1,  $n - 1$ ], RAND[1,  $n - 2$ ], ...

instead of

RAND[1,  $n$ ], RAND[2,  $n$ ], RAND[3,  $n$ ], ...

(It fills in the permutation from the right.) Figure it out and write it down.

5. ⟨1⟩ Consider Table 4.2.
  - a) The last line is not necessary; the permutation does not change from the previous line. Why not?
  - b) How can Algorithm PERMUTE-3 be changed to take advantage of this fact?
  - c) How is its efficiency analysis affected?
6. ⟨2⟩ Argue as carefully as you can that the permutation produced by Algorithm PERMUTE-3 is random.
7. ⟨2⟩ Explain why Algorithm PERMUTE-4 does or does not generate a random permutation of  $1, 2, \dots, n$ .

### Algorithm

---

**Input**  $n$

**Output**  $\text{Perm}(1), \dots, \text{Perm}(n)$

**Algorithm** PERMUTE-4

**for**  $i = 1$  **to**  $n$

$\text{Perm}(i) \leftarrow i$

[Initialize]

**endfor**

**for**  $k = n$  **downto** 2

$\text{Perm}(1) \leftrightarrow \text{Perm}(\text{Rand}[1, k])$

**endfor**

8. ⟨2⟩ Most computer languages and calculators do not actually have our RAND as their basic (pseudo)random number generator. More often, they have a command, call it RAN, which

generates a decimal number strictly between 0 and 1, where every decimal number is equally likely. How could you define our RAND[ $n, m$ ] in terms of RAN?

9. (2) For  $n = 5$  what are the first 10 permutations output by Algorithm ALLPERM? The last 10?

10. (1) Step through Algorithm ALLPERM when  $n = 3$ . Make a table like Table 4.3, but also depict the process of creating each permutation from the previous one by showing the result of exchanging two entries and then showing the result of reversing part of the sequence. That is, show steps 2 and 3 in the text description of ALLPERM separately.

11. (1) Modify our three random permutation algorithms to produce random permutations of  $n$  things taken  $k$  at a time.

12. (4) Devise an algorithm to construct all permutations of  $n$  things taken  $k$  at a time.

13. (1) Develop an algorithm to find a random combination of  $n$  things taken  $k$  at a time. *Hint:* You don't have to wander very far from a permutation algorithm. On the printed page, a combination has to appear in some order, but nobody said anything about caring what that order is.

14. (3) Develop an algorithm to output all combinations of  $n$  things taken  $k$  at a time, without listing any combination twice. *Suggestion:* If for no other reason than to keep from repeating things, it is now important to output each combination in a standard way, say, in increasing order.

15. (2) We didn't define lexical order precisely in the text but instead relied on your prior understanding of odometer order. Give a precise definition. A recursive definition is one possibility.

16. (2) One way to generate all the  $n$ -permutations is to take any algorithm for generating single random permutations and run it until all permutations have been found, discarding repeats along the way. If we count each call of the random permutation algorithm as one step, what is the Order of this algorithm?

17. (3) Consider the rank and unrank functions for Algorithm ALLPERM.

a) What is Rank(462135)?

b) When  $n = 5$ , what is Unrank(90)?

c) Devise a general procedure for computing the unrank function.

18. (2) The coefficients  $a_k$  in Eq. (3) that arise out of Algorithm ALLPERM are intimately related to Algorithm PERMUTE-2. Explain how.

19. (1) In Eq. (3) one may sum from  $k = 1$  to  $n$  instead of 1 to  $n - 1$ . Explain. (Summing to  $n$  adds an unnecessary  $n$ th term, but for one of us it makes the formulas more esthetic.)

20. (3) **Reverse lexical** order is just lexical order run backwards. For instance, if the symbols are digits, reverse lexical order lists numbers from largest to smallest. Do [17] again for reverse lexical order. Also come up with a general procedure (not using the lexical rank) for computing the reverse lexical rank function.

21. (2) Suppose that 100 students sign up for course CS101 and 90 are male. Also suppose that there is room for only 30 students in the course, and the CS Department decides to make a random selection.

a) The CS Department Chair is worried that the random selection might result in all 10 females being excluded and that there will be claims of discrimination. He would like to see what typical random selections look like with regard to gender. Provide him with 10 such random selections.

b) Generate 100 random selections and use these to estimate the probability that there will be no females in the class; at most 1 female in the class. (If you know how to compute the probability exactly, do so and compare your answers with your experimental results.)

22. (2) Make up a traveling salesman problem with six cities. Make up the distances between each pair of cities. Write a computer program similar to ALLPERM to test the length of each tour and thus find the shortest one.

23. (3) Make up a traveling salesman problem with 20 cities. Make up the distances between each pair of cities. (This is a lot of pairs — let the computer generate the distances using its random number generator.) Write a computer program similar to PERMUTE-3 which tests  $N$  random tours and outputs the shortest tour found. Run this program for  $N = 10, 100, 1000$ . Report on your results.

24. (3) Run your program from [23] again, but now run it on the six cities of [22], for which you know the exact answer. Do some experimenting. How

many random tours do you need to pick before you get within 10% of the optimal answer?

25.  $\langle 1 \rangle$  In the text we indicated how to find an approximate solution to the traveling salesman problem by using random permutations. Another plausible approach is to try a greedy algorithm: Start with the shortest edge and grow a path, at each step adding the shortest edge from either end of the path to vertices not on the path. When all vertices have been reached, close the path by adding the edge between the ends.

Unfortunately, this need not work well. Consider the complete graph in Fig. 4.14. The shortest cycle happens to be 1–2–3–4–5–1, with length 900. What cycle, with what length, does the greedy algorithm obtain?

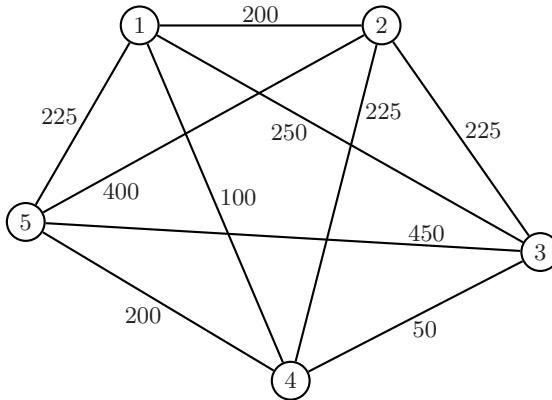


FIGURE 4.14

Greed need not pay

26.  $\langle 2 \rangle$  For [25], verify that the cycle claimed to be shortest is the shortest. This is bearable by hand (24 tours), but use a computer, basing your program on Algorithm ALLPERM.
27.  $\langle 1 \rangle$  Write an algorithm to output a random ordered pair  $(i, j)$  where each entry is between 1 and  $n$ , inclusive.
28.  $\langle 1 \rangle$  Write an algorithm to output all ordered pairs  $(i, j)$  with entries between 1 and  $n$ , inclusive.
29.  $\langle 2 \rangle$  Here is an algorithm which purports to output a random increasing ordered pair  $(i, j)$ , where  $1 \leq i \leq j \leq n$ .

**Input**  $n$

**Algorithm** RISING-PAIR

$i \leftarrow \text{RAND}[1, n]$

$j \leftarrow \text{RAND}[i, n]$

**Output**  $(i, j)$

This outputs an increasing ordered pair, all right. What's wrong with the algorithm?

30.  $\langle 3 \rangle$  Write an algorithm which outputs a truly random, *strictly* increasing ordered pair with entries between 1 and  $n$ . That is, it should output  $(i, j)$  where  $1 \leq i < j \leq n$ .
31.  $\langle 3 \rangle$  Write an algorithm which outputs a truly random, increasing ordered pair with entries between 1 and  $n$ . That is, it should output  $(i, j)$  where  $1 \leq i \leq j \leq n$ .
32.  $\langle 4 \rangle$  Algorithm ALLPERM sometimes requires a lot of reassignments of the permutation entries to get between the current permutation and the next one. It turns out there is another algorithm which generates the permutations in an order (*not* lexical order) such that each permutation differs from the previous one by the minimum possible reassignment — the exchange of two adjacent entries. For instance, here is a listing of the six permutations of  $\{1, 2, 3\}$  with this “minimal exchange” property:

321    231    213    123    132    312.

(Underlining indicates those two digits that were just exchanged.)

- a) Think recursively to devise such an algorithm.  
*Hint:* Suppose that you already have a minimal exchange listing of the permutations on the set  $\{1, 2, \dots, n-1\}$ . For each permutation in the list, consider all the ways to insert the symbol  $n$ . Do it so as to maintain the minimal exchange property.
- b) Find a recursive formula for the rank function for your algorithm.
- c) Use (one less than) the rank function to devise a representation theorem for the numbers from 0 to  $n! - 1$  that is different from the theorem in the text.
- d) Find an iterative algorithm which outputs the  $n$ -permutations in the same order as your algorithm in part a).

## 4.10 Algorithmic Pigeonholes

In addition to putting balls into urns, mathematicians like to put pigeons into holes. This hobby is based on a very simple observation.

### The Pigeonhole Principle

If  $n + 1$  or more pigeons are put into  $n$  holes, at least two pigeons must be in the same hole.

This principle has many applications, some quite surprising. Most of the applications are “structure” results. They tend to say: No matter how disordered you try to be, there is some structure in what you create. This vague statement will make more sense after some examples.

What’s “algorithmic” got to do with it? You’ll see.

#### EXAMPLE 1

You have five pairs of socks, all different colors, jumbled together as ten individual socks in your dresser drawer. You go into your room before sunrise to get a pair and discover the light is broken. How many socks must you pull out to be certain to have a matched pair among them?

**Solution** The socks are pigeons and the colors are holes. There are five holes and you want to have two pigeons in some hole. Thus to be certain, you have to take six socks. ■

#### EXAMPLE 2

You and a friend play the following game. The friend gets to pick any ten integers from 1 to 40. You win if you can find two different sets of three from those ten that have the same sum. Prove that you can always win this game.

**Solution** Let the pigeons be the possible 3-sets. Let the holes be the possible sums. There are  $C(10, 3) = 120$  pigeons. The smallest possible sum is  $1+2+3 = 6$  (and that’s possible only if your friend picked 1, 2, and 3). Likewise, the largest possible sum is  $38+39+40 = 117$ . Therefore there are at most  $117-5 = 112$  holes. So at least two pigeons (3-sets) go in the same hole (have the same sum). ■

We claimed that the Pigeonhole Principle is about disorder and structure. Now we can explain better what we meant. In Example 1, the socks were disordered, but you couldn’t help but find order (a matching pair) if you took six. In Example 2, your friend can pick the ten numbers from 1 to 40 any way at all, with irregular gaps between their values. But this can’t keep you from finding a certain order to things, specifically, two triples that have the same sum.

Note that the Pigeonhole Principle does not say *which* pigeons share a hole, or *how* to find them. For this reason, the principle has usually been regarded as a

purely existential result; see the discussion of existential and constructive proofs in Section 3.3. However, it doesn't have to be viewed existentially. Instead of asking

Given anything of type A, prove that it contains something of type B,

instead ask

Devise an algorithm that, given anything of type A, outputs something of type B if it's there.

The second form is the reason for our section title. Moreover, the second is easier to get started on, and if you are successful, as a by-product you will usually have an answer for the first form as well.

We illustrate this with a problem that is very hard when treated the existential way but is easier, at least to get started, when treated algorithmically. It concerns subsequences of a sequence. Given a sequence of numbers, a **subsequence** is a subset considered in the same order as the numbers appear in the original. Thus 1, 5, 2 is a subsequence of  $S = (1, 3, 5, 7, 6, 4, 2)$ , but 1, 2, 5 is not. Given any sequence of *distinct* numbers, a subsequence is **monotonic** if it is always increasing or always decreasing. Thus 1, 3, 5, 7 and 6, 4, 2 are both monotonic subsequences of  $S$ , but 1, 5, 2 is not. (The reason for limiting ourselves to distinct numbers in this section is simply so that we don't have to decide whether sequences with repeats, like 6, 4, 4, 2, are monotonic or not. In fact, with the right definition of monotonic, the results we get below work for sequences with repeats as well [15].

The problem is:

Given a sequence of distinct real numbers, what can you say about the longest monotonic subsequence?

We have stated this so open-endedly because, in fact, there are two quite different forms in which this problem has circulated. The existential version goes:

Show that any sequence of length such-and-such has a monotonic subsequence of length so-and-so.

(“Such-and-such” and “so-and-so” are specific expressions, but we don't want to give them away. That's the problem with the existential approach; you are presented with the results as a fait accompli and have no idea how you could find them yourself.) The algorithmic version goes:

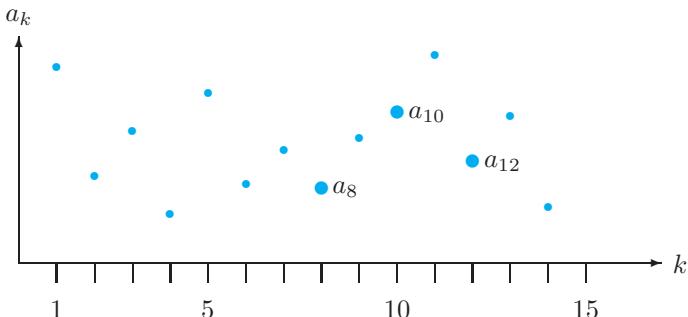
Devise an algorithm that takes any sequence and finds the length of its longest monotonic subsequence.

Let's try to solve the algorithmic problem. In so doing, our goal is to gain insight about the existential problem as well. Use the recursive paradigm! Suppose the given sequence is  $S = (a_1, a_2, \dots, a_m)$ . Let's begin, as we often do, by generalizing. For every  $k$  from 1 to  $m$  we ask: How can we find the longest monotonic subsequence in  $S_k$ , the initial subsequence  $(a_1, a_2, \dots, a_k)$  of  $S$ ? Next we suppose we know how to find the longest monotonic subsequence for every  $S_j$  with  $j < k$  and use

this supposition to develop a recurrence for  $S_k$ . Finally, we apply this recurrence iteratively to  $k = 1, 2, \dots, m$ , and when we reach  $m$  we are done.

So, we are looking for (the length of) a longest monotonic subsequence in  $S_k$ . (We say “a longest”, not “the longest”, because several subsequences might tie for longest.) Either all longest monotonic subsequences of  $S_k$  end at  $a_k$  or some longest monotonic subsequence of  $S_k$  doesn’t end at  $a_k$ , in which case it is a longest monotonic subsequence of  $S_{k-1}$  and we are supposing we already know how to find it.

So let’s assume we are in the former case and let  $T$  be a longest monotonic subsequence of  $S_k$ . Consider Figure 4.15, where  $k = 12$ . Some  $a_j$  is the next to last term in  $T$ . If  $a_j > a_k$  (for instance, see  $j = 10$  in the figure), then  $T$  must be a decreasing sequence; if  $a_j < a_k$  (see  $j = 8$  in the figure), then  $T$  must be increasing. Now, let  $T'$  be  $T$  with the final term  $a_k$  deleted. If  $T$  is decreasing, then  $T'$  must be a longest decreasing sequence ending at  $a_j$ , for if  $T''$  were a longer decreasing sequence ending at  $a_j$ , we could substitute  $T''$  for  $T'$  in  $T$  and get a monotonic subsequence longer than  $T$ . Similarly, if  $T$  is increasing, then  $T'$  must be a longest increasing sequence ending at  $a_j$ .



The graph of part of a sequence. The value of a term is indicated by its height. The term  $a_{12}$  may be appended to any decreasing subsequence ending at  $a_{10}$  or to any increasing subsequence ending at  $a_8$ .

What have we discovered? That it is not enough to suppose we know the longest monotonic subsequence in  $S_{k-1}$ . Because we don’t know which  $a_j$  is the next to last term in  $T$ , we must find separately for each  $j < k$  the longest increasing length and longest decreasing length *ending* at  $a_j$ . And therefore, to make the induction work going forward, for  $S_k$  we must use this information to compute both longest increasing and decreasing lengths ending at  $a_k$ .

So, suppose we already knew the lengths  $u_j$  and  $d_j$  of the longest up and the longest down subsequences ending at  $a_j$ , for each  $j$  from 1 to  $k-1$ . Then the recurrence is

**FIGURE 4.15**

$$u_k = \max\{u_j + 1\} \text{ for } j < k \text{ with } a_j < a_k, \\ d_k = \max\{d_j + 1\} \text{ for } j < k \text{ with } a_j > a_k.$$

Algorithm 4.7, MONOTONIC, carries this recurrence out in the inner  $j$  loop, using the usual device for finding the max of many values by maxing with just one more value each time. The outer loop then proceeds to compute  $u_k$  and  $d_k$  for each  $k$  in turn. Finally, recall that the longest monotonic subsequence of  $S$  need not end at  $a_m$ , so the last step is to find the max of all the  $u$ 's and  $d$ 's. Table 4.4 gives an example.

### Algorithm 4.7

#### Monotonic

|                  |                                                                         |                    |
|------------------|-------------------------------------------------------------------------|--------------------|
| <b>Input</b>     | $a_1, a_2, \dots, a_m$                                                  | [Distinct numbers] |
| <b>Output</b>    | $length$                                                                |                    |
| <b>Algorithm</b> | MONOTONIC                                                               |                    |
|                  | $u_1 \leftarrow 1; d_1 \leftarrow 1$                                    |                    |
|                  | <b>for</b> $k = 2$ <b>to</b> $m$                                        |                    |
|                  | $u_k \leftarrow 1; d_k \leftarrow 1$                                    | [Initialize]       |
|                  | <b>for</b> $j = 1$ <b>to</b> $k-1$                                      |                    |
|                  | <b>if</b> $a_j < a_k$ <b>then</b> $u_k \leftarrow \max\{u_k, u_j + 1\}$ |                    |
|                  | <b>else</b> $d_k \leftarrow \max\{d_k, d_j + 1\}$                       | $[a_j > a_k]$      |
|                  | <b>endfor</b>                                                           |                    |
|                  | <b>endfor</b>                                                           |                    |
|                  | $length \leftarrow \max\{u_1, u_2, \dots, u_m, d_1, d_2, \dots, d_m\}$  |                    |

All right, we can solve the algorithmic problem, but so far our method doesn't seem to suggest any patterns we can use for the existential version. That is, it hasn't shown us any nice relationship between  $m$ , the length of the original sequence, and  $n$ , the length of the longest monotonic subsequence. Besides, where are the pigeons and where are the holes?

Answering the last question will lead us to a pattern. Since an assignment of pigeons to holes is a function mapping one set (pigeons) to another (holes), look to see if Algorithm MONOTONIC sets up any functions. Yes, for each  $k$  it assigns the couplet  $(u_k, d_k)$ . Furthermore, this function is one-to-one (defined on p. 16). To see why, look at the two lines inside the inner loop; they say that, for each  $k$ ,  $(u_k, d_k)$  is different from all previous  $(u_j, d_j)$  — either  $u_k \geq u_j + 1$  or  $d_k \geq d_j + 1$ . So let the pigeons be the  $k$ 's (the indices of the sequence), and let the holes be ordered pairs of positive integers. The length of the longest monotonic subsequence is the maximum value of all the  $u_j$ 's and  $d_j$ 's for the holes actually occupied by pigeons. If this maximum is  $n$ , then at most the  $n^2$  holes from  $(1,1)$  to  $(n,n)$  are occupied. Since each hole has at most one pigeon, there are at most  $n^2$  pigeons. In other words, we have solved the existential problem by proving the "structure" theorem of numerical sequences given below (Theorem 1).

**TABLE 4.4**  
**Trace of Algorithm MONOTONIC for the sequence 7, 3, 8, 4, 6**

---

| $k$ | $u_k$ | $d_k$ | $j$     | $u_4$ | $d_4$ |
|-----|-------|-------|---------|-------|-------|
| 1   | 1     | 1     | initial | 1     | 1     |
| 2   | 1     | 2     | 1       | 1     | 2     |
| 3   | 2     | 1     | 2       | 2     | 2     |
| 4   | 2     | 2     | 3       | 2     | 2     |
| 5   | 3     | 2     |         |       |       |

for  $k = 4$

---

The box on the left shows values at the end of each pass of the outer loop; the box on the right shows values at the end of each pass of the inner loop for  $k = 4$ . In this inner loop,  $u_4$  and  $d_4$  start at 1, representing the sequence consisting of  $a_4 = 4$  alone. Then the sequence ending at  $a_4$  with  $a_1 = 7$  as the previous term is considered. Since it is monotonic down, and  $d_1 = 1$ ,  $d_4$  increases to 2. Next,  $a_2=3 < a_4$ , so sequences ending at  $a_4$  with  $a_2$  as the previous term must be monotonic up, and  $u_4$  increases to  $u_2+1 = 2$ . Finally,  $a_3 > a_4$ , so  $d_3+1 = 2$  is the length of a down sequence ending at  $a_4$ . Since  $d_4$  was already 2, it does not change.

Now that you know how to obtain this existential theorem via the algorithmic approach, you too can impress and mystify your friends by telling them only the theorem and the following short proof. (This is all that is typically said if one presents the problem existentially.)

---

**Theorem 1.** Let a sequence consist of distinct numbers. If the longest monotonic subsequence has length  $n$ , then the sequence has at most  $n^2$  terms. Conversely, a sequence of  $n^2 + 1$  (or more) distinct numbers necessarily has a monotonic subsequence of at least  $n + 1$  terms.

---

**PROOF** Let the sequence be  $a_1, \dots, a_{n^2+1}$ . For each  $k$ , let  $u_k$  (respectively  $d_k$ ) be the length of the longest increasing (respectively decreasing) subsequence ending at  $a_k$ . Note that  $u_k$  and  $d_k$  are both positive integers. The mapping  $k \rightarrow (u_k, d_k)$  is one-to-one, because for any pair  $j < k$ , either the longest up sequence or the longest down sequence to  $a_j$  can be continued on to  $a_k$  (depending on whether  $a_j < a_k$  or vice versa). Therefore there are  $n^2 + 1$  distinct pairs  $(u_k, d_k)$ , so some  $u_k$  or  $d_k$  must be at least  $n + 1$ . ■

Sometimes the Pigeonhole Principle is useful because we *don't* want to put more than one pigeon in a hole, or we know for a fact that we haven't. For instance, the solution of the monotonic subsequence problem hinged on realizing that each pigeon was necessarily in a different hole. In such cases, it's the contrapositive form of the principle that we use:

If each of  $n$  pigeons fits in a different hole, then there are at least  $n$  holes.

We conclude with an easy example.

### EXAMPLE 3

In order for a keyboard to send its output to a computer, there must be a code for each character – uppercase and lowercase letters, numerals, punctuation marks and such commands as “carriage return”. Each of these characters must be converted into a string of 0's and 1's. Are binary strings of length six enough to encode all characters? If not, what's the minimum length needed?

**Solution** There are about 100 actual symbols and a variable number of function keys. These symbols and function keys are the pigeons and the 6-bit strings are the holes. It is essential that each pigeon go in a different hole. (Why?) Unfortunately, there are only  $2^6 = 64$  6-bit strings. Therefore 6 bits are not enough. On the other hand, unless you have a keyboard with a lot of special function keys, 7 bits are enough. And sure enough, the American Standard Code for Information Interchange (ASCII) is 7 bits. ■

## Problems: Section 4.10

---

1. **(1)** The **Extended Pigeonhole Principle** says: If \_\_\_\_\_ pigeons are put in  $n$  holes, then some hole contains at least  $k + 1$  pigeons. Fill in the blank. Explain.
2. **(1)** In Example 1, instead of one pair of each of five colors, suppose that you have seven identical pairs of each color. Does the answer change?
3. **(2)** Repeat [2], except that you want to pull out three pairs all of the same color. How many socks must you take to be certain of success?
4. **(3)** Repeat [3], except that the three pairs need not be the same color.
5. **(2)** Same as Example 1, except that you want the red pair. How many socks must you draw to be certain? (This problem shows that if you start specifying *which* pigeonhole you want, the Pigeonhole Principle doesn't apply.)
6. **(3)** Consider Example 2 (equal sums of 3-sets

when ten numbers are picked between 1 and 40).

- a) Suppose your friend picks ten numbers between 1 and 50. Show that the sort of pigeonhole analysis in the solution is now insufficient to prove that you can always win. That is, show that the number of 3-sets is not greater than the number of possible sums. *Note:* Showing this does not prove that you *can't* force a win. A much more subtle analysis might still show that you can always beat your friend; see part g) for an example. If you can't always beat him, a natural way to prove this would be to exhibit a particular 10-set of  $\{1, 2, \dots, 50\}$  such that all 120 of its 3-subsets have different sums.
- b) Show that the pigeonhole analysis of the solution is insufficient to show you can always win if your friend picks nine numbers between 1 and 40.

- c)** Suppose we want to show that, when any 10 integers are picked from 1 to  $m$ , then some two 3-subsets of the 10-set necessarily have the same sum. What is the largest  $m$  for which the sort of analysis in Example 2 shows this?
- d)** Suppose we want to show that, when any  $n$  integers are picked from 1 to 100, then some two 3-subsets of the  $n$ -set necessarily have the same sum. What is the smallest  $n$  for which the sort of analysis in Example 2 shows this?
- e)** Same as part c), except now consider 4-subsets instead of 3-subsets.
- f)** Same as part d), but consider 4-subsets.
- g)** Suppose your friend now picks ten integers between 1 and 24, and you try to find two *pairs* of his numbers with the same sum. There are  $C(10, 2) = 45$  pairs and 45 possible sums (from  $1+2=3$  to  $23+24=47$ ). So the pigeonhole analysis alone does not prove that you can always win. Nonetheless, you can. Prove it. *Hint:* Show that your friend, to have a chance to force a win, must include in his 10-set 1, 2, 23, and 24. So what?
- 7.** (3) Example 2 was presented in the existential form. An algorithmic form would ask for an algorithm which, given  $n$  distinct numbers chosen from 1 to  $m$ , finds all 3-subsets with the same sum (if there are any).
- a)** Outline such an algorithm.
- b)** Explain how finding such an algorithm might lead someone to solve the existential version, that is, to identify the pigeons and the holes and to find  $m$  and  $n$  (like  $m = 40$  and  $n = 10$ ) which guarantee the existence of 3-subsets with equal sums.
- 8.** (2) Show that, given *any*  $n + 1$  integers from the set  $\{1, 2, \dots, 2n\}$ , one of the integers is a multiple of another. (In fact, one of them is a power of 2 times another.) *Hint:* Look at the largest odd factor of each number.
- 9.** (1) If computers used trits instead of bits, how long would strings have to be to transfer keyboard input? (See Example 3. Trits are the “digits” 0, 1, and 2.)
- 10.** (2) We devised Algorithm MONOTONIC recursively, yet we wrote it iteratively. Why? When we need information about the  $j$ th case, why not obtain it by a recursive procedure call?
- 11.** (2) In Algorithm MONOTONIC, why did we initialize  $u_k$  and  $d_k$  to 1? Find a situation in which any other initialization would lead to a wrong answer.
- 12.** (2) Algorithm MONOTONIC can be refined.
- a)** Modify MONOTONIC so that it returns the terms of a longest monotonic subsequence (instead of just its length).
- b)** If there are several longest monotonic subsequences, you could return all of them, but it’s better to return one that is uniquely determined by further properties. Think of at least one property that would uniquely determine one longest subsequence and modify MONOTONIC further to output that subsequence.
- 13.** (3) Theorem 1 didn’t really finish the job on monotonic subsequences. It shows that  $n^2 + 1$  terms force a monotonic subsequence of length  $n + 1$ , but it doesn’t show that some lengths less than  $n^2 + 1$  don’t also force the same conclusion. Maybe some even more subtle pigeonhole argument replaces  $n^2 + 1$  by a smaller value. Show that Theorem 1 is the best possible by exhibiting a sequence of  $n^2$  distinct numbers with no monotonic subsequence of length more than  $n$ . For your sequence, verify that the pairs  $(u_k, d_k)$  take on every value from  $(1, 1)$  to  $(n, n)$ .
- 14.** (3) Theorem 1 speaks only about sequences of a special length,  $m = n^2 + 1$ . This seems like a very special consequence of Algorithm MONOTONIC. This problem draws a more general conclusion about longest monotonic subsequences. Throughout the problem, sequences are understood to consist of distinct numbers.
- a)** Show that the pigeonhole argument of the theorem actually proves this: For any positive integer  $m$ , every sequence  $a_1, a_2, \dots, a_m$  has a monotonic subsequence of length at least  $\lceil \sqrt{m} \rceil$ .
- b)** Prove that the bound in a) is *tight*, that is, for every  $m$  construct an  $m$ -sequence in which the longest monotonic subsequence has length exactly  $n = \lceil \sqrt{m} \rceil$ .
- c)** Show that the smallest  $m$  for which every  $m$ -sequence is guaranteed to have a monotonic subsequence of length  $n$  is  $m = (n-1)^2 + 1 = n^2 - 2n + 2$ .

- d) Using c), what is the smallest  $m$  for which every subsequence has a monotonic subsequence of length  $n + 1$ ?

15. ⟨2⟩ How do the analysis and conclusions of the monotonic subsequence problem change if the terms of the original sequence need not be distinct? *Note:* The previous conclusions can be true or false, depending on your definition of monotonic when there are repeats.

16. ⟨3⟩ Consider a sequence of numbers, not necessarily distinct. A subsequence  $s_1, s_2, \dots, s_k$  is **strictly increasing** if  $s_{i+1} > s_i$  for each  $i = 1, 2, \dots, k-1$ . A subsequence is **strictly decreasing** if  $s_{i+1} < s_i$  for each  $i = 1, 2, \dots, k-1$ . A subsequence is an *equality subsequence* (not a standard term) if all terms of the subsequence are equal. Finally, call a subsequence *special* if it is either strictly increasing, strictly decreasing, or an equality subsequence.

a) Create an algorithm that finds, for any finite sequence of numbers, a longest special subsequence.

b) State and prove a theorem of the form: Every sequence of numbers of length  $L$  has a special subsequence of length  $L'$ . (You have to figure out  $L$  and  $L'$  before you state the theorem.)

c) Using your  $L, L'$  from part b), construct a sequence of length  $L - 1$  that does not have any special subsequence of length  $L'$ .

17. ⟨4⟩ Consider a sequence

$$(a_1, b_1), (a_2, b_2), \dots, (a_m, b_m)$$

of ordered pairs of real numbers. Such a “vector sequence” is monotonic if the  $a$  entries are monotonic and the  $b$  entries are monotonic (it’s OK if, say, the  $a$ ’s are increasing and the  $b$ ’s are decreasing). Solve the longest monotonic subsequence problem again for such vector sequences. Devise both an algorithmic and an existential solution.

18. ⟨2⟩ Call a sequence of ordered pairs of real numbers

$$(a_1, b_1), (a_2, b_2), \dots, (a_m, b_m)$$

**monotonic increasing** if  $a_{i+1} \geq a_i$  and  $b_{i+1} \geq b_i$  for each  $i = 1, 2, \dots, m-1$ . The definition of monotonic decreasing is similar. Prove the best theorem you can of the form “Every sequence of ordered pairs of numbers of length  $L$  has a monotonic increasing or monotonic decreasing subsequence of length  $L'$ .” Explain why your choices of  $L, L'$  are the best possible.

19. ⟨2⟩ The Pigeonhole Principle hardly requires a formal proof, but it is interesting to note that one can give a formal proof (of the contrapositive form), using the Sum Rule. Do so.

20. ⟨2⟩ In a manner similar to [19] give a formal proof of the Extended Pigeonhole Principle of [1].

## Supplementary Problems: Chapter 4

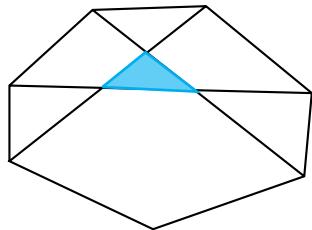
1. ⟨3⟩ Investigate the pattern of odd and even entries in Pascal’s Triangle. It will help to replace each even entry by 0 and each odd entry by 1.

2. ⟨3⟩ Twenty scientists are working on a secret project. The government trusts the scientists collectively, but not individually. An administrator decides that any group of five or more of the scientists should be allowed to enter the documents room together, but that no smaller group should be allowed to do so. The idea is to put a number of locks on the door and distribute keys to individual scientists so that each group of five will have all the keys among them, but no smaller group will.

Is this a workable decision? At the minimum, how many locks are necessary? How many keys must be made for each lock? How many keys must each scientist carry? *Hint:* Show that there must be a one-to-one correspondence between 4-sets of scientists and locks.

3. ⟨3⟩ How many triangles are formed using chords and sides of an  $n$ -gon, where the vertices of the triangle need not be vertices of the  $n$ -gon. For instance, one such triangle is highlighted in Figure 3. Assume that no three chords meet at the same interior point. *Hint:* Relate different sorts of triangles to different-sized sets of vertices of the

polygon.



**FIGURE 4.16**

4. ⟨3⟩ Repeat [3], but now only chords of the  $n$ -gon can be sides of the triangle.

5. ⟨4⟩ Solve Example 3, Section 4.4 (number of intersection points of the chords of a polygon) by the first method outlined in the text. Let  $c_k$  be the number of chords which have exactly  $k$  intersection points with other chords of the polygon. Use the Division Rule to express the answer in terms of  $S = \sum c_k$ . Now, if you can deduce a formula for  $c_k$  and then figure out how to simplify  $S$ , you're in business.

6. ⟨2⟩ Give a combinatorial argument to prove

$$\sum_{r=k}^{n-k} \binom{r}{k} \binom{n-r}{k} = \binom{n+1}{2k+1}.$$

*Hint:* When picking  $2k + 1$  people from a row of  $n + 1$ , decide first who will be the middle person in the subrow that you pick.

7. ⟨4⟩ Find a combinatorial argument for

$$n^n = \sum_{k=1}^n \binom{n-1}{k-1} n^{n-k} k!$$

8. ⟨2⟩ As Eq. (2), Section 4.6 shows, the number of odd subsets of an  $n$ -set equals the number of even subsets.

- a) When  $n$  is odd, this fact has a particularly simple one-to-one correspondence proof using complementation. Give it. Why does it break down when  $n$  is even?  
b) Here is a one-to-one correspondence between the odd subsets of an  $n$ -set and the even subsets — a correspondence which works whether  $n$  is odd or even. Let  $x$  be any one element in the  $n$ -set,  $N$ . Then for any  $S \subset N$ , define

$$f(S) = \begin{cases} S \cup \{x\} & \text{if } x \notin S, \\ S - \{x\} & \text{if } x \in S. \end{cases}$$

Show that  $f$  is a correspondence between odd and even sets.

9. ⟨3⟩ Show by a 1-to-1 correspondence that the number of (vertex) labeled simple graphs on  $n$  vertices in which every vertex has even degree is the same as the total number of labeled simple graphs on  $n-1$  vertices. (The latter class of graphs is easily counted; see [19, Section 4.4], where labeling of graphs is also reviewed.)

Show by an example that this claim is no longer correct if all occurrences of the word labeled are removed.

10. ⟨4⟩ Show that  $(n!)^{n+1}$  is a divisor of  $(n^2)!$  by a combinatorial argument. Specifically, devise a counting problem for which  $(n^2)!$  is the answer and for which the things counted can be partitioned into subsets of size  $(n!)^{n+1}$ .

11. ⟨2⟩ In how many ways can five indistinguishable balls be put in three distinguishable urns if:  
a) The first urn must hold exactly three balls.  
b) The first urn must hold at least three balls.  
c) The first two urns each must hold at least one ball.  
d) The first two urns together must hold at least one ball.  
e) No urn may be empty.

12. ⟨3⟩ Do [11] again for distinguishable balls. Which parts get much harder?

13. ⟨4⟩ Counting binary trees. In [26, Section 3.7], you were asked to find a recurrence for the number of binary trees with  $n$  vertices. The recurrence we had in mind was

$$b_{n+1} = \sum_{k=0}^n b_k b_{n-k} \tag{1}$$

because each such tree with  $n + 1$  vertices has two binary trees dangling from its root (one of which may be empty) which, between them, have  $n$  vertices.

The right-hand side of Eq. (1) is a convolution. Use this fact to translate Eq. (1) into a quadratic equation in the “unknown”

$$B(s) = \sum_{k=0}^{\infty} b_k s^k.$$

Solve this quadratic using the quadratic formula and then expand the solution to find  $b_n$ . To do this expansion you will need the Binomial Series Theorem of Section 4.6. To get the result in its most convenient form you will have to expand a binomial coefficient with upper index  $\frac{1}{2}$  and use it to simplify the expression.

- 14.** **(3)** Exponential generating functions. Given a sequence  $\{a_n\}$ , its **exponential generating function** is defined as

$$E(s) = \sum_{n=0}^{\infty} \left( \frac{a_n}{n!} \right) s^n.$$

- a) Find the exponential generating function for  $a_n = n!$ .
- b) (Requires calculus) Find the exponential generating function for  $a_n = 1$ .
- c) Suppose that  $A(s)$  and  $B(s)$  are the exponential generating functions for  $\{a_n\}$  and  $\{b_n\}$ . Show that  $A(s)B(s)$  is the exponential generating function for

$$c_n = \sum_{k=0}^n \binom{n}{k} a_k b_{n-k}. \quad (2)$$

- d) Suppose that  $a_n$  is the number of distinguishable lists (ordered arrangements) of  $n$  things from some set  $A$ . Likewise, suppose that  $b_n$  is the number of distinguishable lists of  $n$  things from some set  $B$ . Show that if  $A \cap B = \emptyset$  (with every item in  $A$  distinguishable from every item in  $B$ ), then  $c_n$  as in Eq. (2) is the number of ways to pick a distinguishable list of  $n$  things from  $A \cup B$ . You have proved the exponential analog of Theorem 1, Section 4.8.
- e) State and prove an analog to Theorem 2, Section 4.8. Also state the  $m$ -fold generalization.
- f) Let  $A$  be an infinite set of identical blue balls. Let  $B$  be an infinite set of identical red balls. Use the earlier parts of this problem to determine the number of distinguishable  $n$ -sequences from  $A \cup B$ . (Of course, you already know the answer, but our goal here is merely to introduce exponential generating functions. As with ordinary generating functions, they can also be used to solve some problems that our previous methods couldn't.)

- 15.** **(2)** The Sum Rule with Overlaps (Section 4.3) gives the Inclusion-Exclusion formula for 2- and 3-sets. Conjecture the statement for  $n$ -sets. *Suggestion:* Call the sets  $A_i$ ,  $i = 1, 2, \dots, n$ , instead of  $A, B, C, \dots$ . You may want to invent other notation as well to keep the statement from being terribly long. If these suggestions don't help, at least come up with a verbal statement of Inclusion-Exclusion.

- 16.** **(4)** This problem derives the general Inclusion-Exclusion formula and proves it as it discovers it.

For each  $A \subset U$ , define the function

$$A(x) : U \rightarrow \{0, 1\}$$

by

$$A(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{if } x \notin A. \end{cases}$$

$A(x)$  is called the **characteristic function** of set  $A$ .

- a) Explain why  $\sum_{x \in U} A(x) = |A|$ .
- b) Show that  $A(x)B(x) = (A \cap B)(x)$ . In words, the product of the characteristic functions of two sets is the characteristic function of their intersection.
- c) Show that  $1 - A(x) = \overline{A}(x)$ . ( $\overline{A}$  is the complement of  $A$  in  $U$ .) Also, restate this equation in words.
- d) Show that  $N(x) = \prod_{i=1}^n (1 - A_i(x))$  is the characteristic function of  $\bigcap_{i=1}^n \overline{A}_i$ , the set of elements of  $U$  in none of the sets  $A_i$ .
- e) Explain why  $1 - N(x)$  is the characteristic function for  $\bigcup_{i=1}^n A_i$ , the set we want to count in Inclusion-Exclusion.
- f) Derive the Inclusion-Exclusion formula by evaluating

$$\sum_{x \in U} (1 - N(x)) = \sum_{x \in U} \left[ 1 - \prod_{i=0}^n (1 - A_i(x)) \right]$$

two ways: from the LHS and, after multiplying out the product, from the RHS.

- g) Let  $n = 3$  in part f) and check that the general formula agrees with the specific formula in the Sum Rule with Overlaps in Section 4.3.
- h) By the method of part f), but slightly simpler, one can also get an Inclusion-Exclusion type formula for  $|\bigcap_{i=1}^n \overline{A}_i|$ . Find this formula.

**17.**  $\langle 4 \rangle$  To verify the formula in Fig. 4.13 for the number of ways to put  $b$  distinguishable balls into  $u$  indistinguishable urns, we need to justify Eq. (6) in Section 4.7 for  $T(b, u)$ , the number of ways to put  $b$  distinguishable balls into  $u$  indistinguishable urns *with no urn empty*. To do this, let  $U$  be the set of all ways to put the  $b$  balls in the  $u$  urns, and let  $A_i \subset U$  be the ways in which urn  $i$  is *empty*. Use [16h] to get the needed formula for  $T(b, u)$ .

**18.**  $\langle 3 \rangle$  How many nonnegative integral solutions are there to  $x+y+z = 20$  if no unknown can be 10 or greater? *Hint:* The number of solutions in which  $x$ , say, is 10 or greater is the same as the number of all nonnegative solutions to  $x' + y + z = 10$ . Now use Inclusion-Exclusion.

**19.**  $\langle 3 \rangle$  The **Stirling numbers  $s_{nk}$  of the first kind** are defined to be the coefficients of  $x^k$  when the

falling factorial function  $x_{(n)}$  (see Eq. (2), Section 0.2) is expanded in powers of  $x$ . That is,

$$x_{(n)} = \sum_{k=0}^n s_{nk} x^k.$$

Put another way, the Stirling numbers  $s_{1n}, s_{2n}, \dots$  are the sequence for which  $x_{(n)}$  is the generating function. Note that  $s_{nk}$  is certainly 0 except for  $0 \leq k \leq n$ . This observation will “smooth out the rough edges” in b).

- a) Compute  $s_{nk}$  from its definition for  $0 \leq k \leq n \leq 3$ .
- b) Find a recurrence for  $s_{n+1,k}$  in terms of  $s_{nj}$  for various  $j$  (you figure out which  $j$ ). *Hint:*  $x_{(n+1)} = x_{(n)}(x - n)$ .
- b) Use b) to compute  $s_{nk}$  in a table for  $0 \leq k \leq n \leq 5$ .

# Difference Equations

## 5.1 Introduction

---

A **difference equation** is any definition of the general term in a sequence in terms of one or more previous terms and initial cases. Such a definition is also called a **recurrence**.

For example, in Section 1.5 we let  $h_n$  be the minimum number of moves to solve  $n$ -ring Towers of Hanoi and showed that

$$h_{n+1} = 2h_n + 1 \quad \text{for } n \geq 1, \quad h_1 = 1.$$

This is a difference equation. It is a special case of the general linear difference equation

$$a_n = \sum_{i=1}^k c_i a_{n-i} + f(n), \quad a_1, a_2, \dots, a_n \text{ specified,}$$

which is the topic of Sections 5.5 and 5.7. Here  $a_n$  is the general term of the unknown sequence, and the coefficients  $c_1, \dots, c_k$  and the function  $f(n)$  are known.

As another example, consider

$$a_n = \max\{a_{\lfloor n/2 \rfloor}, a_{\lceil n/2 \rceil}\}, \quad a_1 = 0.$$

This is nonlinear because  $\max$  is a nonlinear function. It is a special type of solvable nonlinear recurrence that shows up in studying recursive algorithms; see Section 5.8. More typical of the behavior of nonlinear recurrences is the quadratic difference equation

$$P_{n+1} = (1+r)P_n - rP_n^2, \tag{1}$$

which we study in Section 5.10.

Note that each but the last of the above examples consists of more than one equation. Technically speaking, the first equation in each display above is the

difference equation and the rest is an **initial condition** or initial conditions, but we will often regard such pairs as a unit and refer to that unit as a difference equation. You need initial conditions to get a specific sequence as the solution. Otherwise, as in Eq. (1), you get a class of solutions.

The purpose of this chapter is to introduce you to difference equations. This used to mean: learn how to solve them (get a closed form for the  $n$ th term), by hand. But now that CASs can often solve them faster, the emphasis in a good introduction has changed. We will spend considerable time on setting up difference equations, on interpreting closed form solutions, and on getting information without a closed form.

The purpose of studying any sequence is to *understand* it. We seek difference equations because they are often the best way to summarize the behavior of a sequence. If we can then get an explicit formula, that will help us understand it much better — if the closed form isn’t too complicated. For TOH, the explicit form is  $h_n = 2^n - 1$ , and this form is very understandable. But if the closed form is very complicated, or doesn’t exist, we can often get the information we want about the sequence without it, by producing a table of values, or a plot of values, or inequalities, or qualitative information like the rate of growth. To find such things the difference equation is often enough. Maybe you just want a specific term in the sequence (in the Prologue we wanted  $T(13)$ ), in which case the difference equation is certainly enough to guide a computation — no closed form required.

The fact is: relatively few difference equations can be solved in closed form, and today these can mostly be solved with a CAS. So there is no point spending a lot of time doing complicated examples by hand.

Still, there is reason to learn the *theory* of difference equations, pushing the pencil for a while to help learn it. First, the theory will teach you how to solve some simpler recurrences quickly. That works a lot faster than either guessing (as we’ve done before) or fiddling with your calculator or computer. Second, the theory leading to these easy solutions is pretty, and the methods will help you in several other branches of mathematics.

Third, the theory behind the harder recurrences your computer can solve is also very pretty, and some of it is very recent. Not only are there algorithms to find closed forms, there are algorithms that prove it when closed forms are impossible to find.

Finally, problems are often given initially in verbal form, in which case someone has to *find* a difference equation before one can worry about how to seek a solution. That someone is likely to be you, not a computer. You’ve had some practice at this in earlier chapters, but a lot more practice is in order.

Here’s the outline of the chapter. In Section 5.2 we work on **modeling** with difference equations, that is, setting them up from the words. In Section 5.3 we show how we can often answer our questions directly from the difference equation. Another purpose of these sections is to show you the variety of applications of difference equations. The examples of difference equations in earlier chapters have been from mathematics and computer science, and mostly concern algorithms. Now we cast our net more broadly.

The next four sections treat constant coefficient linear difference equations. For these recurrences the theory is central enough that you should know it and simple enough that you can solve many of them by hand. The solutions to these difference equations are usually geometric sequences or sums of geometric sequences, and the theory explains this good fortune. For the most part, we develop the theory using the method of *characteristic equations*, but at the end of Section 5.5 we briefly introduce an alternative approach, based on the generating functions introduced in Chapter 4.

After covering constant coefficient linear difference equations, you will have the knowledge to look more deeply into the use of difference equations in the analysis of algorithms; that's Section 5.8.

Once one gets beyond either constant coefficients or linearity, the theory gets much more complicated. Section 5.9 deals with variable coefficients, and introduces some exciting developments of the last 20 years — much of this subject has been completely automated. Section 5.10 uses the simplest quadratic recurrence to introduce the lively topic of *chaotic dynamics*. These two sections scratch the surface of some very active research.

In the final section, 5.11, we turn to a very classical topic, *finite differences*. By introducing the difference operator  $\Delta$ , we can explain why recurrences are called difference equations and outline the very strong parallels between the theory of differences and the theory of derivatives (calculus).

One more point: What we are solving for in this chapter are *sequences*, not individual numbers. If there wasn't a sequence, there wouldn't be different terms to relate via difference equations. But since the sort of solution we are looking for is a formula that holds for all terms, we often talk about solving for a typical term, say  $a_n$ , instead of for the sequence  $\{a_n\}$ . Such term-talk is quite common — usually it reads better and causes no confusion. But never forget that we are really talking about sequences and that  $a_n$ , say, is referring to the typical term in the sequence.

## 5.2 Modeling with Difference Equations

---

The purpose of this section is to take you through the process of setting up difference equations. We won't solve the equations we set up, but defer that until Section 5.3 or later. We have set up a few difference equations before, so in this section we'll emphasize examples that are harder or that illustrate applications not touched on earlier. However, it's a good idea to start with something simple and familiar.

### EXAMPLE 1

Describe arithmetic and geometric sequences using difference equations.

**Solution** Probably you have seen arithmetic sequences defined as sequences of the form

$$a, \quad a + d, \quad a + 2d, \quad \dots, \quad a + nd, \quad \dots$$

and geometric sequences as sequences of the form

$$a, \quad ar, \quad ar^2, \dots, ar^n, \dots$$

An arithmetic sequence is one that starts at some value  $a$  and goes up by some fixed difference  $d$  each time. Well, this last sentence is just a difference equation in words. So  $s_0, s_1, \dots$  is an arithmetic sequence if and only if

$$s_n = s_{n-1} + d, \quad n \geq 1; \quad s_0 = a.$$

Similarly, a geometric sequence is one that starts at some value  $a$  and is changed by a fixed multiplicative ratio  $r$  each time. So  $t_0, t_1, \dots$  is a geometric sequence if and only if

$$t_n = rt_{n-1}, \quad n \geq 1; \quad t_0 = a. \blacksquare$$

## EXAMPLE 2

Describe arithmetic and geometric *series* using difference equations.

**Solution** A **series** is the sum of terms from a sequence. So an arithmetic series is

$$a + (a+d) + (a+2d) + \cdots + (a+nd),$$

and a geometric series is

$$a + ar + ar^2 + \cdots + ar^n.$$

Even if we are only interested in these sums for one specific value of  $n$ , it is worthwhile to consider all the values  $n = 0, 1, 2, \dots$ . This gives us a new sequence, the **sequence of partial sums**, and so we can look for a difference equation to describe the typical term in that sequence. Specifically, for  $n \geq 0$  define

$$\begin{aligned} S_n &= a + (a+d) + (a+2d) + \cdots + (a+nd) = \sum_{k=0}^n (a+kd), \\ T_n &= a + ar + ar^2 + \cdots + ar^n = \sum_{k=0}^n ar^k. \end{aligned}$$

We want difference equations for the sequences  $\{S_n\}$  and  $\{T_n\}$ .

Now, the very definition of  $\sum$  is recursive:  $S_n$  is just  $S_{n-1}$  with one more term,  $s_n = a + nd$ , added to it. Therefore we may describe  $\{S_n\}$  by

$$S_n = S_{n-1} + (a+nd), \quad n \geq 1; \quad S_0 = a.$$

Similarly,

$$T_n = T_{n-1} + ar^n, \quad n \geq 1; \quad T_0 = a. \blacksquare$$

This same technique specifies the partial sums for *any* sequence. Hence, for any sequence  $\{a_n\}$ ,

$$A_n = \sum_{k=0}^n a_k \iff [A_0 = a_0 \text{ and } A_n = A_{n-1} + a_n, n \geq 1]. \quad (1)$$

But so what? The goal is presumably to find a closed form for these sums, and all we've obtained is difference equations. True, but that will change in Section 5.7,

which covers solutions for recurrences like Eq. (1). The method of difference equations is not the only way to find formulas for series, nor is it necessarily the simplest, but probably it is the most systematic.

### EXAMPLE 3

Devise a difference equation for the expected (average) number of comparisons in sequential search (Algorithm SEQSEARCH from Section 1.5) when the new word  $w$  is known to be on the list and it is equally likely to be at any position in the list. (To do this example you need some knowledge of probability, at least on the intuitive level of Section 1.5. We will do much more with probability in Chapter 6.)

**Solution** Recall that the algorithm simply marched through the list until it found  $w$ . Let  $E_n$  be the expected (i.e., average) number of comparisons when there are  $n$  words in the list. Fix  $k$  to be some integer between 1 and  $n - 1$ . We may now think of sequential search as a two-stage, recursive process. The first stage is sequential search applied to the first  $k$  words. If  $w$  is found, the process ends. Otherwise, we do stage two, sequential search on the remaining  $n - k$  words.

What is the expected number of comparisons in the first stage? If  $w$  is among the first  $k$  words, the answer by definition is  $E_k$ . If it is not, the answer is  $k$ : We check all  $k$  first-stage words. Furthermore, the probability that  $w$  is in the first  $k$  is  $k/n$ , since it is assumed that  $w$  is equally likely to be any one of the  $n$  words. Thus we weight the two possibilities ( $w$  among the first  $k$  or not) by their probabilities and get

$$\text{Average no. of comparisons for stage 1} = \frac{k}{n}E_k + \frac{n-k}{n}k.$$

What about the second stage? The average number of comparisons *during* stage two is  $E_{n-k}$ , but we only get there with probability  $(n-k)/n$ . So again we weight the two possibilities (get there or not) by their probabilities and get

$$\text{Average no. of comparisons for stage 2} = \frac{k}{n}0 + \frac{n-k}{n}E_{n-k}.$$

Adding, we conclude that

$$E_n = \frac{k}{n}E_k + \frac{n-k}{n}(k + E_{n-k}). \quad (2)$$

Finally, since this equation holds for any  $k$  less than  $n$ , let's pick one that's easy to compute with, namely,  $k = 1$ . It's easy to see that  $E_1 = 1$ . (Why?) Substituting 1 for  $k$  and simplifying, we find

$$E_n = \frac{n-1}{n}E_{n-1} + 1. \quad (3)$$

Compute  $E_2$  and  $E_3$  from Eq. (3) and you'll see what the explicit formula for  $E_n$  is. Sure enough, it checks with the formula we obtained by completely different methods in Example 4, Section 1.5.

The probability manipulations in this example have been quite subtle; do not worry if you are not convinced they are right. We haven't formally analyzed the concept of averages, and we won't until Chapter 6. When you get there, check out especially [42, Section 6.5]. ■

## EXAMPLE 4

Is an IRA (Individual Retirement Account) a good investment? Set up some difference equations that can help answer this question.

**Solution** IRAs are investment vehicles available to US taxpayers that shield income from current taxes. There are now many sorts; for now we limit ourselves to “traditional” IRAs. If you put some money (your “principal”) in an interest-bearing IRA account, the interest is not taxed until it is withdrawn many years later, whereas in an ordinary account the interest is taxed each year. In some cases the tax on the principal itself is also deferred.

Tax deferral is supposed to save you money. Since you pay a penalty for early withdrawal from an IRA, IRAs have the disadvantage that your money is effectively “locked up” for a long time. Thus the savings should be substantial to be worth the wait. Are they?

Individual circumstances vary, so let’s make some simplifying assumptions. (Other assumptions may appeal to you more; see [20].) Assume you have just two choices: an ordinary interest-bearing account and an interest-bearing IRA. (A person can also have IRA funds in stocks, bonds, etc.) Assume both accounts earn interest at a fixed rate of  $r$  a year (e.g.,  $r = .1$  means 10%). Assume you are not allowed to defer the tax on your principal. Thus the choice of an ordinary account or an IRA has no effect on your taxes this year, so you would have the same amount of after-tax money to start either account. Call this amount  $P$ . Assume you open the account at time 0 and plan to withdraw all the proceeds after exactly  $N$  years. Finally, assume your tax rate is  $t$ , now and in the future, under either option. For each choice, you want to compute how much money you get after paying taxes in year  $N$ .

So far we have no sequence. Let  $P_k$  be the amount in the account exactly  $k$  years from the start if you set up an ordinary account. Let  $\hat{P}_k$  be the amount after exactly  $k$  years if you set up an IRA. ( $\hat{P}$  is read “P hat”. Remember our choice this way: Hats make things bigger, as IRAs are supposed to do.) For the IRA, the compounding of interest is uninterrupted, so

$$\hat{P}_{k+1} = \hat{P}_k(1+r), \quad k \geq 0; \quad \hat{P}_0 = P. \quad (4)$$

For the ordinary account,  $t$  times the interest is taxed away each year, leaving less money to draw interest the next year. That is,

$$P_{k+1} = P_k(1+r) - P_krt, \quad k \geq 0; \quad P_0 = P. \quad (5)$$

These are the difference equations. Let  $V$  and  $\hat{V}$  be the after-tax values upon withdrawal at year  $N$ . For the ordinary account,  $V = P_N$ , because all taxes, including taxes in year  $N$ , have already been factored in. For the IRA,

$$\hat{V} = \hat{P}_N - (\hat{P}_N - P)t,$$

because all tax on the accumulated interest,  $P_N - P$ , is paid after withdrawal. So the analysis will be easy to complete as soon as we solve the difference equations. Do you recognize them as a type you have seen before? ■

The next example isn’t so much a single example as an explanation of how a whole subfield of economics can be treated with difference equations.

**EXAMPLE 5****National Income Accounting**

Use difference equations to model a national economy.

**Solution** Every student of introductory economics learns these equations:

$$Y = C + I + G,$$
$$C = cY. \quad (0 < c < 1)$$

What do they mean? First, that national income  $Y$  (the sum of all personal incomes) equals national consumption  $C$  (the sum of all personal spending), plus investment  $I$ , plus government spending  $G$ . Second, that consumption is a fraction  $c$  of national income.

Why are these equations right? The second is pretty clear; the only question is whether  $c$  is a constant, and if not, what it depends on. For simplicity, in introductory economics  $c$  is assumed to be a constant; we will assume the same. As for the first equation, a brief explanation goes like this (economists please forgive us!). All money that is spent eventually goes into the pockets of *people*. For instance, money invested in municipal bonds might be spent to build a bridge, in which case all the money eventually gets to the contractors and their employees, to the workers who make the steel girders, etc.

One trouble with this model (there are lots of troubles) is that it assumes all these transfers take place instantaneously. In fact, if you buy a bridge bond, it may be a year before a steelworker takes some of that money home. A better model builds in such time lags. To model this with discrete mathematics, we divide up time into periods, say, years or quarters. Let  $Y_n$  be the national income in period  $n$ , and likewise for  $C$ ,  $I$  and  $G$ . Then a slightly more reasonable model is

$$\begin{aligned} Y_n &= C_{n-1} + I_{n-1} + G_{n-1}, \\ C_n &= cY_{n-1}. \end{aligned} \tag{6}$$

Just as intro econ students solve the simpler model for  $Y$ , let us try to solve for  $Y_n$ . First let us take the case that  $I$  and  $G$  do not vary from period to period. Then substituting the second equation above into the first (with  $n$  replaced in the second equation by  $n - 1$ ), we get

$$Y_n = cY_{n-2} + (I + G),$$

where  $I + G$  is constant. So, once we know  $Y$  for two consecutive periods, we can figure it out for all future periods.

To take a slightly more complicated version, let's continue to assume that government spending is constant, but now let

$$I_n = A + dC_{n-1}, \tag{7}$$

where  $A$  and  $d$  are constants. That is, we now assume there is a rock-bottom amount  $A$  of investment even if the economy is bust ( $C = 0$ ), but that the more people have spent in the previous period, the more confidence people and business will have for investing in the current period.

Substituting Eq. (7) into the first line of Eq. (6), and then substituting the second line of Eq. (6) into the first twice, we find that

$$Y_n = cY_{n-2} + cdY_{n-3} + (A + G).$$

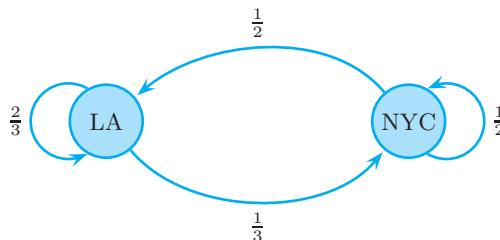
The actual economic models on which public policy is based are much more complicated than these, but these illustrate the basic technique. Economic forecasting involves lots of difference equations.

## EXAMPLE 6

### Business inventory

Trusty Rent-A-Car has offices in New York City and Los Angeles. It allows customers to make local rentals or one-way rentals to the other location. Each month, it finds that half the cars that start the month in NYC end it in LA, and one third of the cars that start the month in LA end it in NYC. (See Fig. 5.1.) If at the start of operations Trusty has 1000 cars in each city, what can it expect  $n$  months later? Model this situation with a difference equation.

(Of course, no car rental company has just two offices, certainly not two so far apart. Our example is fanciful, but problems like it are real, and difference equations in many variables are used to solve them, along the lines below.)



**FIGURE 5.1**

A digraph of monthly car movements for Trusty Rent-A-Car.

**Solution** Let  $N_n$  be the number of cars in New York at the beginning of month  $n$ , with the start of operations considered to be the beginning of month 0. Define  $L_n$  similarly. Then we have the linked difference equations

$$\begin{aligned} N_n &= \frac{1}{2}N_{n-1} + \frac{1}{3}L_{n-1}, \\ L_n &= \frac{1}{2}N_{n-1} + \frac{2}{3}L_{n-1}, \end{aligned} \tag{8}$$

with initial conditions

$$L_0 = N_0 = 1000.$$

We say “linked” because each variable depends on previous values of the other variable as well as of itself. It’s considerably harder to separate the variables here than in Example 5. Try it! It can be done [31], but it’s easier to study this sort of problem by matrix methods not treated in this book (but see [8]).

In any event, it’s not hard at all to compute some terms of  $\{N_n\}$  and  $\{L_n\}$  and look for a pattern [7]. In fact, do you have any intuition about what will happen? Will Trusty suffer wild oscillations in the locations of its cars or will things “settle down”? ■

## EXAMPLE 7

Find a recurrence for the number of indistinguishable balls in distinguishable urns.

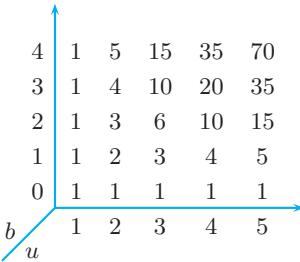
We solved this problem in Section 4.7, but we promised other approaches and here is one. Let us denote the answer by  $C^*(u, b)$ . (We could use double-index notation like  $c_{u,b}$  instead. You should get accustomed to “function notation” and “index notation” for both one-variable and two-variable difference equations.) We use  $C^*$  and put  $u$  first because we showed in Section 4.7 that the answer we seek now is the number of combinations of  $b$  things from  $u$  allowing repetitions, which we have already called  $C^*(u, b)$ .

**Solution** We devise a difference equation and from that compute a chart of values to look at for conjectures. Think recursively. Let’s pick one of the urns; since they’re distinguishable we can give it a number, say, 1. Either some balls go in urn 1 or none do. If none do, we must put all  $b$  balls in the other  $u - 1$  urns. There are  $C^*(u-1, b)$  ways to do this. If some balls go in urn 1, let’s begin by dropping exactly one ball in — any one. There’s only one way to do this, not  $b$  ways, since balls are indistinguishable. Now what? We have  $b - 1$  balls to put in how many urns? The answer is  $u$ , not  $u - 1$ , because now we are allowed to use urn 1 again. Thus there are  $C^*(u, b-1)$  ways to finish up. By the Sum Rule we have the recurrence

$$C^*(u, b) = C^*(u, b-1) + C^*(u-1, b). \quad (9)$$

This argument makes sense so long as  $u \geq 2$  and  $b \geq 1$ . (Why?) So we must compute the base cases  $u = 1$  or  $b = 0$  directly. This is easy.  $C^*(1, b) = 1$  because there is exactly one way to “distribute”  $b$  balls into one urn, even if  $b = 0$ . Also,  $C^*(u, 0) = 1$  for any  $u \geq 1$  because there is exactly one way to distribute no balls into  $u$  urns: Do nothing!

These base cases give us the first column and row of Fig. 5.2. Now we use Eq. (9) to fill in the rest; it says that each entry is the sum of the entry directly below it and the entry directly to the left. Note that the first variable indexes columns, and the second indexes rows, with the row number increasing as you go up the page, just as in the standard  $(x, y)$  coordinate system.



**FIGURE 5.2**

The number of ways to put  $b$  indistinguishable balls into  $u$  distinguishable urns, computed from Eq. (9).

In any event, can you look at this chart, use the inductive paradigm, and complete the solution yourself? We present our solution in Example 5, Section 5.3. ■

## EXAMPLE 8

Discuss the use of difference equations in population models.

**Solution** This is an enormous field. We will sketch a few ideas for setting up models and let you do some more in the problems.

The simplest population assumption is constant growth rate. Let  $P_n$  be the numerical size of some population at time period  $n$  (measured in some convenient units, maybe thousands or millions, or maybe some other unit suggested by the problem, as we will see shortly). If the population grows by 2% over each time period, then we have

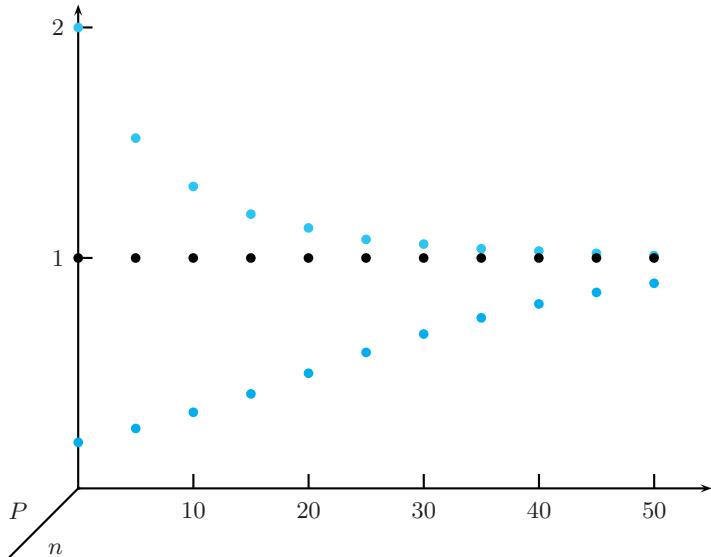
$$P_{n+1} = (1.02)P_n.$$

This, of course, gives a geometric sequence (the discrete analog of exponential functions in calculus).

However, there may be a crowding effect. A better equation might be

$$P_{n+1} = P_n \left[ 1 + r \left( 1 - \frac{P_n}{C} \right) \right]. \quad (10)$$

where the positive constant  $r$  would be the growth rate if there were no crowding effect, and  $C$  is a positive constant that enables the factor  $1 - (P_n/C)$  to model the crowding. So long as  $P_n < C$ , the population increases from  $P_n$  to  $P_{n+1}$ , but the closer  $P_n$  is to  $C$ , the smaller the amount of increase. Similarly, if  $P_n > C$ , then  $P_{n+1} < P_n$ , but the closer  $P_n$  is to  $C$  the smaller the decline. Finally, if  $P_n = C$ , then  $P_{n+1} = P_n$ ; the population doesn't change. See Fig. 5.3, which illustrates all three situations. Do you see why  $C$  is called the **carrying capacity** of the system? Also, look particularly at the plot that starts at  $P_0 = .2$  (dark color). Notice how the population rises more and more steeply for a while, and then less and less steeply as it approaches the carrying capacity. This is very common population behavior. Data like this is said to trace a “logistic” curve (or S-curve).



Graphical solution to population model (10) with  $r = .07$ ,  $C = 1$  (in thousands, say) and initial conditions  $P_0 = .2$  (dark color),  $P_0 = 1$  (black) and  $P_0 = 2$  (light color). Every fifth term is shown.

**FIGURE 5.3**

There is no reason to suppose a real-world population follows exactly this model of crowding. Instead of the term  $r(1-(P_n/C))$  there is probably some more general function  $f(P_n)$ . The key requirement is that  $f$  be decreasing with a zero at some number  $C$ . However, even with the simple form of Eq. (10), lots can happen; see [24–26]. Eq. (10) is a quadratic difference equation: if you multiply it out, you get a term  $P_n^2$ . This is the simplest sort of **nonlinear difference equation**; for more complex functions  $f(P_n)$  we would get a more general nonlinear equation.

One more observation about Eq. (10). We can simplify it further as follows. First divide both sides by  $C$ :

$$\frac{P_{n+1}}{C} = \frac{P_n}{C} \left[ 1 + r \left( 1 - \frac{P_n}{C} \right) \right].$$

Now redefine  $P_n$  (for any index) by  $P_n \leftarrow P_n/C$ ; the result is

$$P_{n+1} = P_n [1 + r(1-P_n)] = (1+r)P_n - rP_n^2. \quad (11)$$

Let us call this the *normalized* form of (10). By rescaling  $P_n$  by  $C$ , we have in effect declared the carrying capacity to be our unit of population and eliminated the need for the letter  $C$ . For instance, if  $P_n = 1$ , that means the population is at its capacity.

We return to nonlinear systems in Section 5.10, where we use the normalized form (11).

Whenever sequences are used to model change over *time* (such as with population change), the subject is often called **discrete dynamical systems**. This name is especially common for the study of the wilder behavior introduced in Section 5.10.

For population studies it is a crude approximation to assume that births are some fraction of the whole population, as we have assumed above. Except for single-cell organisms, only part of the population is of “child bearing” age. Thus, to refine the model a bit, let  $C_n$  and  $A_n$  be, respectively, the number of children and adults at time  $n$ . Then you might have a set of difference equations like this:

$$\begin{aligned} C_n &= aC_{n-1} + bA_{n-1}, \\ A_n &= cC_{n-1} + dA_{n-1}, \end{aligned} \quad (12)$$

where  $a, b, c, d$  are positive constants. For instance,  $a < 1$  but probably close to 1, taking into account that a few children die and some graduate into adulthood. The constant  $b$  takes into account that a fraction of adults give birth to children. And so on. Even this is crude. For human populations, demographers break things down, at the least, into age groups like 5–9, 10–14, etc.

Let us briefly mention the case of competing populations. Let  $W_n$  and  $R_n$  be the number of wolves and rabbits on a certain range during time period  $n$ . It seems reasonable to suppose that

$$R_n = aR_{n-1} + bW_{n-1},$$

because rabbits produce rabbits and wolves eat rabbits; note therefore that this time  $b < 0$  ( $a$  is still positive). Indeed, this competition scenario leads to a linked linear system as in Eqs. (12) — what’s a plausible equation for  $W_n$ ? ■

Finally, we can't leave the subject of difference equations in population models without introducing the most famous population problem ever: Fibonacci's rabbit problem. In 1202, the Italian mathematician Fibonacci (also known as Leonardo of Pisa) posed the following:

### EXAMPLE 9

Suppose at the end of every month a pair of rabbits gives birth to one baby pair of rabbits, and that these baby pairs in turn give birth to a pair at the end of every month, starting at the end of their second month, and so on with each generation. If you start with one pair of newborn rabbits, how many pairs will you have exactly  $n$  months later? Assume that rabbits never die (not an unreasonable simplifying assumption if the values of  $n$  that really interest you are less than the lifespan of rabbits).

**Solution** Probably you've already guessed that the answer is the Fibonacci sequence we defined in Example 2, Section 2.7. But let's apply the recursive paradigm from scratch. Let  $r_n$  be the number of pairs alive exactly  $n$  months after you start with the newborn pair. Thus  $r_0 = 1$ . Can we express  $r_n$  in terms of earlier  $r$ 's? Of course. Those pairs alive exactly  $n$  months after the start are just those alive  $n - 1$  months after the start plus those born exactly  $n$  months after the start. We can write this as

$$r_n = r_{n-1} + b_n,$$

but this gives us two variables,  $r$  and  $b$ . The key additional observation is that the number *born*  $n$  months after the start ( $b_n$ ) equals the number *alive*  $n - 2$  months after the start — because there is one newborn pair for each previous pair that is at least two months old, and a pair is at least two months old at time  $n$  iff it was alive at time  $n - 2$ . Thus

$$r_n = r_{n-1} + r_{n-2}.$$

This is the Fibonacci recurrence from Chapter 2. Using the initial conditions  $r_0 = 1$  and  $r_1 = 1$  (where does the latter come from?) we can easily compute as many terms as we wish:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \dots \blacksquare \tag{13}$$

This sequence shows up in an amazing number of places having nothing to do with rabbits. For instance, it shows up in the complete answer to a question touched on in Example 4 of Section 1.1: how efficient is Euclid's gcd algorithm? See [15], which gives just a hint of the connection to Fibonacci numbers.

*Caution.* For us, the initial conditions for the Fibonacci numbers are  $r_0 = r_1 = 1$ , but some authors use  $r_1 = r_2 = 1$ , that is, for them the sequence (13) starts with  $r_1$  instead of  $r_0$ . For instance, if you want to publish in the *Fibonacci Quarterly*, and you want to call the Fibonacci sequence  $\{f_n\}$  (lowercase), then you are required to let  $f_1 = f_2 = 1$ . Formulas differ slightly depending on which convention is used.

## Problems: Section 5.2

The purpose of this exercise set is to have you practice setting up difference equations. Even where a problem seems to ask for an explicit solution (e.g., [16]), it suffices to set up the appropriate difference equation and initial conditions. If you can guess an explicit formula from your equation, great; if you can prove it, super; but neither is part of the assignment — yet.

1.  $\langle 1 \rangle$  Devise a difference equation for the series  $1 + 4 + 9 + \dots + n^2$ .
2.  $\langle 2 \rangle$  Consider the sequence  $1, 3, 7, 13, 21, 31, \dots$ . Devise a difference equation for the sequence. Devise a difference equation for the associated series.
3.  $\langle 1 \rangle$  An **arithmetic-geometric** sequence is one that starts at some value  $a$ , and thereafter each term is obtained from the previous one by first multiplying by a constant ratio  $r$  and then adding a constant difference  $d$ . Find a difference equation for this sort of sequence.
4.  $\langle 1 \rangle$  In Example 3, what recurrence do you get if you use  $k = n - 1$  in Eq. (2)? Also, find a recurrence relating  $E_{2n}$  to  $E_n$  by replacing  $n$  by  $2n$  and  $k$  by  $n$ .
5.  $\langle 2 \rangle$  Modify the second equation of the national income model, Eq. (6), so that  $C_n = cY_n$ . That is, we now think of personal income as being consumed instantaneously, at least relative to the rate at which  $C + I + G$  turns into people's income.
  - a) Find the difference equation for  $Y_n$  (involving no other variables) assuming that  $I + G$  is constant.
  - b) Find the difference equation for  $Y_n$  assuming that  $G$  is constant but  $I$  satisfies Eq. (7).
6.  $\langle 2 \rangle$  Modify the national income model so that  $G$  is no longer constant. Specifically, assume that  $G_n = \bar{G} - kY_n$ , where  $\bar{G}$  and  $k$  are constants.
  - a) Explain why this difference equation for  $G_n$  is plausible if the government has the philosophy that it should try to counter economic cycles. What do  $\bar{G}$  and  $k$  represent?
  - b) Find the difference equation for  $Y_n$  (involving no other variables) assuming  $I$  is constant.
  - c) Find the difference equation for  $Y_n$  if  $I$  satisfies Eq. (7).
7.  $\langle 2 \rangle$  In Example 6, compute  $N_k$  and  $L_k$  through

$k = 5$ , or until you think you see what's happening. Assume  $N_0 = L_0 = 1000$ . You'll start to get fractional values, which couldn't be true of an exact model of real rental cars, but best not to round for now. If your hand calculator or CAS can do matrix calculations, see [8] to speed up the work.

8.  $\langle 2 \rangle$  With the notation of Example 6, define  $\mathbf{u}_n = (N_n, L_n)^T$  and define  $A = \begin{bmatrix} 1/2 & 1/3 \\ 1/2 & 2/3 \end{bmatrix}$ .
  - a) Verify that the linked system of equations (8) is just the *single* matrix equation  $\mathbf{u}_n = A\mathbf{u}_{n-1}$ .
  - b) Thus, show by induction that  $\mathbf{u}_n = A^n \mathbf{u}_0$ . So the rental car problem is just a geometric sequence problem, except that the “base” is a matrix  $A$  instead of a number  $r$ .
9.  $\langle 1 \rangle$  Los Angeles suffers an earthquake and becomes less popular. Trusty Rent-a-Car now finds that only  $1/4$  the cars in NYC end up in LA the next month, whereas  $4/5$  the cars rented in LA end up in NYC. Revise the linked difference equations of Example 6 to reflect this change.
10.  $\langle 1 \rangle$  Developing countries have a problem with too many people moving to cities. Country X does a study and finds that, in a given year,  $10\%$  of the rural population moves to a city, but only  $1\%$  of the urban population goes back to the country. Describe this situation with difference equations.
11.  $\langle 2 \rangle$  In [10], let  $r_k, c_k$  be the rural and city population, respectively, at the start of year  $k$ . Compute  $r_k$  and  $c_k$  through  $k = 5$ , or until you think you see what's happening. Assume  $N_0 = L_0 = 1$  million.
12.  $\langle 2 \rangle$  Change Fibonacci's problem slightly. Once a pair of rabbits starts having children, assume they have two new pairs every month. Also, assume

that you start at  $n = 0$  with three newborn pairs. What is the recurrence now? What are the first ten terms? Do you see a pattern?

13. ⟨3⟩ Change Fibonacci’s problem again. At age two months, a pair has one new pair; every month after that it has two pairs. What is the recurrence now?

14. ⟨4⟩ Change Fibonacci’s problem once more. As in the original problem (Example 9), rabbits give birth to a new pair at the start of every month beginning when they are 2 months old, but now each pair dies, just before they turn 12 months old. Find a recurrence for  $r_n$ . *Caution:* It’s tempting to say that

$$r_n = r_{n-1} + r_{n-2} - r_{n-12}$$

because  $r_{n-2}$  counts the new births at month  $n$ , and  $r_{n-1} - r_{n-12}$  counts the pairs which continue living from the previous month — but this analysis is wrong!

15. ⟨3⟩ Let  $F_n$  be the  $n$ th Fibonacci number. How many iterations does Algorithm EUCLID (Example 4, Section 1.1) use to compute  $\gcd(F_n, F_{n-1})$ ? Let  $a_n$  be the answer, and find a difference equation for it.

16. ⟨3⟩ How many  $n$ -symbol code words of dots and dashes are there if dashes may not be consecutive?

17. ⟨3⟩ How many monotonic increasing sequences of positive integers are there which end at  $n$  and do not contain any consecutive integers? For instance, when  $n = 1$  there is just one, the singleton sequence (1). When  $n = 4$  there are three: (1,4), (2,4) and (4).

18. ⟨4⟩ Suppose  $n$  planes float in space in **general position** — every two intersect in a line, every three intersect in a single point, but no four have a point in common. Into how many solid regions,  $S_n$ , do the planes divide space? *Hint:* Look at the intersection of the  $n$ th plane with the others; this consists of  $n - 1$  lines dividing the  $n$ th plane into regions. You can find a recurrence for  $S_n$  in terms of the number of regions,  $r_{n-1}$ , formed on the  $n$ th planes by these  $n - 1$  lines. This latter question was addressed in Example 5, Section 2.6.

- a) Using the closed form for  $r_n$  from that example, come up with a recurrence for  $S_n$ .
- b) Using just the recurrence for  $r_n$  from that example, come up with a *semilinked* system of

two equations in  $S_n$  and  $r_n$ . It is semilinked because the equation for  $S_n$  involves lower-indexed terms for both  $S$  and  $r$ , but the equation for  $r_n$  involves only lower indexed terms for  $r$ .

19. ⟨2⟩ If your income is low enough, you may qualify to defer taxes on the principal of a traditional IRA as well as on the investment income. In this case, you can put more principal into an IRA than into an ordinary account (contrary to the assumption in the text) because you need to put less money aside to pay taxes. Redo the difference equation analysis in the text for this case. Find difference equations for  $P_k$  and  $\hat{P}_k$  and formulas for  $V$  and  $\hat{V}$ .
20. ⟨3⟩ We don’t claim that the IRA analysis in the text and in [19] cover all possibilities. For instance, there is a cap on IRA contributions, depending on your age and the year. Also, our analysis in [19] seems to assume that people spend a fixed amount and save whatever is left over; other assumptions may be more reasonable. Another point is that we assumed a fixed tax rate. Many people expect to have a lower tax rate when they withdraw the money, usually during retirement. Come up with at least one other IRA model, incorporating whatever changes you think are most needed.
21. ⟨3⟩ Actually, you don’t just start an IRA one year and let it sit. You are allowed to put in money every year (up to a limit that depends on the year and your age). Do Example 4 again under the assumption that you have a fixed amount  $P$  available (and allowed) every year for augmenting your principal. As in the text, assume that  $P$  is the same whether you use an IRA or an ordinary account. Let  $P_n$  be the total amount in your account at the *end* of year  $n$ , just after you accrue interest and pay taxes for the year, and just before you put in another  $P$  at the start of year  $n + 1$ . (The initial index is  $n = 0$  and  $P_0 = 0$ .) Assume you withdraw your money at the end of year  $N$ .
22. ⟨1⟩ In a newer sort of IRA, the Roth IRA, taxes on contributions are never deferred, but there is no tax on either principal or interest upon withdrawal. Redo the analysis of Example 4 substituting a Roth IRA for a traditional IRA.
23. ⟨2⟩ Let  $c_{u,b}$  be the number of ways to put  $b$  indistinguishable balls into  $u$  distinguishable urns if

each urn can hold at most one ball. Find a difference equation and initial conditions for  $c_{u,b}$ . Create a chart of values like Fig. 9. Do you recognize these numbers?

24. (2) Consider Eq. (10), the model of population growth with crowding. Compute several terms of the sequence for  $C = 1$ ,  $P_0 = .5$  and  $r =$

- a) .5      b) 1      c) 1.5  
d) 2      e) 2.5      f) 3

25. (1) Do [24] again, except use  $P_0 = 1$ . (You should be able to solve all the parts at once.)

26. (2) In light of [24, 25], what do you think it means for a difference equation to have an *equilibrium point*? To be *stable*? For what values of  $r$  does it seem that the Eq. (10) is stable? Such ideas are discussed further in Section 5.10.

27. (2) Consumer loans work as follows. The lender gives the consumer a certain amount  $P$ , the principal. At the end of each payment period (usually each month) the consumer pays the lender a fixed amount  $p$ , called the payment. Payments continue for a prearranged number  $N$  of periods, say 60 months = 5 years. The value of  $p$  is calculated so that, with the final payment, the principal and all interest due have been paid off exactly. During each payment period, the amount that the consumer still owes increases by a factor of  $r$ , the period interest rate; but it also decreases at the end of the period by  $p$ , since the consumer has just made a payment.

Let  $P_n$  be the amount the consumer owes just after making the  $n$ th payment. Find a difference equation and **boundary conditions** for  $P_n$ . (Boundary conditions are initial conditions that apply to non-consecutive terms.)

28. (4) The payment  $p$  in [27] is considered to have two parts: payment of interest and return of principal. (This distinction is important for tax purposes.) The interest is the increase in the amount due during the period (from multiplying by  $r$ ); the return of principal is the rest. These amounts vary from period to period, but their sum is always  $p$ .

- a) Let  $I_n$  and  $R_n$  be the interest and return of principal in period  $n$ . Express these in terms of  $P_n$  (or  $P_{n-1}$  if that's simpler).

- b) Consumers are often surprised when the lender shows them a table of values of  $I_n$  and  $R_n$ . For the longest time, it seems that you mainly pay interest, but towards the end  $R_n$  shoots up rapidly. This suggests that  $R_n$  grows exponentially. Show that this is correct by finding a difference equation for  $R_n$ .

- c) Find a difference equation for  $I_n$ .

29. (3) In demography (human population studies) it is sufficient to keep track of the number of females, because only females give birth. Models used for serious projections usually divide females into groups by age in years. Let  $P(n, a)$  be the number of females of age  $a$  years in the United States at the start of calendar year  $n$ . (As usual,  $a$  is an integer, the floor of one's exact age; e.g., someone 23 years and 4 months old is age 23.) For each age  $a$  (say, from 0 to 99), such a model would express  $P(n+1, a)$  as a function of  $P(n, b)$  for one or more  $b$ 's. What is a reasonable difference equation for  $P(n+1, 0)$ ? For  $P(n+1, a)$  where  $a \geq 1$ ?

30. (3) Using the notation  $[n] = \{1, 2, \dots, n\}$ , define  $a_n = \sum_{P \subset [n]} |P|$ . Find a difference equation relating  $a_{n+1}$  and  $a_n$ . Hint: For every subset  $P$  of  $[n]$  you get two subsets of  $[n+1]$ , namely  $P$  and  $P \cup \{n+1\}$ .

31. (3) Unlink the difference equations in Example 6. Find an expression for  $N_n$  in terms of  $N_{n-1}$  and  $N_{n-2}$ . No values of  $L$  should appear.

32. (3) Even when a quantity has an explicit formula, numerical calculations may be more reliable if you use difference equations. This is most likely to be true when the quantity is doubly indexed, such as  $C^*(u, b)$ . Develop difference equations for

- a)  $P(n, k)$  and  $P^*(n, k)$  from Section 4.4 (permutations and permutations with repetitions). Note that  $P(n, k) = n!/(n-k)!$ , but when, say,  $n = 100$  and  $k$  is small, this explicit formula can cause overflow even though  $P(n, k)$  is not that large and your difference equation approach easily computes the value.

- b)  $B(b, u)$ , the number of ways to put  $b$  distinguishable balls into  $u$  indistinguishable urns; see Eq. (5) in Section 4.7.

## 5.3 Getting Information from Difference Equations

This section is about the use of the inductive paradigm in the analysis of difference equations. Continuing with several examples from the previous section, we will guess or otherwise concoct closed forms and then prove them correct by induction. We also show that you can often establish very useful information even when you can't find a closed form. As we explained in Section 5.1, the approaches illustrated in this section are important because many difference equations cannot be solved by the systematic methods of later sections. Even when those systematic methods apply, the approaches of this section are sometimes faster.

### EXAMPLE 1

Conjecture and prove an explicit formula for  $E_n$ , the expected length of sequential search from Example 3, Section 5.2.

**Solution** In Section 5.2 we showed that  $E_1 = 1$  and

$$E_n = \frac{n-1}{n} E_{n-1} + 1. \quad (1)$$

From this equation it is easy to compute

$$E_2 = \frac{3}{2}, \quad E_3 = 2, \quad \text{and} \quad E_4 = \frac{5}{2},$$

from which we guess

$$E_n = \frac{n+1}{2}.$$

We will now prove this equality by induction. Let  $P(n)$  be the assertion that the guess is correct for the particular value  $n$ . We already know that  $P(1)$  is true. Assuming  $P(n-1)$  and substituting in Eq. (1), we have

$$E_n = \frac{n-1}{n} \left( \frac{(n-1)+1}{2} \right) + 1 = \frac{n-1}{2} + 1 = \frac{n+1}{2}. \blacksquare$$

Even when you can't guess an explicit formula for a series by computing values from its difference equation, you can still use the difference equation to prove a formula if you discover it by some other means. For instance, for the geometric series  $T_n = \sum_{k=0}^n ar^k$  we showed (Example 2, Section 5.2) that

$$T_n = T_{n-1} + ar^n, \quad T_0 = a. \quad (2)$$

Perhaps you seem to recall from some earlier course that

$$T_n = \frac{a(r^{n+1} - 1)}{r - 1}, \quad (3)$$

but you are not sure it's correct. If we can prove Eq. (3) inductively using (2), it's correct.

### EXAMPLE 2

Prove Eq. (3) by induction using Eq. (2).

**Solution** The initial case of Eq. (3),  $n = 0$ , is easy: does  $a = a(r-1)/r-1$ ? Yes, at least when  $r \neq 1$ , which reminds us that Eq. (3) is only correct when  $r \neq 1$ . The inductive step is the following calculation:

$$\begin{aligned} T_n &= \frac{a(r^{(n-1)+1} - 1)}{r-1} + ar^n = \frac{ar^n - a}{r-1} + \frac{ar^n(r-1)}{r-1} \\ &= \frac{(ar^n - a + ar^{n+1} - ar^n)}{r-1} = \frac{a(r^{n+1} - 1)}{r-1}. \quad \blacksquare \end{aligned}$$

Even if you misremember a formula, induction can help you recover. For instance, if you try to prove a wrong explicit formula for  $T_n$  using Eq. (2), it won't work, and why it doesn't work may suggest what you are misremembering [6].

### EXAMPLE 3

Complete the analysis of IRAs from Example 4, Section 5.2.

**Solution** Recall the notation we set up in Section 5.2:  $P$  is the original principal,  $P_k$  the amount of money in the account at the end of  $k$  years,  $r$  is the yearly interest rate,  $t$  the tax rate upon withdrawal, and  $V$  the amount after taxes upon full withdrawal after  $N$  years. Finally, hats on  $P$  and  $V$  are for an IRA account, and no hats are for an ordinary interest-bearing account. We found that

$$\begin{aligned} P_{k+1} &= P_k(1+r) - P_krt, \tag{4} \\ \hat{P}_{k+1} &= \hat{P}_k(1+r), \\ V &= P_N, \\ \hat{V} &= \hat{P}_N - (\hat{P}_N - P)t = \hat{P}_N(1-t) + Pt. \end{aligned}$$

Clearly  $\{\hat{P}_n\}$  is a geometric sequence, so

$$\begin{aligned} \hat{P}_N &= P(1+r)^N, \\ \hat{V} &= P[(1+r)^N(1-t) + t]. \end{aligned}$$

Now the key observation: The sequence  $\{P_k\}$  is also geometric, because Eq. (4) may be rewritten as

$$P_{k+1} = P_k[1 + r(1-t)].$$

Thus

$$V = P_N = P[1 + r(1-t)]^N.$$

So which investment is better? It makes sense to look at  $\hat{V}/V$ . We find that

$$\frac{\hat{V}}{V} = \frac{(1+r)^N(1-t) + t}{[1 + r(1-t)]^N}. \tag{5}$$

This equation doesn't simplify readily, but if we delete the positive term  $t$  from the numerator we get

$$\begin{aligned} \frac{\hat{V}}{V} &> \frac{(1+r)^N(1-t)}{[1 + r(1-t)]^N} \\ &= \left[ \frac{1+r}{1+r(1-t)} \right]^N (1-t). \end{aligned} \tag{6}$$

Consider the bottom line in display (6). The factor

$$\frac{1+r}{1+r(1-t)}$$

is a constant greater than 1, so if the power  $N$  is large enough,

$$\left[ \frac{1+r}{1+r(1-t)} \right]^N (1-t)$$

will be greater than 1 even though the final factor  $1-t$  is less than 1. In fact, the larger  $N$  is, the more  $1-t$  will be overwhelmed.

For example, consider the values  $r = .05$  and  $t = .25$ . If you start an IRA at age 35 and retire upon reaching 65, then  $N = 65 - 35 = 30$  and inequality (6) becomes

$$\frac{\hat{V}}{V} > \left[ \frac{1.1}{1.075} \right]^{30} (.75) \approx 1.074.$$

(Using Eq. (5) rather than inequality (6), we find that  $\hat{V}/V$  is 1.157.) And had you been perspicacious enough to start your IRA at age 30 instead of 35, then from (6) we would find  $\hat{V}/V > 1.141$ . (The value from Eq. (5) is 1.209.) These are not tremendous gains over non-IRA investments, but every penny counts.)

If you are allowed to defer taxes on principal as well, the advantage of an IRA is greater [10]. Also, you can fund a traditional IRA with riskier investments and if they do get a high return, the gain over the same investment outside an IRA is substantial. Finally, more recent types of IRAs, like Roth IRAs, are generally taxed less and so have a higher gain over non-IRA investments even with today's lower  $r$  values [11]. ■

## EXAMPLE 4

### Rent-A-Car Inventory

In Example 6, Section 5.2, we determined that

$$\begin{aligned} N_n &= \frac{1}{2}N_{n-1} + \frac{1}{3}L_{n-1}, \\ L_n &= \frac{1}{2}N_{n-1} + \frac{2}{3}L_{n-1}, \\ L_0 &= N_0 = 1000, \end{aligned}$$

where  $N_k$  and  $L_k$  are the number of cars in NYC and LA, respectively, at time  $k$ . Although individual cars will keep moving from NYC to LA and back, the *number* of cars in each place might stabilize. If we owned the rental agency, that would certainly make life simpler! If stability occurred, we would say that the agency was in **equilibrium**. What would this equilibrium look like?

**Solution** If an equilibrium is achieved, then for all  $n$  after some point,  $N_{n-1} = N_n = N$  and  $L_{n-1} = L_n = L$ . Substituting into the difference equations we get

$$\begin{aligned} N &= \frac{1}{2}N + \frac{1}{3}L \\ L &= \frac{1}{2}N + \frac{2}{3}L, \end{aligned} \tag{7}$$

both of which can be rewritten as

$$\frac{1}{2}N - \frac{1}{3}L = 0.$$

Then using  $N + L = 2000$  to solve this, we conclude: *If* the numbers stabilize, they must stabilize at  $N = 800$ ,  $L = 1200$ .

What we now know with certainty is this: If Trusty Rent-A-Car starts with  $2/5$  of its cars in NYC (800 of its 2000), it will continue to have  $2/5$  of its cars there at the end of each period. It is also true, but we aren't prepared to prove it here, that no matter how the cars are distributed initially, in time the distribution converges on this  $(2/5, 3/5)$  equilibrium. ■

In Example 6, the total number of cars was not changing. Difference equations for such “closed” systems usually result in sequences which head towards equilibrium. Such closed difference equation systems are called **Markov Chains**.

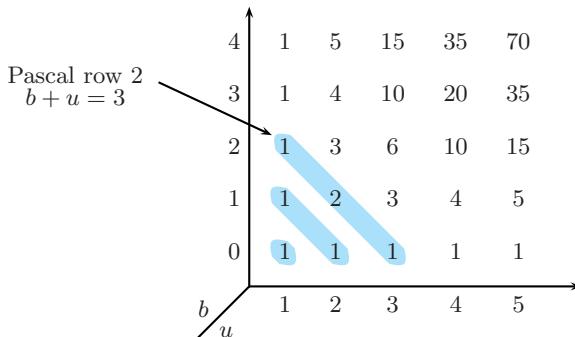
## EXAMPLE 5

Determine the number of ways to put  $b$  indistinguishable balls in  $u$  distinguishable urns.

**Solution** In Example 5, Section 5.2, we obtained the recurrence

$$C^*(u, b) = C^*(u, b-1) + C^*(u-1, b) \quad (8)$$

and computed the values shown in Fig. 5.2, repeated here as Figure 5.4.



**FIGURE 5.4**

Ways to put  $b$  indistinguishable balls into  $u$  distinguishable urns, with Pascal’s Triangle noted in color.

Voilà, this is Pascal’s Triangle, only arranged in a funny way. Thus  $C^*(u, b)$  equals  $C(n, k)$  for some choice of  $n, k$  that we need to figure out. On each Pascal Triangle row (the first few marked in color)  $u+b$  is a constant and thus corresponds to an  $n$  value. For instance, the 1 2 1 diagonal, where  $u+b=3$ , is the  $n=2$  row of Pascal’s Triangle — remember, Pascal’s Triangle starts with the  $n=0$  row. On such a diagonal  $b$  goes from 0 (lower right) to  $n$  (upper left), as does  $k$ . Thus  $C(n, k) = C(u+b-1, b)$ . So

$$C^*(u, b) = \binom{u+b-1}{b} = \frac{(u+b-1)!}{b!(u-1)!}. \quad (9)$$

Having discovered Eq. (9), we prove it by induction using Eq. (8). Let  $P(u, b)$  be the statement that Eq. (9) holds for the particular pair  $(u, b)$ . (Note:  $P(u, b)$  is a *proposition*, not a count of permutations.) Equation (8) suggests that we should assume the previous cases  $P(u, b-1)$  and  $P(u-1, b)$ . Then

$$C^*(u, b) = C^*(u, b-1) + C^*(u-1, b)$$

[Eq. (9)]

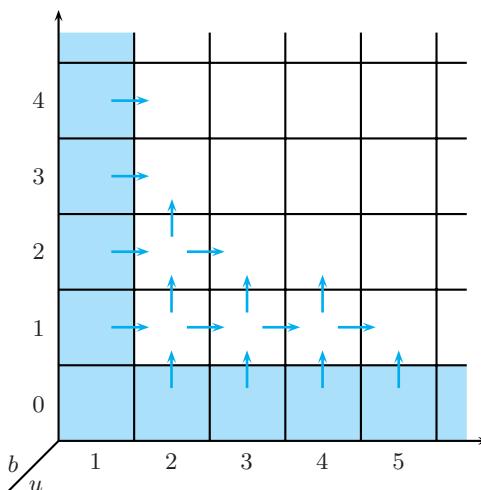
$$= \binom{b+u-2}{b-1} + \binom{b+u-2}{b}$$

$[P(u-1, b), P(u, b-1)]$

$$= \binom{b+u-1}{b}, \quad [\text{Binomial recurrence (Thm. 2, p. 349)}]$$

so  $P(u, b)$  is true.

Is this legit? This **double induction** involves two variables that are both changing, and we didn't treat this case in Chapter 2. Yes, it is legitimate — once we verify an appropriate basis. Look at Figure 5.5. The square in column  $u$  and row  $b$  represents  $P(u, b)$ . The three-line displayed equation above shows that  $P(u, b)$  is true if both  $P(u, b-1)$ , corresponding to the square below the  $(u, b)$  square, and  $P(u-1, b)$ , corresponding to the square to the left of the  $(u, b)$  square, are true. So suppose we can show as a basis that both all  $P(u, 0)$ , corresponding to the bottom of the infinite array, and all  $P(1, b)$ , corresponding to the left side, are true. These are shaded in Figure 5.5. The arrows illustrate why all  $P(i, j)$  are true.  $P(2, 1)$  is true because  $P(2, 0)$  and  $P(1, 1)$  are in the basis. Then  $P(3, 1)$  is true because we've just shown  $P(2, 1)$  to be true and  $P(3, 0)$  is in the basis. Continuing this way (a single-variable induction!), we see that the whole row  $P(m, 1)$  is true. Now we apply the same reasoning to see that the whole row  $P(m, 2)$  is true, and then row  $P(m, 3)$ , and so on (another single-variable induction, this time on the second variable). We say that this proof reduces double induction to **iterated induction**.



**FIGURE 5.5**

A valid double induction principle.

So, to finish this example, we need to verify the basis: is  $P(u, 0)$  true for all  $u \geq 1$  and is  $P(1, b)$  true for all  $b \geq 0$ ? We need to show that the right-hand side of Eq. (9) equals 1 in all these cases, since we showed in Section 5.2 that all the border entries of Figure 5.4 are 1's. Well,  $P(u, 0)$  claims that  $\binom{u-1}{0} = 1$ , and that's always true;  $P(1, b)$  claims that  $\binom{b}{1} = 1$ , and that's always true. ■

## EXAMPLE 6

Analyze the Fibonacci numbers from Example 9, Section 5.2, further.

**Solution** It's practically impossible to guess a formula for the Fibonacci numbers. But since we can compute them so easily using the associated difference equation,  $F_n = F_{n-1} + F_{n-2}$ , we can analyze them numerically as far as we wish. It is a semimystical principle in mathematics that it is always revealing to look at rates of growth. To do that, we look at ratios, as in Examples 1 and 3 and [26] in Section 2.6. Table 5.1 shows the first ten ratios  $F_{n+1}/F_n$ . Clearly the ratio seems to approach a limit. Therefore we conclude that the rabbit population grows quite regularly after the first few months at about 62% a month. This knowledge, so easy to comprehend, may be more useful than a closed-form expression for the exact number.

---

TABLE 5.1  
Fibonacci Numbers and  
their Ratios

---

| $n$ | $F_n$ | $F_{n+1}/F_n$ |
|-----|-------|---------------|
| 0   | 1     | 1             |
| 1   | 1     | 2             |
| 2   | 2     | 1.5           |
| 3   | 3     | 1.667         |
| 4   | 5     | 1.6           |
| 5   | 8     | 1.625         |
| 6   | 13    | 1.615         |
| 7   | 21    | 1.619         |
| 8   | 34    | 1.618         |
| 9   | 55    | 1.618         |

---

Have you seen this ratio before? The ancient Greeks called it the **Golden Ratio** [20] more than 1500 years before Fibonacci. ■

## Problems: Section 5.3

---

1. {2} Consider the difference equation

$$a_n = a_{n-1} - a_{n-2} \quad \text{for } n > 1,$$
$$a_0 = a_1 = 1.$$

Compute several terms. Do you see a pattern?  
Can you prove it?

2. {2} Consider the same recurrence as in [1], but

now let  $a_0$  and  $a_1$  be arbitrary. What happens? Can you prove it?

3.  $\langle 3 \rangle$  Consider the difference equation  $a_{n+2} = 5a_{n+1} - 6a_n$ . Compute the first 6 terms if:
- a)  $a_0 = 1, a_1 = 2$
  - b)  $a_0 = 5, a_1 = 10$
  - c)  $a_0 = 1, a_1 = 3$
  - d)  $a_0 = 2, a_1 = 5$
  - e)  $a_0 = 0, a_1 = 1$ .

Do you see a pattern? Can you prove it?

4.  $\langle 3 \rangle$  In [4, Section 5.2], you were asked to find a formula relating  $E_{2n}$  and  $E_n$  for Algorithm SEQ-SEARCH. Use that formula and the condition  $E_1 = 1$  to find and prove an explicit formula for  $E_n$  when  $n$  is a power of 2. (This may seem silly, since we already found a formula for  $E_n$  for *any*  $n$ , but it provides more practice, and it previews a “powers of 2” approach that is very important in Section 5.8.)
5.  $\langle 3 \rangle$  Use Eq. (2), Section 5.2, to find a recurrence relating  $E_{n+2}$  to  $E_n$ . Use this recurrence to find and prove an explicit formula for  $E_{2n}$ .
6.  $\langle 2 \rangle$  Suppose that you remembered Eq. (3) to be the negative of what it really is. Try doing the inductive proof using Eq. (2). Things won’t cancel right. Do you see how this might suggest to you where your error is?

7.  $\langle 1 \rangle$  Using the IRA analysis of Example 3 bound and then compute  $\hat{V}/V$  for
- a)  $r = .05, t = .25, N = 5$
  - b)  $r = .07, t = .25, N = 30$
  - c)  $r = .05, t = .3, N = 30$ .

Notice we have changed just one of the three parameters  $r, t, N$  at a time from the example in the text. From this you can make general conjectures about, say, the effect on the advantage of IRAs when  $t$  alone changes.

8.  $\langle 3 \rangle$  You may withdraw money from a traditional IRA early, but under most circumstances you must then pay a penalty of 10% of all the interest withdrawn, in addition to the regular taxes upon withdrawal from your IRA. A regular interest-bearing account can be withdrawn without penalty at any time. For this reason people are wary of putting money into an IRA if they think they may need it for an emergency before retirement.

Assume as in the text that you cannot defer tax on the principal. With  $r = .05$  and  $t = .25$ , what is the first year  $N$  for which it is advantageous

to withdraw money from a traditional IRA even with a penalty? (*Note:* When traditional IRAs were created, interest rates were much higher so  $N$  would have been much lower.)

9.  $\langle 3 \rangle$  Using the IRA analysis of Example 3 verify that  $\hat{V}/V = 1$  for  $N = 1$  and  $\hat{V}/V > 1$  for  $N > 1$ . *Hint:* Expand Eq. (5) by the Binomial Theorem.
10.  $\langle 2 \rangle$  Redo the analysis of Example 3 if you are permitted to defer tax on your IRA principal until withdrawal. (This builds on [19, Section 5.2]; the additional work is actually slightly simpler than what we had to do in the text.) Include numerical evaluations for the same values of  $r, t$ , and  $N$  considered in the text.
11.  $\langle 2 \rangle$  In a Roth IRA, you may never defer tax on the principal, but the entire withdrawal is tax free. Redo Example 3 for a Roth IRA. (*Note:* Not everyone is eligible for a Roth IRA.)
12.  $\langle 3 \rangle$  Consider an IRA to which you add  $P$  each year, but otherwise the hypotheses of Example 3 are unchanged. (This builds on [21, Section 5.2].) You are not yet prepared to find closed forms for  $V$  and  $\hat{V}$  in this model, but there is no reason why you can’t compute them numerically. Use  $r = .05$ ,  $t = .25$ , and  $N = 30$  and  $35$  as in Example 3. Do you expect  $\hat{V}/V$  to be more or less than in the one-time deposit model of the text?
13.  $\langle 1 \rangle$  Show that there is a unique equilibrium point for Trusty Rent-A-Car for the post-earthquake conditions described in [9, Section 5.2].
14.  $\langle 1 \rangle$  Consider underdeveloped Country X from [10, Section 5.2]. If migration continues the same way and equilibrium is reached, what fraction of the population will live in the cities?
15.  $\langle 2 \rangle$  Here is a double induction principle different from the one used in Example 5. To prove  $P(m, n)$  for all natural numbers  $m \geq m_0$  and  $n \geq n_0$ , it suffices to prove
- i) Basis:  $P(m, n_0)$  for all  $m \geq m_0$  and  $P(m_0, n)$  for all  $n \geq n_0$ ;
  - ii) Inductive Step: For all  $n > n_0$  and  $m > m_0$ ,  $[P(m-1, n) \text{ and } P(m-1, n-1)] \implies P(m, n)$ .

Justify this principle by a figure and an argument similar to that used for the double induction principle in the text.

16.  $\langle 2 \rangle$  State the double induction principle from Example 5 in a general way. See [15].

17.  $\langle 2 \rangle$  Figure 5.6 displays values of a function  $q = f(m, n)$  defined for pairs of positive integers with  $n \leq m$ . Figure out a formula for  $f$ .

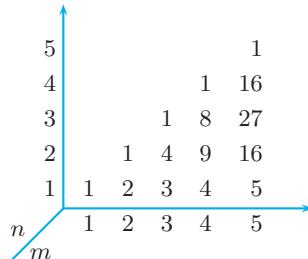


FIGURE 5.6

18.  $\langle 2 \rangle$  Set up a value and ratio table like Table 5.1 for the following modifications of the Fibonacci numbers.

- a) The **Lucas numbers**  $\{L_n\}$ : these numbers satisfy the same recurrence as the Fibonaccis, but the initial conditions are  $L_0 = 1$  and  $L_1 = 3$ .  
b) The sequence in [12, Section 5.2].  
c) The sequence in [13, Section 5.2]. (Start with one pair of rabbits.)

19.  $\langle 3 \rangle$  Let  $r_n = F_{n+1}/F_n$ , the ratio of consecutive Fibonacci numbers.

- a) Find a difference equation relating  $r_n$  and  $r_{n-1}$ . Use it to check the ratios in Table 5.1 without actually computing any Fibonacci numbers. You also need the initial condition  $r_0 = 1$ .  
b) If  $r_n$  really does go to a limit, call it  $r$ , then  $r_{n-1}$  goes to  $r$  as well. Thus rewrite the recurrence in a) as an equation in  $r$  and solve. You should get two roots, but one of them could not possibly arise from the Fibonacci numbers.

Why? Does the remaining root agree with the data in Table 5.1?

20.  $\langle 3 \rangle$  According to the ancient Greeks, the most pleasing rectangle is one where the ratio of length to width is the same as the ratio of the sum of length and width to length alone. The Greeks knew that all rectangles with this property have the same ratio of length to width, so they called it the Golden Ratio. Find a formula for this ratio.

21.  $\langle 2 \rangle$  Let  $G_n = \sum_{i=0}^n F_i$ , where  $F_i$  is again the  $i$ th Fibonacci number. Compute the first several terms of  $\{G_n\}$ . Find a pattern. Prove it.

22.  $\langle 3 \rangle$  Consumer loans continued. In [27, Section 5.2] you were asked to develop the equation  $P_{n+1} = P_n(1+r) - p$  for the amount of principal owed at the end of period  $n+1$ . Payment  $p$  is initially an unknown; you know how much you want to borrow,  $P = P_0$ ; you know the period interest rate,  $r$ , the lender will charge; and you know the number of periods,  $N$ , after which the loan must be completely repaid. Compute  $P_1$ ,  $P_2$ , and  $P_3$  in terms of  $P_0$ ,  $p$ , and  $r$  by repeatedly substituting into the difference equation. At this point you should see a pattern. Now express  $P_N$  in terms of  $P_0$ ,  $p$  and  $r$  and solve for  $p$ .

23.  $\langle 3 \rangle$  Consider the difference equation

$$A_0 = 0$$

and for  $n > 0$  and  $c$  constant,

$$A_n = A_{n-1} + \left(\frac{1}{n+1}\right)^2 (c - A_{n-1}).$$

Find and prove a formula for  $A_n$ .

24.  $\langle 2 \rangle$  Consider the difference equation

$$a_{n+1} = \frac{1}{2} \left[ a_n + \frac{4}{a_n} \right], \quad n \geq 0.$$

Compute the first ten terms (use a calculator) for various  $a_0 > 0$  and then various  $a_0 < 0$ . What happens?

## 5.4 Terminology and a Fundamental Theorem

Before we can give general methods for solving classes of difference equations, we need some words to describe the classes we are talking about. The key phrases we need to define are  $k$ th order, linear, constant coefficient, and homogeneous.

As noted in Section 5.1, a difference equation relates terms in a sequence. Different terms have different indices (subscripts). The **order** of a difference equation is the difference between the highest and lowest indices in the equation. Consider

- (A)  $a_{n+1} = \frac{1}{2}\left(a_n + \frac{4}{a_n}\right)$
- (B)  $a_n = a_{n-1}a_{n-2}$ ,
- (C)  $a_n = 4a_{n-7} - a_{n-10}$ ,
- (D)  $w_n = 1 + w_{\lfloor n/2 \rfloor}$ .

Equation (A) is first order because  $(n+1) - n = 1$ . Equation (B) is second order. Another way to say this is that (B) relates terms that are at most two apart in the sequence. Equation (C) is tenth order. Finally, (D) has no fixed order because the difference between the highest and lowest indices is  $n - \lfloor n/2 \rfloor = \lceil n/2 \rceil$ , which changes as you move along the sequence.

A difference equation is **linear** if it expresses a term in the sequence as a sum of multiples of previous terms of the sequence plus, optionally, a **nonhomogeneous part** that does not involve terms in the sequence. When a difference equation has a nonhomogeneous part, then the whole equation is said to be **nonhomogeneous**. Any difference equation like (C) without a nonhomogeneous part is said to be **homogeneous**.

The following are linear:

- (E)  $a_{n+1} = 3a_n + na_{n-1} + 2^n$ ,
- (F)  $S_n = S_{n-1} + n^3 - 2n^2 - 3n + 2$ ,
- (G)  $P_{n+1} = P_n(1+r) - p$ .

For instance, (E) expresses  $a_{n+1}$  as a multiple of  $a_n$  (the multiple is 3) plus a multiple of  $a_{n-1}$  (the multiple is  $n$ ) plus the nonhomogeneous part  $2^n$ , which does not involve any terms of the sequence (no  $a$ 's). Notice that the multipliers and the nonhomogeneous part need not be constants; they may involve the index variable  $n$ . In (F) the nonhomogeneous part is the cubic. In (G) the multiplier of  $P_n$  is  $1+r$ . Since the only variable is  $n$ ,  $1+r$  is a constant.

Equation (A) is nonlinear because  $(a_n)^{-1}$  appears. (B) is nonlinear because  $a_{n-1}$  and  $a_{n-2}$  are multiplied together. (Similarly,  $z = xy$  is not a linear equation in algebra because  $x$  and  $y$  are multiplied.) Continuing, (C) and (D) above are linear. In short, in a linear difference equation each  $a_k$  appears to the first power and is not multiplied by any  $a_j$ . For instance, a linear difference equation cannot involve  $2^{a_n}$ ,  $a_k^3$  or  $\ln a_{n-1}$ .

A linear difference equation has **constant coefficients** if all the multipliers of sequence terms are constants. Thus, (C), (D), (F) and (G) have constant coefficients. Equation (E) does not. Since (D)–(G) have nonhomogeneous parts, let us note that whether a nonhomogeneous part is constant or not is irrelevant in the definition of constant coefficients, because the nonhomogeneous part is not a coefficient. Thus in (F) the nonhomogeneous part  $(n^3 - 2n^2 - 3n + 2)$  is not constant,

but (F) does have constant coefficients — the only coefficient is the implicit 1 in front of  $S_{n-1}$ .

To summarize, a  $k$ th order linear difference equation is of the form  $a_n = (\sum_{i=1}^k c_{i,n} a_{n-i}) + d_n$ , where, as indicated by the notation, the coefficients  $c_{i,n}$  and the nonhomogeneous part  $d_n$  (if present) may depend on  $n$ . When they don't, we write just  $c_i$  and  $d$ .

The most common type of difference equation in the rest of this chapter will be a constant coefficient linear difference equation, or CCLDE for short. If a CCLDE is homogeneous, it is a CCHLDE.

## Existence and Uniqueness

Before we look for solution methods for various classes of difference equations, it's good to know in advance that there *are* solutions and to know to what extent they are unique.

Initial conditions are intended to specify a unique solution. For instance, many sequences satisfy the recurrence

$$h_{n+1} = 2h_n + 1, \tag{1}$$

(namely, all those of the form  $A2^n - 1$  with  $A$  a constant), but one and only one sequence also satisfies  $h_1 = 1$  (the TOH sequence). If we are given only the recurrence, we speak of finding the **general solution**, that is, a description of all sequences satisfying the recurrence. If we are also given initial conditions which specify a unique sequence, we talk about *the* solution for the initial conditions, or perhaps the specific solution.

Often, the general solution can be described in a simple way. For instance, it turns out that, as noted above, all solutions to Eq. (1) have the form  $h_n = A2^n - 1$ , for  $A$  an arbitrary constant. This means that for each value of  $A$ , the sequence  $\{A2^n - 1\}$  is a solution, and each solution is a sequence of this form for some  $A$ .

The following is a quite general theorem about when a recurrence and (one or more) initial conditions define a unique solution. Loosely stated, it says that a  $k$ th-order recurrence and  $k$  consecutive initial conditions uniquely determine a sequence.

**Theorem 1.** Suppose  $k$  is a positive integer and  $f(x_1, x_2, \dots, x_k, n)$  is a real-valued function defined for all real numbers  $x_1, x_2, \dots, x_k$  and all integers  $n \geq k$ . Then for any  $k$  real numbers  $c_0, c_1, \dots, c_{k-1}$ , there exists one and only one sequence  $a_0, a_1, a_2, \dots$  satisfying

$$a_n = f(a_{n-1}, a_{n-2}, \dots, a_{n-k}, n), \quad \text{for all } n \geq k \tag{2}$$

and

$$a_0 = c_0, a_1 = c_1, \dots, a_{k-1} = c_{k-1}. \tag{3}$$

This theorem is completely parallel to the Principle of Mathematical Induction. Induction says: Once the initial case or cases are checked, the inductive step settles the claim for all remaining cases. This theorem says: Once the initial terms are specified as in Eqs. (3), then Eq. (2) uniquely specifies all the remaining terms.

**PROOF** Intuitively this theorem is clear enough: the terms  $a_m$  are determined one at a time, first from (3), then from (2), and each time there is one and only one choice given the previous choices. However, we are dealing with an infinite set of unknowns (all the  $a$ 's) that solve an infinite set of equations (displays (2) and (3)), and just as there are subtleties going from finite sums to infinite sums, so are there subtleties going from finite systems of equations to infinite systems. So we will give more detail.

Note that Eqs. (2) and (3) include, for each  $m \geq 0$ , exactly one equation with  $a_m$  on the left-hand side. We will refer to this as equation  $m$ . (Note, therefore, that the initial equation is equation 0. Best not to call it the first equation, since that might just as well mean equation 1.)

Now, what does it mean that  $a_0, a_1, \dots$  satisfy an infinite system of equations? It means that they satisfy each of the equations. Thus, for each  $m$ , we must show that  $a_0, a_1, \dots$  satisfy equation  $m$ . This is true because we always set  $a_m$  to the value that makes equation  $m$  true. Thus we have proved that there is at least one sequence that solves all the equations. How do we prove that the sequence is unique? We reduce to the finite case. For each  $m$  we show that  $a_m$  is already uniquely determined by a finite subset of the equations. Therefore, when we consider all the equations (infinitely many), this value of  $a_m$  is the only value that might still work.

This uniqueness argument is by induction. Let  $Q(m)$  be the claim that the  $m+1$  values  $a_0, a_1, \dots, a_m$  are uniquely determined by the  $m+1$  equations 0 through  $m$ . The basis case,  $Q(k-1)$ , is true by display (3). For the inductive step, we prove  $Q(m-1) \implies Q(m)$ , for  $m \geq k$ . By  $Q(m-1)$ , equations 0 through  $m-1$  have uniquely determined  $a_0, \dots, a_{m-1}$ , and now we also consider equation  $m$ ,  $a_m = f(a_{m-1}, a_{m-2}, \dots, a_{m-k}, m)$ . Since the inputs to  $f$  are uniquely determined (and in the domain of  $f$ ), and since  $f$  is a function (and so has a unique output), we conclude that  $a_m$  is uniquely determined as well when we consider this equation along with the others. This completes the inductive step. ■

Why does the theorem fuss about  $f$  having *all* real numbers  $x_1, x_2, \dots, x_k$  in its domain? Because otherwise at some point in the iteration we might have values  $a_{m-1}, \dots, a_{m-k}$  for which  $f$  is not defined [13]. Why is Eq. (2) given in a form with  $a_n$  by itself on one side? Because otherwise there may be *many* sequences satisfying these equations [14]. Fortunately, the assumptions about  $f$  in Theorem 1 do obtain in most cases of interest. For instance, for constant-coefficient homogeneous linear difference equations we have  $f = \sum_{i=0}^k c_i x_i$  with each  $c_i$  constant, so  $f$  really is defined for all real numbers  $x_1, \dots, x_k$ .

Theorem 1 is an *existence and uniqueness* theorem. It says there is a sequence (existence) but only one (uniqueness). The existence part will serve us mostly as a form of inspiration; since we know there must be a solution, there might be a closed form and it's worth looking for. The uniqueness part will serve us more directly. It

tells us this:

**Key Observation.** If *you* have a sequence that satisfies Eqs. (2) and (3), and *we* find a formula for some sequence that satisfies (2) and (3), then we have found a formula for your exact sequence.

The following example shows a typical use of this Observation.

## EXAMPLE 1

Use the Key Observation to prove the geometric series formula:

$$\sum_{k=0}^n ar^k = \frac{a(r^{n+1} - 1)}{r - 1}, \quad r \neq 1. \quad (4)$$

**Solution** We must cast this problem in the form where you know a difference equation for a sequence and we know a formula that satisfies the difference equation. If you denote the left-hand side of (4) by  $T_n$ , then as noted in Example 2, Section 5.2,

$$T_n = T_{n-1} + ar^n \quad \text{for } n > 0, \quad (2')$$

$$T_0 = ar^0 = a. \quad (3')$$

Thus this difference equation is in the form of Theorem 1 with  $k = 1$ ,  $a_n = T_n$ ,  $f(a_{n-1}, n) = a_{n-1} + ar^n$ , and  $a_0 = a$ . Therefore, to prove Eq. (4) it suffices to check that the formula  $T_n = a(r^{n+1}-1)/(r-1)$ , also satisfies (2') and (3'). This is easy: (3') becomes

$$\frac{a(r^{0+1} - 1)}{r - 1} = a. \quad \checkmark$$

(2') becomes:

$$\begin{aligned} \frac{a(r^{n+1} - 1)}{r - 1} &\stackrel{?}{=} \frac{a(r^n - 1)}{r - 1} + ar^n \\ &\stackrel{?}{=} \frac{ar^n - a}{r - 1} + \frac{ar^{n+1} - ar^n}{r - 1} \\ &\stackrel{\checkmark}{=} \frac{a(r^{n+1} - 1)}{r - 1}. \quad \blacksquare \end{aligned}$$

It is instructive to compare this proof of Eq. (4) with the previous one in Example 2, Section 5.3. The algebra is about the same, but it serves a different purpose. There we proved by induction that, for each  $n$ ,

$$T_n = \frac{a(r^{n+1} - 1)}{r - 1}.$$

Here we prove that the two sequences

$$\{T_n\}, \quad \left\{ \frac{a(r^{n+1} - 1)}{r - 1} \right\}$$

satisfy the same recurrence and initial condition.

Theorem 1 has become even more fundamental in the last 20 years, for it has become the foundation for major research breakthroughs in automated theorem proving about formulas for sequences. See Section 5.9.

## Problems: Section 5.4

For each difference equation in [1–10], state its order and whether it is linear. If it is linear, state if it is homogeneous; if it has constant coefficients.

1.  $\langle 1 \rangle a_n = 2/a_{n-1}.$

2.  $\langle 1 \rangle w_n = \max\{w_{n-1}, w_{n-3}\}.$

3.  $\langle 1 \rangle h_{n+1} = 2h_n + 1.$

4.  $\langle 1 \rangle a_{n-2} = 2^n a_{n-6} + (-3)^n.$

5.  $\langle 1 \rangle F_n = F_{n-1} + F_{n-2}.$

6.  $\langle 1 \rangle P_{k+1} = P_k(1+r) - P_k r t.$

7.  $\langle 1 \rangle d_n = (n-1)(d_{n-1} + d_{n-2}).$

8.  $\langle 1 \rangle E_n = \frac{3}{n} + \frac{n-2}{n}(2 + E_{n-2}).$

9.  $\langle 1 \rangle p_{n+1} = \sum_{k=0}^n p_k.$

10.  $\langle 1 \rangle t_n = \sum_{k=1}^{n-1} t_k t_{n-k}.$

11.  $\langle 2 \rangle$  The text defined classifications only for difference equations for a single sequence involving a single index variable. Nonetheless, we think you can figure out reasonable classifications for the following. Do so.

a) 
$$\begin{cases} N_n = \frac{1}{2}N_{n-1} + \frac{1}{3}L_{n-1} \\ L_n = \frac{1}{2}N_{n-1} + \frac{2}{3}L_{n-1} \end{cases}$$

b)  $C^*(u, b) = C^*(u-1, b) + C^*(u, b-1)$

12.  $\langle 2 \rangle$  Show that every second-order, nonhomogeneous, constant-coefficient difference equation with geometric nonhomogeneous part meets the assumptions of Theorem 1. (That is, you have to show that  $f$  is defined wherever it is supposed to be.) A geometric nonhomogeneous part is an expression of the form  $ar^n$  for some nonzero constants  $a$  and  $r$ .

13.  $\langle 1 \rangle$  Consider the difference equation

$$r_n = \frac{r_{n-2}}{r_{n-1} - 2}, \quad n > 1.$$

Suppose that we set initial conditions  $r_0 = 1$ ,  $r_1 = 2$ . Show that there is no infinite sequence satisfying these conditions — because one can't even define  $r_2$ . Why isn't Theorem 1 violated?

14.  $\langle 2 \rangle$  Consider the difference equation

$$a_n^2 = a_{n-1}^2 \quad \text{for } n > 1, \quad a_1 = 1.$$

Find all infinite sequences that satisfy this difference equation. Why isn't Theorem 1 violated?

15.  $\langle 2 \rangle$  Consider the difference equation

$$a_n = 2a_{n-2} \quad \text{for } n > 2, \quad a_1 = 2, a_3 = 3.$$

Show that no sequence satisfies these conditions. Why isn't Theorem 1 violated?

16.  $\langle 2 \rangle$  Theorem 1 shows that, in general,  $k$  consecutive terms define the rest of a  $k$ th-order difference equation uniquely. Show that fewer than  $k$  consecutive terms don't. Hint: Fewer than  $k$  terms are given, so you can make up the  $k$ th term.

17.  $\langle 2 \rangle$  Use Theorem 1 to prove the formula for the sum of an arithmetic series:

$$\sum_{k=1}^n a + (k-1)d = an + \frac{n(n-1)}{2}d.$$

18.  $\langle 3 \rangle$  Let  $d_n$  be the number of ways to cover a strip of length  $n$  and width 1 by  $1 \times 1$  squares and  $2 \times 1$  dominoes. For instance,  $d_3=3$  because, working from left to right, you can either use 3 squares, or a domino with a square on its right, or a square with a domino on its right. Prove by a combinatorial argument that

a)  $d_{m+n} = d_m d_n + d_{m-1} d_{n-1}.$  Hint: Either a piece ends at  $m$  units from the beginning of the strip or we are in the middle of a domino.

b)  $\sum_{k=0}^n d_k = d_{n+2} - 1.$  Hint: Let  $k$  be the position of the left end of the right-most domino.

c)  $d_n = d_{n-1} + d_{n-2}$  and  $d_1 = 1, d_2 = 2.$  Therefore,  $\{d_n\}$  is what famous sequence? Note that in parts a) and b) you have therefore proved some identities for this sequence which might have been hard to prove directly.

19.  $\langle 3 \rangle$  We've proved the formula for a geometric series twice, in Example 2, Section 5.3, and in Example 1 in this section. The first proof used induction, the second didn't. Or did it?

## 5.5 Constant Coefficient Homogeneous Linear Difference Equations

In this section, “difference equation” (or recurrence) means constant coefficient homogeneous linear difference equation (CCHLDE). We will show how to solve all such difference equations. That is, for each such recurrence, we will show how to find all sequences which satisfy it. We develop the solution method through examples and then summarize it as Theorem 2. Almost all our examples will involve second-order equations; such problems exhibit all the issues that arise in solving the general case.

In a final subsection, we briefly introduce an alternative solution method using generating functions.

To get an idea of what the solutions should look like, it is best to start with the simplest case, the general first-order CCHLDE,

$$a_n = ra_{n-1}. \quad (1)$$

We know the general solution is  $a_n = ar^n$ , where  $a = a_0$ . (See Example 1, Section 5.2, and also [3].) Eq. (1) is not the only difference equation we have met thus far with geometric growth. We saw that the Fibonacci numbers, which satisfy a second-order equation, grow with a ratio closer and closer to geometric as  $n$  increases. Also, the difference equation in [3, Section 5.3] was second order, but some of the solutions were exactly geometric. Let’s guess, therefore, that *every* difference equation (linear, homogeneous, constant coefficient) has a geometric solution. If this turns out to be correct, then we’ll worry about combining geometric solutions to get other solutions.

### EXAMPLE 1

Find all *geometric* solutions to

$$a_n = a_{n-1} + 2a_{n-2}, \quad n > 1. \quad (2)$$

This difference equation arose in [12, Section 5.2], a problem about rabbits.

**Solution** If  $a_n = r^n$  is a solution, then substituting in (2) we get

$$r^n = r^{n-1} + 2r^{n-2}, \quad n > 1.$$

We need to find all values  $r$  for which this is true. One immediate solution is  $r = 0$ , but this is the trivial solution — all terms 0. Excluding this as uninteresting, we can divide out the common factor  $r^{n-2}$  to obtain

$$r^2 = r + 2 \quad \text{or} \quad r^2 - r - 2 = (r - 2)(r + 1) = 0.$$

This polynomial equation (either form) is called the **characteristic equation** of the difference equation and the left-hand side of the second form is called the **characteristic polynomial**. Its solutions are  $r = -1, 2$ , so  $a_n = (-1)^n$  and  $a_n = 2^n$  are both solutions to Eq. (2). Indeed, since every step of this algebraic derivation reverses, we have that the sequence  $\{r^n\}$  is a nontrivial geometric solution  
 $\iff r = -1, 2.$  ■

*Note.* We use “ $r$ ” in the characteristic equation because it suggests both “ratio” (of successive terms of the geometric solution sequence) and “root” (of the characteristic equation).

## EXAMPLE 2

Find all geometric solutions to the Fibonacci recurrence.

**Solution** Proceeding as before, from  $F_n = F_{n-1} + F_{n-2}$  we obtain

$$\begin{aligned} r^n &= r^{n-1} + r^{n-2}, \\ r^2 &= r + 1, \\ r^2 - r - 1 &= 0. \end{aligned}$$

This characteristic equation doesn’t factor in an obvious way, but the quadratic formula gives

$$r = \frac{1 \pm \sqrt{5}}{2} = 1.618\dots \text{ and } -.618\dots$$

So there are two geometric solutions  $\{r^n\}$ , one for each value of  $r$  just displayed. ■

In both Example 1 and Example 2 we obtained two geometric solutions. However, in neither example is either solution the one we looked at in Section 5.2. In [12, Section 5.2], corresponding to Example 1, the solution was a sum involving both  $(-1)^n$  and  $2^n$ . As for the Fibonacci numbers, corresponding to Example 2, 1.618 seems to be just the ratio we observed at the end of Section 5.3. Moreover,  $(-.618)^n$  gets closer and closer to zero. Could  $F_n$  just be a multiple of  $1.618^n$  perturbed by a multiple of  $(-.618)^n$ ? The answer is Yes. This is a special case of a general result, so we state it in the general case ( $k$ th order).

**Theorem 1.** If  $\{b_n\}$  and  $\{b'_n\}$  are both solutions to the difference equation

$$a_n = \sum_{i=1}^k c_i a_{n-i} \tag{3}$$

for given constants  $c_1, c_2, \dots, c_k$  and all  $n \geq k$ , then  $\{Bb_n + B'b'_n\}$  is also a solution for arbitrary constants  $B$  and  $B'$ .

The sequence  $\{Bb_n + B'b'_n\}$  is called a **linear combination** of the sequences  $\{b_n\}$  and  $\{b'_n\}$ . Theorem 1 is summarized by saying that the set of solutions to Eq. (3) is **closed** under linear combinations.

**PROOF** To say that  $\{b_n\}$  is a solution is to say

$$b_n = \sum_{i=1}^k c_i b_{n-i}$$

for all  $n \geq k$ . To say that  $\{b'_n\}$  is a solution is to say

$$b'_n = \sum_{i=1}^k c_i b'_{n-i}$$

for the same  $n$ . Multiplying the last equation by  $B'$ , the previous by  $B$ , and adding, gives

$$\begin{aligned} Bb_n + B'b'_n &= \sum_{i=1}^k Bc_i b_{n-i} + \sum_{i=1}^k B'c_i b'_{n-i} \\ &= \sum_{i=1}^k c_i (Bb_{n-i} + B'b'_{n-i}). \end{aligned} \quad [\text{Regrouping}]$$

But this last display says that  $\{Bb_n + B'b'_n\}$  satisfies Eq. (3). ■

This theorem is about linear combinations with two terms, because for the examples so far there are only two “starter” solutions to add. However, there will soon be more. You should be able to generalize Theorem 1 to linear combinations of an arbitrary number of solutions [30].

So we now know how to parlay any number of geometric solutions into an infinite set of solutions. But is this set large enough to include the specific solution for any given initial conditions? To get at the answer, let us extend the previous examples.

### EXAMPLE 3

In Example 1, the two geometric solutions to Eq. (2) were  $(-1)^n$  and  $2^n$ . Is there a linear combination of these two solutions that also satisfies the initial conditions  $a_0 = a_1 = 3$ ? (These were the initial conditions of [12, Section 5.2].)

**Solution** We are asking if there are specific constants  $A$  and  $B$  for which

$$\begin{aligned} A(-1)^0 + B2^0 &= A + B = a_0 = 3, \\ A(-1)^1 + B2^1 &= -A + 2B = a_1 = 3. \end{aligned} \quad (4)$$

Solving these equations, one finds there is the single solution  $A = 1$ ,  $B = 2$ . So the answer is Yes, there is exactly one linear combination of  $(-1)^n$  and  $2^n$  that satisfies the initial conditions, namely  $a_n = (-1)^n + 2^{n+1}$ . ■

Notice that by Theorem 1, Section 5.4, we can say that this linear combination is in fact the *only* solution of *any form* to Eq. (2) and the initial conditions  $a_0 = a_1 = 3$ . Moreover, there is nothing special about these initial conditions. No matter what  $a_0$  and  $a_1$  are, the two equations in (4) can be solved uniquely for  $A$  and  $B$  (why?). Thus  $A(-1)^n + B2^n$  is the *general* solution to Eq. (2): everything of this form is a solution and there are no others. For this reason,  $\{(-1)^n\}$  and  $\{2^n\}$  are said to be a **basis** of the solutions to Eq. (2).

*Remark.* In the examples so far we have needed two steps that will occur over and over again in the rest of this chapter: finding the roots of a polynomial and solving a system of linear equations. We will rarely show the details of these steps. For polynomials, we assume you know how to solve quadratics. For higher degree

polynomials we expect you will use a calculator or software. (While there are exact formulas for the roots of cubics and quartics, they are almost never used because of their complexity. As well, there are methods that simplify the search for rational roots of all polynomials but we do not assume you know these.) For systems of linear equations, there are many methods, both for hand and for machine, and we assume you know at least one. We also assume you know that most linear systems have a unique solution, but that some systems have no solution and some have infinitely many solutions [22].

#### EXAMPLE 4

Find a closed-form formula for the  $n$ th Fibonacci number.

**Solution** From Example 2 and Theorem 1, every sequence of the form

$$F_n = A\left(\frac{1+\sqrt{5}}{2}\right)^n + B\left(\frac{1-\sqrt{5}}{2}\right)^n$$

is a solution to  $F_n = F_{n-1} + F_{n-2}$ . Since the initial conditions for the Fibonaccis are  $F_0 = F_1 = 1$ , it is sufficient to find  $A$  and  $B$  that satisfy

$$\begin{aligned} A\left(\frac{1+\sqrt{5}}{2}\right)^0 + B\left(\frac{1-\sqrt{5}}{2}\right)^0 &= A + B = F_0 = 1, \\ A\left(\frac{1+\sqrt{5}}{2}\right)^1 + B\left(\frac{1-\sqrt{5}}{2}\right)^1 &= A\left(\frac{1+\sqrt{5}}{2}\right) + B\left(\frac{1-\sqrt{5}}{2}\right) \quad (5) \\ &= F_1 = 1. \end{aligned}$$

After considerable algebra (see Note 1 below), one finds that

$$A = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right), \quad B = -\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right). \quad (6)$$

Therefore

$$F_n = \frac{1}{\sqrt{5}} \left[ \left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1} \right]. \quad (7)$$

*Note 1.* These days, one usually solves ugly equations like (5) with a CAS. However, the solutions you get may not look like (6); we have left a radical in the denominator, which is not standard form. However, this form for  $A$  and  $B$  is most pleasing (or at least traditional) for use in Eq. (7). This “disagreement” illustrates an important limitation of CAs; they don’t know what form of output will be best for your particular application. You must learn to be the master, guiding your CAS to the form you want. Of course, whatever form you got with your CAS, your answer is correct too, unless you hit some wrong keys. Here’s a general procedure for finding out if your answer agrees with ours. If  $E$  is your expression for some quantity (say,  $A$  above), and  $E'$  is our expression, type in  $E - E'$  and hit Simplify. If your expression is correct, you should get 0.

*Note 2.* Fibonacci never discovered Eq. (7); it was first obtained 650 years later! It’s certainly not very guessable. In fact, you may find it hard to believe that

the right-hand side is always an integer, but it must be since all Fibonacci numbers are integers. This matter is pursued further in [9–10, Supplement].

*Note 3.* If you use the other Fibonacci convention that  $F_1 = F_2 = 1$ , then the exponents in Eq. (7) are  $n$ , not  $n + 1$ . (See page 422.)

So far the examples in this section have had characteristic equations with distinct roots. A wrinkle arises when there is a multiple root.

## EXAMPLE 5

Find the general solution to

$$a_n = 4a_{n-1} - 4a_{n-2}, \quad (8)$$

and the specific solution when  $a_0 = 5$ ,  $a_1 = 4$ .

**Solution** Substituting  $r^n$  for  $a_n$  and simplifying, we obtain

$$\begin{aligned} r^n &= 4r^{n-1} - 4r^{n-2}, \\ r^2 - 4r + 4 &= (r-2)^2 = 0. \end{aligned}$$

There is a double root,  $r = 2$ , so  $2^n$  is the only geometric solution we get using the characteristic equation. However, we need two basic solutions if we are going to satisfy two initial conditions by taking a linear combination. What's the other?

It turns out the answer is  $n2^n$ . While this result is hard to discover, it can be verified with just a little algebra [17]. (More generally, if  $r_1$  is a double root, then  $nr_1^n$  is a basic solution [20].) Thus by Theorem 1 everything of the form  $A2^n + Bn2^n$  is a solution. To show that it is the general solution, we must show that for any initial conditions  $a_0$  and  $a_1$ , there are values for the constants  $A, B$  that work. Substituting the cases  $n = 0$  and  $n = 1$  in the putative general solution, we obtain

$$\begin{aligned} A &= a_0, \\ 2A + 2B &= a_1. \end{aligned}$$

Direct calculation shows that this pair of equations has a unique solution for  $A$  and  $B$ , whatever  $a_0$  and  $a_1$  are. For instance, when  $a_0 = 5$  and  $a_1 = 4$  as in this example,  $A = 5$  and  $B = -3$ . In other words, the particular solution is  $a_n = 5 \cdot 2^n - 3n2^n$ . ■

These examples illustrate the general pattern. A  $k$ th order difference equation corresponds to a polynomial equation of degree  $k$ . If the roots  $r_1, r_2, \dots, r_k$  are all distinct, then the general solution is  $a_n = \sum_{i=1}^k A_i r_i^n$ . If instead some  $r_i$  is a double root, then both  $r_i^n$  and  $nr_i^n$  appear as basic solutions in the linear combination. If  $r_i$  is a triple root, then  $r_i^n$ ,  $nr_i^n$  and  $n^2r_i^n$  are basic solutions. And so on. In any event, once you have the general solution, to find a specific solution, you substitute the general solution into the initial conditions and solve for the constant  $A_1, A_2, \dots, A_k$ .

For easy reference, let us state all this again carefully with a definition and a theorem.

**Definition 1.** Let  $c_1, c_2, \dots, c_k$  be given constants. The **characteristic equation** of the difference equation

$$a_n = \sum_{i=1}^k c_i a_{n-i}$$

is

$$r^k - \sum_{i=1}^k c_i r^{k-i} = 0.$$

This is the polynomial equation obtained by replacing each  $a_m$  in the difference equation with  $r^m$ , dividing out by the highest common power of  $r$ , and putting all terms on the left. The polynomial itself is called the **characteristic polynomial**.

---

For instance, the characteristic equation of

$$s_{n+1} = s_n + 3s_{n-2},$$

is

$$r^3 - r^2 - 3 = 0.$$

Note that in this instance we used  $s$  and  $n+1$  instead of the  $a$  and  $n$  that appear in the definition. The key thing is to replace subscripts by powers and to divide out by the highest common power.

---

**Theorem 2.** Suppose the characteristic equation of a  $k$ th-order linear, constant coefficient, homogeneous difference equation is

$$\prod_{i=1}^j (r - r_i)^{m_i} = 0.$$

That is, we suppose that the characteristic equation has distinct roots  $r_1, r_2, \dots, r_j$ , with multiplicities  $m_1, m_2, \dots, m_j$ , respectively (so that  $k = \sum_{i=1}^j m_i$ ). Then every solution  $\{a_n\}$  to the difference equation can be expressed uniquely as a linear combination of the following  $k$  solutions:

$$\begin{aligned} & \{r_1^n\}, \quad \{nr_1^n\}, \dots, \{n^{m_1-1}r_1^n\}, \\ & \{r_2^n\}, \dots, \{n^{m_2-1}r_2^n\}, \\ & \quad \quad \quad \cdot \quad \cdot \quad \cdot \\ & \{r_j^n\}, \dots, \{n^{m_j-1}r_j^n\}. \end{aligned}$$

Put another way, for each choice of  $k$  consecutive values  $a_0, a_1, \dots, a_{k-1}$ , there is a unique linear combination of these solutions which meets these  $k$  initial conditions and satisfies the difference equation.

---

This is a mouthful, and we have proved only part of it (as discussed below), so let's clarify the theorem with some examples.

First, what does it say in the case  $k = 2$ ? Every second degree characteristic polynomial factors as either  $(r-r_1)(r-r_2)$  or  $(r-r_1)^2$ . In the first case the Theorem claims that every solution is uniquely of the form  $a_n = Ar_1^n + Br_2^n$ , that is, every sequence of this form satisfies the difference equation and, for each choice of initial conditions for  $a_0$  and  $a_1$ , there is exactly one solution sequence of this form. In the second case the claim is that every solution is uniquely of the form  $a_n = Ar_1^n + Bnr_1^n$ . In the examples so far you have verified these claims for specific quadratics. In [20] we ask you to prove them for arbitrary  $r_1$  and  $r_2$ , thus proving the theorem for  $k = 2$ .

For a higher degree example of Theorem 2, suppose the characteristic polynomial of a fourth-order difference equation turns out to be  $(r+1)(r-5)^3$ . Then Theorem 2 claims that the general solution to the difference equation is

$$A(-1)^n + B5^n + Cn5^n + Dn^25^n.$$

If the initial conditions are  $a_0 = 1$ ,  $a_1 = 3$ ,  $a_2 = 4$ ,  $a_3 = -7$ , then there will be unique choices for  $A, B, C, D$  that work. How do you find them? By solving for  $A, B, C, D$  from

$$\begin{aligned} A(-1)^0 + B5^0 + C05^n + D0^25^n &= 1, \\ A(-1)^1 + B5^1 + C15^n + D1^25^n &= 3, \\ A(-1)^2 + B5^2 + C25^n + D2^25^n &= 4, \\ A(-1)^3 + B5^3 + C35^n + D3^25^n &= -7. \end{aligned} \tag{9}$$

These equations are ugly, but any calculator with matrix capabilities can solve them. Theorem 2 guarantees that the solution exists and is unique.

Actually, any difference equation problem this complicated should probably be done from the start with a CAS, in which case system (9) will be solved behind the scenes after you type in the original difference equation and the initial conditions.

We should be forthright about what's involved in a complete proof of Theorem 2. The fact that

$$r^n \text{ is a (nonzero) solution} \iff r \text{ is a root of the characteristic equation}$$

was shown in the solution to Example 1. (We showed it there for a particular second-order difference equation, but the method of proof was general.) The fact that linear combinations of solutions are solutions was proved in Theorem 1. (The theorem talks only about linear combinations of two solutions, but induction shows that the same result holds for any number of solutions [30].) Consequently, we *have* shown that everything of the form  $a_n = \sum_{i=1}^k A_i r_i^n$  is a solution. What we have left out is an explanation of where the additional basic solutions come from in the multiple root case, and a proof in either case that linear combinations of the basic solutions give us *all* solutions. As for filling the first gap, the proof in [20] for  $k = 2$  hints at the general method, as does [23], but, at the least, more sophisticated notation is needed to carry out a general proof. To fill the second gap one must

show that all systems of linear equations arising from Theorem 2, like (9), have unique solutions. Filling this gap requires substantial linear algebra theory.

*Note 1.* In all the examples we have given, the initial conditions were for the consecutive terms  $a_0, a_1, \dots, a_{k-1}$ , where again  $k$  is the order of the difference equation. Very commonly, they are given for  $a_1, a_2, \dots, a_k$  which is a distinction without a difference if we just set  $b_j = a_{j+1}$ . Thus for any *consecutive*  $k$  initial conditions, there is still a unique solution.

However, sometimes the initial conditions are different in a more substantial way. For instance, for a second-order difference equation you might be told that  $a_0 = 1$  and  $a_{100} = 0$ . (Such “initial conditions” are more properly called **boundary conditions**.) The method of solution is still the same: find the general solution using Theorem 2 and then solve for the coefficients using the initial conditions. However, for nonconsecutive initial conditions Theorem 1 of Section 5.4 does *not* guarantee that there is a unique solution. In some cases there will be infinitely many solutions, in some other cases none [22]. Also, if there are either more or fewer than  $k$  initial conditions, there may again be no or infinitely many solutions.

*Note 2.* Theorem 2 is not restricted to real numbers. Some examples with complex roots are treated in [44], in Section 5.6, and in [27–28, Supplement].

## Generating Functions

We now show how the generating functions introduced in Section 4.8 can provide an alternative treatment of CCHLDEs. Indeed, they can be used to solve many sorts of difference equations, including some not solvable by any of the methods of this chapter. (See [13, Ch. 4 Supplement] to learn how generating functions can handle convolution recurrences.) Consequently, there are whole books on generating functions.

Recall that every sequence can be turned into an algebraic object, its generating function. Putting together the pieces of a problem may be simpler in the generating function world, after which you return to the sequence world to read off your answer. In Chapter 4 we turned counting problems into sequences in order to use generating functions. But here in Chapter 5, solving difference equations, we have sequences right from the start.

### EXAMPLE 6

Use a generating function to solve

$$a_n - 3a_{n-1} + 2a_{n-2} = 0; \quad a_0 = 0, \quad a_1 = 1. \tag{10}$$

**Solution** The generating function for the sequence  $\{a_k\}$  defined by Eq. (10) is, by definition,

$$G(s) = \sum_{k=0}^{\infty} a_k s^k. \tag{11}$$

Now, look very carefully at how we line things up in the next display:

$$\begin{aligned}
G(s) &= a_0 + a_1 s + a_2 s^2 + a_3 s^3 + a_4 s^4 + \dots, \\
-3sG(s) &= -3a_0 s - 3a_1 s^2 - 3a_2 s^3 - 3a_3 s^4 - \dots, \\
2s^2 G(s) &= 2a_0 s^2 + 2a_1 s^3 + 2a_2 s^4 + \dots.
\end{aligned}$$

The second line above is  $-3s$  times  $G(s)$ , and  $-3$  is the coefficient in the second term of Eq. (10),  $-3a_{n-1}$ . The third line is  $2s^2$  times  $G(s)$ , and  $2$  is the coefficient in the third term of Eq. (10),  $2a_{n-2}$ . Also notice that the powers of  $s$  in the second and third lines, respectively,  $1$  and  $2$ , are how much is subtracted from  $n$  in the index of the second and third terms of (10). Finally, notice that the point of shifting the RHS of the latter two equations is so that all terms with the same power of  $s$  line up vertically.

What's the point of writing these three equations, and in just this way? Add them up! By Eq (10), for every power  $s^k$  on the right where there are three vertical terms, they add to  $0s^k = 0$ . Thus the sum is

$$(1 - 3s + 2s^2)G(s) = a_0 + a_1 s - 3a_0 s.$$

Furthermore, by the initial conditions in (10),

$$a_0 + a_1 s - 3a_0 s = s.$$

Solving for  $G(s)$  we therefore have

$$G(s) = \frac{s}{1 - 3s + 2s^2}. \quad (12)$$

We have now solved the problem in the generating function world, but to get back to the sequence world we must expand this closed form in powers of  $s$  so that we can see explicitly the form of  $a_k$ . The denominator of  $G(s)$  is a quadratic, which factors into  $(1 - s)(1 - 2s)$ . It turns out that rational functions always have a **partial fraction** decomposition using the factors of the denominator. For  $G(s)$  it is

$$\frac{s}{(1 - s)(1 - 2s)} = \frac{-1}{1 - s} + \frac{1}{1 - 2s}.$$

(Even if you have no idea how to discover this identity, you can easily verify it by putting the terms on the RHS over a common denominator.) Thus we have

$$G(s) = \frac{-1}{1 - s} + \frac{1}{1 - 2s}. \quad (13)$$

Then using the geometric series identity

$$\frac{1}{1 - s} = \sum_{k=0}^{\infty} s^k$$

and its analog with  $s$  replaced by  $2s$ , we may write (13) as

$$\begin{aligned}
G(s) &= -(1 + s + s^2 + s^3 + \dots) + [1 + 2s + (2s)^2 + (2s)^3 + \dots] \\
&= (1 - 1) + (2 - 1)s + (2^2 - 1)s^2 + \dots + (2^k - 1)s^k + \dots.
\end{aligned} \quad (14)$$

The coefficient of  $s^n$  in  $G(s)$  is just  $a_n$ , so from Eq. (14) it follows that

$$a_n = 2^n - 1. \quad \blacksquare$$

This result is the same as we would have gotten using the characteristic equation method — it has to be since we have proved that solutions are unique. And while this is only one example, in fact every CCHLDE can be handled by this method. Every CCHLDE corresponds to a rational generating function, and every rational generating function can be analyzed using partial fractions (with complications for multiple roots and hard-to-factor polynomials that closely parallel the complications for the characteristic equation method). For the details on partial fractions, see [29–34, Supplement].

We suspect that you find the characteristic equation method more congenial, and indeed, we think that's right for problems to be done by hand. But keep in mind:

- Generating functions reduce solving CCHLDEs to *standard* algebra, so a CAS can handle it easily.
- Generating functions can handle other sorts of difference equations.
- Sometimes you don't just want to know a formula for the typical term  $a_n$ , but rather you actually want a value for  $\sum a_k s^k$  for particular values of  $s$ . In this case, the generating function is precisely what you want; you just substitute in the particular value of  $s$ . For instance, this situation arises when treating expected values in probability; see Example 4, Section 6.5.

So we strongly urge you to take a second course in discrete mathematics, where generating functions are almost always a major topic.

## Problems: Section 5.5

---

In [1–14], find the solution by the method of this section, the specific solution if initial conditions are given, the general solution otherwise.

1.  $\langle 1 \rangle \quad e_n = 3e_{n-1}$  (Of course, this you can do in your head, but practice the method of the section.)
2.  $\langle 1 \rangle \quad d_{n+1} = -2d_n, \quad d_1 = 3$
3.  $\langle 2 \rangle$  In Example 1, Section 5.2, we assumed it was clear from past experience that  $a_n = ar^n$  is the general solution to

$$a_n = ra_{n-1}. \quad (15)$$

However, it's worth a proof, or two, for practice.

- a) Verify that  $a_n = ar^n$  satisfies (15), where  $a = a_0$ . (Thus by Theorem 1, Section 5.4, all solutions are of this form.)
- b) Use the characteristic equation method of this section to prove that  $ar^n$  is the general solution to (15).
4.  $\langle 1 \rangle \quad a_n = a_{n-1} + 2a_{n-2}, \quad a_1 = 2, \quad a_2 = 3$ .

5.  $\langle 2 \rangle \quad a_n = a_{n-1} + a_{n-2}, \quad a_0 = 1, \quad a_1 = 3$ .
6.  $\langle 1 \rangle \quad a_{n+1} = 5a_n - 6a_{n-1}, \quad a_1 = -1, \quad a_2 = 1$ .
7.  $\langle 1 \rangle \quad a_n = -2a_{n-1} - a_{n-2}, \quad a_0 = 1, \quad a_1 = 2$ .
8.  $\langle 1 \rangle$  Same as [6] except  $a_1 = 5, a_2 = 7$ .
9.  $\langle 1 \rangle$  Same as [7], but  $a_0 = 1, a_1 = -2$ .
10.  $\langle 1 \rangle$  Solve [3acd, Section 5.3].
11.  $\langle 3 \rangle$  (Requires complex numbers) Solve [1, Section 5.3].

For the next three problems, you may want a calculator or a CAS to analyze the characteristic polynomial and solve any systems of linear equations.

12.  $\langle 2 \rangle \quad c_n = -3c_{n-1} - 3c_{n-2} - c_{n-3}$
13.  $\langle 2 \rangle \quad e_n = 3e_{n-2} - 2e_{n-3}$
14.  $\langle 2 \rangle$  For the recurrence in [12] find the particular solution for
  - a)  $c_0 = 1, \quad c_1 = -1, \quad c_2 = 1$
  - b)  $c_0 = c_1 = c_2 = 1$ .

15.  $\langle 2 \rangle$  Suppose that a difference equation has order  $k$ . Explain why the degree of its characteristic polynomial is also  $k$ .
16.  $\langle 2 \rangle$  Explain why
- The leading coefficient of a characteristic polynomial, as defined, is always 1.
  - The constant term is always nonzero.
  - Zero will never be a root of a characteristic equation.
- If  $r = 0$  could be a root, then the “expressed uniquely” part of Theorem 2 would be false.
17.  $\langle 2 \rangle$  Verify that  $\{n2^n\}$  is indeed a solution to Eq. (8).
18.  $\langle 2 \rangle$  In the solution to Example 1, and again in the proof of Theorem 2, we dismissed the case  $r = 0$  in order to divide by  $r$  and get a quadratic. However, the null sequence  $\{0^n\}$  is a solution to the original difference equation, at least if we declare  $0^0$  to be 0 when used in formulas for difference equations, or if we start our sequences at  $a_1$  instead of  $a_0$ . In Theorem 2 we claim that we have all solutions, so the formula there had better include the null sequence even if  $r = 0$  is never a root of the characteristic polynomial. Does it include the null sequence?
19.  $\langle 2 \rangle$  Ever wonder why we usually start our sequences at  $a_0$  instead of  $a_1$ ? One reason is because it makes the algebra less tedious when you have to find the particular solution from initial conditions *by hand*. Check this out with Examples 3 and 5. In each case, do the algebra to find  $A$  and  $B$  using  $a_0$  and  $a_1$  as the initial values. Then find the same  $A$  and  $B$  again using  $a_1$  and  $a_2$ .
20.  $\langle 3 \rangle$  Prove the case  $j = 2$  of Theorem 2 as follows: First, let the difference equation be
- $$a_n = c_1 a_{n-1} + c_2 a_{n-2}, \quad (16)$$
- so the characteristic polynomial is  $r^2 - c_1 r - c_2$ .
- Case 1:
- If the characteristic polynomial factors as  $(r-r_1)(r-r_2)$  with  $r_1 \neq r_2$ , show that  $a_n = r_1^n$  and  $a_n = r_2^n$  both satisfy (16). Hint:  $c_1 = r_1 + r_2$  and  $c_2 = -r_1 r_2$ . Use these facts when plugging  $r_1^n$  and  $r_2^n$  into (16).
  - By Theorem 1 we now know that  $Ar_1^n + Br_2^n$  is a solution sequence for every choice of constants  $A, B$ . To show that there is a unique choice of  $A, B$  for every pair of initial conditions  $a_0, a_1$ , show that the system of linear equations obtained by plugging  $n = 0, 1$  into  $Ar_1^n + Br_2^n = a_n$  has a unique solution. Remember,  $a_0, a_1, r_1, r_2$  are constants (and  $r_1 r_2 \neq 0$ );  $A, B$  are the unknowns.
- Case 2:
- If the characteristic polynomial is  $(r-r_1)^2$ , show that  $a_n = r_1^n$  and  $a_n = nr_1^n$  both satisfy (16). (Start by mimicking the hint in a.)
  - Show that there is a unique choice of  $A, B$  for every pair of initial conditions  $a_0, a_1$ .
21.  $\langle 3 \rangle$  Show: For any second-order linear difference equation, there is always a unique solution given any two consecutive values  $a_k$  and  $a_{k+1}$  as initial conditions, if
- the characteristic equation has distinct roots;
  - it has a double root.
22.  $\langle 2 \rangle$  Give an example of two linear equations in two unknowns that have no solution. By changing the constants on the right-hand side only, obtain a system that has infinitely many solutions.
23.  $\langle 3 \rangle$  Consider the difference equation with characteristic equation  $(r-c)^2(r-d) = 0$ , where  $c \neq d$ . Verify that the sequences  $\{c^n\}$ ,  $\{nc^n\}$ , and  $\{d^n\}$  are solutions.
24.  $\langle 3 \rangle$  We now have two ways to specify the Fibonacci numbers: the recurrence
- $$F_n = F_{n-1} + F_{n-2}, \quad F_0 = F_1 = 1,$$
- and Eq. (7). Which do you feel is more efficient to use, and why, if you want to
- compute the first 25 Fibonacci numbers;
  - analyze the long term growth rate of the Fibonacci sequence;
  - know the order of magnitude of  $F_{100}$ ;
  - know the exact value of  $F_{100}$ .
25.  $\langle 3 \rangle$  Prove that  $F_n$  is the closest integer to
- $$\frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^{n+1}$$
- for all  $n \geq 0$ .
26.  $\langle 3 \rangle$  Consider the directed graph in Fig. 5.7. Count the number of directed paths from  $v_0$  to  $v_{10}$ . Hint: Generalize the problem to ask for a sequence of counts and find a difference equation for the sequence.



**FIGURE 5.7**

27. ⟨2⟩ Suppose national income in each period is the average of national income in the preceding and succeeding periods.

- a) Model this situation with a difference equation and find the general solution.
- b) Find the particular solution if U.S. national income is \$1 trillion in period 0 and \$1.05 trillion in period 1.

28. ⟨3⟩ Consider binary sequences of length  $n$  in which 1's may not appear consecutively except in the rightmost two positions. In Example 4, Section 2.8, we incorrectly proved that there are  $2^n$  such sequences. Find the right number. Hint: Consider how to get sequences of length  $n+1$  by putting 0 or 1 to the left of sequences of length  $n$ .

29. ⟨3⟩ Repeat [28], except now consecutive 1's may not appear anywhere.

30. ⟨2⟩ Prove: If any linear combination of two solutions to a homogeneous linear difference equation is a solution, then any linear combination of any number of solutions is a solution.

31. ⟨2⟩ Find a difference equation whose characteristic polynomial is  $(r - 3)(r + 4)$ .

32. ⟨2⟩ Find a difference equation whose characteristic polynomial is  $(r - 2)^3$ .

In the text we worked from difference equations to polynomials to explicit solutions, but the process can be reversed. Problems [31, 32] asked you to reverse this partly, from polynomial to difference equation; [33–40] ask you to go all the way. Each problem gives you a sequence. Name the simplest difference equation you can (including initial conditions) for which it is a solution.

33. ⟨1⟩  $a_n = 2^n$

34. ⟨1⟩  $b_n = 4 \cdot 3^n$

35. ⟨2⟩  $c_n = 2^n + 3^n$

36. ⟨2⟩  $d_n = 7 + (-1)^n$

37. ⟨2⟩  $e_n = 2^n - (-2)^n + 3$

38. ⟨2⟩  $f_n = 2^n - n 2^n + 3$

39. ⟨3⟩  $f_n = r_1^n + r_2^n$ , where  $r_1$  and  $r_2$  are the zeroes of  $x^2 - 5x + 3$ . You can answer this problem

without ever computing  $r_1$  and  $r_2$ . Finding the recurrence is easy; finding the initial conditions is a little harder.

40. ⟨4⟩  $g_n = r_1^n + r_2^n + r_3^n$ , where  $r_1$ ,  $r_2$  and  $r_3$  are the roots of  $x^3 - x^2 + 2x - 3$ . (The idea behind [39, 40] is known as the method of Newton sums.)

For the rest of the problems in this set, solid knowledge of complex numbers is very useful or even necessary.

41. ⟨3⟩ Consider the recurrence  $a_{n+1} = a_n - a_{n-1}$ , where  $a_0$  and  $a_1$  are anything you please. Show that  $\{a_n\}$  is periodic — starting with a certain  $a_n$ , it simply repeats itself every 6 terms. If you already showed this result in [2, Section 5.3] by induction, this time do it using the theory of this section.

42. ⟨3⟩ Let  $\{a_n\}$  satisfy the recurrence of [41] and satisfy the boundary conditions  $a_0 = 1, a_6 = 1$ . Show that  $\{a_n\}$  is not unique; in fact, there are infinitely many sequences meeting these conditions.

43. ⟨3⟩ Again let  $\{a_n\}$  satisfy the recurrence of [41], but now let the boundary conditions be  $a_0 = 1, a_6 = 2$ . Show that no sequences satisfy these conditions. (Use Theorem 2.)

44. ⟨2⟩ Find a linear difference equation for which  $\{\sin n\theta\}$  and  $\{\cos n\theta\}$  are solutions. Hint:

$$\begin{aligned}\cos n\theta &= (e^{in\theta} + e^{-in\theta})/2 \\ &= \frac{1}{2}r_1^n + \frac{1}{2}r_2^n,\end{aligned}$$

where  $r_1 = e^{i\theta}$  and  $r_2 = e^{-i\theta}$ .

45. ⟨3⟩ Write a computer or calculator program which plots a circle by plotting the 360 points  $(\sin \theta, \cos \theta)$  for integer degrees  $\theta$  from 1 to 360. Have the program compute the values of  $\sin \theta$  and  $\cos \theta$  from initial values using your recurrence from [44]. Which program do you think will run faster, and why: your program or a more straightforward program of the form

```
for θ = 1 to 360
    plot (sin θ°, cos θ°)
endfor
```

To check your conclusion, run and time both programs on some machine. In order for the difference in run times to be visually obvious, you might want to run them on a (slower) graphics calculator instead of a computer.

**46.** *(3)* The simplest correct recurrences for [44] are second order. Come up with a system of linked first-order difference equations relating  $\sin(n+1)\theta$  and  $\cos(n+1)\theta$  to  $\sin n\theta$ ,  $\cos n\theta$  and some initial values. Write a program for plotting a 360-point circle using this linked recurrence. Compare its speed to that of the two programs in [45].

**47.** *(2)* Use generating functions to solve each of the following difference equations.

a)  $r_n = 2r_{n-2}$ ,  $r_0 = 0$ ;  $r_1 = 1$ .

b)  $2r_{n+1} = r_n + r_{n-1}$ ,  $r_0 = 0$ ;  $r_1 = 1$ .

c)  $r_{n+1} = 2r_n - r_{n-1}$ ,  $r_0 = r_1 = 1$ .

d) The TOH recursion:

$$h_{n+1} = 2h_n + 1, \quad h_0 = 0.$$

**48.** *(3)* In Example 6 the quadratic in the denominator of  $G(s)$  in Eq. (12) is very closely related to the characteristic polynomial for the difference equation, (10). State the exact relationship and explain why it holds for any CCHLDE.

**49.** *(3)* Show that if the characteristic polynomial of a second-order CCHLDE is  $r^2 + cr + d$ , and it factors

as  $(r-r_1)(r-r_2)$  where  $r_1 \neq r_2$ , then the partial fraction decomposition of the generating function is

$$\frac{A}{1-r_1s} + \frac{B}{1-r_2s},$$

where  $A, B$  are constants that depend on the first two terms  $a_0, a_1$  of the sequence. Verify that this partial fraction approach gives the same solution sequences as the characteristic polynomial method.

**50.** *(4)* Show that if the characteristic polynomial of a second-order CCHLDE is  $(r-r_1)^2$ , then the partial fraction decomposition of the generating function is

$$\frac{A}{1-r_1s} + \frac{B}{(1-r_1s)^2},$$

where  $A, B$  are constants that depend on the first two terms  $a_0, a_1$  of the sequence. Verify that this partial fraction approach gives the same solution sequences as the characteristic polynomial method.

## 5.6 Qualitative Analysis

Sometimes it's not important to have an exact value for  $a_n$ . Sometimes it's even misleading: if the model that leads to a difference equation is not very exact, it would be wrong to believe in the exactness of the answer that comes from that difference equation. In such cases one might be more interested in qualitative results. For instance, what happens to the sequence in the long run? Does  $a_n$  go to zero, head to some nonzero limit, oscillate indefinitely, go to plus or minus infinity, or what? In this section we will analyze long-term behavior. Occasionally we will assume basic familiarity with complex numbers.

Theorem 2, Section 5.5, says that every solution is a sum of geometric sequences or close relatives. Since the qualitative behavior of any one geometric series is easy to analyze, it is possible to piece together the behavior of any difference equation satisfying Theorem 2.

### EXAMPLE 1

Analyze the long-term behavior of the following sequences:

$$a_n = 2\left(\frac{1}{3}\right)^n + 4\left(\frac{1}{2}\right)^n,$$

$$b_n = 3 - 5\left(\frac{1}{2}\right)^n,$$

$$c_n = (-2)^n + 5\left(\frac{1}{2}\right)^n,$$

$$d_n = 2^n - 5(1.5)^n.$$

**Solution** For  $\{a_n\}$ , the two geometric sequences are  $\{2(\frac{1}{3})^n\}$  and  $\{4(\frac{1}{2})^n\}$ . Since the ratio in both cases is between 0 and 1, both sequences head towards 0. Therefore their sum goes to 0 as well. This is shown graphically in Figure 5.8a.

For  $\{b_n\}$ , the first geometric sequence is the constant sequence with  $n$ th term  $3 = 3 \cdot 1^n$ . Since the second geometric sequence goes to 0, we see that  $b_n \rightarrow 3$ . This is shown in color in Figure 5.8a.

For the third sequence, the second term again dies out. The first term has ratio  $-2$ , and therefore alternates positive and negative terms, growing without bound in absolute value. This is sometimes called **exponential oscillation**. Therefore, the combined effect of both terms is also explosive oscillation.

For  $d_n$ , both ratios are greater than 1. The first term,  $2^n$ , is always positive, while the second term,  $-5(1.5)^n$ , is always negative. The first term has the slightly greater ratio, but the second has a larger multiplicative constant. Which wins out in the long run? The answer is: the larger ratio. To see this, rewrite the formula for  $d_n$  as follows:

$$d_n = 2^n \left[ 1 - 5\left(\frac{1.5}{2}\right)^n \right].$$

Now note that  $(1.5/2)^n$  goes to 0 as  $n$  gets large, so the quantity inside the brackets approaches 1. Therefore, in the long run  $d_n$  looks essentially like  $2^n$ . This is illustrated in Figure 5.8b. Note how initially  $d_n$  is negative — the term  $-5(1.5)^n$  is initially more significant. But then the sum turns positive and start to grow like  $2^n$ . ■

Example 1 involved sequences that arise from difference equations with distinct roots. Now let us look at sequences arising from multiple roots.

## EXAMPLE 2

Analyze the long-term behavior of the following sequences:

$$\begin{aligned} e_n &= -3 \cdot 2^n + .7n2^n, \\ f_n &= 3(.6)^n + 4n(.6)^n. \end{aligned}$$

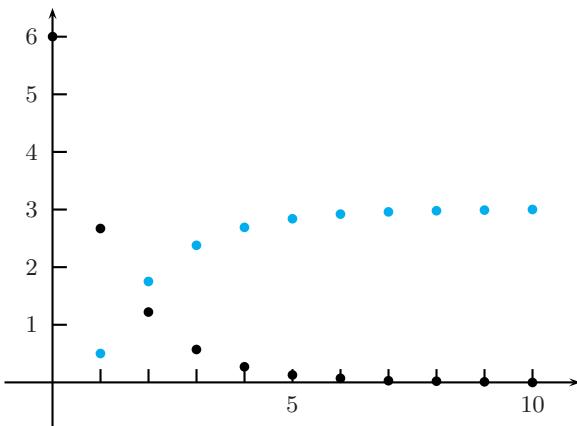
**Solution** We may rewrite the first formula as  $(.7n-3)2^n$ . As  $n$  gets large, both factors go to positive infinity, so their product does also. (Note that  $e_n$  is initially negative, since  $3 > .7n$  for small  $n$ , but eventually  $.7n$  is much larger than 3.)

As for  $f_n$ , the same sort of factoring yields  $(4n+3)(.6)^n$ . Now the first factor goes to infinity as  $n$  increases, but the second goes to 0. Which wins out? Answer: the second. More generally, whenever a polynomial  $p(n)$  is multiplied by an exponential  $b^n$  with  $0 < b < 1$ , the limit is 0 as  $n \rightarrow \infty$ . This fits in with the general rule in Section 0.2 that exponentials overpower polynomials, but it also follows from the specific result there that

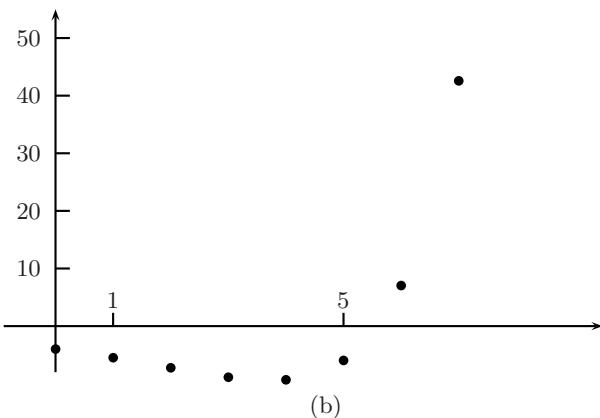
$$\lim_{n \rightarrow \infty} \frac{n^k}{a^n} = 0 \quad \text{for } a > 1.$$

Just set  $b = 1/a$  and note that  $p(n)b^n$  is just a linear combination of terms  $n^k b^n$ .

Another way to think about the long-run behavior of  $f_n$  intuitively is this: when  $n$  is large, each increase by 1 in  $n$  reduces  $(.6)^n$  by 40%, but the relative



(a)



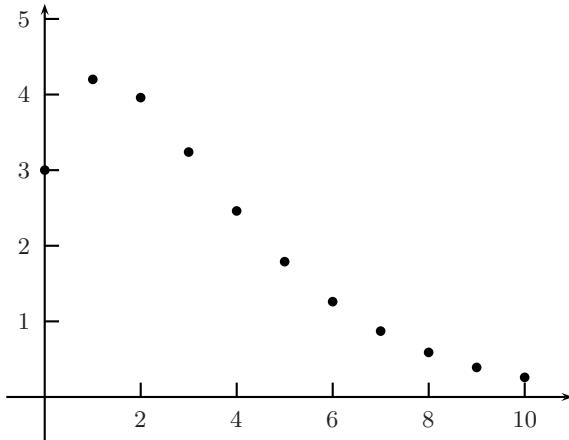
(b)

- (a) Graph of  $a_n = 2\left(\frac{1}{3}\right)^n + 4\left(\frac{1}{2}\right)^n$  (black) and  $b_n = 3 - 5\left(\frac{1}{2}\right)^n$  (color).
- (b) Graph of  $d_n = 2^n - 5(1.5)^n$ .

**FIGURE 5.8**

increase in  $(4n+3)$  is very small, so in the product the decrease wins out. This is illustrated graphically in Figure 5.9. Note that  $f_n$  increases at first. ■

From these examples the basic pattern is clear: the significant term in the long run is the one whose geometric ratio is greatest in absolute value (and if several terms share this highest ratio, the one which includes the highest power  $n^k$  is most significant). But not so fast. This doesn't tell the whole story, even if we restrict ourselves (as we will for the rest of this section) to the simpler case of distinct roots. What happens, for instance, if there is a tie for largest absolute value?



**FIGURE 5.9**

Graph of  $f_n = 3(.6)^n + 4n(.6)^n$

Consider the simple case  $a_n = 1^n + (-1)^n$ . Then the sequence alternates 2, 0, 2, 0 . . . . Another way to say this is that we have a sequence with **period 2**. In this case the geometric ratios both have absolute value 1. Related behavior occurs for

$$b_n = 3(1.2)^n + 2(-1.2)^n.$$

Here both ratios again have the same absolute value but it's not 1. When  $n$  is even, the negative sign drops out and we have

$$b_n = 5(1.2)^n \quad n \text{ even.}$$

When  $n$  is odd, the negative sign stays and we get

$$b_n = (1.2)^n \quad n \text{ odd.}$$

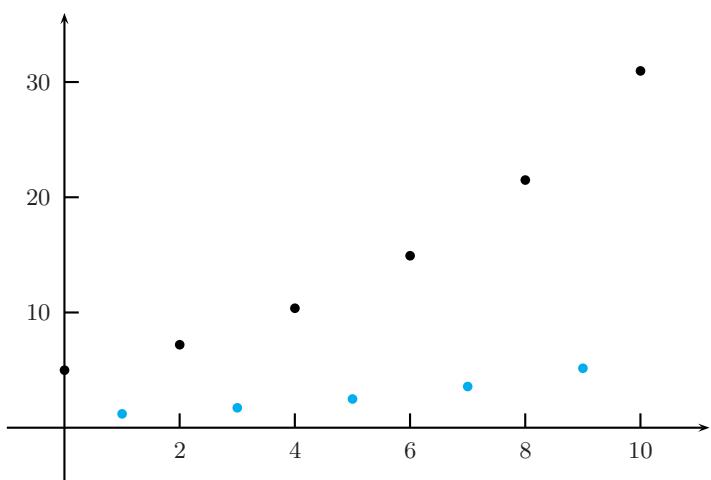
The graph is shown in Figure 5.10. Instead of alternating between two numbers, this sequence alternates between two simple patterns. Both patterns are geometric, differing only by the constant.

The story is still not over. The geometric ratios in solutions to difference equations are roots of polynomials; what if these roots are complex numbers? Figure 5.11 shows two examples, for the sequences

$$c_n = i^n + (-i)^n \quad \text{and} \quad d_n = (.8i)^n + (-.8i)^n.$$

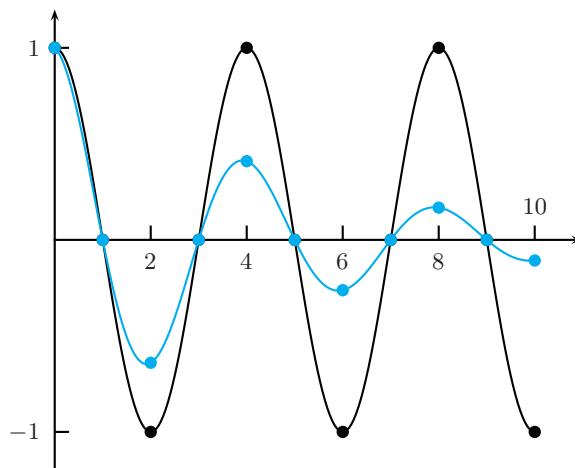
First note that the  $c$ 's and  $d$ 's are real numbers, despite being expressed using complex numbers. Next notice that the  $c$  sequence is periodic of period 4, and the  $d$  sequence also has a pattern of length 4, but each set of 4 has values closer to 0 than the previous set. In fact, these are just discrete analogs of the “harmonic” and “damped harmonic” motion one studies in calculus. To show this, we have superimposed the appropriate cosine curves over the discrete graph.

There is nothing special about period 4. With appropriate complex roots, it is possible to get any period and certain more general types of “cyclic” behavior.



The sequence  $b_n = 3(1.2)^n + 2(-1.2)^n$  alternates between  $b_n = 5(1.2)^n$  for  $n$  even (black) and  $b_n = (1.2)^n$  for  $n$  odd (color).

**FIGURE 5.10**



The sequences  $c_n = i^n + (-i)^n$  (black) and  $d_n = (.8i)^n + (-.8i)^n$  (color). These sequences are discrete analogs of the continuous circular functions  $y = \cos(\pi x/2)$  and  $y = (.8)^x \cos(\pi x/2)$ .

**FIGURE 5.11**

Suffice it to say this: if you know the algebra of complex numbers well, including the concepts of modulus and argument and the Theorem of DeMoivre, the analysis of geometric series with complex ratios is not much harder than for real ratios. The same basic result applies: if  $|r| > 1$ , then  $ar^n$  is unbounded; if  $|r| < 1$ , then  $ar^n \rightarrow 0$ ; and if  $|r| = 1$ , then one gets cyclic motion.

Let us get back to the essential point. The qualitative behavior of the solution to a linear difference equation depends only on the roots of the characteristic equation — in fact, the root(s) with the largest absolute value. Knowing the formula for the solution is not necessary.

### EXAMPLE 3

Give a qualitative analysis of the general solution to

$$a_n = a_{n-1} + 6a_{n-2}.$$

**Solution** The characteristic polynomial is  $r^2 - r - 6 = (r+2)(r-3)$ . Thus the general solution is

$$A3^n + B(-2)^n.$$

The first term will predominate, except in the rare case that initial conditions force  $A = 0$ . Even if  $A$  is tiny and  $B$  is huge, eventually the term  $A3^n$  will predominate. Therefore, the sequence may show oscillating behavior at first (if  $B$  is large and thus  $B(-2)^n$  is more significant for small  $n$ ) but eventually the sequence will stay positive and growing (if  $A > 0$ ) or stay negative and growing (if  $A < 0$ ). ■

Although our examples in this subsection have involved the sum of just two geometric sequences, nothing in our analysis requires this. To highlight this fact, we state the most important case as a theorem. In [4] you are asked to prove it.

**Theorem 1.** Suppose  $\{a_n\}$  is the solution to a linear, constant coefficient, homogeneous difference equation whose characteristic equation has distinct roots. Assume that  $r_1$  is the root with the largest absolute value and that no other root has the same absolute value. Assume further that the coefficient of  $r_1^n$  in the explicit formula for  $a_n$  is not 0. Then in the long run,  $a_n$  behaves like a multiple of the geometric sequence  $r_1^n$ . ■

## Problems: Section 5.6

1.
  - 1) Determine the long-range behavior of the following sequences. You should be able to do this in your head, though you might confirm your answers with a graphics calculator.
    - a)  $2^n + 3^n$
    - b)  $3 + (.5)^n$
    - c)  $3(1.1)^n - (1.2)^n$
    - d)  $3(1.1)^n + (-1.2)^n$
    - e)  $2(1.2)^n + 3(-1.2)^n$
    - f)  $(.9)^n + 2(-.9)^n$
    - g)  $3(\frac{1}{2})^n + 5n(\frac{1}{2})^n$
    - h)  $2(1.2)^n - 3n(1.2)^n$
    - i)  $2^n - 13(3^n) + .1(4^n)$
    - j)  $(2 + 4n)2^n + (-2.1)^n$
  2.
    - 1) Show that in the long term  $a_n = 2^n - 13(3^n) + .1(4^n)$  behaves like  $.1(4^n)$  by rewriting  $a_n$  in the form  $4^n$  times something.
    - 2) Consider the recurrence
$$a_n = a_{n-1} + a_{n-2} + a_{n-3},$$

with  $a_0 = a_1 = 1$ ,  $a_2 = 2$ . This comes from the modified rabbit problem of [12, Section 5.2]. Find, at least approximately, the long-term rate of growth for  $a_n$ . Note: This recurrence cannot

be solved by hand (unless you know about the general formula for roots of cubics), but there are several ways to answer the question using modest calculator power.

4. (3) Prove Theorem 1. The concepts and methods of Section 0.3 should help. The first step is to decide what the precise meaning of “behaves like” should be.

## 5.7 Nonhomogeneous Difference Equations

---

Throughout this section, “difference equation” still means “constant coefficient, linear difference equation”. But now we allow the difference equation to be nonhomogeneous. Thus

$$a_n = a_{n-1} + as_{n-2} + n^2.$$

is nonhomogeneous with nonhomogeneous part  $n^2$ . If we drop off the nonhomogeneous part, we have the **associated homogeneous** difference equation

$$h_n = h_{n-1} + 2h_{n-2}.$$

We have used  $h_n$  for mnemonic reasons ( $h$  for homogeneous) but, of course, it’s irrelevant what we call the members of a sequence.

To solve homogeneous difference equations it was enough to find basis solutions. For nonhomogeneous difference equations we need only find any one solution and then add to it the general solution to the associated homogeneous equation. The result is the complete solution. In this section we will see why, and give examples.

First let’s see why the approach we used for *homogeneous* difference equations doesn’t work. Why can’t we just continue to take linear combinations of solutions, as in Theorem 1, Section 5.5? Let’s find out by trying to mimic the proof of that theorem.

Suppose, for example, that both  $\{a_n\}$  and  $\{b_n\}$  satisfy

$$s_n = s_{n-1} + 2s_{n-2} + n^2. \tag{1}$$

Substituting first  $a_n$  and then  $b_n$  for  $s_n$ , and then adding and regrouping, we obtain

$$a_n + b_n = (a_{n-1} + b_{n-1}) + 2(a_{n-2} + b_{n-2}) + 2n^2.$$

Thus the equation that  $\{a_n+b_n\}$  satisfies has a different nonhomogeneous part,  $2n^2$ , and so solution sets for nonhomogeneous difference equations are *not* closed under linear combinations.

However, if adding makes the nonhomogeneous part larger, then subtracting might make it smaller. In fact, it will go away:

$$b_n - a_n = (b_{n-1} - a_{n-1}) + 2(b_{n-2} - a_{n-2}). \tag{2}$$

This is good, because if we define  $h_n = b_n - a_n$ , Eq. (2) becomes

$$h_n = h_{n-1} + 2h_{n-2}, \tag{3}$$

which is just the associated homogeneous equation of Eq. (1), which we know how to solve from Section 5.5. (This process — making problems simpler by subtraction — is one that occurs frequently in mathematics.)

Thus, if we know *any one* solution  $a_n$  of Eq. (1), then *any other* solution  $b_n$  can be found by adding to  $a_n$  some solution of the associated homogeneous equation. Or putting this another way, to get the general solution of the nonhomogeneous equation, add any one solution of the nonhomogeneous equation to the general solution of the associated homogeneous equation. We shall show this explicitly in the proof of Theorem 1 below.

### EXAMPLE 1

Find the general solution to

$$R_n = R_{n-1} + 2R_{n-2} + 1. \quad (4)$$

**Solution** It happens that the constant sequence  $a_n = -\frac{1}{2}$  is one solution — plug it in and see. (We'll explain how to find such “starter” solutions later.) Since the associated homogeneous equation for Eq. (4) is again Eq. (3), it follows from the discussion above that the complete set of solutions to Eq. (4) are those  $b_n$  of the form  $b_n = a_n + h_n$ , where  $h_n$  ranges over all solutions to Eq. (3). By the characteristic polynomial method (Section 5.5), we know that the general solution for Eq. (3) is  $A2^n + B(-1)^n$ . Therefore every solution to Eq. (4) is of the form

$$b_n = -\frac{1}{2} + A2^n + B(-1)^n. \blacksquare$$

We summarize with a theorem.

**Theorem 1.** Let  $\{a_n\}$  be any one solution to the nonhomogeneous, constant coefficient linear difference equation

$$s_n = \sum_{i=1}^k c_i s_{n-i} + f(n). \quad (5)$$

(Remember: “one solution” means a solution of this difference equation but not one that necessarily satisfies any initial conditions.) Then

$$b_n = a_n + h_n$$

is the general solution (i.e., the most general sequence satisfying the difference equation), where  $\{h_n\}$  is the general solution to the associated homogeneous difference equation

$$h_n = \sum_{i=1}^k c_i s_{n-i}. \quad (6)$$

**PROOF** If  $a_n$  is any solution to (5) and  $b_n$  is any other solution, then by subtraction,  $h_n = b_n - a_n$  satisfies

$$\begin{aligned} b_n - a_n &= \left( \sum_{i=1}^k c_i b_{n-i} + f(n) \right) - \left( \sum_{i=1}^k c_i a_{n-i} + f(n) \right) \\ &= \sum_{i=1}^k c_i (b_{n-i} - a_n) + f(n) - f(n), \end{aligned}$$

that is,

$$h_n = \sum_{i=1}^k c_i h_{n-i}.$$

Thus  $h_n$  satisfies Eq. (6). Conversely, if we take any  $h_n$  that satisfies Eq. (6) and now define  $b_n$  by  $b_n = a_n + h_n$ , then by addition  $b_n$  satisfies

$$\begin{aligned} a_n + h_n &= \left( \sum_{i=1}^k c_i a_{n-i} + f(n) \right) + \left( \sum_{i=1}^k c_i h_{n-i} \right) \\ &= \sum_{i=1}^k c_i (a_{n-i} + h_n) + f(n), \end{aligned}$$

that is,

$$b_n = \sum_{i=1}^k c_i b_{n-i} + f(n).$$

In short, for any solution  $a_n$  to Eq. (5), everything of the form  $a_n + h_n$ , and only sequences of this form, is also a solution. ■

*Remark:* The one nonhomogeneous solution you have to find first is often called “a particular” solution, because you need just one, but it can be any one.

We have now shown how to find the general solution to a nonhomogeneous difference equation. To find the solution that also satisfies the given initial conditions, proceed as in Section 5.5: first find the general solution and then use the initial conditions to determine the coefficients.

## EXAMPLE 2

Find the solution to

$$a_n - 2a_{n-1} = 3^n \tag{7}$$

for which  $a_0 = 4$ .

**Solution** To find the general solution, we first need to find some one solution; it need not satisfy  $a_0 = 4$ . Let’s guess — intelligently! Unless  $a_n$  involves  $3^n$  in some way, it seems unlikely that subtracting  $2a_{n-1}$  is going to create  $3^n$ . So let’s guess  $a_n = A3^n$ , where  $A$  is to be determined. Substituting in the LHS of Eq. (7) gives

$$A3^n - 2(A3^{n-1}) = A3^n - \frac{2}{3}A3^n = \frac{A}{3}3^n.$$

Therefore

$$\frac{A}{3}3^n = 3^n \quad \text{or} \quad A = 3.$$

Thus  $a_n = 3(3^n) = 3^{n+1}$  is one solution of the difference equation but, as you can easily verify, it does not satisfy the initial condition. Since the general solution of the associated homogeneous equation is  $B2^n$  (why?), the general solution to Eq. (7) is

$$a_n = 3^{n+1} + B2^n.$$

Now, to satisfy the initial condition, plug in  $n = 0$  and find

$$4 = a_0 = 3 + B \quad \text{or} \quad B = 1.$$

Therefore the specific solution for the initial condition is

$$a_n = 3^{n+1} + 2^n. \blacksquare$$

*Finding that first solution.* You generally don't have to guess, because, for many standard forms for the nonhomogeneous part, there are procedures that will pick a good guess for you and then find the right coefficients. Such general procedures are built into good CAS software but they are too complicated for us to discuss here.

Nonetheless, it's good to know how to guess right in simple cases, so that you don't run to your calculator when you can guess the right answer easily yourself. So we will explain enough to give you an idea of the key theorem and to allow you to guess right in simple cases.

First, note that in both examples so far, the right guess was a multiple of the nonhomogeneous part. In Example 1, the starter solution was  $a_n = -\frac{1}{2}$  and the nonhomogeneous part was 1. In Example 2, the starter was  $3^{n+1}$  and the nonhomogeneous part was  $3^n$ . Indeed, the right guess is usually  $Af(n)$ , where  $f(n)$  is the nonhomogeneous part and  $A$  is a constant. However, suppose Eq. (7) had been

$$a_n - 2a_{n-1} = 2^n. \tag{8}$$

Then  $A2^n$  cannot be a solution because instead it is a solution to the associated homogeneous equation  $a_n - 2a_{n-1} = 0$  (the characteristic polynomial is  $r - 2$ ). In this case the right guess is  $An2^n$ , that is, you go up to the next basic solution in Theorem 2, Section 5.5. Had  $n2^n$  also been a solution to the homogeneous form of Eq. (8), that is, if the LHS of Eq. (8) had  $(r-2)^2$  in its characteristic polynomial, then you would guess  $An^22^n$ , and so on.

**EXAMPLE 3** Find the solution to Eq. (8) for which  $a_0 = 4$ .

**Solution** Plug the guess  $An2^n$  into Eq. (8):

$$\begin{aligned} An2^n - 2(A(n-1)2^{n-1}) &= 2^n \\ A(n2^n - (n-1)2^n) &= 2^n \\ A2^n &= 2^n. \end{aligned}$$

Thus  $A = 1$  makes the guess right. Therefore, a particular solution is  $a_n = n2^n$  and

the general solution is  $a_n = n2^n + B2^n$ . To determine  $B$  that satisfies the initial condition, plug in  $n = 0$ :

$$4 = a_0 = 0 + B \quad \text{or} \quad B = 4.$$

Therefore the specific solution for the initial condition is

$$a_n = (n+4)2^n. \blacksquare$$

For which CCLDEs can this method of guessing a particular solution be systematized and always lead to a closed form general solution? Answer: Whenever the nonhomogeneous part is itself the solution to some *homogeneous* constant coefficient difference equation. For a proof of a special case that illustrates the general idea behind this claim, see [11].

## Problems: Section 5.7

Some of the problems below, or problem parts, are tedious to do by hand. In that case use a computer algebra system. However, it is worthwhile to do as much as you can in your head and by hand, to get a feel for what's happening.

1.  $\langle 2 \rangle$  Find the general solution to  $b_n = 3b_{n-1} + f(n)$ , where  $f(n)$  is

- a)  $2^n$       b)  $n2^n$       c)  $3^n$   
 d) 1      e)  $(-2)^n$       f)  $5 \cdot 3^n$ .

2.  $\langle 2 \rangle$  Find the general solution to

$$a_n = 3a_{n-1} - 2a_{n-2} + f(n),$$

for each choice of  $f(n)$  in [1].

3.  $\langle 3 \rangle$  Find the general solution to  $c_{n+1} = 4c_n - 4c_{n-1} + f(n)$  for each choice of  $f(n)$  in [1].

4.  $\langle 1 \rangle$  Solve the TOH recurrence using the method of this section.

5.  $\langle 1 \rangle$  Solve  $s_n = 3s_{n-1} + 2$ ,  $s_1 = 2$  (the Straight-line TOH recurrence) using the method of this section.

6.  $\langle 2 \rangle$  Solve [22, Section 5.3] (consumer loans) by the methods of this section.

7.  $\langle 2 \rangle$  Except for the nonhomogeneous part, Eq. (4) is the difference equation for the rabbit problem [12, Section 5.2]. Make up a plausible story about rabbits that fits Eq. (4), including the nonhomogeneous part.

8.  $\langle 3 \rangle$  In Theorem 1, is it necessary that  $a_n$  and  $b_n$

satisfy a *constant coefficient* nonhomogeneous linear difference equation? Is it necessary that they satisfy a *linear* difference equation? Why?

9.  $\langle 2 \rangle$  If  $\{a_n\}$  satisfies  $s_n = s_{n-1} + 2s_{n-2} + n^2$ , show that  $\{3a_n\}$  does not satisfy this same equation. What difference equation does  $\{3a_n\}$  satisfy? (This problem shows that solutions to nonhomogeneous linear difference equations are not closed under multiples.)

10.  $\langle 2 \rangle$  The systematic method mentioned in the last paragraph of text in the section works only if you can find a CCHLDE that your nonhomogeneous part  $f(n)$  satisfies. But it's not hard to tell when  $f(n)$  has this property because you already know all possible solutions to such equations. Describe them. We'll call these functions **polygeometric** and we'll refer to them in the next two problems.

11.  $\langle 4 \rangle$  Suppose  $\{f_n\}$  satisfies

$$f_n - 3f_{n-1} = 0. \tag{9}$$

(Of course,  $f_n$  must be a multiple of  $3^n$ , but thinking about a closed form would simply obscure the point in what follows.) Suppose further that  $\{a_n\}$  satisfies

$$a_n - 2a_{n-1} - 3a_{n-2} = f_n. \tag{10}$$

Substitute Eq. (10) into Eq. (9) to obtain a *homogeneous* equation  $E$  for  $\{a_n\}$ . Without regrouping  $E$  in standard form, determine its characteristic polynomial.

a) Explain why we now know that every solution to Eq. (10) must be polygeometric (as defined in [10]) and how we know how to limit our guess for a particular solution.

b) Explain how the characteristic polynomial for equation  $E$  is related to the polynomial  $q(r)$  for Eq. (9) and the polynomial  $p(r)$  for the homogeneous difference equation associated with Eq. (10)?

12. (3) For any function  $f$  with domain  $N$ , define  $F$  by

$$F(n) = \sum_{k=1}^n f(k).$$

a) Using the last paragraph in the section text, and the definition of polygeometric in [10], explain why  $F(n)$  is guaranteed to have a closed form if  $f$  is polygeometric.

Use the method of this section to find the closed form for the following sums.

b)  $\sum_{k=1}^n k$

c)  $\sum_{k=1}^n k2^k$

d)  $\sum_{k=1}^n k^2/2^k$

13. (3) Suppose  $f(n)$  is not of a form which solves a homogeneous difference equation. That doesn't mean a nonhomogeneous problem involving  $f(n)$  cannot be solved. Theorem 1 still applies. So if by hook or by crook you can find one solution to your nonhomogeneous difference equation, you automatically get them all. For instance, consider

$$a_n = 2a_{n-1} + \frac{1}{n}. \quad (11)$$

By playing around — setting  $a_0 = 0$  and using the inductive paradigm to compute terms and look for a pattern — you might soon conjecture that one solution is

$$a_n = \sum_{k=1}^n \frac{2^{n-k}}{k}.$$

Prove this and then find all solutions to Eq. (11).

14. (3) Find all solutions to  $b_n = 3b_{n-1} + \log n$ . See [13].

## 5.8 Applications to Algorithms

When you want to analyze algorithms (determine how many steps they take), difference equations occur naturally. They are especially natural if the algorithms are recursive.

We are not done yet introducing methods to solve difference equations, but we have enough for the analysis of algorithms. While we have analyzed many algorithms earlier in this book, now we can be more systematic and at the same time briefer. Below, once we get a recurrence, we will often simply state the solution; you know how to find it by hand or machine. (In fact, most of the recurrences below are easy to solve by hand, or even in your head.)

We warm up with two simple, iterative algorithms for evaluating polynomials. Then we move on to several recursive algorithms. One of the points we will demonstrate is that a recursive algorithm, though easy to understand, is often *not* the most efficient way to do a computation. Often it is horrendously inefficient. We will demonstrate this with the Fibonacci numbers.

Finally, we give several examples of a special class of recursive algorithms that are often surprisingly efficient — **Divide and Conquer** algorithms (D&C). The recursive counting method that goes with Divide and Conquer is also special, and we give several examples.

OK, let's evaluate polynomials. Algorithm 5.1, STRAIGHTFORWARD, needs no introduction because it corresponds exactly to the way most people think to evaluate a polynomial: You compute each term in order and add them up.

## Algorithm 5.1

### Straightforward

**Input**  $a_0, a_1, \dots, a_n, x$   
**Output**  $S = \sum_{k=0}^n a_k x^k$   
**Algorithm** STRAIGHTFORWARD

```
 $S \leftarrow a_0$ 
for  $k = 1$  to  $n$ 
     $S \leftarrow S + a_k x^k$ 
endfor
```

The next algorithm needs some explanation. Think recursively. Suppose we already know how to evaluate polynomials of degree  $n-1$  and now wish to evaluate

$$P(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n. \quad (1)$$

The obvious way is

$$\left( a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} \right) + a_n x^n.$$

That is, first evaluate the  $(n-1)$ st degree polynomial in parentheses and then add  $a_n x^n$ . This method of evaluation is easy to implement iteratively, and it gives us Algorithm STRAIGHTFORWARD. But it's also possible to "grow  $P(x)$  from the right". Namely, consider the  $(n-1)$ st degree polynomial that uses the right-most  $n$  coefficients from Eq. (1):

$$a_1 + a_2 x + a_3 x^2 + \cdots + a_{n-1} x^{n-2} + a_n x^{n-1}.$$

(Note that the subscripts on  $a$  and the powers on  $x$  differ by 1.) Then

$$P(x) = a_0 + \left( a_1 + a_2 x + \cdots + a_{n-1} x^{n-2} + a_n x^{n-1} \right) x.$$

(Note the  $x$  at the end.) This method of evaluation also can be implemented iteratively, and is known as **Horner's Algorithm**.<sup>†</sup>

## EXAMPLE 1

Compare the efficiency of Algorithms STRAIGHTFORWARD and HORNER by counting the number of multiplications.

**Solution** It's easy to eyeball the answers here, but let's practice thinking systematically. Let  $M_k$  be the number of multiplications in STRAIGHTFORWARD after  $k$

<sup>†</sup>William Horner, 1786–1837, was an English mathematician and school headmaster. He used this algorithm as part of "Horner's Method", a now obsolete paper-and-pencil procedure for computing roots of polynomials to several decimal places without going completely mad. Horner's algorithm, however, is far from obsolete, as you are about to see. The traditional hard copy format can be found in many high school algebra books under the name "synthetic division" or "synthetic substitution".

## Algorithm 5.2

### Horner

**Input** and **Output** same as for STRAIGHTFORWARD

#### Algorithm HORNER

```
S ← an
for k = n - 1 downto 0
    S ← Sx + ak
endfor
```

passes through the loop, and let  $M_k^*$  be the corresponding number for HORNER. In STRAIGHTFORWARD the  $k$ th pass through the loop requires  $k$  multiplications ( $k-1$  multiplications for  $x^k$  and one for the multiplication by  $a_k$ ). In HORNER each pass through the loop requires just one multiplication. Therefore we have the difference equations

$$\begin{aligned} M_k &= M_{k-1} + k, \quad k \geq 1; & M_0 &= 0; \\ M_k^* &= M_{k-1}^* + 1, \quad k \geq 1; & M_0^* &= 0. \end{aligned}$$

Since both algorithms pass through their loops  $n$  times, we want  $M_n$  and  $M_n^*$ . It's easy to see from what we just said that

$$M_n^* = n \quad \text{and} \quad M_n = \sum_{i=1}^n i = \frac{n(n+1)}{2}.$$

However, if for some reason we didn't see these closed forms right away, we could obtain them by the methods for nonhomogeneous linear difference equations (Section 5.7). In any event, note how much more efficient HORNER is: It's an  $\text{Ord}(n)$  algorithm while STRAIGHTFORWARD is  $\text{Ord}(n^2)$ . In fact, it is known that HORNER is best possible in terms of the number of multiplications. ■

Why did we analyze STRAIGHTFORWARD and HORNER in terms of multiplications, instead of, say, additions or assignments? Because both algorithms are  $\text{Ord}(n)$  for both additions and assignments [1]. Thus multiplication is the operation that separates one algorithm from the other. Since multiplications typically take longer in computers than additions or assignments (though not by much these days), multiplication is clearly the operation to focus on in comparing STRAIGHTFORWARD and HORNER.

For each algorithm below, we will analyze it in terms of what is typically regarded as the most appropriate measure. However, using difference equations works equally well no matter what aspect of an algorithm you wish to measure, as several problems will explore.

## EXAMPLE 2

Consider the following two, slightly different recursive procedures for Towers of Hanoi. Call them Procedures TOH and TOH\*. (We omit the rest of the algorithms in which these procedures reside.) Is one more efficient than the other?

```

procedure TOH(in  $n, i, j$ ) [Move  $n$  rings from pole  $i$  to pole  $j$ ]
  if  $n > 1$  then TOH( $n-1, i, 6-i-j$ ) endif
  move Ring  $n$  from Pole  $i$  to Pole  $j$ 
  if  $n > 1$  then TOH( $n-1, 6-i-j, j$ ) endif
endpro

```

```

procedure TOH*(in  $n, i, j$ )
  if  $n = 1$  then move Ring  $n$  from Pole  $i$  to Pole  $j$ 
  else
    TOH*( $n-1, i, 6-i-j$ )
    move Ring  $n$  from Pole  $i$  to Pole  $j$ 
    TOH*( $n-1, 6-i-j, j$ )
  endif
endpro

```

Both procedures are correct (for  $n \geq 1$  rings); indeed, both perform exactly the same minimal sequence of moves. This result is easy to show by induction [7]. The question is whether one requires more peripheral activity than the other. The first has fewer lines. Does that make it better?

**Solution** Let us use as our measure of efficiency the number of times  $n$  is compared to 1; in Procedure TOH these comparisons take the form “if  $n > 1 \dots$ ”, and in TOH\* the form “if  $n = 1 \dots$ ”. We use this measure for two reasons. First, comparison is the only operation that occurs more often in the statement of one algorithm than in the other, suggesting (correctly) that it is the only operation for which the actual count will be different. Second, these comparisons are all that either procedure does, other than invoking itself and moving rings. (Actually, invoking itself, that is, doing the overhead for setting up recursion, is relatively substantial in actual computers, so computing the number of procedure calls in the two versions is probably a better measure. But our decision to count comparisons illustrates the method well enough, which is the main point here.) Let  $C_n$  be the number of comparisons made by TOH( $n, i, j$ ), including comparisons in subcalls. Define  $C_n^*$  analogously. Then we have

$$\begin{aligned} C_1 &= 2 & \text{and} & \quad C_n = 2 + 2C_{n-1} & \text{for } n > 1, \\ C_1^* &= 1 & \text{and} & \quad C_n^* = 1 + 2C_{n-1}^* & \text{for } n > 1. \end{aligned}$$

For instance, the first of these lines is correct because Procedure TOH has two explicit comparisons of  $n$  and 1, and for  $n > 1$  it also results in two calls of TOH with  $n-1$  rings.

From our knowledge of nonhomogeneous recurrences, we conclude that

$$C_n = A2^n + B, \quad C_n^* = A^*2^n + B^*,$$

where  $A, B, A^*, B^*$  are constants to be determined. It turns out that

$$C_n = 2 \cdot 2^n - 2, \quad C_n^* = 2^n - 1.$$

In short, Procedure TOH makes twice as many comparisons, making it run longer (though not by much relatively, because as noted earlier, the overhead from procedure calls actually takes the most time, and both procedures make the same large number of procedure calls [8]). ■

### EXAMPLE 3

Analyze Algorithms FIBA and FIBB, which find the  $n$ th Fibonacci number,  $F_n$ .

#### Algorithm 5.3

##### FibA

|                                                                                                                                                                                              |              |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| <b>Input</b> $n$<br><b>Output</b> $F_n$<br><b>Algorithm</b> FIBA<br>$F_0 \leftarrow 1; F_1 \leftarrow 1$<br><b>for</b> $k = 2$ to $n$<br>$F_k \leftarrow F_{k-1} + F_{k-2}$<br><b>endfor</b> | $[n \geq 0]$ |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|

#### Algorithm 5.4

##### FibB

|                                                                                                                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Input</b> and <b>Output</b> same as for FIBA<br><b>Algorithm</b> FIBB<br><b>Function</b> $F(k)$<br><b>if</b> $k = 0$ or $1$ <b>then</b> $F \leftarrow 1$<br><b>else</b> $F \leftarrow F(k-1) + F(k-2)$<br><b>endif</b><br><b>endfunc</b><br>$F_n \leftarrow F(n)$ |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Solution** Let us use as our measure the number of assignments of values. Let  $A_n$  be the number of assignments for Algorithm FIBA, and  $B_n$  the number for FIBB. In both algorithms there are two initial cases:

$$A_0 = A_1 = 2, \quad B_0 = B_1 = 1.$$

So FIBB is more efficient for the initial cases. For  $n > 1$ , in Algorithm FIBA we simply make one more assignment for each pass through the loop. Thus

$$A_n = A_{n-1} + 1, \quad n > 1.$$

It follows that

$$A_n = n + 1, \quad n > 1.$$

In Algorithm FIBB, when  $n > 1$  we make the one assignment on the else-line *plus* all assignments incurred by calling the function for  $n-1$  and  $n-2$ . That is,

$$B_n = 1 + B_{n-1} + B_{n-2}, \quad n > 1. \quad (2)$$

Thus the recurrence for  $\{B_n\}$  is identical to that for  $\{F_n\}$ , the Fibonacci numbers themselves, *except* that the  $B$ -recurrence has an additional, nonhomogeneous constant 1.

We can solve Eq. (2) by our standard methods, but it's a mess by hand —  $\sqrt{5}$  appears many times, even more than in solving the ordinary Fibonacci recurrence (Example 4, Section 5.5). Fortunately, all we need is qualitative information: How do  $B_n$  and  $A_n = n+1$  compare? We can answer this question pretty much in our heads, by showing that  $B_n \geq F_n$ , for we know that  $F_n$  is approximately  $(1.618)^n$  and thus much larger than  $n+1$  for all but quite small  $n$ .

Why is  $B_n \geq F_n$ ? Because  $B_0 = F_0$ ,  $B_1 = F_1$ , but the  $B$ -recurrence has an additional positive term (the 1) on its right-hand side. It follows by induction [13] that  $B_n > F_n$  for  $n > 1$ . (To obtain a formula for  $B_n$  by hand without all the messy algebra alluded to, see [14, 15].)

Conclusion: Algorithm FIBB is lousy compared to FIBA. In fact, it's lousy period. ■

Why is this recursion so slow? Because the algorithm repeats itself flagrantly. For instance, suppose we use Algorithm FIBB to compute  $F(10)$ . It first must compute  $F(8)$  and  $F(9)$ . In computing  $F(8)$  it will first compute  $F(7)$ . Later, in computing  $F(9)$ , it will compute  $F(7)$  and  $F(8)$  again, both *from scratch*. The smaller  $k$  is, the more times FIBB recomputes  $F(k)$  as part of its main call.

In general, if a recursive function or procedure calls on itself more than once in its definition, the total amount of work to run it will be exponential. Sometimes this can't be avoided — every algorithm for Towers of Hanoi takes at least  $2^n - 1$  moves. But often it can be avoided: Don't use recursion (or, if you are a knowledgeable programmer, save values of  $F_n$  as you compute them, thereby avoiding making the same call more than once for any value of  $n$ ).

All this is too bad. Recursive algorithms are relatively easy to prove correct and to analyze for the number of steps. Iterative algorithms are not always so easy to analyze as Algorithm FIBA. Sometimes, therefore, it is useful to find a recursive algorithm, even if inefficient, as a first step. This will prove that a problem can be constructively solved and thus give you the fortitude to search further for a more efficient iterative algorithm. But for other problems, as we now discuss, recursive algorithms are surprisingly more efficient than the natural iterative algorithms.

## Divide and Conquer Algorithms

The D&C approach is a special case of the recursive paradigm. Instead of supposing that you already know how to do the immediately preceding case, or all preceding cases, you assume that you know how to do the cases approximately half the size of the current case (and, of course, a basis case). Recursive algorithms built on this assumption often run much faster than more straightforward algorithms. In this subsection we give several examples of the D&C approach and show you how to analyze their run time.

First, here are some sketches of D&C solutions.

#### EXAMPLE 4

Find if a word  $w$  is on an alphabetical list of  $n$  words.

**Solution** We already gave a D&C solution to this problem with Algorithm BINSEARCH in Section 1.5, even though the algorithm was written iteratively. Recall the idea: Check if  $w$  is the middle word. If not, and it comes earlier alphabetically, search for it on the first half of the list. (By recursion you already know how to do this.) If  $w$  is later alphabetically, search for it on the second half. Also recall the analysis: Whereas the more obvious sequential search took  $n$  comparisons in the worst case, BINSEARCH at worst took about  $\log n$  comparisons, a very substantial saving for large  $n$ . (We gave an intuitive argument for  $\log n$  in Section 1.5 and in [20–21, Section 2.5] we asked you to show by induction that  $1 + \log_2 n$  is an upper bound on the number of comparisons.) ■

#### EXAMPLE 5

Find the maximum of  $n$  numbers.

**Solution** Our previous solution, Algorithm MAXNUMBER in Section 1.5, was iterative, but a D&C solution is equally easy to devise. Suppose we already know how to find the maximum of a list of  $n/2$  numbers. We then divide the list of  $n$  in half, find the maximum of each recursively, and compare the two winners. To be more precise, that's what we do if  $n$  is even. If  $n$  is odd, we divide the list into lists of sizes  $\lfloor n/2 \rfloor$  and  $\lceil n/2 \rceil$ . This illustrates why we used the phrase “approximately half the size” in the first paragraph of this subsection. ■

#### EXAMPLE 6

Given a real number  $x$  and a positive integer  $n$ , compute  $x^n$ .

**Solution** The obvious approach is to start with  $x$  and keep multiplying by  $x$  a total of  $n - 1$  times; e.g.,  $x^3 = x \cdot x \cdot x$ , which involves two multiplications. But think recursively. If  $n$  is even, find  $x^{n/2}$  and do just one more multiplication to square it. If  $n$  is odd, find  $y = x^{\lfloor n/2 \rfloor}$  and then do just two more multiplications to compute  $x^n = y^2x$ . This recursion is easy to set up as a function in our algorithmic language (no annoying details about passing lists [37]). Here it is:

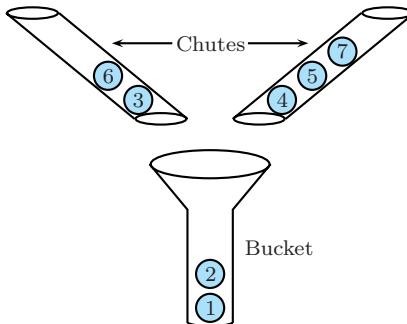
```
function P(x, n)                                [x real, n ∈ N+]
    if n = 1          then P ← x
    n/2 = ⌊ n/2 ⌋ then P ← (P(x, n/2))2
    n/2 ≠ ⌊ n/2 ⌋ then P ← x(P(x, ⌊ n/2 ⌋))2
    endif
endfunc ■
```

#### EXAMPLE 7

Arrange  $n$  numbers in increasing order.

**Solution** This sorting problem is a classic of computer science. It's not hard to outline a D&C approach: Divide the list in half and sort each half by recursion.

Combine half lists, maintaining the increasing order. The smallest number of all is the smaller of the smallest numbers in the two half lists. Having picked this number off, the next smallest of all is the smaller of the smallest numbers *remaining* in the half lists. Think of the half lists as numbers held in chutes (larger numbers higher up) above a narrow “bucket” (see Fig. 5.12). One at a time, the bottom numbers in the chutes are compared and the smaller one drops into the bucket. Numbers in the bucket are always in increasing order, so when all have dropped into the bucket the algorithm is finished. This method is known as **Merge Sort**. We show later in this section that this method takes around  $n \log n$  comparisons. The methods many people learn first (e.g., Insertion Sort or Bubble Sort; see Section 7.7) take  $n^2$ . ■



**FIGURE 5.12**

The merge phase of Merge Sort.

Now let's write down the difference equations for these D&C methods. For BINSEARCH, let  $A_n$  be the worst number of comparisons when there are  $n$  words. Then

$$A_n = 1 + \max\{A_{\lfloor \frac{n-1}{2} \rfloor}, A_{\lceil \frac{n-1}{2} \rceil}\}, \quad A_1 = 1. \quad (3)$$

The argument is: You compare the search word  $w$  with the middle word in any case (that gives the 1); in the worst case,  $w$  is on whichever side is itself worse. Since there are  $n - 1$  words left, and the search word is as close to the middle as possible, the two halves have size  $\lfloor \frac{n-1}{2} \rfloor$  and  $\lceil \frac{n-1}{2} \rceil$ . (Surely  $A_n$  should be an increasing function of  $n$ , in which case (3) simplifies to  $A_n = 1 + A_{\lfloor \frac{n-1}{2} \rfloor}$ , but at this stage we don't really know it's increasing and must stick with (3). But see [30].)

As for Example 5, finding the maximum takes just one more comparison after solving the half problems, but unlike for binary search both half problems must be solved. Also, the case where  $n = 1$  takes no work. So for  $B_n$  defined as the number of comparisons when there are  $n$  numbers, we get

$$B_n = 1 + B_{\lfloor \frac{n}{2} \rfloor} + B_{\lceil \frac{n}{2} \rceil}, \quad B_1 = 0. \quad (4)$$

Turning to the computation of  $x^n$ , let  $C_n$  be the number of multiplications to get  $x^n$ . Our explanations of the D&C method translates to

$$C_n = C_{\lfloor n/2 \rfloor} + t, \quad \text{where } t = \begin{cases} 1 & \text{if } n \text{ even} \\ 2 & \text{if } n \text{ odd} \end{cases} \quad \text{and} \quad C_1 = 0. \quad (5)$$

To recall where  $t = 1, 2$  and  $C_1 = 0$  come from, see the case-if in the algorithm display in Example 6.

Finally, for Merge Sort, the two subproblems are as close to  $n/2$  as possible and the number of comparisons in the merge phase is at most  $n - 1$ . (Why? How could it be less? [42]) Since it is not easy to nail down the number of comparisons in the merge phase more exactly, we settle for an upper bound on the worst case by solving the recurrence

$$D_n = D_{\lfloor n/2 \rfloor} + D_{\lceil n/2 \rceil} + n - 1, \quad D_1 = 0. \quad (6)$$

Fine and dandy. We have recurrences. Unfortunately, they are very elaborate — full of floors and ceilings and tiny little symbols you can hardly read. More to the point, they don't seem to be the sort that you have learned how to solve in this chapter. Fortunately, there is a simple procedure that converts all these recurrences to kinds that we do know how to solve. We introduce it with an example.

In this example, and for the rest of the section, logs appear often. They will always be base 2.

## EXAMPLE 8

Find a solution to Eq. (5) for computing  $x^n$ .

**Solution** We need to simplify the recurrence. Before treating the main culprits — the floors and ceilings — let's see if anything else is unwieldy. Yes,  $t$  varies between 1 and 2. Let's replace it with its greatest value, 2. That will cause our answer to err on the high side, but that's OK. In analysis of algorithms, erring on the low side is the thing to avoid.

As for those floors and ceilings, we would like them to go away. If  $n$  is divisible by 2, they do go away, and we get

$$C_n = C_{n/2} + 2, \quad C_1 = 0. \quad (7)$$

Now, here's the main idea. Suppose we could find a subsequence of indices  $\{n_k\}$  such that for each  $k$ ,  $n_{k-1} = n_k/2$  exactly. Then setting  $c_k = C_{n_k}$  would reduce (7) immediately to a first-order difference equation in  $c_k$ . But of course we do know such a subsequence:  $n_k = 2^k$ . Therefore Eq. (7) becomes

$$c_k = c_{k-1} + 2, \quad c_0 = 0.$$

This is so simple we can solve it by sight:

$$c_k = 2k.$$

Finally, substitute  $C$ 's and  $n$ 's back in. Since  $n = 2^k$ , then  $k = \log n$  and

$$C_{2^k} = 2k \quad \text{or} \quad C_n = 2 \log n. \blacksquare$$

*Warning.* We don't claim our answer is exactly correct, even as an upper bound. The argument presented shows only that  $2 \log n$  is an upper bound when  $n$  is a power of 2. In fact, whenever an answer involves logs, it *can't* be exactly correct for other  $n$  because then  $\log n$  is not even an integer!

Nonetheless, it is a pleasant fact that the formula obtained by this powers-of-2 method generally *is* correct, in one of the following two senses:

1. This formula, or a slight variant, really is correct for all  $n$  (either as an exact value or as an upper bound), but one needs to show this by a separate proof, usually a strong induction;
2. The formula, call it  $f_n$ , is within a constant multiple of the correct answer  $a_n$  for all  $n$  (that is,  $a_n = \text{Ord}(f_n)$ , as defined in Section 0.3), or it is within a constant multiple of a correct upper bound (that is,  $a_n = O(f_n)$ ), *assuming*  $a_n$  is an increasing function of  $n$ . This assumption is generally correct (since we are measuring the work done by an algorithm) but it needs a separate proof.

We illustrate both methods below, the first in Example 9 and the second in Example 10. The second method is more common in the literature, as its weaker claim is more often true and it is easier to use once you know how (see Example 11).

But before reading Example 10, please review the definitions of  $\text{Ord}$  and  $O$  from Section 0.3 (Definitions 1 and 3 there), since they will be used quite precisely.

## EXAMPLE 9

Verify that the upper bound on computing  $x^n$  obtained for  $n = 2^k$  in Example 8 is valid in general. That is, show that  $C_n \leq 2 \log n$  for *all* positive integers  $n$ .

**Solution** By strong induction, with  $P(n)$  the statement  $C_n \leq 2 \log n$ . The basis case is easy: We know that  $C_1 = 0$ , and sure enough,  $2 \log 1 = 0$ , too. As for the inductive step, since we are assuming the conclusion for all  $i < n$ , assume it in particular for  $\lfloor n/2 \rfloor$ . Then by Eq. (5)

$$\begin{aligned} C_n &\leq 2 \log \lfloor n/2 \rfloor + 2 = 2(\log \lfloor n/2 \rfloor + 1) \\ &\leq 2[\log(n/2) + 1] && [\lfloor n/2 \rfloor \leq n/2] \\ &= 2 \log n. \blacksquare && [1 = \log 2] \end{aligned}$$

## EXAMPLE 10

Show that the worst case number of comparisons for Merge Sort is  $O(n \log n)$ .

**Solution** First, to get fewer terms and less calculation, consider a related sequence by replacing  $n-1$  in Eq. (6) by  $n$ :

$$D'_n = D'_{\lfloor n/2 \rfloor} + D'_{\lceil n/2 \rceil} + n, \quad D'_1 = 0. \tag{8}$$

This means  $D'_n$  is only an upper bound, but that was true anyway of  $D_n$  in (6):  $n-1$  was a worst case for the merge phase already. (More formally, we are claiming that  $D'_n \geq D_n$  for all  $n \in N^+$ , which is easily proved by induction without finding a closed form for either sequence [32].) Now let  $n = 2^k$ , let  $d_k = D'_{2^k}$ , and rewrite (8) in terms of  $k$ :

$$d_k = 2d_{k-1} + 2^k, \quad d_0 = 0. \tag{9}$$

This one is hard to eyeball, but our nonhomogeneous methods apply. The right guess for a particular solution is  $Ak2^k$  (not  $A2^k$ , which is a solution of the associated homogeneous equation). Plugging this guess into Eq. (9) we find that  $A = 1$  [33]. So the general solution is  $B2^k + k2^k$ . Using the initial condition, we find that  $B = 0$ . Thus the particular solution to Eq. (9) is simply  $d_k = k2^k$ . Substituting  $n$  back in, we find that  $D'_n = (\log n)n$  for  $n = 2^k$ .

Now we get to a new sort of argument. Assuming that  $D'_n$  increases with  $n$  (proved by induction [31]), we will show that, for any  $n$ ,  $D'_n$  differs from  $n \log n$  by at most a multiplicative factor, so that  $D'_n = \text{Ord}(n \log n)$ . In other words, using Definition 1 in Section 0.3, we have to show that there are positive constants  $L$  and  $U$  such that

$$L \leq \frac{D'_n}{n \log n} \leq U, \quad n \in N^+. \quad (10)$$

We need only consider  $n \neq 2^k$ , since  $D'_n/(n \log n) = 1$  otherwise. To determine  $U$ , we must make  $D'_n$  as large as possible and  $n \log n$  as small as possible. Well, for any positive  $n$  that is not a power of 2, there is a unique integer  $k \geq 1$  such that  $2^k < n < 2^{k+1}$ . Since  $D'_n$  increases, for  $n$  in this interval we have  $D'_n \leq D'_{2^{k+1}} = 2^{k+1} \log 2^{k+1} = (k+1)2^{k+1}$ . Likewise, since  $n \log n$  increases, the smallest it can be in this interval is  $2^k \log 2^k = k2^k$ . Therefore

$$\frac{D'_n}{n \log n} \leq \frac{(k+1)2^{k+1}}{k2^k} = 2 \frac{k+1}{k} \leq 4,$$

since  $\frac{k+1}{k} \leq 2$  for  $k \geq 1$ . So any  $U \geq 4$  will do in (10). To get  $L$ , make  $D'_n$  as small as possible for this interval and  $n \log n$  as large as possible. We get

$$\frac{D'_n}{n \log n} \geq \frac{k2^k}{(k+1)2^{k+1}} = \frac{k}{2(k+1)} \geq \frac{1}{4}.$$

since  $\frac{k}{k+1} \geq \frac{1}{2}$  for  $k \geq 1$ . Thus any positive  $L \leq \frac{1}{4}$  will do. So  $D'_n = \text{Ord}(n \log n)$ .

Finally, recall that  $D'_n \geq D_n$ , and the recurrence for  $D_n$  may itself overstate the worst case of Merge Sort. Therefore, we conclude only that the worst case is  $O(n \log n)$ . ■

In fact, by an entirely different argument [45] one can show that *any* method of sorting  $n$  items by comparisons must take *at least*  $\text{Ord}(n \log n)$  comparisons in the worst case. Therefore, the worst case of Merge Sort can be no better than  $\text{Ord}(n \log n)$ . Therefore, the worst case of Merge Sort is precisely  $\text{Ord}(n \log n)$ .

The reasoning in Example 10 leads to the following general theorem about the solutions to D&C recurrences.

**Theorem 1.** Suppose  $\{A_n\}$  is an increasing sequence satisfying a recurrence which, for  $n$  any power of 2, simplifies to

$$A_n = bA_{n/2} + cn^d, \quad (11)$$

where  $b \geq 1$  and  $c > 0$ . Then for all  $n$ ,

$$A_n = \begin{cases} \text{Ord}(n^d) & \text{if } b < 2^d, \\ \text{Ord}(n^{\log b}) & \text{if } b > 2^d, \\ \text{Ord}(n^{\log b} \log n) & \text{if } b = 2^d. \end{cases}$$

Several comments are in order before we indicate the proof. First, this theorem may seem too good to be true; it claims a result for all  $n$  and yet requires nothing about the form of  $A_n$  except for powers of 2. However, it compensates by requiring that the sequence be increasing for all  $n$ . As we have seen, to prove that the sequence is increasing you usually need to do induction using the recurrence as defined for all  $n$ .

Second, what is this  $n^{\log b}$  term? We've seen algorithms that are  $\text{Ord}(n)$  or  $\text{Ord}(n^2)$ , but never weird powers of  $n$ . You will see in the proof that  $n^{\log b}$  is just the way to express the familiar function  $b^k$  when we convert from  $k$  back to  $n = 2^k$ . In any event, some very important algorithms are  $\text{Ord}(n^{\log b})$  for various  $b$  values. There was quite a stir in 1969 when an algorithm was announced for computing the product of two  $n \times n$  matrices with  $\text{Ord}(n^{2.81})$  multiplications instead of the usual  $n^3$ . This was achieved by a D&C algorithm and  $2.81 = \log 7$ . (See [29]. Since then even faster matrix multiplication algorithms have been found. The fastest currently is  $\text{Ord}(n^{2.376})$ .)

## EXAMPLE 11

Analyze  $D'_n$  from Eq. (8) using Theorem 1.

**Solution** When  $n = 2^k$  (indeed, whenever  $n$  is even), Eq. (8) becomes  $D'_n = 2D'_{n/2} + n$ , so in terms of the theorem,  $b = 2$  and  $c = d = 1$ . Since  $2 = 2^1$ , the third case applies, and we conclude (since  $D'_n$  is increasing) that  $D'_n = \text{Ord}(n^1 \log n)$ , the same conclusion we reached with a lot of work in Example 10. ■

**PROOF OF THEOREM 1** We prove only the case  $b = 2^d$ , which is the most complicated. (In all cases, the key method is the same as in Example 10.)

First, a little fact about logs that is used over and over in algorithm analysis: If  $k = \log n$ , then

$$\begin{aligned} b^k &= b^{\log n} = (2^{\log b})^{\log n} = (2^{\log n})^{\log b} \\ &= n^{\log b}. \end{aligned} \tag{12}$$

Now, we solve Eq. (11) exactly for  $n = 2^k$  as we have done before. Let  $a_k = A_{2^k}$  so that (11) becomes

$$\begin{aligned} a_k &= ba_{k-1} + c(2^k)^d \\ &= ba_{k-1} + c(2^d)^k \\ &= ba_{k-1} + cb^k, \end{aligned} \tag{13}$$

since  $b = 2^d$ . The right guess of a particular solution is  $Pkb^k$ , not  $Pb^k$ , since  $b^k$  is a solution of the associated homogeneous equation. You can verify that  $P = c$  [34]. Therefore the general solution to (13) is

$$a_k = (m+ck)b^k. \tag{14}$$

We can't compute the constant  $m$  because we haven't been given an initial condition, but it doesn't matter because the theorem assumes  $c > 0$  so the term  $ckb^k$  is

the dominant term in the order analysis. That is, there are positive constants  $L$  and  $U$  such that with  $L$  small enough (but positive) and  $U$  large enough

$$L kb^k \leq a_k \leq U kb^k. \quad (15)$$

Substituting  $n = 2^k$  back into (14–15) and using the log fact (12), we get, since  $A_n = A_{2^k} = a_k$ ,

$$A_n = (m + c \log n)n^{\log b} \quad \text{for } n = 2^k, \quad (16)$$

and

$$L n^{\log b} \log n \leq A_n \leq U n^{\log b} \log n \quad \text{for } n = 2^k. \quad (17)$$

For convenience, we now define

$$f(n) = n^{\log b} \log n.$$

Note that  $f(2^k) = kb^k$  (use the log fact in reverse [35]). The advantage of this notation is that inequality (17) can now be summarized as

$$A_n = \text{Ord}(f(n)) \quad \text{for } n = 2^k.$$

But the theorem claims that  $A_n = \text{Ord}(f(n))$  for all  $n \in N^+$ . Thus we need to show that  $A_n/f(n)$  is bounded above and below for all such  $n$ .

As in Example 10, for  $n$  not a power of 2 there is a  $k$  such that  $2^k < n < 2^{k+1}$ . Because the theorem assumes that  $A_n$  is increasing, and because  $f(n)$  is certainly increasing (why?),

$$\frac{A_n}{f(n)} \leq \frac{A_{2^{k+1}}}{f(2^k)} \leq \frac{U f(2^{k+1})}{f(2^k)} \leq \frac{Ub^{k+1}(k+1)}{b^k k} \leq Ub \frac{k+1}{k} \leq 2Ub.$$

Similarly,

$$\frac{A_n}{f(n)} \geq \frac{A_{2^k}}{f(2^{k+1})} \geq \frac{Lf(2^k)}{f(2^{k+1})} \geq \frac{Lb^k k}{b^{k+1}(k+1)} \geq \frac{L}{b} \frac{k}{k+1} \geq \frac{L}{2b}.$$

Since these bounds work for all  $n \in N^+$ ,  $A_n = \text{Ord}(f(n))$ , proving the case  $b = 2^d$ . ■

One often gets the impression that this theorem completely answers all questions about solving D&C recurrences, but that's not true:

1. As already noted, you have to prove the hypothesis that the sequence is increasing.
2. The theorem only gives the Ord of the sequence. Other methods can sometimes give more information, even an exact formula.
3. The theorem only applies directly if, when the recurrence that holds for all  $n$  is simplified for  $n = 2^k$ , the only terms of the sequence that remain are  $A_n$  and  $A_{n/2}$ .

The condition in Item 3 doesn't always hold. For instance, when  $n = 2^k$  the recurrence (3) for BINSEARCH becomes

$$A_n = 1 + \max\{A_{\frac{n}{2}-1}, A_{\frac{n}{2}}\};$$

Theorem 1 does not apply to this. Fortunately there is a workaround: Apply the theorem instead to the sequence  $\{A'_n\}$  that satisfies

$$A'_n = 1 + A'_{n/2}$$

because  $A'_n \geq A_n$ . But for other D&C recurrences it may not be so obvious how to do workarounds [50].

To be fair, Item 3 above is a stumbling block as well for the method of substituting  $2^k$  (Example 8), but we feel that method offers more possibilities for workaround [41]. In any event, we urge you in the problems to use the substitution method instead of just quoting Theorem 1, at least until you get a good feel for what's going on. But truth be told, most practitioners simply replace anything close to  $n/2$  with  $n/2$  and use the theorem. Most of the time the conclusion you obtain is right, and there's much less work.

## Problems: Section 5.8

*Note:* All logs in this problem set are base 2 unless a different explicit base is given.

1.  $\langle 2 \rangle$  Compute the number of additions in the two polynomial evaluation algorithms, STRAIGHTFORWARD and HORNER

- a) Counting just those additions caused by a plus sign.
- b) Also counting subtractions and all increments and decrements of counters (which amount to adding  $\pm 1$ ).

Either way, your answer should confirm the claim in the text that both algorithms are  $\text{Ord}(n)$  for additions.

2.  $\langle 1 \rangle$  Count the number of assignments of variables in STRAIGHTFORWARD and HORNER. Consider each quantity in the Input to be one assignment.
3.  $\langle 3 \rangle$  It seems that HORNER does much better than STRAIGHTFORWARD — the former is as good as the latter on assignments and additions and does much better on multiplications. Why, then, do we use STRAIGHTFORWARD in practice? “That’s what our silly teachers taught us” is not a good enough answer.
4.  $\langle 3 \rangle$  Prove that HORNER is correct, that is, it really does output the value of the polynomial at  $x$ .
5.  $\langle 2 \rangle$  Consider Algorithm 5.5, POWERPOLY, which evaluates polynomials and attempts to save work

on evaluating  $x^k$  by computing lower powers first and saving the results. Compare this algorithm for efficiency with the two in Example 1.

### Algorithm 5.5. POWERPOLY

**Input** and **Output** same as in Example 1

#### Algorithm POWERPOLY

```
P ← x
S ← a0 + a1x
for k = 2 to n
    P ← Px
    S ← S + akP
endfor
```

6.  $\langle 3 \rangle$  Algorithms can be implemented on calculators as well as on computers, but the appropriate measures of efficiency are different. For calculators the obvious measure is the number of button pushes. For your favorite calculator, compute the number of button pushes for Algorithms
  - a) STRAIGHTFORWARD
  - b) HORNER
  - c) POWERPOLY from [5].Since calculators differ, you must describe in words or “button pictures” how your calculator would handle these algorithms, as well as show your counting calculations. Also, the number of pushes depends on the number of digits in your input data. For simplicity, assume each input quantity

requires  $P$  button pushes each time you punch it in.

7.  $\langle 1 \rangle$  Prove that Procedures TOH and TOH\* perform the same sequence of moves, given the same input  $(n, i, j)$ .
8.  $\langle 1 \rangle$  Measure the relative efficiency of Procedures TOH and TOH\* by the number of procedure calls needed to run them with  $n$  rings. For instance, running TOH( $n, i, j$ ) requires the initial call plus two subcalls with  $n-1$  rings, plus whatever calls those subcalls make.
9.  $\langle 2 \rangle$  Explain intuitively why Procedure TOH requires exactly twice as many comparisons of  $n$  and 1 as Procedure TOH\*.
10.  $\langle 2 \rangle$  Compare Algorithms FIBA and FIBB for the Fibonacci numbers (Example 3) on the basis of how many additions they each require.
11.  $\langle 3 \rangle$  Compare FIBA and FIBB on the basis of the number of additions and subtractions they perform. In FIBB, when function  $F(k)$  calls  $F(k-1)$ , a subtraction is performed in figuring out the calling value. Similarly, subtractions are performed in FIBA to evaluate subscripts.
12.  $\langle 4 \rangle$  Consider the recursive function  $F(k)$  in FIBB. Suppose that it is modified so that the first time  $F$  is called for any particular  $k$  it stores the answer, and for each later call it simply looks up that answer instead of computing it again.
- a) Make a tree of the calls when the main call is  $F(4)$ ; when it is  $F(6)$ .
- b) Actually, there are two trees in each case, depending on whether, on the line
- $$F \leftarrow F(k-1) + F(k-2),$$
- Algorithm FIBB first calls  $F(k-1)$  or first calls  $F(k-2)$ . Whatever assumption you made in part a) about the order of calls, now draw the trees for the other assumption. (Represent the first call by the edge to the left child.)
- c) Count the total number of calls in each tree and make a conjecture. Prove it.
13.  $\langle 2 \rangle$  Prove that  $B_n > F_n$  for all  $n > 1$ , where  $B_n$  is as in Eq. (2). Use induction and the recursive definitions of  $B_n$  and  $F_n$ . Do not use closed-form formulas.
14.  $\langle 3 \rangle$  Solve for  $B_n$  in Eq. (2) by the general method of Section 5.7. It's actually not so much work if
- you make use of work already done in Example 4, Section 5.5. At some point you will have to solve equations of the form
- $$Cg^0 + D\bar{g}^0 = e_1,$$
- $$Cg^1 + D\bar{g}^1 = e_2,$$
- for  $C$  and  $D$ , where
- $$g = (1 + \sqrt{5})/2, \quad \bar{g} = (1 - \sqrt{5})/2,$$
- and  $e_1, e_2$  will be known constants. If you now compare your equations with the similar equations solved in Example 4, Section 5.5, you should be able to simply write down the answer.
15.  $\langle 2 \rangle$  Here is another trick for solving for  $B_n$  in (2). Note that
- $$B_n = 1 + B_{n-1} + B_{n-2}$$
- may be rewritten as
- $$(B_n + 1) = (B_{n-1} + 1) + (B_{n-2} + 1).$$
- Now set  $G_n = B_n + 1$ . What are the recurrence and initial condition for  $G_n$ ? Solve for  $B_n$  by solving for  $G_n$ .
16.  $\langle 3 \rangle$
- a) Prove
- $$a^{\log_b c} = c^{\log_b a}.$$
- b) Use a) to give another proof that if  $n = 2^k$ , then  $b^k = n^{\log_b b}$ .
- Problems 17–29 present D&C recurrences. Solve them either by the “temporarily substitute  $2^k$ ” method (as in Example 8) or by using Theorem 1, but we think you will learn more if you use the first method. Besides, some of these problems don’t fit the theorem. You may replace anything close to  $n/2$  by  $n/2$  but keep in mind that the validity of the result is therefore sometimes in question. You need not determine the validity, though we hope you will think about it for each problem you do.
17.  $\langle 1 \rangle$   $A_n = A_{\lfloor n/2 \rfloor} + A_{\lceil n/2 \rceil}, \quad A_1 = 1$
18.  $\langle 1 \rangle$   $A_n = A_{\lfloor n/2 \rfloor} + A_{\lceil n/2 \rceil} + 1, \quad A_1 = 1$
19.  $\langle 2 \rangle$   $B_n = 3B_{n/2}, \quad B_1 = 1 \quad (\text{See [16].})$
20.  $\langle 2 \rangle$   $C_n = C_{\lfloor (n-1)/2 \rfloor} + C_{\lfloor n/2 \rfloor} + C_{\lceil n/2 \rceil} + C_{\lceil (n+1)/2 \rceil}, \quad C_1 = 1$
21.  $\langle 2 \rangle$   $D_n = 3D_{n/2} + 2, \quad D_1 = 1$
22.  $\langle 2 \rangle$   $D_n = 3D_{n/2} + n, \quad D_1 = 1$
23.  $\langle 3 \rangle$   $F_n = 4F_{n/2} + n^2, \quad F_1 = 1$

24.  $\langle 2 \rangle G_n = G_{\lfloor n/2 \rfloor} + G_{\lceil n/2 \rceil} + n, \quad G_1 = 1$
25.  $\langle 3 \rangle H_n = 5H_{n/2} + \log n, \quad H_1 = 1$
26.  $\langle 3 \rangle I_n = I_{\lfloor (n-1)/2 \rfloor} + I_{\lceil (n-1)/2 \rceil} + n \log n, \quad I_1 = 1$
27.  $\langle 3 \rangle J_n = 3J_{n/2} + n \log n, \quad J_1 = 1$
28.  $\langle 2 \rangle K_n = 2K_{n/2} + \frac{1}{n}, \quad K_1 = 1$
29.  $\langle 2 \rangle L_n = 7L_{n/2} + (4.5)n^2, \quad L_1 = 1$ . This recurrence may seem completely cooked up, but it's not. It counts the number of operations in a D&C method for multiplying  $n \times n$  matrices announced in 1969 by Strassen. Show that it beats the order  $n^3$  standard algorithm. (See also [7, Final Problems] and Example 4, Section 4.2.)
30.  $\langle 3 \rangle$  Prove that the sequence  $A_n$  defined by Eq. (3) is increasing, so that the simpler recurrence mentioned in the paragraph after the equation may be used.
31.  $\langle 3 \rangle$  Prove that the sequence  $\{D'_n\}$  defined by Eq. (8) is increasing.
32.  $\langle 2 \rangle$  Prove that  $D'_n \geq D_n$  for all  $n \in N^+$  if  $D_n$  satisfies (6) and  $D'_n$  satisfies (8).
33.  $\langle 2 \rangle$  In the solution to Example 10, verify the claims that  $A = 1$  and  $B = 0$ , needed in finding the solution to Eq. (9).
34.  $\langle 2 \rangle$  Verify the claim that  $P = c$  in the solution of recurrence (13) in the proof of Theorem 1.
35.  $\langle 1 \rangle$  Verify the claim that  $f(2^k) = kb^k$  in the proof of Theorem 1.
36.  $\langle 2 \rangle$  Verify that function  $P(x, n)$  in Example 6 correctly computes  $x^n$  for  $x$  real and  $n \in N^+$ .
37.  $\langle 3 \rangle$  Use our algorithmic language to write up the D&C recurrence discussed in
  - Example 4. *Suggestion:* For the recursive procedure, use as parameters the index of the first and last word in the subsequence to be searched. Make the list entries Inputs and thus global.
  - Example 5.
  - Example 7.
38.  $\langle 4 \rangle$  If  $n$  is input in binary form, then our D&C algorithm to compute  $x^n$  can be reformulated iteratively with a single loop. Do so. By "input in binary" we mean that you are given  $n$  as a sequence of bits  $a_m, a_{m-1}, \dots, a_0$ .
39.  $\langle 2 \rangle$  Consider Eq. (4) for the D&C maximum algorithm.
40.  $\langle 2 \rangle$  Solve Eq. (3) for Algorithm BINSEARCH by the method of replacing everything close to  $n/2$  by  $n/2$  and then substituting  $n = 2^k$ . You will *not* get an exact answer for  $A_n$ , the worst case; for that see [22, Section 2.5]. Verify that your answer is nonetheless a valid upper bound for  $A_n$  for all  $n$ .
41.  $\langle 4 \rangle$  Problem [40] shows that replacing everything near  $n/2$  with  $n/2$  doesn't give the best possible result for the BINSEARCH recurrence. Here is a variant method that does better. Find a subsequence  $n_k$  of  $n$  values (different from  $n_k = 2^k$ ) for which all the floors and ceilings in Eq. (3) do vanish exactly. That is, we want a subsequence  $\{n_k\}$  where, for each  $k$ ,  $n_{k-1} = (n_k - 1)/2$ , for then Eq. (3) reduces exactly to a first-order recurrence.
  - In our subsequence, we also want  $n_1 = 1$ . Why?
  - You know such a sequence! What is it?
  - Now define  $a_n = A_{n_k}$  for your sequence  $\{n_k\}$  and solve the BINSEARCH recurrence exactly for those values of  $n$ . If you did [22, Section 2.5], compare with your answer there.
42.  $\langle 2 \rangle$  If ordered lists of size  $p$  and  $q$  are merged as in Merge Sort, what is the least number of comparisons necessary before the order of the entire list is known? Give an example with  $p = 2$  and  $q = 4$  to show that this result could actually happen.
43.  $\langle 4 \rangle$  Solve the Merge Sort recurrence Eq. (6) for powers of 2 without simplifying  $n - 1$  to  $n$ . You can't use Theorem 1 (why not?), but the other method works — find an exact formula for  $n = 2^k$  — though the algebra is more complicated than with the simplification. (It happens that the answer you get is *not* correct as an upper bound for  $n \neq 2^k$ ; check it out by hand for  $n = 3$ .)
44.  $\langle 3 \rangle$  We said that the original Merge Sort recurrence, Eq. (6) for  $D_n$ , might itself be an upper bound, because  $n - 1$  is an upper bound, not always attained, on the number of comparisons to merge the two half lists. But in fact, there do exist lists for which, in every subcall of Merge Sort,  $n - 1$  comparisons are needed (where  $n$  is the number of items in the list obtained at the end of the call).

Therefore, there exist lists for which Merge Sort takes exactly  $D_n$  steps.

Find such a list with  $n = 8$  and show that you are right.

45. ⟨2⟩ Show by the decision tree analysis in [4, Section 4.3] that any method of sorting by 2-way comparisons takes at least  $\log n!$  comparisons. (*Hint:* How many cases must every method distinguish when there are  $n$  inputs?) It then follows by [18, Section 0.3] that sorting by comparison has worst case complexity at least  $\text{Ord}(n \log n)$ .

46. ⟨2⟩ Consider Eq. (5) for finding  $x^n$ . By replacing  $t$  in that equation with 1, find a lower bound on  $C_n$ . Verify that this bound is correct for all  $n$ .

47. ⟨3⟩ Where in the proof of Theorem 1 did we use that  $b \geq 1$ ?

48. ⟨5⟩ Prove the case  $b < 2^d$  in Theorem 1.

49. ⟨5⟩ Prove the case  $b > 2^d$  in Theorem 1.

50. ⟨4⟩ Figure out how to apply Theorem 1 to

$$B_n = 1 + B_{\lceil \frac{n+1}{2} \rceil}, \quad B_1 = 1.$$

And why can't you just replace it by

$$B'_n = 1 + B'_{n/2}, \quad B'_1 = 1?$$

51. ⟨4⟩ The proof of Theorem 1 is usually presented in texts by a dot-dot-dot expansion rather by the sort of substitution we have used. To limit the mess, consider the case  $A_n = bA_{n/2} + n$ . The argument goes:

$$\begin{aligned} A_n &= bA_{n/2} + n \\ &= b(bA_{n/4} + \frac{n}{2}) + n = b^2A_{n/4} + n(1 + \frac{b}{2}) \\ &= b^2(bA_{n/8} + \frac{n}{4}) + n(1 + \frac{b}{2}) = \dots \\ &= \dots \end{aligned}$$

Next, if  $n = 2^k$ , then the expansion using  $A_{n/2}, A_{n/4}, \dots$  eventually gets down to  $A_1$ , which is a constant, and at that point all the terms can be grouped and from that one gets the appropriate Ord statement.

Finish carrying out this plan for  $A_n = bA_{n/2} + n$ . There will be three cases:  $b < 2$ ,  $b = 2$ , and  $b > 2$ .

52. ⟨5⟩ Consider the problem of finding both the maximum and the minimum of  $n$  numbers.

- a) Devise an iterative solution, based on Algorithm MAXNUMBER from Section 1.6.

- b) Devise a D&C solution. *Suggestion:* You'll get a better algorithm if you have two base cases,  $n = 1$  and  $n = 2$ .

- c) Determine and compare the efficiency of the two approaches.

53. ⟨3⟩ The **median** of a set of numbers is the number such that half (or half  $\pm 1$ ) of the other numbers are smaller. Is D&C a good approach for finding medians? Explain.

54. ⟨3⟩ Although dividing in two is usually very good, it is not always best. Sometimes dividing in three is better.

- a) Determine the exact number of multiplications to obtain  $x^{27}$  by the method of Example 6. (The upper bound solution we obtained is of no use here. Instead, actually carry out the recursion.)

- b) Figure out a 3-way division recurrence for  $x^{27}$  and count the number of multiplications.

55. ⟨3⟩ Theorem. Any algorithm that correctly finds the maximum of  $n$  numbers, using comparisons of two of the numbers at a time, must use at least  $n-1$  comparisons. Proof: Let  $G$  be a graph whose vertices are the numbers and which has an edge between two numbers if they get compared at any point in the algorithm. We claim that if the algorithm is correct, the graph must be connected. Why? How does this prove the theorem?

56. ⟨3⟩ To our knowledge, there is still no known formula or efficient algorithm for the *exact* minimum number of multiplications to find  $x^n$  for arbitrary  $n$ . Call this minimum number  $M(n)$ .

- a) Show that  $M(2^k) = k$ .

- b) Show that  $\log n \leq M(n) \leq 2 \log n$  for all  $n$ .

## 5.9 Variable Coefficients, Sums, and Recent Advances in Computer Algebra

---

In the next two sections, we go beyond constant coefficient linear difference equations. In this section, we allow coefficients that vary. In Section 5.10 we treat nonlinear recurrences.

We begin with first-order variable coefficient linear recurrences, which have the form

$$a_n = c_n a_{n-1} + d_n. \quad (1)$$

where  $c_n$  and  $d_n$  can both be functions of  $n$ . Let's try to use the inductive paradigm to see if we can find a general solution of (1). Using (1) to write out the first few terms of  $\{a_n\}$ , we find

$$\begin{aligned} a_0 &= a_0, \\ a_1 &= c_1 a_0 + d_1, \\ a_2 &= c_2 a_1 + d_2 = c_2(c_1 a_0 + d_1) + d_2 \\ &= c_2 c_1 a_0 + c_2 d_1 + d_2, \\ a_3 &= c_3 a_2 + d_3 = c_3(c_2 c_1 a_0 + c_2 d_1 + d_2) + d_3 \\ &= c_3 c_2 c_1 a_0 + c_3 c_2 d_1 + c_3 d_2 + d_3. \end{aligned} \quad (2)$$

We have arranged the terms in the final expression for each  $a_n$  to make the pattern clearer. Do you see it? All terms except the first and the last in the expansion of each  $a_n$  have the form  $(\prod_{i=j+1}^n c_i) d_j$  for some  $j$ . Even the first term for each  $a_n$  is of this form if we set  $d_0 = a_0$ , which we can do since only  $d_1, d_2, \dots$  have been defined. And also the last term of  $a_n$  is of this form if we note that  $\prod_{i=n+1}^n c_i = 1$  since this is an empty product, which we have defined in Section 0.4 to be 1. Generalizing from this pattern, we infer that

$$a_n = \sum_{j=0}^n \left( \prod_{i=j+1}^n c_i \right) d_j. \quad (3)$$

Of course, we need a *proof* that (3) is correct — by induction of course [19]. For easy reference, we restate this result as a theorem

---

**Theorem 1.** If the sequence  $\{a_n\}$  satisfies

$$a_n = c_n a_{n-1} + d_n, \quad n \geq 1; \quad a_0 = d_0,$$

then

$$a_n = \sum_{j=0}^n \left( \prod_{i=j+1}^n c_i \right) d_j. \quad (4)$$

---

Eq. (4) is a notational triumph, but it is only a formula of sorts, because it is not a closed form as we usually understand it: for one thing, the number of terms in the sum varies with  $n$ , and closed forms usually have a fixed number of terms. So, does Eq. (4) tell us anything? The answer is: sometimes. The following examples show why.

### EXAMPLE 1

Apply Theorem 1 to the general arithmetic sequence,  $a_n = a_{n-1} + d$ . Do we get the explicit formula for the  $n$ th term?

**Solution** For  $n \geq 1$ , substitute 1 for  $c_n$  and  $d$  for  $d_n$  in Eq. (4). Recall that  $d_0 = a_0$  in every use of Eq. (4) by definition. We find that the inside product on the right in Eq. (4) is always 1 and thus

$$a_n = \sum_{j=0}^n d_j = a_0 + \sum_{j=1}^n d = a_0 + nd,$$

the explicit formula we wanted. Of course, we hardly needed Theorem 1 for this simple problem. ■

### EXAMPLE 2

Apply Theorem 1 to the problem of expressing the general sum  $a_n = \sum_{k=1}^n d_k$ .

**Solution** We know that  $a_n = a_{n-1} + d_n$ , so set  $c_k = 1$  for all  $k$  and  $a_0 = 0$ . From the theorem we obtain

$$a_n = a_0 + \sum_{j=1}^n d_j = \sum_{j=1}^n d_j.$$

This just gets us back to where we started. If  $d_k$  is especially nice, the variable length sum may collapse to a closed form (e.g., if  $d_j = d$ , then the sum is  $nd$ ), but in such cases Theorem 1 generally doesn't tell us anything we wouldn't have found without it. ■

### EXAMPLE 3

Apply Theorem 1 to the problem of expressing the general product  $a_n = \prod_{i=1}^n b_i$ .

The right choices for the  $c$ 's and  $d$ 's are:

$$c_n = b_n, \quad \text{and} \quad d_n = 0, \quad \text{for } n \geq 1; \quad d_0 = a_0 = 1.$$

Sure enough, plugging into Theorem 1 gives

$$a_n = \prod_{j=1}^n b_j,$$

but no more. ■

### EXAMPLE 4

Apply Theorem 1 to the general arithmetic-geometric sequence defined in [3, Section 5.2] by  $a_n = ca_{n-1} + d$ .

**Solution** In Eq. (4) set  $d_j = d$  for  $n > 0$  and  $d_0 = a_0$ . Then

$$\begin{aligned} a_n &= \sum_{j=0}^n \left( \prod_{i=j+1}^n c \right) d_j = \left( \prod_{i=1}^n c \right) a_0 + \sum_{j=1}^n \left( \prod_{i=j+1}^n c \right) d \\ &= a_0 c^n + \sum_{j=1}^n c^{n-j} d = a_0 c^n + d \left( \sum_{k=0}^{n-1} c^k \right). \end{aligned}$$

Using our knowledge of geometric series on the last parenthesized expression and assuming  $c \neq 1$  (the case  $c = 1$  is handled by Example 1), we obtain

$$a_n = a_0 c^n + d \left( \frac{c^n - 1}{c - 1} \right) = c^n \left( a_0 + \frac{d}{c - 1} \right) - \frac{d}{c - 1}. \quad (5)$$

Eq. (5) is a new formula. We could also obtain it by the method of Section 5.7; by that method we know that  $a_n = ca_{n-1} + d$  must have its solution in the form  $Ac^n + B$ . However, that method requires solving for  $A$  and  $B$  as a second stage. Here their values fall out as specific expressions involving  $a_0$ . Thus Theorem 1 does do some good. ■

## EXAMPLE 5

Apply Theorem 1 to

$$a_n = na_{n-1} + n!, \quad a_0 = 1.$$

This recurrence cannot be treated systematically by our previous methods. (Of course, you might be able just to look at the equation, guess the solution and then prove your guess correct, either by induction or by checking that it meets the initial condition and recurrence and is thus the only solution.)

**Solution** Setting  $c_n = n$  and  $d_n = n!$  in Theorem 1, we get

$$\begin{aligned} a_n &= \sum_{j=0}^n \left( \prod_{i=j+1}^n i \right) j! = \sum_{j=0}^n \left( \frac{n!}{j!} \right) j! \\ &= \sum_{j=0}^n n! = (n+1)n! \quad [n! \text{ is a constant relative to } j] \\ &= (n+1)!. \end{aligned}$$

So in this example Theorem 1 actually finds the solution. ■

## Higher Order Variable Coefficient Linear Recurrences

So Theorem 1 sometimes finds a closed form solution and sometimes does not. Indeed, that's the general situation with variable coefficient  $k$ th-order linear difference equations, which may be written

$$\sum_{i=0}^k c_{i,n} a_{n-i} = d_n. \quad (6)$$

Only sometimes is there a closed form solution. But when? Until 1991 the answer was not known. If you couldn't find a closed form solution to your specific difference

equation, maybe that just meant you weren't smart enough. But then the Slovenian mathematician Petkovšek made a great breakthrough. For the case when the coefficients  $c_{0,n}, c_{1,n}, \dots, c_{k,n}$  are *polynomials* in  $n$ , he *completely determined* when (6) has a closed form. Specifically, he developed and proved correct an *algorithm*, called HYPER, which, given any specific instance of (6) with polynomial coefficients, either returns a closed form solution or announces that none exists.

Interestingly, HYPER's main role has not been for solving difference equations per se, but rather as the final piece in a related development: the automation of finding and proving combinatorial identities. We take two paragraphs to introduce this related subject, and then show how HYPER fits in.

From research culminating around 1990, we now know that vast classes of combinatorial identities, particularly involving sums, have a two-step proof that can be automated:

1. Find a linear recurrence (with polynomial coefficients) satisfied by the complicated side of the identity (e.g., the sum); then
2. Check that the other side satisfies the same recurrence and initial conditions, and thus conclude that the two sides are equal by Theorem 1 of Section 5.4.

The key result that makes this method work was a proof by algorithm that large classes of combinatorial sums *do* satisfy linear recurrences.

Here is a simple illustration. Suppose we seek a closed form for  $S_n = \sum_{k=0}^n \binom{n}{k}$ . Imagine we have conjectured the identity  $\sum_{k=0}^n \binom{n}{k} = 2^n$  but haven't proved it yet. We feed the sum on the LHS of this conjecture into the recurrence-finding algorithm and back comes

$$S_{n+1} = 2S_n. \tag{7}$$

Great, we say, because it's easy to check that the RHS of our conjecture,  $2^n$ , also satisfies this recurrence and has the same initial condition,  $S_0 = 1$ . Therefore, our conjecture is correct.

So where does HYPER come in? The theory above works fine if you already *have* a conjecture for the simpler side of the equation ( $2^n$  in our case). What if you don't? Well, feed the recurrence (Eq. (7) in our case) into HYPER. Either it will spit back a closed form, or it will announce that there is none. In the first case HYPER has discovered the identity for you; in the latter it has told you that there is no closed-form identity for the sum you started with.

The last several paragraphs barely scratch the surface of the rapidly developing field of automated theorem proving in discrete math. Nor do they make precise exactly what has been automated. For one thing, before one can prove what HYPER does, one needs a precise definition of "closed form", and we haven't given one. Likewise, before one can prove that "large classes" of sums satisfy recurrences, one has to define those classes precisely. It turns out that both definitions hinge on the same key concept, *hypergeometric term*. We define this concept (in the simplest, one-variable case) in [22]. But you don't have to stop there. The best place to find out more about this whole subject is [www.cis.upenn.edu/~wilf/AeqB.html](http://www.cis.upenn.edu/~wilf/AeqB.html). For instance, from there you can link to the entire book *A = B*, which is the

main exposition of this theory, written by the three people who created most of it, Petkovšek, Wilf, and Zeilberger. From the website you can also download HYPER for free (if it is not yet implemented in your favorite CAS), along with other software.

## Problems: Section 5.9

Solve all of the first-order difference equations that follow by the method of this section, even though some are easier to do by other means.

1.  $\langle 1 \rangle b_n = 2b_{n-1} + 2^n, \quad b_0 = 1$

2.  $\langle 1 \rangle c_n = 3c_{n-1}, \quad c_0 = 1$

3.  $\langle 1 \rangle d_n = nd_{n-1}, \quad d_0 = 1$

4.  $\langle 1 \rangle f_n = [(n-1)/n] f_{n-1}, \quad f_0 = 1$

5.  $\langle 1 \rangle g_n = [(n+1)/n] g_{n-1}, \quad g_0 = 1$

6.  $\langle 3 \rangle r_n = [(n-1)/n] r_{n-1} + n + 1, \quad r_0 = 1$

7.  $\langle 2 \rangle h_n = nh_{n-1} - n!, \quad h_0 = 1$

8.  $\langle 2 \rangle m_n = 2nm_{n-1} + n!2^n, \quad m_0 = 1$

9.  $\langle 2 \rangle$  Solve  $a_n = c_n a_{n-1} + d_n$  if

a)  $c_n = c^n, d_n = 0, a_0 = 1$

b)  $c_n = c^n, d_n = a_0 = 1$

10.  $\langle 2 \rangle$  Solve  $t_n = \frac{1}{n}t_{n-1} + \frac{1}{n!}, \quad t_0 = 1$ .

11.  $\langle 3 \rangle$  Solve

$$S_1 = 1;$$

$$S_n = \frac{n-1}{n}S_{n-1} - \frac{1}{n^2(n-1)}, \quad n \geq 2.$$

Note that this sequence starts with  $n = 1$ , not  $n = 0$ , so you have to adjust Theorem 1 accordingly.

12.  $\langle 1 \rangle$  Solve for the average time of sequential search (Eq. (3) of Section 5.2) using the method of this section. Note that  $E_0 = 0$ .

13.  $\langle 1 \rangle$  Solve the TOH recurrence by the method of this section.

14.  $\langle 1 \rangle$  Solve the Straightline TOH recurrence (from [5, Section 2.4]) by the method of this section.

15.  $\langle 2 \rangle$  Obtain Eq. (5) by the theory of Section 5.7.

16.  $\langle 3 \rangle$  In [21, Section 5.2] we asked you to come up with a difference equation for both an IRA and an ordinary interest-bearing account if you deposit principal  $P$  each year. (For the IRA, it was assumed you could defer taxes on interest only.) At

that point you couldn't solve the equations to obtain explicit formulas for the final values of the accounts, but now you can.

a) Do so.

b) Using your formulas in a), find the numerical value of the amounts at withdrawal in both the ordinary and IRA accounts if  $r = .1$ ,  $t = .25$ ,  $P = \$2000$ , and  $N = 30$ .

17.  $\langle 2 \rangle$  Find and simplify a closed-form formula for  $a_n$ , if  $a_n = ra_{n-1} + s^n$  and  $a_0 = 1$ , where  $r$  and  $s$  are constants and

a)  $r \neq s$

b)  $r = s$ .

18.  $\langle 4 \rangle$  Solve the following problems from Section 5.3 by the method of the current section.

a) [22]

b) [23]

19.  $\langle 3 \rangle$  Prove Theorem 1.

20.  $\langle 3 \rangle$  The  $n$ th partial sum of the general geometric series,  $\sum_{k=0}^n ar^k$ , is also the  $n$ th term of a certain arithmetic-geometric sequence. Describe that sequence recursively, find a closed form for the  $n$ th term using the methods of this section, and verify that this closed form is equivalent to the usual formula for the sum of a geometric series.

21.  $\langle 2 \rangle$  Here is a trick for solving  $a_n = ca_{n-1} + d$ , ( $c \neq 1$ ), without using the theory of this section or Section 5.7. Namely, there is a constant  $k$  (you find it), so that, after adding  $k$  to both sides and rewriting the RHS, the given equation becomes

$$a_n + k = c(a_{n-1} + k).$$

Now define  $b_n = a_n + k$  and rewrite the recurrence in terms of  $b$ 's. A closed-form formula for  $b_n$  in terms of  $b_0$  should now be clear. Work backwards to a formula for  $a_n$ . Check that the formula is equivalent to the formula obtained in Example 4.

22.  $\langle 2 \rangle$  A **hypergeometric term** (in one variable) is any term in a sequence  $\{h_n\}$  that satisfies a first-order recurrence  $h_{n+1} = r(n)h_n$  where  $r(n)$  is a rational function. For instance,  $n!$  is a hypergeometric term because  $n!$  satisfies  $h_{n+1} = \left(\frac{n+1}{n}\right)h_n$ .

Also,  $2^n$  is a hypergeometric term, because  $2^n$  satisfies  $k_{n+1} = 2k_n$ ; 2 is a pretty simple rational function of  $n$ .

Look again at Eq. (4). Suppose  $c_n$  and  $d_n$  are rational functions of  $n$ . Explain why each product  $(\prod_{i=j+1}^n c_i)d_j$  in Eq. (4) is a hypergeometric term.

- 23.**  $\langle 2 \rangle$  In the text, when we discussed the automated discovery of a closed form for  $S_n = \sum_{k=0}^n \binom{n}{k}$ , we said that the method first finds a recurrence for  $S_n$ . Actually, it first finds a recurrence for the summand, here  $\binom{n}{k}$ . For this summand, it finds, not surprisingly,

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}.$$

The next step is to sum both sides from  $k = -\infty$  to  $\infty$ . Why is taking an infinite sum legitimate here? What is the result (in terms of  $S_n$ )?

- 24.**  $\langle 4 \rangle$  We apply the automated method to a sum where maybe you don't know the answer already:  $a_n = \sum_{k=0}^n \binom{n}{k}^2$ .

- a)** The first thing the method does is find a linear recurrence for the summands,  $\binom{n}{k}^2$ . It happens to find

$$\begin{aligned} n \binom{n}{k}^2 - (2n-1) \left( \binom{n-1}{k} + \binom{n-1}{k-1} \right) \\ + (n-1) \left( \binom{n-2}{k} - 2 \binom{n-2}{k-1} + \binom{n-2}{k-2} \right) \\ = 0. \end{aligned} \quad (8)$$

This is a second-order linear recurrence in two variables.

- a)** Verify that (8) is correct. You may wish to get a CAS to do it. Even if you do it by hand, note that the verification, while tedious, is routine. Finding such recurrences was not routine, at least not until this theory was developed.
- b)** Sum (8) over all integers  $k$ . You should get a much simpler first-order linear recurrence for  $a_n$ .
- c)** Verify that  $b_n = \binom{2n}{n}$  satisfies the same recurrence as in b), and furthermore,  $a_0 = b_0$ . Therefore, what?
- d)** Give a combinatorial argument to prove the identity you proved in c). We confess we still like combinatorial arguments better, but first, there's no guarantee that anyone will ever find one, while there is a guarantee that the automated method will work. Second, once you study and understand the automated theory, you will find beauty in it too, both in its breadth and in the guiding insights behind its lengthy computations.

## 5.10 Nonlinear Difference Equations

In Section 5.2, Example 8, we looked briefly at a nonlinear difference equation as a population model. We will now study that difference equation further, in order to make some points about nonlinear problems, and to introduce you to an area of active research.

There are linear and nonlinear problems in all branches of mathematics. It's a rule of thumb that where linear problems are easy, nonlinear problems are hard, and where linear problems are hard, nonlinear problems are impossible (except perhaps for approximate solutions).

For difference equations the situation is this. So long as a nonlinear equation is given in the form of Theorem 1 of Section 5.4, a solution exists. However, in general no explicit formulas are known, and even qualitative information is hard to come by. We will illustrate this with the recurrence

$$P_{n+1} = P_n [1 + r(1 - P_n)] = (1+r)P_n - rP_n^2, \quad r > 0, P_0 \geq 0. \quad (1)$$

This is the normalized population equation (11) of Section 5.2 with rate of population growth  $r$  and carrying capacity  $C = 1$ . We shall consider values for  $r$  that would generally be unreasonable for rates of population growth but which are applicable to real-world concerns such as turbulence in fluid flow. As a result, we will sometimes get values of  $P_n$  that are unreasonable for populations.

The quadratic recurrence in Eq. (1) is a very simple nonlinear difference equation, and can be made even simpler by certain substitutions [23–25], yet we will be unable to obtain an explicit solution. Instead we consider long-term qualitative behavior. For instance, in the population case ( $r$  positive but small) we observed and expected that  $P_n$  would approach 1; we can ask if this happens more generally. Even for this qualitative question we will be able to obtain an answer only for certain values of  $r$ . This whole section is really one long numerical exploration of Eq. (1), punctuated by provable results when we can get them.

Let's start right in. Since none of the theory from Section 5.5 about generating all solutions from basic solutions applies [8], the best thing to do first is generate some data (inductive paradigm). Table 5.2 shows the four sequences obtained from Eq. (1) for  $r = 1$  and 1.5, and  $P_0 = .2$  and  $.7$ . Note that  $P_n$  goes rapidly to 1 in each case, the same behavior we noted graphically in Fig. 5.3, Section 5.2, which used  $r = .07$  and values of  $P_0$  both above and below 1. Note, though, that for  $r = 1.5$  in Table 5.2,  $P_n$  does not stay on the same side of 1 as it starts on. We return to this point later.

---

**TABLE 5.2**  
**Values of  $P_n$  for  $n$  from 1 to 10 Using Eq. (1); the Values of  $r$  and  $P_0$  are Indicated above each Column**

---

| $n$ | $r=1, P_0=.2$ | $r=1, P_0=.7$ | $r=1.5, P_0=.2$ | $r=1.5, P_0=.7$ |
|-----|---------------|---------------|-----------------|-----------------|
| 1   | .360          | .910          | .440            | 1.015           |
|     | .590          | .992          | .810            | .992            |
|     | .832          | 1.000         | 1.041           | 1.004           |
|     | .972          | 1.000         | .977            | .998            |
| 5   | .999          | 1.000         | 1.011           | 1.001           |
|     | 1.000         | 1.000         | .994            | 1.000           |
|     | 1.000         | 1.000         | 1.003           | 1.000           |
|     | 1.000         | 1.000         | .999            | 1.000           |
|     | 1.000         | 1.000         | 1.001           | 1.000           |
| 10  | 1.000         | 1.000         | 1.000           | 1.000           |

---

It seems that  $P_n \rightarrow 1$  in each column, and we'd like to understand why. Now, the limit for which it is easiest to explain convergence (or divergence) is 0, because

in addition to the general limit theorems, for limit 0 you may also use the following *ratio argument*: If, say,  $|a_{n+1}/a_n| \leq .9$  for all  $n$ , then  $a_n \rightarrow 0$ , because  $a_n$  decreases at least as fast as the geometric sequence  $.9^n$ . Therefore, we will look at  $1 - P_n$  and try to explain why it goes to 0. From the first equality in (1) we find

$$\begin{aligned} 1 - P_{n+1} &= 1 - P_n - rP_n(1 - P_n) \\ &= (1 - P_n)(1 - rP_n), \end{aligned} \tag{2}$$

and so

$$\frac{1 - P_{n+1}}{1 - P_n} = 1 - rP_n.$$

Thus, it makes sense to analyze the ratio of successive values of  $1 - P_n$  to see if  $1 - P_n$  itself goes to 0. Specifically, here is what the ratio  $1 - rP_n$  tells us about the transition from any one value  $1 - P_n$  to the next value  $1 - P_{n+1}$ :

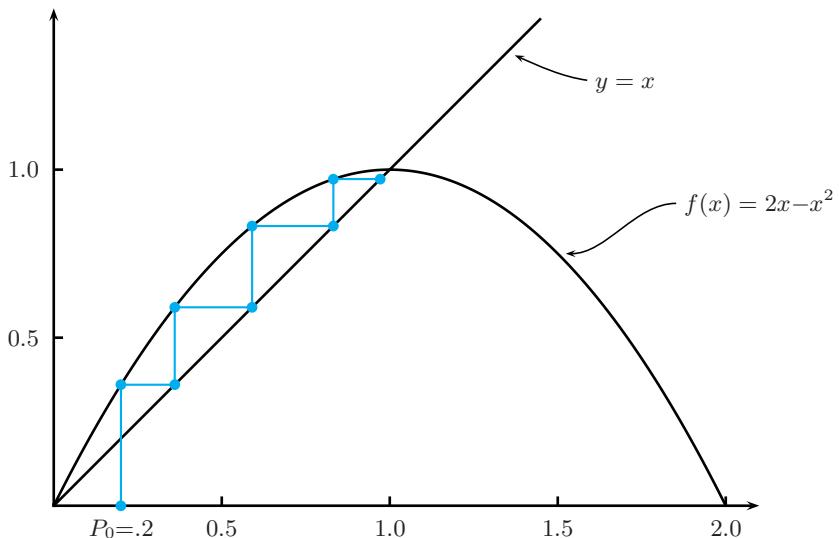
1. If  $0 < 1 - rP_n < 1$ , then  $1 - P_{n+1}$  is smaller than  $1 - P_n$ , and has the same sign, so  $P_{n+1}$  is closer to 1 than  $P_n$  and on the same side of 1;
2. If  $-1 < 1 - rP_n < 0$ , then  $1 - P_{n+1}$  is smaller than  $1 - P_n$  but has the opposite sign, so  $P_{n+1}$  is closer to 1 than  $P_n$  but on the other side of 1;
3. If  $1 - rP_n = 0$ , then  $1 - P_{n+1} = 0$  so  $P_{n+1} = 1$ ;
4. If  $1 - rP_n \geq 1$ , then  $P_{n+1}$  is as far or farther from 1 than  $P_n$  and on the same side of 1. (This case, which requires that  $P_n$  be negative, since  $r$  is always nonnegative, never occurs in any examples in the text of this section but does occur in several problems; see [14].)
5. If  $1 - rP_n \leq -1$ , then  $P_{n+1}$  is as far or farther from 1 than  $P_n$  but on the other side.

Now so long as  $0 < r \leq 1$  and  $P_0 < 1$ , case 1 applies — and by induction continues to apply for all  $P_n$ . This is the key observation that explains what happens in the first two columns of Table 5.2. So in those columns all terms of  $\{P_n\}$  are between 0 and 1 and each term is closer to 1 than the previous one. In fact, as  $n$  increases,  $P_n$  approaches 1 at an increasing rate, in that the ratio  $(1 - P_{n+1})/(1 - P_n) = 1 - rP_n$ , always between 0 and 1, gets smaller, but remains positive, as  $P_n$  increases. Thus  $\lim_{n \rightarrow \infty} P_n = 1$ , with convergence at least as fast as for a geometric sequence.

There's also a graphical approach to studying the long-term behavior of first-order difference equations, which is often very effective for nonlinear problems. (It's also effective for some linear difference equations [17], but we didn't use it for them because the algebraic approach provided much more information — an explicit formula.)

Figure 5.13 shows the method. If the difference equation is  $P_{n+1} = f(P_n)$ , you first plot the curve  $y = f(x)$ . For instance, if the difference equation is Eq. (1), then  $f(x) = (1 + r)x - rx^2$ . In Fig. 5.13 we consider the particular case  $r = 1$ . Next you plot the  $45^\circ$  line  $y = x$ . Then you mark  $P_0$  on the horizontal axis and draw a vertical line from it until it intersects the  $f$  curve. From there you draw a horizontal line until it hits the  $45^\circ$  line. Now iterate — vertical line, horizontal line, over and over. If this zig-zag line approaches a limit, then the  $x$ -coordinate of the

limit point is  $\lim_{n \rightarrow \infty} P_n$ . (Since the limit point will be on the line  $y = x$ , we could also say that the  $y$ -coordinate is the limit of  $P_n$ .)



Graphical method for generating the terms of a first-order recurrence  $P_{n+1} = f(P_n)$  in the case  $f(x) = 2x - x^2$  (that is,  $f(x) = (1+r)x - rx^2$  with  $r = 1$ ). If we put  $P_n$  on the  $x$ -axis, draw a vertical line to the curve  $f$ , and then a horizontal line to  $y = x$ , the new  $x$ -coordinate is  $P_{n+1}$ .

**FIGURE 5.13**

Why does this method work to generate successive terms? When we draw a vertical line from  $P_0$  on the  $x$ -axis to the curve  $f$ , the  $y$ -coordinate of the point we reach is simply  $f(P_0) = P_1$ . Then when we go horizontally to the line  $y = x$ , we reach the point  $(P_1, P_1)$  with  $x$ -coordinate  $P_1$ . Thus we may iterate the process, now drawing the vertical line from  $P_1$ , etc. We may then look for a pattern.

For instance, in Fig. 5.13 it seems pretty clear visually, for the specific function  $f(x) = 2x - x^2$  used there, that no matter what value between 0 and 1 we choose for  $P_0$ , the zig-zag moves up towards the point  $(1,1)$ , where  $y = f(x)$  and  $y = x$  intersect [15].

Two Observations. First, the points where the  $f$  curve and  $y = x$  intersect are special. Since  $f(x) = x$  at these points, if we start with  $P_0$  equal to one of these  $x$  values, the sequence stays there forever. Thus, such points are called **fixed points** or **equilibrium points**. Eq (1) has two, for any value of  $r$ , as you would expect from a quadratic:

$$x = (1 + r)x - rx^2,$$

$$0 = rx(1 - x),$$

so the fixed points are  $x = 0$  and  $x = 1$ . Both make sense in terms of population models. If  $P_0 = 0$ , then there is no population to grow. If  $P_0 = 1$ , the carrying

capacity, you'd expect the population to remain constant. But there is a qualitative difference between the points. As Fig. 5.13 indicates, if you start with  $P_0$  a little bit off from 1,  $P_n$  still approaches 1; whereas if you start a little bit off from 0,  $P_n$  moves away from 0. We say that 1 is a **stable fixed point** and that 0 is an **unstable fixed point**.

Second, note that the graphical approach uses a *continuous* curve. There's a good reason for this: since we have no idea what values will appear in  $\{P_n\}$ , we need to graph all possible values. This is another illustration of the fruitful interplay of discrete and continuous methods.

We promised to return to the fact that, in columns 3 and 4 of Table 5.2,  $P_n$  jumped over 1 and then oscillated around 1. Actually, we've already explained this, in case 2 after Eq. (2) above. If  $1-rP_n < 0$ , then  $P_{n+1}$  is on the opposite side of 1 from  $P_n$ . For instance, in the 4th column,  $P_0 = .7$  and  $r = 1.5$ , so sure enough,

$$1 - rP_0 = 1 - (1.5)(.7) = -0.5 \quad \text{and} \quad P_1 = 1 + .05(1 - P_0) = 1.015.$$

In this example,  $1-rP_0 > -1$  but as case 5 indicates, we can run into trouble (with regard to convergence) if we have a sequence where at some point  $1-rP_n < -1$ . This is more likely to happen if  $r$  is larger than in the examples so far. Therefore, we consider some examples with  $r = 2, 2.25$  in Table 5.3.

Compared to the previous table, it's harder to see patterns here, which is why we provide more terms. In the second column, it looks like  $P_n$  is oscillating about 1, but converging to it, albeit pretty slowly. In column 1, it looks instead as if  $P_n$  doesn't go to one limit but rather two, oscillating back and forth from 1.019 to .980. But don't be hasty! First of all, these entries are rounded to three decimal places. Note that the 19th entry was .981 instead of .980; this could be a roundoff error, but it could instead indicate that all the entries listed as .980 are really an increasing subsequence, with the increase so small that so far it has affected only digits not shown. Why not write a program [1] to print out more digits and more terms? As for columns 3 and 4, it definitely looks like there is a period 2 cycle, called a **limit cycle**, but here too we should be cautious.

Let's try to prove that  $\{P_n\}$  converges for  $r = 2$  and any  $P_0$  in  $[0,1]$ . This attempt will show you how rapidly this difference equation gets tough to analyze as  $r$  increases. Can we show that  $P_{n+1}$  is always closer to 1 than  $P_n$  is? Since  $0 < P_0 < 1$ , we find that the factor  $1 - rP_0$  satisfies  $1 > 1 - 2P_0 > -1$ , so  $P_1$  is indeed closer to 1 than  $P_0$  is. For instance, in column 2,  $P_1 = 1.120$ , which differs from 1 by .120, whereas  $P_0 = .7$  differed from 1 by .300. However, now that  $P_1 > 1$ , we find that  $1 - 2P_1 < -1$ , so  $P_2$  will be farther from 1 than  $P_1$  is. Indeed, looking in the table we find  $P_2 = .851$ , which differs from 1 by .149. So it does not look like we are getting convergence.

Still, let's not give up. Now that  $0 < P_2 < 1$ , the factor  $c = 1 - rP_2$  will again satisfy  $1 > c > -1$ , so  $P_3$  will be closer to 1 than  $P_2$  was. Indeed,  $P_3 = 1.105$ , only .105 away from 1. In fact, not only is  $P_3$  closer than  $P_2$ , it is closer than  $P_1$ . If we can show that  $P_{n+2}$  is always closer to 1 than  $P_n$  is, then it follows that the sequence converges to 1 after all. Indeed, for  $r$  satisfying  $1 < r \leq 2$ , this does always happen, but the proof is very subtle [21]. (For  $r < 2$  the decrease every two

**TABLE 5.3**  
**Values of  $P_n$  for  $n$  from 1 to 20 Using Eq. (1); Same Choices for  $P_0$  as Before but Now  $r$  is Larger, 2 and 2.25**

| $n$ | $r=2, P_0=.2$ | $r=2, P_0=.7$ | $r=2.25, P_0=.2$ | $r=2.25, P_0=.7$ |
|-----|---------------|---------------|------------------|------------------|
| 1   | .520          | 1.120         | .560             | 1.172            |
|     | 1.019         | .851          | 1.114            | .717             |
|     | .980          | 1.105         | .828             | 1.174            |
|     | 1.019         | .874          | 1.149            | .715             |
|     | .980          | 1.094         | .764             | 1.174            |
| 5   | .980          | .888          | 1.170            | .715             |
|     | 1.019         | 1.087         | .723             | 1.174            |
|     | .980          | .898          | 1.174            | .715             |
|     | 1.019         | 1.081         | .715             | 1.174            |
|     | 1.019         | .905          | 1.173            | .715             |
| 10  | .980          | 1.077         | .715             | 1.174            |
|     | 1.019         | .912          | 1.174            | .715             |
|     | .980          | 1.073         | .715             | 1.174            |
|     | 1.019         | .917          | 1.174            | .715             |
|     | .980          | 1.069         | .715             | 1.174            |
| 15  | 1.019         | .921          | 1.174            | .715             |
|     | .980          | 1.067         | .715             | 1.174            |
|     | 1.019         | .925          | 1.174            | .715             |
|     | .981          | 1.064         | .715             | 1.174            |
|     | 1.019         | .928          | 1.174            | .715             |

terms in  $1 - P_n$  is essentially geometric, hence rapid; but for  $r = 2$  it is extremely slow. This is why column 1 of Table 5.3 is ambiguous.)

Column 1 of Table 5.3 gave you an example of the fact that even large amounts of data do not always make the true story immediately clear. It seemed at first that the sequence alternated between two limits but, as just noted, that isn't so. Indeed, there were probably a lot of false starts before mathematicians got a good sense of what the simple Eq. (1) is really "up to". It helps to generate some theory concurrently with data, as we have done.

So, before making heavy bets about what seems to be obviously going on in columns 3 and 4 of Table 5.3, it is instructive to look at data with even larger  $r$ 's. Table 5.4 shows examples with  $r = 2.5$  and  $2.7$ .

TABLE 5.4

More Values of  $P_n$  for  $n$  from 1 to 20 Using Eq. (1); for  $r = 2.5$  we use  $P_0 = .1$  Instead of .2 to Avoid a Misleading Special Case

| $n$ | $r=2.5, P_0=.1$ | $r=2.5, P_0=.7$ | $r=2.7, P_0=.2$ | $r=2.7, P_0=.7$ |
|-----|-----------------|-----------------|-----------------|-----------------|
| 5   | .325            | 1.225           | .632            | 1.267           |
|     | .873            | .536            | 1.260           | .354            |
|     | 1.150           | 1.158           | .376            | .971            |
|     | .719            | .701            | 1.009           | 1.047           |
|     | 1.224           | 1.225           | .985            | .913            |
|     | .538            | .536            | 1.025           | 1.127           |
|     | 1.160           | 1.158           | .955            | .741            |
|     | .697            | .701            | 1.071           | 1.259           |
|     | 1.225           | 1.225           | .866            | .378            |
|     | .536            | .536            | 1.179           | 1.012           |
| 10  | 1.158           | 1.158           | .609            | .979            |
|     | .701            | .701            | 1.252           | 1.035           |
|     | 1.225           | 1.225           | .400            | .938            |
|     | .536            | .536            | 1.048           | 1.095           |
|     | 1.158           | 1.158           | .912            | .814            |
|     | .701            | .701            | 1.128           | 1.223           |
|     | 1.225           | 1.225           | .738            | .488            |
|     | .536            | .536            | 1.260           | 1.162           |
|     | 1.158           | 1.158           | .375            | .654            |
|     | .701            | .701            | 1.008           | 1.265           |

There is a pattern in column 1: do you see it? If not, look at Table 5.5, where we repeat column 1, but this time we have split it into four columns; read across and then down as with ordinary prose. The pattern is now obvious — the sequence approaches a limit cycle of period 4. Now check column 2 of Table 5.4 and you will see the same thing; in fact, it looks as if the cycle is entered immediately (but see [4]). The “right” format for displaying data always helps!

Now that we see that we can obtain a sequence with limit period 4 when  $r = 2.5$ , it seems reasonable to believe that the two sequences in Table 5.3 with  $r = 2.25$  really do have a limit period 2. Indeed, one can show this by applying the graphical method, not to Eq. (1), but to the “second iterate” of Eq. (1). That is, using  $f(P_n)$  again as the RHS of Eq. (1), one draws the graph for the sequence

**TABLE 5.5**  
**The First Column of Table 5.4 Split in Four; Read Across  
 and then Down as with Ordinary Prose**

| $n$ | $P_n$ | $n$ | $P_n$ | $n$ | $P_n$ | $n$ | $P_n$ |
|-----|-------|-----|-------|-----|-------|-----|-------|
| 1   | .325  | 2   | .873  | 3   | 1.150 | 4   | .719  |
| 5   | 1.224 | 6   | .538  | 7   | 1.160 | 8   | .697  |
| 9   | 1.225 | 10  | .536  | 11  | 1.158 | 12  | .701  |
| 13  | 1.225 | 14  | .536  | 15  | 1.158 | 16  | .701  |
| 17  | 1.225 | 18  | .536  | 19  | 1.158 | 20  | .701  |

defined by  $Q_{n+1} = f(f(Q_n))$ . The two limit values .715 and 1.174 are in fact fixed points of the equation  $x = f(f(x))$ ; see [22].

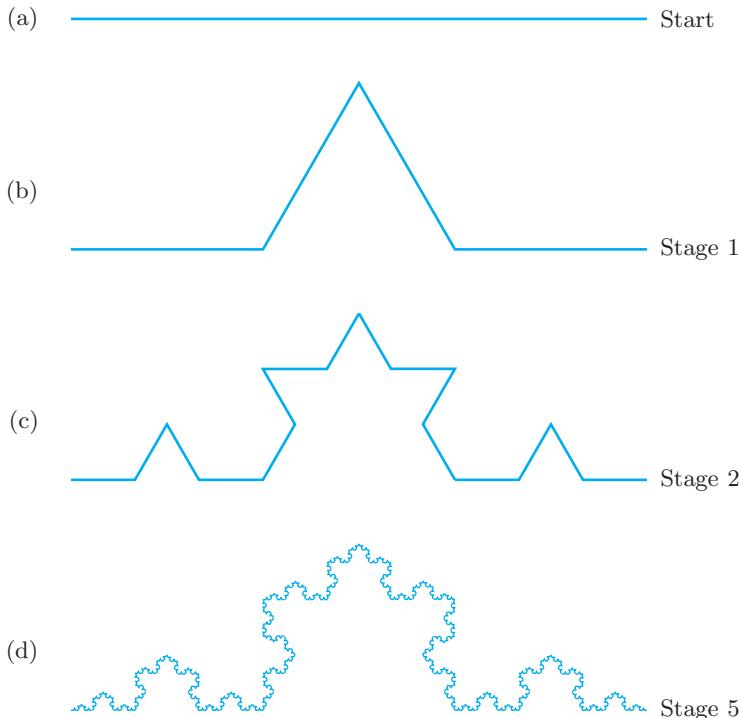
We've reached the end of what we will even attempt to explain. The fact is, there is an infinite sequence of numbers  $2 = r_0 < r_1 < r_2 < \dots$ , with limit  $r^* = 2.692$ , such that, for sequences generated from Eq. (1) with  $r_{k-1} < r < r_k$  and  $0 < P_0 < 1$ , the sequence converges on a limit cycle with  $2^k$  elements.

Furthermore, for  $r \geq r^*$ , the behavior of the sequence is much more complicated than that, so complicated that it is called **chaotic**. Columns 3 and 4 of Table 5.4 illustrate this — there seems to be no pattern. In the last 30 years it has been discovered that this phenomenon of doubling followed by chaos is very widespread. For one-variable problems such as ours, the reasons for this behavior and the nature of the chaos is at last fairly well understood, but much remains open, especially for several variables. This subject is often called **discrete dynamical systems**. (There are also plenty of continuous dynamical systems which exhibit chaos, and these continuous systems are also an active area of research.)

We want to say a little more about the nature of this chaos. Sequences like column 1 of Table 5.4 don't have a limit *point*, but they do have a limit *set*. If you start with  $P_0$  in  $S = \{.701, 1.225, .536, 1.158\}$ , you stay there (in fact, visiting each member), and if you start outside but nearby  $S$ , you approach each member of  $S$  infinitely often. The set  $S$  is therefore called an **attractor**. The doubling of the size of the limit cycle as  $r$  increases is a doubling of the size of the attractor set. Eq. (1) still has an attractor for many  $r \geq r^*$ , but now the attractor can be a very complex infinite set — hence dubbed a **strange attractor**. It can be complex in at least two ways: it has a *recursive structure* and it has *fractional dimension*.

We can explain these claims best by taking a simpler and more easily visualized example than what results from Eq. (1). Start with a line segment as in Fig. 5.14a and replace it by four segments  $1/3$  as long as in Fig. 5.14b. Now replace each of these segments by four segments, as in Fig. 5.14c, and so on. Fig. 5.14d shows the

result of five levels of replacements. Now imagine doing this forever. The result is called the **Koch curve**. Put recursively, the Koch curve is what you get by taking a segment, replacing it by four segments as shown, and then making each of them into a Koch curve.



If you carry out the transformation from (a) to (b) forever, you get the Koch curve; (d) shows the fifth level approximation.

Notice that this recursion digs an infinite hole. If you actually wanted to implement it as a graphics algorithm, you would have to stop at some level, as we did in Fig. 5.14d. However, the infinite recursion makes sense and results in a *limit curve*. The infinite recursion implies that the limit curve is **self-similar** in that pieces of the curve are similar to the whole curve. For the Koch curve, this means if you take one-quarter of it and magnify it by a factor of 3, you get the whole Koch curve again.

Self-similar curves are one type of **fractal**. Fractals have fractional dimension in the following sense. A straight line segment is 1-dimensional because if you magnify your picture by a factor of 3, the measure of the segment (its length) gets 3 times as big. Likewise, a square is 2-dimensional because if you magnify a square by 3, its measure (area) gets  $9 = 3^2$  times bigger. Now, what happens if you magnify the Koch curve by 3? It gets 4 times bigger, since, as above, you get 4 pieces each *congruent* to the original. In other words, the measure of the Koch curve grows

**FIGURE 5.14**

faster than for a 1-dimensional object but slower than for a 2-dimensional object, so we say that its dimension is fractional. (What exactly do you think its dimension should be? [27])

To conclude, we return to the essential point, the behavior of nonlinear difference equations. We have investigated one particular example, a very simple quadratic difference equation, and it has taken us beyond the point where even qualitative analysis is manageable. Are all nonlinear recurrences this bad? No, but most are. Every once in a while a nonlinear recurrence can be solved in closed form by a clever substitution, one that reduces it to another difference equation (often linear) that can be solved. We give some examples in the problems [29–33]. However, there are no known general rules for finding such substitutions.

## Problems: Section 5.10

---

For the first 7 problems, use a computer or hand calculator to create some more tables for Eq. (1) with the values of  $r$  and  $P_0$  indicated.

1.  $\langle 2 \rangle$   $r = 2$ ,  $P_0 = .2$  as in Table 5.3, but display more decimal places and more terms. Now would you conjecture that the sequence is converging to 1, or does it still seem to have a period 2 cycle?
2.  $\langle 2 \rangle$   $r = 1.9$  and  $P_0 = .2, .7$ .
3.  $\langle 3 \rangle$   $r = 2.5$ ,  $P_0 = .2$ . This is the case that really should have been in column 1 of Table 5.4, since that table was created to be parallel to Table 5.3. What was special about this case that made us not include it in the text? Now try modifying  $P_0$  ever so slightly from .2, keeping  $r = 2.5$ . Is the special behavior stable?
4.  $\langle 2 \rangle$   $r = 2.5$ ,  $P_0 = .7$ , but display more digits than in Table 5.4. Is the 4-cycle entered immediately?
5.  $\langle 2 \rangle$   $r = 2.55$  and various values of  $P_0$ , at least  $P_0 = .5$  and 1.4. Do you find periodic behavior?
6.  $\langle 2 \rangle$   $r = 3$  and various values of  $P_0$ , at least  $P_0 = .5$  and 1.4.
7.  $\langle 2 \rangle$  Redo the first and third columns of Table 5.4 using 5 decimal places. Then do these two columns again changing  $P_0$  ever so slightly to .10001 in the first and .20001 in the third. What do you observe? If a difference equation has the property that minute changes in the initial conditions can lead to large and seemingly random changes in later terms, then humans will be forever incapable of making accurate long term predictions

for any real-world system obeying that difference equation. Explain.

8.  $\langle 2 \rangle$  Show that solutions to Eq. (1) are not closed under either multiples or sums. There are two ways to do this. One way is to take two solutions, say the first two columns of Table 5.2, and show that neither twice the first nor their sum satisfies Eq. (1). Or you can show where the proof of Theorem 1 of Section 5.5 breaks down when applied to Eq. (1).
9.  $\langle 2 \rangle$  For the most part we chose to represent the sequences in this section by tables instead of graphs. Usually graphs are easier to comprehend. Why did we use tables?
10.  $\langle 2 \rangle$  In light of Eq. (2), does anything special happen to later terms in  $\{P_n\}$  if at some point
  - a)  $1 - rP_n = 1$ ?
  - b)  $1 - rP_n = -1$ ?
11.  $\langle 2 \rangle$  By induction prove the claim in the text that if  $\{P_n\}$  satisfies Eq. (1) with  $0 < r \leq 1$  and  $0 < P_0 < 1$ , then  $\{P_n\}$  is a strictly increasing sequence with all terms between 0 and 1.
12.  $\langle 3 \rangle$  In Fig. 5.3, Section 5.2, all three sequences  $\{P_n\}$  converge monotonically to 1. The case  $P_0 = .2$  can be explained, just as early in this section, by noting that  $0 < r \leq 1$  and  $0 < P_0 < 1$  implies that  $0 < 1 - rP_n < 1$  for all  $n$ . However, in the other two cases in the figure,  $P_0 \not< 1$ . Show that, nonetheless  $0 < 1 - rP_n < 1$  for all  $n$ .
13.  $\langle 2 \rangle$  We argued that if  $P_0 = 0$  or 1, then  $P$  never changes. In fact, if at any point  $P_n = 0$ , then  $P$

never changes after that. But could this happen? Could  $P_0$  be neither 0 nor 1, and yet at some point  $P_n$  is exactly 0 or 1?

- a) Use case 3 in the analysis of the ratio  $1 - rP_n$  to show that, yes,  $P_0$  can be neither 0 nor 1 and yet  $P_1$  can be 1.
- b) Use Eq. (2) directly to show that, yes,  $P_0$  can be neither 0 nor 1 and yet  $P_1$  can be 0.

14. (2) Case 4 in the analysis of the ratio  $1 - rP_n$  may have seemed far fetched, because for  $1 - rP_n$  to be  $> 1$  when  $r > 0$  and the  $P$  sequence starts out with  $P_0$  positive, it must be that at some point  $P_n$  is negative. Come up with an example where, in fact,  $P_1 < 0$ , and so case 4 applies with  $n = 1$ .

15. (3) For the specific function  $f(x) = 2x - x^2$  in Fig. 5.13, show by algebra that if  $P_{n+1} = f(P_n)$  and  $0 < P_0 < 1$ , then  $\{P_n\}$  is an increasing sequence and  $\lim P_n = 1$ . That is, show that what is visually evident in the figure is correct. Hint: To show that  $\{P_n\}$  is increasing, show that  $f(x) > x$  for all  $x$  with  $0 < x < 1$ . To show that  $\lim P_n = 1$ , note that  $r = 1$  in Eq. (1) and now use the ratio argument explained above Eq. (2). Specifically, show that  $|P_{n+1}/P_n| \leq 1 - P_0 < 1$ .

16. (2) Consider Eq. (1) with  $r = 1$ .

- a) Show that if  $1 \leq P_0 < 2$ , then  $P_n$  approaches 1 in the limit.
- b) Show that if  $P_0 < 0$ , then  $P_n$  gets more and more negative without bound.
- c) What happens if  $P_0 = 2$ ?
- d) What happens if  $P_0 > 2$ ?

17. (2) Consider the linear difference equation  $a_n = \frac{1}{2}a_{n-1} + 1$ . For any initial condition,  $\{a_n\}$  has the same limit as  $n$  gets big. Show this, and find the limit, two ways:

- a) By the graphical method of this section;
- b) By finding the explicit formula for  $a_n$  in terms of  $a_0$ , and analyzing the limit algebraically.

18. (2) Consider the difference equation  $a_n = 2a_{n-1}$ . This is easy to analyze algebraically for any initial value, so it provides a good check on the graphical method to see that the graphical method works for it too. Show graphically that any solution has a fixed point, but that the fixed point is unstable: if  $a_0$  is not the fixed point, then  $a_n$  goes off to  $\pm\infty$ .

19. (2) Consider  $a_n = 2a_{n-1} + 1$ . Use the graphical method to draw the same conclusions as in [18].

20. (2) Attempt a graphic analysis for Eq. (1) with  $r = 2$ . (You may want to get a computer to draw the figure for you.) What in the figure “explains” how the sequence  $\{P_n\}$  can jump from below 1 to above 1?

21. (5) Suppose  $\{P_n\}$  satisfies Eq. (1) with  $0 < r \leq 2$ . Show that, if  $P_n$  is close to 1, then  $P_{n+2}$  is even closer to 1 and on the same side. Also, use your analysis to explain why  $r = 2$  is a “dividing point”. Hint: Write  $P_n$  as  $1 + \epsilon$ , and now compute  $P_{n+2}$  by plugging into Eq. (1) twice. Regroup to get  $P_{n+2}$  in the form  $1 + \epsilon'$ , where  $\epsilon'$  will be some complicated expression in  $r$  and  $\epsilon$ . Now consider only those terms with the lowest power of  $\epsilon$ . If the claimed conclusion is correct if there were no other terms, then you are done, because for  $\epsilon$  sufficiently small the ignored terms cannot overcome the effect of the lowest power terms. You may wish to do all this with a CAS.

22. (3) Let  $f(x) = (1 + 2.25)x - (2.25)x^2$ , so that Eq. (1) with  $r = 2.25$  is  $P_{n+1} = f(P_n)$ .

- a) Graph  $y = x$  against  $y = f(f(x))$  to find the roots of  $x = f(f(x))$ . (You should use a highly accurate computer or hand-calculator plotter.)
- b) Graph  $y = x$  against  $y = f(x)$ . Apply the graphical method both to this figure and the figure from part a). Use these figures to explain why columns 3 and 4 of Table 5.3 have a period 2 limit cycle.
- c) Is 1 an equilibrium point for Eq. (1) with  $r = 2.25$ ? Is it a stable equilibrium? Why?

23. (2) Show that the difference equation

$$p_{n+1} = sp_n - p_n^2 \quad (3)$$

can be obtained from Eq. (1) by the substitutions  $s = 1 + r$  and  $p_n = rP_n$ . Thus Eq. (1) is equivalent to another quadratic difference equation that is even simpler, and so this simpler difference equation exhibits the same complicated behavior.

24. (3) Find substitutions that turn Eq. (1) into

$$q_{n+1} = sq_n + q_n^2. \quad (4)$$

See [23] for ideas.

25. (2) Show that Eq. (4) in [24] can be simplified further. By making a substitution  $z_n = q_n + b$ , Eq. (4) reduces to

$$z_{n+1} = z_n^2 + c, \quad (5)$$

where  $b, c$  are constants you are to determine.

Eq. (5) is the one used in most discussions relating fractals to dynamical systems.

26. ⟨3⟩ (assumes calculus) In this section, we saw that solutions to Eq. (1) could, at least for large enough  $r$ , jump across the stable equilibrium value  $P = 1$  and even jump from positive to negative. These phenomena are the starting point for chaotic behavior.

The continuous analog of the recurrence in (1) is the “logistic” differential equation

$$P' = rP(1 - P), \quad (6)$$

where  $P$  is now a continuous function of time and  $r > 0$  is now the instantaneous rate of growth. Show either by the graphical method of vector fields or by solving (6) explicitly, that no solution crosses over  $P = 1$  or crosses from positive to negative. In fact, all solutions are well-behaved functions; there is no chaotic behavior for the logistic equation, even for large  $r$ . So this is a case where the continuous model is much better behaved than the discrete model. However, add a few more variables and continuous models can show chaotic behavior too.

27. ⟨2⟩ For self-similar sets one may define dimension as follows. A set has dimension  $\alpha$  if, when its linear dimensions are magnified by  $c$ , one obtains  $c^\alpha$  copies of the original size. What is the dimension of the Koch curve?
28. ⟨3⟩ The Koch curve is part of the Koch snowflake.  
a) What do you think that is? *Hint:* Start with an equilateral triangle.

- b) In some recursive computer language with good graphics, write a program to produce (an approximation of) the Koch curve or the Koch snowflake.

29. ⟨2⟩ Solve the difference equation  $a_n = (a_{n-1})^2$ .  
*Hint:* write out a few terms and you will see a pattern in terms of powers of  $a_0$ . Prove it by induction.

30. ⟨3⟩  
a) By taking logarithms of both sides of the equation and then making an appropriate substitution, solve the equation

$$p_{n+1} = ap_n^m, \quad a, m \text{ constant}, \quad m \neq 1.$$

- b) As a special case, come up with a formula for perhaps the simplest class of all of quadratic recurrences,

$$p_{n+1} = ap_n^2, \quad p_0 = p. \quad (7)$$

31. ⟨2⟩ Solve  $p_{n+1} = 3p_n - p_n p_{n+1}$ ,  $p_0 = 5/2$ , by the substitution  $a_n = 1/p_n$ .

32. ⟨3⟩ The nonlinear equation  $p_n p_{n+1} + ap_n + b = 0$  ( $a, b$  constants) can be converted to a linear equation by the substitution  $c_n = 1/(p_n + \alpha)$ , for the right choice of  $\alpha$ .

- a) Find the right choice(s) for  $\alpha$ .  
b) Solve the equation in the case  $a = 3$ ,  $b = 2$  and  $p_0 = 1$ .

33. ⟨3⟩ For the equation  $p_{n+1} = 2p_n^2 - 1$ , use the substitution  $p_n = \cos x_n$  to obtain an equation that can be solved for  $x_n$ , and hence for  $p_n$ . It will help if you remember your trigonometric identities!

## 5.11 Finite Differences

Ever wonder why recurrences are called difference equations? Because they are related to something called finite differences. We want to tell you enough about finite differences to make the connection, but we also have another purpose.

Differences and antidifferences provide a striking discrete mathematics parallel to differentiation and integration in calculus. We want to show you just enough so that, if you have had calculus, you will see this parallel almost immediately. In fact, you’ll wonder why difference theory wasn’t taught to you first, since it can all be done with high school algebra – no limits. If you haven’t had calculus, you are one of the lucky ones who will benefit when you get there, because you will have seen the patterns here.

At one time, finite differences played a major role in numerical analysis, the branch of mathematics whose chief concern is to solve problems in continuous mathematics by reformulating them in discrete terms. Numerical analysis, therefore, lies right at the interface between discrete and continuous mathematics. However, the role of finite differences in numerical analysis is now much reduced.

Finite differences were also at one time a primary method for finding closed forms for sums. We'll explain how below. However, this role has also faded, due to advances in computer algebra.

So that leaves the beauty of the parallel with calculus theory as the main draw. We hope you will agree that's draw enough.

As befits a final section, we will move rather quickly.

## Differences, Factorials, and Sums

---

**Definition 1.** Given a sequence  $\{a_n\}$ , the (**forward**) **difference** is the sequence whose  $n$ th term, denoted  $\Delta a_n$ , is

$$\Delta a_n = a_{n+1} - a_n.$$

---

The symbol  $\Delta$  is called the (**forward**) **difference operator**, with “operator” implying that  $\Delta$  “operates” on what follows it to transform it into something new.  $\Delta$  is the Greek capital letter Delta, but in this context one pronounces it “del”.

### EXAMPLE 1

If  $a_n = c$ , then

$$\Delta a_n = c - c = 0.$$

In words, the difference of any constant sequence is the zero sequence. ■

### EXAMPLE 2

Find  $\Delta n^k$ , where  $n$  is the variable and  $k$  is a positive integer.

**Solution** By direct calculation, using the Binomial Theorem, we obtain

$$\begin{aligned}\Delta n^k &= (n+1)^k - n^k = \left[ \sum_{i=0}^k \binom{k}{i} n^i \right] - n^k \\ &= \sum_{i=0}^{k-1} \binom{k}{i} n^i.\end{aligned}$$

In short, the difference of  $n^k$  is a polynomial in  $n$  of degree  $k-1$ . ■

It's too bad that the power functions  $n^k$ , which have such simple derivatives and integrals in calculus, have such complicated differences. Fortunately, there is

an alternative set of functions that interact with differences quite nicely. These are the falling factorials, defined in Section 0.2: For  $n$  arbitrary and  $k$  a positive integer,

$$n_{(k)} = n(n-1)(n-2) \cdots (n-k+1). \quad (k \text{ factors}) \quad (1)$$

By convention,  $n_{(0)} = 1$ . (In Section 0.2 we used  $x$  instead of  $n$ , but in this section we mostly use  $n$  for our variable since we are interested in integer values.) Note that when  $n \geq k$ , then  $n_{(k)} = P(n, k)$ , the number of permutations of  $n$  things taken  $k$  at a time. Also note that if  $n < k \neq 0$ , then  $n_{(k)} = 0$ ; why?

---

**Theorem 1.** For  $k \geq 1$ ,

$$\Delta n_{(k)} = k n_{(k-1)}.$$


---

**PROOF**  $\Delta n_{(k)} = (n+1)_{(k)} - n_{(k)}$ , and the two products on the right have all the same factors except for the greatest factor of  $(n+1)_{(k)}$  and the least factor of  $n_{(k)}$ . Pulling all the common factors out to the right (in the second equation below), we find

$$\begin{aligned} (n+1)_{(k)} - n_{(k)} &= [(n+1)n(n-1) \cdots (n+1-k+1)] \\ &\quad - [n(n-1) \cdots (n+1-k+1)(n-k+1)] \\ &= [(n+1)-(n-k+1)][n(n-1) \cdots (n+1-k+1)] \\ &= k[n(n-1) \cdots (n-[k-1]+1)] \\ &= kn_{(k-1)}. \blacksquare \end{aligned}$$

Theorem 1 suggests that it might be useful if we could write a polynomial as a linear combination of falling factorials because then it would be simple to take differences of the polynomial. Thus we would like to write

$$p(n) = \sum_{j=0}^k c_j n_{(j)} \quad (2)$$

with  $c_0, c_1, \dots, c_n$  suitable constants to be determined. If  $p(n) = q_k n^k + (\text{terms in } n \text{ of lower degree})$ , then, since from Eq. (1)  $n_{(k)} = n^k + (\text{terms in } n \text{ of lower degree})$  and no other falling factorial in Eq. (2) has an  $n^k$  term, we have  $c_k = q_k$ . The other  $c_j$ 's may be similarly determined [34].

The following theorem shows that  $\Delta$  “respects” linear combinations. One therefore says that  $\Delta$  is a **linear operator**.

---

**Theorem 2.** If  $\{a_n\}$ ,  $\{b_n\}$  and  $\{c_n\}$  are sequences such that for all  $n$

$$a_n = Bb_n + Cc_n,$$

where  $B$  and  $C$  are constants, then

$$\Delta a_n = B\Delta b_n + C\Delta c_n.$$

---

**PROOF** Direct calculation gives

$$\begin{aligned}\Delta a_n &= a_{n+1} - a_n \\ &= [Bb_{n+1} + Cc_{n+1}] - [Bb_n + Cc_n] \\ &= B(b_{n+1} - b_n) + C(c_{n+1} - c_n) \\ &= B\Delta b_n + C\Delta c_n.\blacksquare\end{aligned}$$

An immediate consequence of this theorem, Theorem 1 and Eq. (2) is

---

**Corollary 3.** The difference of a polynomial of degree  $k$  is a polynomial of degree  $k - 1$ . Specifically,

$$\Delta p(n) = \sum_{j=1}^k c_j j n_{(j-1)} \tag{3}$$

with the  $c_j$ 's the same constants as in Eq. (2). ■

---

Why is there one less term in Eq. (3) than there was in Eq. (2)? Also, note that the degree of  $\Delta p(n)$  is indeed  $k - 1$  because the coefficient of  $n^{k-1}$  in Eq. (3) is not 0 (why?).

**Antidifferences and Sums.** Recall that when  $s_n$  is defined to be  $\sum_{k=0}^n a_k$ , then  $\{s_n\}$  satisfies

$$s_n = s_{n-1} + a_n.$$

Changing indices, we may instead write this as  $s_{n+1} = s_n + a_{n+1}$  or

$$\Delta s_n = a_{n+1}. \tag{4}$$

In general, if  $\Delta c_n = d_n$ , we say that  $\{c_n\}$  is an **antidifference** of  $\{d_n\}$  and we write  $c_n = \Delta^{-1}d_n$ . We say “an” because the antidifference of a sequence is not unique. By Section 5.7 all antidifference sequences differ by solutions to the associated homogeneous equation  $c_{n+1} - c_n = 0$ , that is, by some constant sequence. The notation  $\Delta^{-1}d_n$  represents any one of them that we find convenient. Therefore, it is called the **indefinite** antidifference. You may know from calculus the corresponding notion of an indefinite integral.

Returning to Eq. (4),  $s_n = \sum_{k=0}^n a_k$  is a specific antiderivative of  $a_{n+1}$ , so once we find one antiderivative  $\Delta^{-1}a_{n+1}$ , then  $s_n$  will differ from what we found by at most a constant. In short, we can find a formula for any sum for which we can find some closed form antiderivative of the summand.

When can we just look at  $a_{n+1}$  and find its antiderivative  $\Delta^{-1}a_{n+1}$ ? Or more simply, when can we find  $\Delta^{-1}a_n$ , after which we just substitute  $n+1$  for  $n$ ? Answer: When  $a_n$  involves falling factorials. For instance,

$$\text{if } a_n = kn_{(k-1)} \text{ then } \Delta^{-1}a_n = n_{(k)}; \quad (5)$$

$$\text{if } a_n = n_{(k-1)} \text{ then } \Delta^{-1}a_n = \frac{1}{k} n_{(k)}; \quad (6)$$

$$\text{if } a_n = n_{(k)} + \frac{1}{2}n_{(k-1)} \text{ then } \Delta^{-1}a_n = \frac{1}{k+1} n_{(k+1)} + \frac{1}{2k} n_{(k)}. \quad (7)$$

Now the nice part: As noted in Eq. (2), *every* polynomial  $p(n)$ , and thus also  $p(n+1)$ , can be written as a sum of multiples of falling factorials. Thus to find the formula for  $\Delta^{-1}p(n+1)$  just replace  $p(n+1)$  expressed as a sum in powers of  $n+1$  by the equivalent sum using falling factorials and then treat each term separately. The following example shows that these sum formulas are sometimes easy to find.

### EXAMPLE 3

Find  $s_n = \sum_{k=0}^n k$  and  $t_n = \sum_{k=0}^n k^2$  by finding antiderivatives.

**Solution** Since  $\Delta s_n = n+1 = (n+1)_{(1)}$ , one antiderivative is  $\frac{1}{2}(n+1)_{(2)} = \frac{1}{2}(n+1)n$  using Eq. (6). This is one antiderivative and by the discussion above,

$$s_n = \sum_{k=0}^n k = \frac{1}{2}(n+1)n + A \quad (8)$$

for some constant  $A$ . To find  $A$ , plug the initial value  $n = 0$  into Eq. (8):

$$\sum_{k=0}^0 k = 0 = \frac{1}{2} \cdot 1 \cdot 0 + A = A.$$

Thus  $A = 0$  and  $\frac{1}{2}(n+1)n$  is the sum, a result we've obtained at various times earlier.

As for summing  $n^2$ ,

$$\Delta t_n = (n+1)^2 = (n+1)n + (n+1) = (n+1)_{(2)} + (n+1)_{(1)}.$$

Thus again using Eq. (6),

$$t_n = \sum_{k=0}^n k^2 = \frac{1}{3}(n+1)_{(3)} + \frac{1}{2}(n+1)_{(2)} + B. \quad (9)$$

By substituting  $n = 0$  we find that  $B = 0$ . We are done, except that Eq. (9) is not the traditional way to write the closed form for the sum. By routine algebra one may now show that  $t_n = n(2n+1)(n+1)/6$  [7]. ■

We are now ready to summarize this method for sums as a theorem. The theorem statement introduces one more insight that simplifies the method even

further: You never have to solve for the constant (which isn't always 0). The proof shows why.

---

**Theorem 4.** Define the notation  $b_k|_p^q$  by

$$b_k|_p^q = b_q - b_p.$$

Then

$$\sum_{k=p}^q a_k = \Delta^{-1} a_k|_p^{q+1}. \quad (10)$$

That is, to evaluate the sum, find any one antiderivative of the summand, evaluate it at  $q + 1$  (sic) and  $p$ , and subtract the latter from the former.

---

*Note 1.* The RHS of Eq. (10) is called the **definite** antiderivative of  $\{a_n\}$ .

*Note 2.* The upper and lower limits  $q + 1$  and  $p$  correspond in calculus to the upper and lower limits of definite integration. That the upper limit is  $q + 1$  and not, as you might expect,  $q$ , is an artifact of moving from the continuous world of integrals to the finite world of sums.

**PROOF** As explained before,

$$\sum_{k=0}^n a_k = \Delta^{-1} a_{n+1} + A$$

for some unknown constant  $A$ . Therefore,

$$\begin{aligned} \sum_{k=p}^q a_k &= \sum_{k=0}^q a_k - \sum_{k=0}^{p-1} a_k \\ &= (\Delta^{-1} a_{q+1} + A) - (\Delta^{-1} a_p + A) \\ &= \Delta^{-1} a_{q+1} - \Delta^{-1} a_p \\ &= \Delta^{-1} a_k|_p^{q+1}. \blacksquare \end{aligned}$$

*Warning.* While the theory of antiderivatives was at one time a major method for finding closed forms for sums, it is no longer the way to go unless it's easy to decompose your summand into terms (like falling factorials) with known antiderivatives. Today, it's generally better to use a CAS, which has more powerful methods (if sometimes less elegant) built in.

## Higher-Order Differences and Difference Tables

Since the difference operator produces sequences from sequences, we can iterate. Thus we define  $\Delta^2 a_n$  to mean  $\Delta(\Delta a_n)$ . More generally, we define

$$\Delta^k a_n = \begin{cases} a_{n+1} - a_n, & \text{if } k = 1; \\ \Delta[\Delta^{k-1} a_n], & \text{if } k > 1. \end{cases} \quad (11)$$

We call  $\Delta^k$  the  **$k$ th-order difference operator**. As an example,

$$\begin{aligned} \Delta^2 a_n &= \Delta(a_{n+1} - a_n) = (a_{n+2} - a_{n+1}) - (a_{n+1} - a_n) \\ &= a_{n+2} - 2a_{n+1} + a_n. \end{aligned}$$

---

**Theorem 5.** If  $p(n)$  is a  $k$ th-degree polynomial, then  $\Delta^{k+1} p(n) = 0$ .

---

**PROOF** By Corollary 3 above,  $\Delta p(n)$  has degree  $k - 1$ . Applying  $\Delta$  for  $k$  times, we get down to a 0th-degree polynomial, that is, a constant. Applying  $\Delta$  once more gives us all 0's. ■

More important, the converse is true: if upon applying differences to a sequence we get down to all 0's after  $k + 1$  iterations, then the sequence has a polynomial formula of degree  $k$ . We shall prove this in Theorem 6. This result will give us a handy way of identifying polynomials from data, so let's turn to formats for carrying out such calculations.

If we wish to write out the terms of a sequence and its differences, it is traditional to prepare a **difference table**. First list the terms of the original sequence  $\{a_n\}$  in a row. Then write out the differences underneath, with each difference halfway between the terms it is the difference of:

$$\begin{array}{ccccccccc} a_0 & a_1 & a_2 & \dots & a_n & a_{n+1} & \dots \\ \Delta a_0 & \Delta a_1 & \dots & & \Delta a_n & \dots & \end{array}$$

The second-order differences would make the next row. We can continue for as many rows as we like. To coordinate names properly, we call the top row the 0th row (the original sequence is the 0th difference). Then the first differences row is row 1,  $\Delta^2$  is the second, and so on.

Now, suppose we are given the  $k$ th row of a difference table and want to reconstruct the higher rows. (Why would we ever want to do that? Be patient.) If in addition we have the left-most entry in row  $k - 1$ , we can reconstruct that entire row. The direct way to see this is as follows: If  $b_0, b_1, \dots$  is row  $k - 1$  (unknown except for  $b_0$ ), and  $c_0, c_1, \dots$  is the known  $k$ th row, then by definition

$$c_0 = b_1 - b_0, \quad c_1 = b_2 - b_1, \quad \dots,$$

so

$$b_1 = c_0 + b_0, \quad b_2 = c_1 + b_1, \quad \dots, \quad (12)$$

and we can reconstruct  $b_1, b_2, \dots$  in order from (12). A more sophisticated way to see it is this: The sequence  $\{b_n\}$  satisfies the recurrence  $b_{n+1} = b_n + c_n$ , so by Theorem 1, Section 5.4, the sequence  $\{b_n\}$  is uniquely determined by the initial condition  $b_0$ .

In any event, by induction we conclude: If we know the last row of a difference table, and we know the left-most entries in every other row, then we can reconstruct the whole table uniquely, working up one row at a time. In particular, the top row is determined.

Now, how would this situation ever arise? Consider the problem of finding a formula for the number of regions into which  $n$  lines divide the plane (Example 5, Section 2.6). For  $n = 0, 1, 2, 3, 4$  one finds by drawing pictures that the numbers are 1, 2, 4, 7, 11. Make the difference table:

|   |   |       |       |       |       |      |
|---|---|-------|-------|-------|-------|------|
| 1 | 2 | 4     | 7     | 11    | . . . |      |
| 1 | 2 | 3     | 4     | . . . |       | (13) |
| 1 | 1 | 1     | . . . |       |       |      |
| 0 | 0 | . . . |       |       |       |      |

The obvious conjecture now is that the third row is 0 forever. Based on that conjecture, we can compute as many terms in the 0th row as we want, which should help us guess a formula. But it's better than that. The difference table method actually produces the formula for us!

---

**Theorem 6.** Consider the difference table for  $a_0, a_1, a_2, \dots$ . Suppose  $\Delta^{k+1}a_n = 0$  for all  $n \in N$  and that no prior row is all 0's. Then  $a_n = p(n)$ , where  $p(x)$  is the polynomial

$$p(x) = \sum_{j=0}^k \frac{\Delta^j a_0}{j!} x_{(j)}. \quad (14)$$

Furthermore,  $\Delta^k a_0 \neq 0$ , so  $p(x)$  is exactly degree  $k$ .

---

Note that  $p(x)$  really is a polynomial, because each  $\Delta^j a_0$  is a constant (why?) and the falling factorial  $x_{(j)}$  is a  $j$ th degree polynomial in  $x$ .

So Theorem 6 is the promised converse of Theorem 5, a very strong converse because it gives an explicit formula for the polynomial.

Eq. (14) is called **Newton's Interpolation Formula** because it can also be used to find the smallest degree polynomial that goes through a given finite set of points [20]. (“Interpolate” means to find a continuous function that goes through a specified discrete set of points, so that (approximate) values of the function can be calculated between the original points.)

**PROOF** Since the top row of a difference table is uniquely determined by its bottom row and its left side, it suffices to show that the difference tables generated by the sequences  $\{a_n\}$  and  $\{p(n)\}$  have the same left side and bottom. We know from Theorem 5 that row  $k+1$  of the  $p(n)$  table will be all 0's, so the bottoms of the two tables agree. As for the left sides, note from the definition of  $x_{(i)}$  that  $x_{(i)}(0) = 0$  when  $i \neq 0$  and that the function  $x_{(0)}$  is 1 by definition. Thus, substituting  $x = 0$  into Eq. (14), all terms on the right zero out except for the term with  $j = 0$ , and we get

$$p(0) = \frac{\Delta^0 a_0}{0!} = a_0.$$

Thus the left-most entry in the 0th row of both tables is  $a_0$ .

Next, by Theorem 2

$$\Delta p(x) = \sum_{j=1}^k \frac{\Delta^j a_0}{j!} \cdot jx_{(j-1)},$$

since by Theorem 1,  $\Delta x_{(j)} = jx_{(j-1)}$ . Now when we substitute  $x = 0$  all terms zero out except when  $j = 1$ , so

$$\Delta p(0) = \frac{\Delta^1 a_0}{1!} = \Delta a_0.$$

Thus the left-most entries in the 1st row of both tables is  $\Delta a_0$ .

Continuing in this way, we have for all  $i$  from 0 to  $k$  that

$$\Delta^i p(x) = \sum_{j=i}^k \frac{\Delta^j a_0}{j!} j_{(i)} x_{(j-i)}$$

and, in particular,

$$\Delta^i p(0) = \frac{\Delta^i a_0}{i!} i_{(i)} = \Delta^i a_0.$$

So both tables agree all along the left side, and thus agree everywhere.

We have now proved that  $a_n = p(n)$ , but we have not proved that  $\Delta^k a_0 \neq 0$ . This is where the hypothesis that row  $k+1$  is the first all-zero row comes in. We do a proof by contradiction. Suppose  $\Delta^k a_0$  were 0. Then  $p(x)$  would have degree at most  $k - 1$ . Thus by Theorem 5 (applied with  $k - 1$  instead of  $k$ ), row  $k$  would be an all-zero 0; contradiction. ■

If you have seen Taylor's Theorem in calculus, do you recognize the strong parallel to Theorem 6, both in the statement and in the proof?

## EXAMPLE 4

Use the difference table in display (13) to conjecture a formula for the number of regions on a plane divided by  $n$  lines.

**Solution** If we are right that the bottom row is all 0's, then by Theorem 6, the right formula is

$$\frac{1}{0!}n_{(0)} + \frac{1}{1!}n_{(1)} + \frac{1}{2!}n_{(2)} = 1 + n + \frac{n(n-1)}{2} = \frac{1}{2}(n^2 + n + 2).$$

This is just the formula we got in Section 2.6. Once again, we have not proved it — time for induction — but we have obtained it more easily than before.

The moral of this subsection is: If you suspect that a sequence is polynomial, make a difference table from it. If you get a row of zeros, your suspicion is right (at least for as many terms as you put in your top row) and you can compute the particular polynomial easily.

## Differences and Recurrences

We can now explain why recurrences, for instance  $a_n = a_{n-1} + a_{n-2}$ , are called difference equations. The reason is that they can all be rewritten to involve  $\Delta$  and the single index  $n$ .

To see this, first rewrite the recurrence in ascending indices:

$$a_{n+2} = a_{n+1} + a_n. \quad (15)$$

Now note that, by definition of  $\Delta$ ,

$$a_{n+1} = a_n + \Delta a_n. \quad (16)$$

This equation shows how to eliminate  $a_{n+1}$ . To eliminate  $a_{n+2}$ , raise the index on Eq. (16) by one and then substitute it into itself as follows.

$$\begin{aligned} a_{n+2} &= a_{n+1} + \Delta a_{n+1} \\ &= (a_n + \Delta a_n) + \Delta(a_n + \Delta a_n) \\ &= a_n + \Delta a_n + \Delta a_n + \Delta(\Delta a_n) \quad [\Delta(c_n + d_n) = \Delta c_n + \Delta d_n] \\ &= a_n + 2\Delta a_n + \Delta^2 a_n. \end{aligned}$$

By induction it follows that every term  $a_{n+k}$  can be rewritten using powers of  $\Delta$  and  $a_n$ . In this particular example we need go no further. Eq. (15) becomes

$$a_n + 2\Delta a_n + \Delta^2 a_n = a_n + \Delta a_n + a_n,$$

or

$$\Delta^2 a_n + \Delta a_n - a_n = 0.$$

Let us call this sort of difference equation — powers of  $\Delta$  and a fixed index — a  **$\Delta$ -equation**.

To go the other way, from a  $\Delta$ -equation to a recurrence, begin with  $\Delta a_n = a_{n+1} - a_n$  and proceed by induction to find non- $\Delta$  expressions for  $\Delta^2, \Delta^3$ , etc. See [26].

## Differences and Differentials

There's one last thing to do in this section. The study of the difference operator is called “finite differences”. Why not just “differences”? The answer is historical:

to contrast this subject with “infinitesimal differences”, which got started a little later. Note that we can uglify  $\Delta a_n$  without changing its value by writing it as

$$\frac{a_{n+h} - a_n}{h}, \quad \text{where } h = 1.$$

This uglification, like many, actually has some use. Think of the index  $n$  as representing time. We now see that  $\Delta a_n$  can be interpreted not only as the *amount* of change from time  $n$  to time  $n + h$ , but as the *average rate* of change. But why restrict  $h$  to be 1? If we take it to be some other amount, we get the  **$h$  (forward) difference**, denoted  $\Delta_h$ . If we take it to be *infinitesimal* — infinitely tiny but not 0 — we get the idea of *derivative* in calculus. This relationship between differences and derivatives helps explain why differencing and antidifferencing play precisely analogous roles in discrete mathematics to the roles of differentiation and integration in calculus.

## Problems: Section 5.11

---

We recommend doing the finite difference computations in this set by hand. While in real life we recommend using a CAS for many problems, doing these text problems by hand will help you understand how finite differences work.

1. ⟨1⟩ Evaluate

- a)  $\Delta(n-1)^2$
- b)  $\Delta n!$
- c)  $\Delta 2^n$
- d)  $\Delta c^n$  ( $c$  constant).

2. ⟨2⟩ Evaluate

- a)  $\Delta n 2^n$
- b)  $\Delta^2 n 2^n$
- c)  $\Delta^k n 2^n$ .

3. ⟨1⟩ What is  $\Delta c_{(n)}$ ? ( $c$  fixed,  $n$  variable)

4. ⟨1⟩ Evaluate  $\Delta^k n^k$ . ( $\Delta n^k$  is a complicated expression, but  $\Delta^k n^k$  is much simpler!)

5. ⟨2⟩ Evaluate, where  $n$  is the variable,

- a)  $\Delta \binom{n}{k}$
- b)  $\Delta^2 \binom{n}{k}$
- c)  $\Delta^j \binom{n}{k}$ .

6. ⟨2⟩ Evaluate

- a)  $\Delta \frac{1}{n}$
- b)  $\Delta^k \frac{1}{n}$ .

7. ⟨1⟩ In Example 3 we showed that

$$\sum_{k=0}^n k^2 = \frac{1}{3}(n+1)_{(3)} + \frac{1}{2}(n+1)_{(2)}.$$

Show that the RHS equals  $\frac{1}{6}(2n+1)(n+1)n$ , which may be a more familiar formula for the sum of squares.

8. ⟨2⟩ Find antidifferences for

- a)  $2n$
- b)  $n^2 - n$
- c)  $(n-1)^2$
- d)  $2n_{(3)}$
- e)  $2^n$
- f)  $3^n$ .

9. ⟨3⟩ Evaluate the following sums using antidifferences and falling factorials.

- a)  $\sum_{k=0}^n (k^2 - k)$
- b)  $\sum_{k=1}^n (k^2 + k)$
- c)  $\sum_{i=0}^n i^3$
- d)  $\sum_{k=2}^{2m} k(k-2)(k-4)$

10. ⟨1⟩ State an explicit formula for

- a)  $\sum_{i=0}^n i_{(k)}$
- b)  $\sum_{k=1}^n k_{(m)}$
- c)  $\sum_{k=0}^{n-1} k_{(m)}$ .

11. ⟨2⟩ While we have applied the difference operator to sequences only, it is easy to state the definition so that it applies to functions with domain  $R$ :

$$\Delta f(x) = f(x+1) - f(x).$$

Restate Theorems 2, 4 and 6 using function notation.

12. ⟨2⟩ The recursive definition for  $\Delta^k$  we gave in display (11) starts at  $\Delta^1$ , yet later in the section we referred to  $\Delta^0$ . Come up with a revised two-line recursive definition that includes  $\Delta^0$ .

13. ⟨1⟩ The proof of Theorem 1 isn’t right for  $k = 0$ , since then there are no factors to pull out. In fact, the statement of the theorem doesn’t immediately make sense for  $k = 0$ , because we haven’t defined  $n_{(-1)}$ . But it doesn’t matter: however we might define  $n_{(-1)}$  (see [22]), the *statement* of Theorem 1 is correct for  $k = 0$ . Why?

14.  $\langle 4 \rangle$  In the text, we assumed that all sequences start with  $a_0$ . But  $a_1$  is just as likely, and other starting points are possible. Fortunately, changing the index of the initial term affects only two parts of this section, the material on sums and the material on difference tables.

a) Show that the effect on sums is minimal. Yes, we want to define  $s_n$  as  $\sum_{k=1}^n a_k$  instead of  $\sum_{k=0}^n a_k$ , but show that the statement of Theorem 4 is correct without change.

b) The subsection on difference tables is affected more. Revise and prove Theorem 6 so that it applies to tables whose left-most entries are  $a_m, \Delta a_m, \Delta^2 a_m, \dots$ .

15.  $\langle 2 \rangle$  In some ways, an even better basis for forward differences than the factorial functions are the combinations viewed as polynomials:  $p_k(x) = C(x, k)$ . How are they better? Rewrite Newton's Interpolation Formula using  $C(x, k)$ .

16.  $\langle 1 \rangle$  Fill in the stars in the following difference table

$$\begin{array}{ccccccc} 3 & * & * & * & * & * & * \\ 5 & 4 & 2 & 8 & -2 & 6 \end{array}$$

17.  $\langle 2 \rangle$  Each of the following sequences gives the values  $p(0), p(1), p(2), \dots$  for a polynomial. Use difference tables to find a formula for  $p(n)$  assuming that the patterns shown below continue for subsequent terms.

a) 0, 2, 6, 12, 20, 30, . . .

b) 0, -1, 4, 21, 56, 115, . . .

c) 1, 5, 7, 7, 5, 1, . . .

18.  $\langle 2 \rangle$  It is widely believed that Newton discovered the formula for distance traveled by a free-falling object as a function of time. Actually, Galileo discovered the right formula 50 years before Newton's work, by collecting data and seeing a pattern. What Newton did was devise theories (the physics theory of forces and the mathematics theory called calculus) that explained *why* Galileo's formula is right.

Here is data Galileo might have collected. It represents the distance an object has fallen by time  $t = 0, 1, 2, 3, 4$  seconds. Help Galileo by finding a formula.

$$0, 16, 64, 144, 256$$

19.  $\langle 2 \rangle$  Consider  $f(x) = \lfloor x \rfloor$  and let  $a_n = f(n)$ . The difference table for  $\{a_n\}$  reaches a row of 0's very

quickly, but  $f(x)$  is not a polynomial. How can you reconcile this with Theorem 6?

20.  $\langle 3 \rangle$

a) Consider the finite sequence 2, 0, -2, 2. Compute the difference table. You'll get down to a single entry in row 3 and it won't be 0. Still, compute  $p(x)$  according to Eq. (14). Verify that  $p(x)$  interpolates this sequence, that is,

$$p(0) = 2, \quad p(1) = 0, \quad p(2) = -2, \quad p(3) = 2.$$

b) Prove the interpolation version of Theorem 6: For any numbers  $a_0, a_1, \dots, a_k$ , there is a polynomial  $p(x)$  of degree at most  $k$  such that  $p(j) = a_j$  for  $j = 0, 1, \dots, k$ , namely

$$p(x) = \sum_{j=0}^k \frac{\Delta^j a_0}{j!} x^{(j)}.$$

*Hint:* This isn't exactly Theorem 6 as stated, because that statement assumes you get to a row of zeroes.

21.  $\langle 3 \rangle$  Intelligence test makers often create problems where they give you a finite sequence and ask you for the next term. For instance, what's the next term for 1, 4, 9, 16?

Why, 37 of course. The pattern is obvious:  $a_n = .5(x^4 - 10x^3 + 37x^2 - 50x + 24)$ . Check it out for  $n = 1, 2, 3, 4, 5$ .

a) Show how to get our formula for  $a_n$ .

b) State the anti-testmaker theorem behind this example.

22.  $\langle 2 \rangle$  (Falling factorial functions of negative order) Let  $k$  be a positive integer. Then  $x_{(-k)}$  is defined by

$$x_{(-k)} = \frac{1}{x(x+1)(x+2)\cdots(x+k-1)}. \quad [k \text{ factors}]$$

a) Evaluate  $\Delta x_{(-1)}$

b) Evaluate  $\Delta x_{(-2)}$

c) Show that Theorem 1 holds for all negative order factorial functions.

23.  $\langle 2 \rangle$  Evaluate in closed form

$$\begin{aligned} \text{a) } & \sum_{k=1}^n \frac{1}{k(k+1)} & \text{b) } & \sum_{n=4}^{16} \frac{1}{n(n+1)(n+2)} \\ \text{c) } & \sum_{j=1}^n j \cdot j! \end{aligned}$$

24.  $\langle 1 \rangle$  Translate the following recurrences into equivalent  $\Delta$ -equations.

a)  $a_n = a_{n-1} + 2a_{n-2}$

b)  $b_n = b_{n-1}b_{n-2}$

25. ⟨1⟩ Translate the following  $\Delta$ -equations into recurrences:

a)  $\Delta a_n = na_n$

b)  $b_n = \Delta b_n - \Delta^2 b_n$

c)  $c_n^2 + c_n \Delta c_n = 1.$

26. ⟨3⟩ Find and prove a formula for  $\Delta^k a_n$  in terms of  $a_n, a_{n+1}, \dots, a_{n+k}$ , for  $k = 1, 2, 3, \dots$ .

27. ⟨3⟩ For  $k = 1, 2, 3, \dots$  find and prove a formula for  $a_{n+k}$  in terms of  $a_n, \Delta a_n, \Delta^2 a_n, \dots$ .

28. ⟨4⟩ Show that, with the correspondence we have set up between recurrences and  $\Delta$ -equations, a  $\Delta$ -equation is linear iff the corresponding recurrence is linear. Is this still true if we add the word “homogeneous”? What about adding “constant coefficient”?

29. ⟨1⟩ The  **$k$ th rising factorial function** is defined for  $x$  arbitrary and positive integers  $k$  by

$$x^{(k)} = x(x+1)(x+2) \cdots (x+k-1).$$

By convention,  $x^{(0)} = 1$ . Evaluate

- a)  $4^{(3)}$       b)  $(n-1)^{(2)}$       c)  $4_{(3)}$   
d)  $(n-1)_{(2)}$       e)  $\Delta n^{(2)}$       f)  $\Delta n^{(k)}$ .

30. ⟨1⟩ The **backward difference** of a sequence  $\{a_n\}$  is defined to be the sequence whose  $n$ th term, denoted  $\nabla a_n$ , is

$$\nabla a_n = a_n - a_{n-1}.$$

Evaluate

- a)  $\nabla n^2$       b)  $\nabla n!$       c)  $\nabla 2^n$   
d)  $\nabla(A n^2 - B n)$       e)  $\nabla n^{(3)}$       f)  $\nabla n^{(k)}$   
g)  $\nabla n_{(3)}$       h)  $\nabla n_{(k)}$ .

31. ⟨2⟩ Backward differences work a little better with sums than forward differences, because if  $s_n = \sum_{k=0}^n a_n$ , then

$$\nabla s_n = a_n,$$

which is a little nicer than  $\Delta s_n = a_{n+1}$ . However, as you discovered in [30], backward differences work better with rising factorials than falling factorials. Do Example 3 over again using rising factorials and anti-backward-differences.

32. ⟨3⟩ State and prove a backward difference analog of Theorem 4.

33. ⟨3⟩ Let  $k$  be a positive integer. What is the right definition of  $x^{(-k)}$  so that it “behaves properly” with backward differences?

34. ⟨4⟩

- a) Prove by induction (or otherwise) that every polynomial can be expressed as a sum of multiples of falling factorials. If you use induction, let  $Q(k)$  be the proposition that any degree  $k$  polynomial can be so expressed.

- b) Write each of the following polynomials as a sum of falling factorials.

- i)  $-4n^2 + 3n - 5$   
ii)  $2n^3 + 3n^2 - 2n - 6$

- c) Derive a formula for, or an algorithm to compute, the  $c_j$  in Eq. (2) in terms of the coefficients  $q_j$  of  $p(n)$ , when it is written using powers, i.e.,  $p(n) = \sum_{j=0}^k q_j n^k$ . If you go for a formula, you might first get special formulas for the cases  $k = 2$ , as in part b-i), and  $k = 3$ , as in part b-ii).

35. ⟨3⟩ The trouble with the statement of Theorem 6 is that it gives you no clue where the formula (14) comes from. There is an additional argument that produces the formula. Namely, conjecture that there is *some* polynomial  $p(x)$  that matches  $\{a_n\}$  and that it can be written as some linear combination of the factorial functions  $x_{(0)}$  through  $x_{(k)}$ :

$$p(x) = \sum_{j=0}^k c_j x_{(j)}. \quad (17)$$

(If you have already done [34] you already know that every polynomial has this form, so the only thing you are left conjecturing is that there *is* a polynomial of degree  $k$  that matches  $\{a_n\}$ .) Then the open issue is: what are the  $c_j$ ? By taking differences in Eq. (17) and evaluating at  $x = 0$ , show that the only candidate for  $c_j$  is  $\Delta^j a_0 / j!$ . Theorem 6 now confirms your conjecture.

36. ⟨3⟩ (assumes calculus) Many results in this section are direct analogs of theorems in calculus books. But one theorem we have not stated an analog to is the Fundamental Theorem of Calculus. State and prove the analog.

## Supplementary Problems: Chapter 5

1. ⟨3⟩ The Fibonacci recurrence can be used to work backwards to define  $F_{-1}, F_{-2}$ , etc., by writing the recurrence as

$$F_{n-2} = F_n - F_{n-1}.$$

Thus  $F_{-1} = 1 - 1 = 0$ ,  $F_{-2} = 1 - 0 = 1$ , and so on. Compute some more terms. Prove what you find.

2. ⟨3⟩ Consider the recurrence of Example 3, Section 5.5:

$$a_n = a_{n-1} + 2a_{n-2}, \quad a_0 = a_1 = 3.$$

Show that this recurrence can be worked backwards in a unique way and compute the terms  $a_{-1}$  to  $a_{-10}$ . Check that the explicit formula obtained in Section 5.5 works for these negatively-indexed terms as well. Why was this bound to happen?

3. ⟨4⟩ Let  $a_n$  satisfy the recurrence

$$a_1 = 2, a_2 = 8 \quad \text{and} \quad a_{n+1}a_{n-1} = 2(a_n)^2.$$

Compute some terms, conjecture an explicit formula for  $a_n$ , and prove it. Does Theorem 1, Section 5.4 apply? If not, why can you be sure that the solution is unique?

4. ⟨2⟩ Suppose the sequence  $\{a_n\}$  repeats itself twice, i.e., for some  $p > 1$ ,  $a_p = a_0$  and  $a_{p+1} = a_1$ . Suppose further that  $\{a_n\}$  satisfies some second-order linear, constant coefficient homogeneous recurrence. Show that  $\{a_n\}$  is periodic.

5. ⟨3⟩ Show that for any second-order linear difference equation whose characteristic equation has distinct roots, there is a unique solution for any two initial values  $a_k$  and  $a_{k+2}$ , so long as one root is not the negative of the other.

6. ⟨4⟩ In [13, Section 4.5], you were asked to write an algorithm to compute the number  $C(n, k)$  by recursively computing earlier entries in Pascal’s Triangle. Assume that this algorithm is set up using a recursive function also named  $C(n, k)$ . Determine how many times this function is called (including the main call) to compute  $C(n, k)$ . Hint: Set up a doubly-indexed difference equation and solve it.

7. ⟨3⟩ Recall that a spanning tree of an undirected graph is a subgraph which is a tree and which includes all the vertices.

- a) The double cycle  $C_n^2$  is a cycle of length  $n$  except that each edge of the cycle is replaced by two identical edges; consider Fig. 5.15a. How many spanning trees does  $C_n^2$  have? Assume that each edge and vertex is labeled, so that trees that look the same but don’t use exactly the same edge or vertex sets are different.

- b) Let  $G_n$  be the graph with vertex and edge sets

$$V = \{x, y, 1, 2, \dots, n\},$$

$$E = \{\{k, x\}, \{k, y\} \mid k = 1, 2, \dots, n\}.$$

See Fig. 5.15b. How many spanning trees does  $G_n$  have? Again assume that everything is labeled.

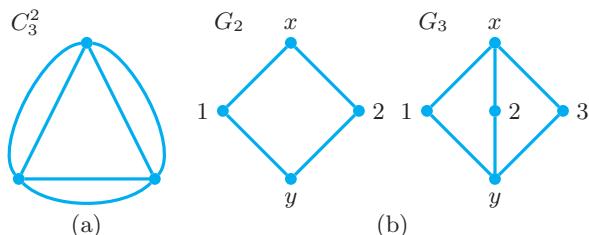


FIGURE 5.15

8. ⟨4⟩ **Annuities** are loans in reverse. You pay a lump sum  $P$  to an insurance company and the insurance company pays you back with interest in the form of a fixed payment  $p$  at the end of each period, usually a month. With “annuities certain” the payment is made for a fixed number of periods (to your heirs if you die in the meantime), at the end of which your “loan” is completely paid back. But more common is an annuity in which you are paid  $p$  per period until you die. This adds an element of chance — for the insurance company. It has to figure out how much  $p$  should be so that it has just enough money to pay everybody. (For simplicity, we’ve ignored profit and operating costs.) The company figures this out by grouping annuitants by age at the beginning of the contract, and by using statistics on mortality.

Consider a group of  $N$  people who buy annuities (payments until death) when they turn 65. Let  $m_n$  be the probability that a person at age 65

will live at least  $n$  more payment periods (months). We claim that an appropriate recurrence for computing the annuity payment  $p$  to persons in this group is

$$P_n = P_{n-1}(1+r) - m_n p,$$

where  $P_n$  is the total amount of money the company has left after payouts during the  $n$ th month, divided by  $N$ . Put another way,  $P_n$  is the average amount left per original annuitant, even though some of these people may have already died. By total amount of money we mean the amount remaining from the original amount  $NP$  received in premiums, in light of interest earned since then and monthly payments already made.

Explain why this recurrence is reasonable. Assume (as insurance companies do) that everybody dies by age 100, so that all the money should be paid back by then. Find a formula involving a summation for the value of  $p$  in terms of  $P$ ,  $r$ , and the various  $m_n$ , where  $p$  is the period payment to annuitants who pay in  $P$  the day they turn 65 and  $r$  is the monthly interest rate.

- 9.**  $\langle 4 \rangle$  It seems hard to believe that the RHS of Eq. (7), Section 5.5, will always be an integer, let alone a Fibonacci number. Prove that, for any integer  $a$  and any positive integers  $b$  and  $n$ , both

$$\left( \frac{a + \sqrt{b}}{2} \right)^n + \left( \frac{a - \sqrt{b}}{2} \right)^n \quad (18)$$

and

$$\frac{1}{\sqrt{b}} \left[ \left( \frac{a + \sqrt{b}}{2} \right)^n - \left( \frac{a - \sqrt{b}}{2} \right)^n \right] \quad (19)$$

are integers — so long as  $a^2 - b$  is divisible by 4.

*Note:* This problem is quite a challenge if you attack it directly, but if you think of expressions (18) and (19) as terms of two sequences and work backwards to a recurrence for the sequences, then the problem is not very hard at all. This problem illustrates an important point. We are all trained to seek closed forms, such as those above. The underlying reason for this training is the assumption that, whatever knowledge we seek, it will be easier to obtain from a closed form. However, sometimes this is simply false! Sometimes a recurrence is not only sufficient to yield the information, but it also yields the information more easily.

- 10.**  $\langle 5 \rangle$  For given integers  $a$  and  $b$ , let  $P(a, b)$  be the proposition that expressions (18) and (19) in [9]

are integers for *any* integer  $n$ , not just for  $n$  a positive integer.

- a)** Prove  $P(1, 5)$ . (*Note:* When  $a = 1$  and  $b = 5$ , expression (19) is the  $(n-1)$ st Fibonacci number.)
- b)** Find general conditions relating  $a$  and  $b$  for which  $P(a, b)$  is true.

- 11.**  $\langle 3 \rangle$  Let  $C_n$  be the number of calls of the function  $F$  needed to compute  $F(n)$  in Algorithm FIBB, Section 5.8. For instance,  $C_0 = 1$ , since there are no calls except the initial call,  $F(0)$ . Find a formula for  $C_n$ .

- 12.**  $\langle 4 \rangle$  Refine [11] as follows. Let  $C_{n,k}$  be the number of calls of  $F(k)$  invoked by Algorithm FIBB when you initially set out to compute  $F(n)$ . For instance,  $C_{0,0} = 1$  and  $C_{2,0} = 1$ . Find a formula for  $C_{n,k}$ .

- 13.**  $\langle 2 \rangle$  From the definitions in [11, 12] we have

$$C_n = \sum_{k=0}^n C_{n,k}.$$

In light of the results of those two problems, this identity becomes an interesting formula concerning the Fibonacci numbers. Can you prove this Fibonacci formula directly?

- 14.**  $\langle 4 \rangle$  Let  $F_k$  be the  $k$ th Fibonacci number. It is a fact that

$$F_n = F_k F_{n-k} + F_{k-1} F_{n-k-1}.$$

- a)** Try to prove this identity by induction on  $n$ .
- b)** Try to prove it by induction on  $k$ .

- 15.**  $\langle 4 \rangle$  For the Fibonacci numbers, prove that

$$F_{2n} = F_n^2 + F_{n-1}^2.$$

If you get stuck, use [14].

- 16.**  $\langle 3 \rangle$  Suppose that national income in any period is the average of national income in the previous two periods and the income in the following one period. Model this relationship with a difference equation and find the general solution. Discuss the qualitative behavior of solutions. For instance, is explosive growth possible? Is oscillation towards a limit possible? Is constant income possible? Are any of these likely?

- 17.**  $\langle 4 \rangle$  Repeat [16], but make national income the average of national income in the previous two periods and the following *two* periods. (*Note:* The characteristic polynomial can be factored.)

18. ⟨2⟩ Consider the recurrence

$$r_n = r_{n-1} + r_{n-2} + \cdots + r_1 + r_0 \quad \text{for } n > 0, \\ r_0 = 1.$$

This is an example of a **variable order** linear difference equation, because the number of terms varies with  $n$ . In many cases it is possible to reduce such problems to fixed order linear difference equations by a substitution. For instance, in this example,

$$r_n = r_{n-1} + (r_{n-2} + \cdots + r_0) \\ = r_{n-1} + r_{n-1}$$

So we are now down to a first-order equation. Note, however, that the first-order recurrence is only valid for  $n \geq 2$ . Why? What, therefore, is the solution to the original problem?

*Note:* With this method, the simpler recurrence typically applies only after a few terms.

19. ⟨5⟩ Solve the following variable order recurrences (see [18]).

- a)  $r_n = 1 + \sum_{k=1}^{n-1} r_k$  for  $n > 1$ ,  $r_1 = 1$ .
- b)  $r_n = r_{n-2} + r_{n-3} + \cdots + r_0$  for  $n \geq 2$ ,  
 $r_1 = r_0 = 1$ .
- c)  $r_0 = 1$  and for  $n > 0$ ,

$$r_n = r_{n-1} + r_{n-3} + r_{n-5} + \cdots + r_\alpha,$$

where  $\alpha = 0$  if  $n$  is odd and  $\alpha = 1$  if  $n$  is even.

- d)  $r_n = \sum_{k=1}^n kr_{n-k}$  for  $n > 0$ ,  $r_0 = 1$ .

20. ⟨4⟩ Solve the following variable order nonhomogeneous recurrences. See [18].

- a)  $a_n = 1 + \sum_{k=1}^n a_{n-k}$  for  $n > 0$ ,  $a_0 = 1$ .
- b)  $b_n = n + \sum_{k=1}^n b_{n-k}$  for  $n > 0$ ,  $b_0 = 1$ .
- c)  $c_n = 2^n + \sum_{k=1}^n c_{n-k}$  for  $n > 0$ ,  $c_0 = 1$ .

21. ⟨3⟩ Here is a device that usually works for solving

$$a_n = \sum_{j=1}^k c_j a_{n-j} + d$$

using only the homogeneous theory. Namely, there is usually a constant  $c$  so that, after adding  $c$  to both sides and rewriting the RHS, the equation becomes

$$a_n + c = \sum_{j=1}^k c_j (a_{n-j} + c).$$

Define  $b_n = a_n + c$  and rewrite the recurrence in terms of  $b$ 's. The general solution for  $b_n$  may now

be obtained by the homogeneous theory. Then subtract  $c$  to get the general solution for  $a_n$ .

- a) Apply this trick to solve

$$a_n = a_{n-1} + 2a_{n-2} + 4.$$

- b) When is it impossible to solve for  $c$ ?

- c) This trick works for difference equations in several variables as well. You happen to know the solution to

$$C(n, k) = C(n-1, k) + C(n-1, k-1), \\ C(n, 0) = C(n, n) = 1.$$

Now find the solution to

$$D(n, k) = D(n-1, k) + D(n-1, k-1) + 1, \\ D(n, 0) = D(n, n) = 1.$$

22. ⟨4⟩ Here is a Divide and Conquer method for evaluating a polynomial, described through an example. To evaluate

$$a_3x^3 + a_2x^2 + a_1x + a_0,$$

rewrite it as

$$(a_3x + a_2)x^2 + (a_1x + a_0).$$

That is, reduce evaluating a four-term polynomial to evaluating two-term polynomials, and continue recursively. Compute the number of multiplications needed to evaluate a polynomial with  $2^k$  terms by this method if

- a) The multiplying factors such as  $x^2$  outside the parentheses are computed by brute force.
- b) Instead they too are computed using Divide and Conquer.

23. ⟨4⟩ Here is another Divide and Conquer method for polynomial evaluation. To evaluate

$$a_3x^3 + a_2x^2 + a_1x + a_0,$$

rewrite it as

$$(a_3x^2 + a_1)x + (a_2x^2 + a_0). \tag{20}$$

(Note that this is not the same regrouping as in [22]. Here we have factored a single power of  $x$  out of half the terms, leaving two polynomials in  $x^2$ .)

- a) Compute the number of multiplications needed to evaluate a polynomial with  $2^k$  terms by this method.

- b) Suppose that you want to evaluate a polynomial with  $2^k$  terms at  $x = \pm 1$ . Do you see that Eq. (20) alone (that is, the first level of the recurrence, without going further) already saves work? Explain. (This is the key idea of the **Fast Fourier Transform**. If the polynomial is to be evaluated at complex roots of unity, the saving at the first-level breakdown can be repeated at later levels as well. Complex roots of unity are the complex solutions of  $x^n = 1$  with  $n$  a positive integer.)

24. {4} Suppose a binary tree has the property that at each internal (non-leaf) node, the longest path going left from that node is one edge longer than the longest path going right. For instance, if a node has one child, it must be a left child and a leaf.

- a) Give two examples of such binary trees.  
 b) Show that for each  $n \geq 0$ , there is a unique binary tree with this property and such that the longest path down from the root contains  $n$  edges.  
 c) Let  $a_n$  be the number of nodes in the unique tree in b). Find a formula for  $a_n$ .

25. {3} Consider the two-index recurrence

$$\begin{aligned} a_{0,0} &= 1, \\ a_{n,k} &= 0 \quad \text{if } k < 0 \text{ or } k > n, \\ a_{n,k} &= \frac{1}{2}(a_{n-1,k-1} + a_{n-1,k}) \quad \text{otherwise.} \end{aligned}$$

Guess a formula for  $a_{n,k}$  by computing some values. Prove it by induction.

26. {4} Define the sequence of functions  $\{F_n(k)\}$ , where  $F_n$  has domain the integers from 0 to  $n$ , by

$$\begin{aligned} F_n(n) &= p^n \quad \text{for } n \geq 0, \\ F_n(0) &= q^n \quad \text{for } n \geq 0, \\ F_n(k) &= pF_{n-1}(k-1) + qF_{n-1}(k) \quad \text{otherwise,} \end{aligned}$$

where  $p$  and  $q$  are constants. Discover an explicit formula for  $F_n(k)$ . Prove it by induction.

27. {3} Let

$$w = \frac{-1 + i\sqrt{3}}{2}, \quad \bar{w} = \frac{-1 - i\sqrt{3}}{2}.$$

(These are the two complex cube roots of 1.)

- a) Compute the first several complex numbers in the sequence defined by

$$a_n = 1 + i\sqrt{3}w^n - i\sqrt{3}\bar{w}^n.$$

What pattern do you observe?

- b) Show how to reach the same conclusion without messy computations by first working backwards from Eq. (21) to the difference equation for  $a_n$ .

28. {5} Suppose  $\{a_n\}$  is a sequence of real numbers that satisfy  $a_n + ba_{n-1} + ca_{n-2} = 0$ , where  $b$  and  $c$  are real numbers and  $c > b^2/4$ .

- a) Show there are real constants  $A$ ,  $B$ , and  $\theta$  such that

$$a_n = c^{n/2}(A \sin n\theta + B \cos n\theta).$$

- b) Show there are also real constants  $D$  and  $\alpha$  so that  $a_n$  has the alternative formula

$$a_n = c^{n/2}(D \sin(n\theta + \alpha)).$$

This equation says every solution is a discrete sinusoidal motion with a “phase shift”,  $\alpha$ , and a multiplier,  $c^{n/2}$ , that causes explosive, steady, or damped behavior.

The next several problems guide you through the theory of partial fractions, which is used in the generating function approach to CCLDEs introduced in the last part of Section 5.5.

29. {3} Let  $P(x)$  be a linear polynomial  $ax + b$  and let  $Q(x)$  be a quadratic polynomial  $cx^2 + dx + e$ , which factors into  $c(x-r)(x-s)$ , with  $r \neq s$ . We want to find constants  $A$  and  $B$  such that

$$\frac{P(x)}{Q(x)} = \frac{A}{x-r} + \frac{B}{x-s},$$

with  $A$  and  $B$  expressed in terms of  $a$ ,  $b$ ,  $c$ ,  $r$ , and  $s$ .

- a) Find  $A$  and  $B$  by putting the right-hand side over a common denominator and equating like powers of  $x$  in the two numerators. Show that what you have done *proves* that the expansion of  $P(x)/Q(x)$  on the right-hand side, called the **partial fraction expansion**, actually exists.  
 b) Then find  $A$  and  $B$  by multiplying both sides of the equation by  $(x-r)$  and then setting  $x = r$  to find  $A$ . Similarly, multiply by  $(x-s)$  to find  $B$ .  
 c) Apply the methods of both a) and b) to i) and ii).  
 i)  $(2x-5)/(x^2-5x+6)$   
 ii)  $(-3x+7)/(x^2+4x)$

- 30.** **(4)** Let  $P_i(x)$  and  $Q_j(x)$  be polynomials of degree  $i$  and  $j$ , respectively, with  $i < j$  and suppose that  $Q_j(x)$  factors into

$$c(x-r_1)(x-r_2) \cdots (x-r_j),$$

where all the  $r_k$ 's are distinct.

- a) By generalizing what you did in [29b], find formulas for the constants  $A_k$ ,  $k = 1, \dots, j$ , such that

$$\frac{P_i(x)}{Q_j(x)} = \sum_{k=1}^j \frac{A_k}{x - r_k}.$$

This expansion is called the partial fraction expansion of  $P_i(x)/Q_j(x)$ .

- b) Use the technique in [29b] to find the partial fraction expansions of:

i)  $(2x^2 - 4x + 7)/[(x-2)(x-3)(x-5)]$

ii)  $\frac{-3x^3 + 4x^2 - 7}{(x-2)(x-4)(x-6)(x-8)(x-10)}$ .

- 31.** **(4)** Things get more complicated when  $Q_j(x)$  has repeated factors or when it has quadratic factors which can't be decomposed into real linear factors. (One version of the Fundamental Theorem of Algebra says that *any* polynomial with real coefficients can be factored into a product of real linear and real quadratic factors.)

- a) If  $Q(x)$  has a repeated linear factor such as  $(x-r)^k$ , the corresponding terms in the partial fraction expansion are

$$\frac{A_1}{x-r} + \frac{A_2}{(x-r)^2} + \cdots + \frac{A_k}{(x-r)^k}.$$

Show why the technique to find the  $A_i$  in [29b] doesn't work in this case.

- b) However, the technique in [30a] still works. Use it to find the partial fraction expansions of:

i)  $(x-5)/[(x-1)(x-2)^2]$

ii)  $(2x^3 - 3x^2 + 4x - 1)/[(x-3)(x-4)^3]$ .

- 32.** **(3)** If the denominator polynomial factors into quadratics which cannot be further factored into real linear factors and if these quadratic factors

are not repeated, the corresponding term in the partial fraction expansion is

$$\frac{ax+b}{cx^2+dx+e}.$$

- a) How can you determine whether  $cx^2 + dx + e$  can be factored into real linear factors? (This question is really a test of whether you remember how to solve quadratic equations.)
- b) Find the partial fraction expansion of:
- i)  $1/[(x-3)(2x^2-4x+5)]$
  - ii)  $(2x+3)/[(x-1)^2(x^2+3x-7)]$ .
- c) What do you suppose happens in the partial fraction expansion when the quadratic factor is repeated? Reason by analogy with what happens for real linear factors (no proof requested).

- 33.** **(4)**

- a) If the degree of the numerator polynomial is greater than or equal to the degree of the denominator polynomial, the method of partial fractions still works, but we can't apply it directly to  $P(x)/Q(x)$ . The argument you used in [29a] to show that the expansion exists won't work now. Why?
- b) If we first perform a long division on  $P(x)/Q(x)$ , we can then do a partial fraction expansion of the remainder. Why?
- c) Apply the idea in b) to:
- i)  $(2x^2 - 4x + 5)/(x^2 + 3x + 7)$
  - ii)  $(4x^4 - 6x^2 + 3x - 5)/(2x^2 - 3x + 2)$ .

- 34.** **(3)** Use the results of [30–33] to state an algorithm to find the partial fraction expansion of *any* rational function (i.e., any ratio of two polynomials). Don't try to make your algorithm too detailed. Just state it in terms of the steps that must be carried out, taking into account the degrees of the numerator and denominator and all the possible factors which the denominator may have. Assume you have a procedure which can factor a polynomial and another procedure which finds the coefficients once the correct sort of partial fraction expansion is identified.

# Probability

## 6.1 Introduction

---

Probability theory is one of the most broadly useful parts of mathematics. To give you a sense of its breadth, we start by stating several problems, each of which we solve at some point in the chapter. After stating them, we will discuss the role of probability in discrete mathematics.

### EXAMPLE 1

A 30-question multiple choice test (say, a section from the College Boards) gives five choices for each answer. You lose  $1/4$  as much credit for each wrong answer as you gain for each right answer. (Problems left unanswered neither gain nor lose you credit.) If you guess at random on questions for which you don't know the answer, what is the probability that you will raise your total score?

### EXAMPLE 2

Medical tests are not perfect. The standard tine test for tuberculosis attempts to identify *carriers* — people who have been infected by the tuberculin bacteria. However, 8% of carriers test negative (a “false negative”) and 4% of noncarriers test positive (a “false positive”). Suppose you take the tine test and get a positive result. What is the probability that you are a carrier? (A carrier need not be sick. The bacteria may have been killed off naturally, or they may be in a latent state, capable of causing serious illness later. However, it is very difficult to distinguish between these two cases, so identified carriers are usually treated.)

Even though tuberculosis is once again becoming a significant threat in advanced nations, this example may not grab you. But the issue of imperfect tests applies to almost any disease. Substitute AIDS for tuberculosis. If you tested positive, wouldn't you want to know exactly what the probability is that you have AIDS and what this probability means? (We use tuberculosis instead of AIDS because the data for tuberculosis are well established.)

### EXAMPLE 3

In Section 4.9, we gave several algorithms for constructing permutations of  $n$  objects. In the first algorithm, at each stage we simply picked one of the objects at random, rejecting it if it had already been picked. If  $k$  of the  $n$  objects have already been placed in the permutation, how many more picks should we expect (on average) before obtaining another previously unpicked object?

### EXAMPLE 4

You receive a form letter in the mail saying, “Congratulations, you may have already won an all-expenses-paid trip to Europe worth \$10,000.” A lottery number is printed on an entry form with your letter. If you return this form to the sponsors — and if the number is the one that has already been chosen by their computer — you win the prize. Should you return the form? (Remember, you will have to pay the postage.)

### EXAMPLE 5

Two people play the following version of “Gambler’s Ruin”. They each start with a certain number of dollars and repeatedly flip a fair coin. Each time the coin comes up heads, person  $A$  gives person  $B$  \$1. Each time it comes up tails,  $B$  gives  $A$  \$1. This continues until one of them is broke (ruined). If  $A$  starts with \$3 and  $B$  starts with \$5, what is the probability that  $A$  goes broke?

### EXAMPLE 6

In Section 1.5 we considered Algorithm 1.12 for finding the largest number on a list  $x_1, \dots, x_n$ . For  $k$  from 2 to  $n$ , that algorithm compares  $x_k$  to the largest number so far, called  $m$ ; initially  $m \leftarrow x_1$  and if  $x_k$  is larger than  $m$ , then  $m \leftarrow x_k$ . Since the number of comparisons is always  $n - 1$ , analyzing this algorithm means answering the question: On average, how many times does  $m$  get assigned a new value?

### EXAMPLE 7

It is common practice in reporting the results of surveys, particularly of candidate preferences during election campaigns, to state the result in a form like the following: 1100 people were polled. 60% prefer candidate G, 40% prefer candidate B and the error in the poll is  $\pm 3\%$ . Now what can that possibly mean? Surely if the electorate is many thousands or millions and 53% of the electorate really favor G, it would have been *possible* to poll 1100 people of whom 60% favor G. So surely the error *could* be greater than 3%. What’s the explanation of this seeming anomaly?

Think about these examples. Only Example 5 was fun and games, and even it can be restated as a serious problem about “random walks” of gas molecules in long tubes, or about competition of biological species for fixed resources [3]. (If you are an addicted gambler, the problem is serious already!) Basically, life is full of situations where information is uncertain and various outcomes are possible. Probability theory is the means to cope with such uncertainty and thus all informed citizens ought to know something about it.

But why treat probability in *this* book? Only two of these examples, Examples 3 and 6, deal with issues we have considered previously.

There are two reasons. First, those two examples concern the core subject in this book, algorithms. For the analysis of algorithm efficiency, we have said that

the average case is most important but also the most difficult. But any analysis of distinct cases depends on the probability of occurrence of the various possible cases. Thus, this chapter gives you the tools to do average case analysis.

The second reason for including probability is this: As just noted, everyone needs to know something about it but not much probability is typically included elsewhere in the first two years of core American college math. It turns out that the discrete approach provides a natural and relatively easy access to the subject.

We should make clear, though, that probability theory is not a subset of discrete math. Like most branches of mathematics, it has both continuous and discrete aspects, and their interplay is often more powerful than both used separately. For instance, in Section 6.4 we will discuss the “binomial distribution”; it models the outcome of repeated coin tossing. As the number of tosses increases, this distribution approaches the famous “normal” distribution, which is continuous. It is usually easier, and sufficient, to model a problem approximately using the normal distribution than exactly with the binomial distribution. We discuss an application of this in Section 6.7.

There is a funny thing about probability. Everybody has an intuitive idea of what it means, but when it comes to getting quantitative, it is easy to go wrong. In many problems, two different arguments lead to very different answers, yet both arguments seem right. (We give one famous example, the second child problem, in [21], Section 6.3.)

When a subject is slippery in this way, a formal approach is called for. Thus in this chapter more than others, we state definitions precisely in displays and solve problems by carrying out calculations in detail using the definitions. (For discrete probability theory, all we need for such precision is the knowledge of sets and functions from Section 0.1.) This formal approach won’t guarantee that you won’t make mistakes (one of us once gave an incorrect analysis of the second child problem in class!), but it helps a lot.

This chapter is organized as follows. In Sections 6.2 through 6.6, we introduce the formulation that mathematicians have devised for discussing probability. In Section 6.2 we define “probability space”. Section 6.3 concerns “conditional probability”, where we show how to modify probabilities when additional information becomes available. In Section 6.4 we introduce “random variables”; despite the name, these are *functions* that allow one to reduce a lot of different-looking probability spaces to a few standard spaces.

In Section 6.5 we treat “expected value”, the generalization of average value. And then in Section 6.6 we discuss variance and standard deviation, which allow us to measure how close values are, on average, to the expected value. In Section 6.7 we use the tools we have developed to introduce the notion of statistical estimation.

In the remaining two sections of this chapter we get some payoffs from all that concept building. In Section 6.8 we treat average case analysis of algorithms by taking examples done intuitively in earlier chapters — we had no choice then but to do them intuitively — and doing them right. Thus in terms of feedback to the core topics of this book, Section 6.8 is the centerpiece of the chapter. We hasten to say, however, that in terms of the general usefulness of probability, payoffs appear

as early as Section 6.3 — we solve Example 2 there. Finally, in Section 6.9 we show that the connection between probability and our core topics is not all one way. The recursive paradigm, one of our core methods, turns out to be just the thing to solve some otherwise very hard types of probability problems, and in Section 6.9 we show how.

Two final points. First, if it struck you that some of the questions in Examples 1–7 are not well defined — you may feel they need some more information or at least a clearer specification of the assumptions before they can be solved — you are absolutely right. Indeed, specifying a question clearly and determining the information needed to get started are often the greater part of the battle in applying mathematics. We will define these questions better at appropriate places.

Second, in most textbooks most probability problems are about tossing coins, throwing dice, or picking balls from urns. These activities are less significant than those in our examples. But there is a reason for such textbook choices: Such situations are simpler to understand. Also, most people find them fun. We too will couch most of our examples in these terms. However, occasional examples like those we've introduced will remind you that probability is useful as well as fun.

## Problems: Section 6.1

---

1. ⟨2⟩ Examples 2 and 4 are among those for which not enough information is given.
  - a) For Example 2, suppose one person out of 100 is a tuberculosis carrier. Can you answer the question now?
  - b) For Example 4, suppose the number on your entry form is 7-493-206. Can you answer the question now?
2. ⟨1⟩ In Example 1, suppose you guess on exactly one problem. What is the probability that you increase your score? What's the probability if you guess on exactly two problems?
3. ⟨2⟩ Suppose two species, say birds, occupy the same “ecological niche”. This means they roost in the same areas, compete for the same food, etc. It is reasonable therefore to suppose that the total number of birds of both species that can be supported in a given locale is fixed. That is, if one more bird of the first species survives in the next generation, then one less of the second species survives. Do you see that we can model the situation with gambler’s ruin? If one species is “fitter” than the other, what simple change in the model accommodates this fact?

It is a theorem that, with probability 1, eventually one of the gamblers is ruined — even if the coin is fair. Thus if the gambling model is appropriate for biology, we conclude that we should never find two species occupying the same niche. (Why?) This conclusion has been confirmed by observation. Biologists refer to it as Gause’s Law.

4. ⟨2⟩ Suppose you are given the results of a poll of some subset of a population. Suppose also that the question asked has only two possible responses (e.g., Yes or No, Prefer candidate B or Prefer candidate G). Obviously the results of the poll itself (i.e., the fraction of those polled who give one answer or the other) will affect whether you believe that the results of the poll really reflect the opinion of the population as a whole. But what other factors besides the result itself will affect your confidence that the poll results are accurate for the whole population?
5. ⟨2⟩ The Monty Hall Problem: This problem, named after a game show host who used a version of it, created a controversy in 1991 that reached the front page of The New York Times on 21 July 1991. During the contretemps several mathematicians found that their intuition had played them

false. This problem comes in several flavors but the most common version is this: As a contestant on a game show, you are told that you have won a prize, either a car or a goat. To claim your prize you must open one of three identical doors. Behind one door is a new car and behind each of the other two is a goat. You must choose one door (which is not opened) and then the host, who knows what is behind each of the three doors, opens one of the other doors, always one with a goat. You may then

either stick with your original choice or switch your choice, after which your chosen door is opened to display your prize.

The question is: Should you stick with your original door, or should you switch, or doesn't it matter? If you can give a mathematical answer, fine. But if not, give an intuitive answer together with your reasoning. (We shall return to this problem in [11-13], Section 6.2.)

## 6.2 Probability Space

In this section we will set up the basic mathematical formulation for probability problems. In any such problem, there are various possible outcomes, or, as we shall call them, **events**. The key step is to identify the events that are indivisible in terms of the analysis you wish to do — these are called **atomic events** — and to represent every other event as a set of atomic events. These nonatomic events are called **compound events**. The set of all atomic events is called the **sample space**, denoted by  $S$ . One immediate advantage of this approach is that we may use set-theory concepts and notation. For instance, events are **disjoint** if they are disjoint as sets of atomic events.

Once we have our events, we can assign probabilities to them. Probabilities will be numbers between 0 and 1, inclusive; the closer the probability of an event is to 1, the more certain that the event will happen. All this is formalized in Definition 1, but we wish to give some examples first. One of the lessons of these first examples is that there isn't always just one right sample space for a problem.

### EXAMPLE 1

A die is to be tossed once. What is an appropriate sample space?

**Solution** Presumably the only thing of interest is which face comes up, in which case the natural sample space is  $S = \{1, 2, 3, 4, 5, 6\}$ . Here,  $\{2\}$  is the atomic event that the face with two dots comes up. The set  $\{1, 3, 5\}$  is the compound event that an odd number comes up. ■

For a fair die, all the atomic events are equally likely; that is, each has probability  $1/6$ .

### EXAMPLE 2

Two dice are to be tossed. What is an appropriate sample space?

**Solution** The answer to this depends, at least, on what question we wish to ask. If the question is, “How many dots come up total?”, then we might as well choose our sample space to be

$$\{2, 3, \dots, 11, 12\}. \tag{1}$$

However, if we are going to ask about the probabilities of more complicated situations:

Does at least one of the dice show two dots?

Does one of the dice show at least twice as many dots as the other?

Are the faces on both dice the same?

then the sample space above just won't do the job. A better choice would be

$$\{ (1, 1), (1, 2), (2, 1), \dots, (5, 6), (6, 5), (6, 6) \}. \quad (2)$$

Here, the atomic event  $\{(2, 5)\}$  is the event that the first die results in a 2, the second in a 5 (but which is the first, which the second? — read on).

Sample space (2) has the advantage that, if the die is fair, each element of the sample space is equally likely. Sample spaces like this one and that in Example 1 are called **equiprobable** and, other things being equal, they are desirable because they make calculating probabilities easier than for non-equiprobable sample spaces.

Thus to determine the probability that both faces are the same, we merely have to divide the *number* of sample space elements in (2) in which both faces are the same by the total number of elements [2]. On the other hand, with sample space (1), the events are not equiprobable so that the probability that should be assigned to each atomic event is not immediately obvious. We'll return to sample space (1) in Example 3.

But, returning to the parenthetic question just above, suppose you throw the two dice and one has two dots and the other five. Is this an instance of event  $\{(2, 5)\}$  or event  $\{(5, 2)\}$ ? If you had labeled one of the dice A and the other B, you would know (assuming that the first number in each ordered pair in (2) referred to die A and the second to die B). But if the dice aren't labelled, that is, if they are for all intents and purposes indistinguishable, then you would have no way of knowing which die is the 2 and which the 5. No matter! You can still answer the kinds of questions we asked above by just counting the number of ordered pairs that satisfy the desired criterion. Both atomic events  $\{(2, 5)\}$  and  $\{(5, 2)\}$  are instances where the face of one die has at least twice as many dots as the other die. And to answer the question about whether at least one of the dice shows two dots, just count all the ordered pairs in (2) that have at least one 2 in them, which includes both events  $\{(2, 5)\}$  and  $\{(5, 2)\}$ .

However, the case of indistinguishable dice suggests yet another sample space in which we replace ordered pairs by sets:

$$\{ \{1, 1\}, \{1, 2\}, \dots, \{1, 6\}, \{2, 2\}, \{2, 3\}, \dots, \{2, 6\}, \{3, 3\}, \dots, \{6, 6\} \}. \quad (3)$$

This sample space has the advantage of having fewer events since the separate events  $\{(a, b)\}$  and  $\{(b, a)\}$ , in (2) when  $a \neq b$ , become the single event  $\{\{a, b\}\}$  in (3). It also has the advantage of avoiding questions about whether the result of tossing the two dice is  $\{(a, b)\}$  or  $\{(b, a)\}$ . But it has the disadvantage that not all events are equiprobable. Usually we would opt for the equiprobable sample space but sometimes we might prefer to use a sample space like (3) where the atomic events  $\{a, a\}$ ,  $a = 1, 2, \dots, 6$  have one probability and all the other atomic events have a second probability, twice as great as the first one (why twice as great?).

It is worth noting, however, that in certain situations in particle physics, such as when considering which excitation state various electrons are in, having one

electron in state 1 and a second in state 2 is *not* twice as likely as having both in state 1.

Indeed, in these situations the atomic events in display (3) are equiprobable and so we would certainly prefer that sample space to (2).

The moral of this example is: You often have a choice of sample spaces and the best one may not be obvious at the outset. ■

Example 2 may have given you the impression that it is always possible to choose a sample space with equiprobable atomic events. But that is not the case. Here's an example. Suppose we are tossing a biased coin once. The natural sample space is still  $\{H, T\}$  (we can't think of any others), but H and T are not equally likely.

It's time now to formalize the general framework for assigning probabilities to events.

---

**Definition 1.** A **probability measure** on a set  $S$  is a real-valued function,  $\Pr$ , with domain the set of subsets of  $S$ , such that

- 1) For any subset  $A$  of  $S$ ,  $0 \leq \Pr(A) \leq 1$ .
- 2)  $\Pr(\emptyset) = 0$ ,  $\Pr(S) = 1$ .
- 3) If subsets  $A$  and  $B$  of  $S$  are disjoint, then

$$\Pr(A \cup B) = \Pr(A) + \Pr(B).$$

The value  $\Pr(A)$  is called the **probability of event  $A$** . A sample space,  $S$ , together with a probability measure, is called a **probability space**.

---

A probability measure, therefore, is a function whose domain is the set of all events and whose range is some subset of the closed interval  $[0, 1]$ . Thus, a probability, which is a number assigned to an event, is just a particular value in the range of a probability measure.

Definition 1 explains what probability *is*, in the sense that it gives all the rules necessary to *use* probabilities; however, it does not say what probability *means*. But this is true of all formal definitions — and it's a good thing too, because it separates the issue of use from the often stickier issue of meaning. In fact, when examined closely, the meaning of probability is quite sticky. We return to this point briefly at the end of this section.

We can give a first illustration of the use of Definition 1 by returning to Example 1 (tossing a die). We have  $S = \{1, 2, \dots, 6\}$  and for  $A \subset S$ ,  $\Pr(A) = |A|/6$ . (Recall that  $A \subset S$  means  $A$  is a subset of  $S$  and  $|A|$  is the number of elements in  $A$ .) You should check that Conditions 1–3 are satisfied. Also check [6] that the probability of the event “an odd number is tossed” is  $1/2$ .

In general, for any finite sample space  $S$ , the function defined by

$$\Pr(A) = \frac{|A|}{|S|}$$

is called the **equiprobable measure**, for it makes all atomic events equally likely. As we have already noted, the advantage of this measure, when it fits your problem, is that computing a probability reduces to counting the number of elements in a set — something you've done a lot!

For any probability measure, it follows by induction from 3) in Definition 1 that if  $A_1, \dots, A_k$  are disjoint, then [15]

$$\Pr\left(\bigcup_{i=1}^k A_i\right) = \sum_{i=1}^k \Pr(A_i). \quad (4)$$

As a special case, if  $A = \{e_1, e_2, \dots, e_k\}$ , where each  $e_i$  is an atomic event, then setting  $A_i = \{e_i\}$  and writing  $\Pr(e_i)$  instead of  $\Pr(\{e_i\})$  to avoid eye strain, we have

$$\Pr(A) = \sum_{i=1}^k \Pr(e_i)$$

or, writing  $e$  for an atomic event in any subset  $A$  of any sample space  $S$ ,

$$\Pr(A) = \sum_{e \in A} \Pr(e). \quad (5)$$

Thus a probability measure on a finite sample space is completely determined by addition from the probability of its atoms.

### EXAMPLE 3

#### Example 2 Continued

What is the correct probability measure for the first sample space 1 in Example 2, in the sense that it corresponds to the equiprobable measure on the second sample space (2)?

**Solution** It suffices to determine probabilities for the atomic events in the first sample space by counting the corresponding events in the second space as follows:

$$\begin{aligned} \{2\} &\longleftrightarrow \{(1, 1)\}, \\ \{3\} &\longleftrightarrow \{(1, 2), (2, 1)\}, \\ \{4\} &\longleftrightarrow \{(1, 3), (2, 2), (3, 1)\}, \\ &\vdots \\ \{7\} &\longleftrightarrow \{(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)\}, \\ \{8\} &\longleftrightarrow \{(2, 6), (3, 5), (4, 4), (5, 3), (6, 2)\}, \\ &\vdots \\ \{12\} &\longleftrightarrow \{(6, 6)\}. \end{aligned}$$

Since the atomic events in the second space are equiprobable, and there are 36 atoms, we conclude that, in the first space,

$$\Pr(2) = \frac{1}{36}, \Pr(3) = \frac{2}{36}, \dots, \Pr(7) = \frac{6}{36}, \dots, \Pr(12) = \frac{1}{36}.$$

This may be stated completely and succinctly as

$$\Pr(n) = \begin{cases} \frac{n-1}{36} & \text{for } 2 \leq n \leq 7, \\ \frac{13-n}{36} & \text{for } 8 \leq n \leq 12. \end{cases}$$

There are all sorts of little rules for computing the probability of one event in terms of related events. The two theorems that follow Definition 2 are examples. When you need such a rule, it's usually best to redevelop it by intuition or by drawing a picture, as we illustrate below. However, all such rules follow logically from Conditions 1–3 in Definition 1.

**Definition 2.** Let  $\sim E$  be the event “not  $E$ ”. That is, if  $S$  is the sample space,

$$\sim E = S - E.$$

For instance, in Example 1, if  $E$  is the event “an odd number shows up”, then  $\sim E$  is the event “an even number shows up”. Note that  $\sim E$  is just the set complement of  $E$  (see Section 0.1), so we could also have written  $\overline{E}$  as we did there but the use of  $\sim$  is standard when discussing probability.

How is  $\Pr(\sim E)$  related to  $\Pr(E)$ ? Think intuitively: Since  $\sim E$  is everything other than  $E$ , it must get all the remaining probability. Since the total probability is 1, this means that  $\Pr(\sim E) = 1 - \Pr(E)$ .

To illustrate that such facts follow from Definition 1, we give this simple fact a formal statement and proof:

**Theorem 1.** For any event  $E$  on a sample space  $S$ ,

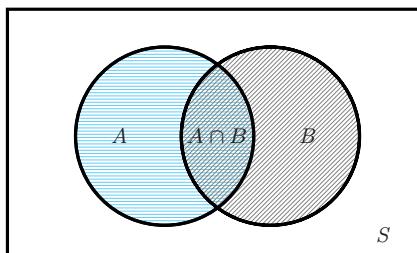
$$\Pr(\sim E) = 1 - \Pr(E).$$

**PROOF**  $S$  is the disjoint union of  $E$  and  $\sim E$ . Therefore by 2) and 3) of Definition 1,

$$1 = \Pr(S) = \Pr(E) + \Pr(\sim E).$$

Now subtract  $\Pr(E)$  from both sides. ■

What can be said about  $\Pr(A \cup B)$  when  $A$  and  $B$  are not disjoint? A Venn diagram tells it all (see Fig. 6.1). Think of the probability of a set as its area in the diagram. If we simply add  $\Pr(A)$  and  $\Pr(B)$ , we count  $\Pr(A \cap B)$  twice. We should subtract  $\Pr(A \cap B)$  once to compensate. Therefore we believe:



$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B)$$

**Theorem 2.** For arbitrary events  $A, B \subset S$ ,

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B).$$

**PROOF** At this point, about the only thing available for starting the proof is Definition 1. The only part of that definition which looks close to what we want is 3), which is about disjoint sets. Thus we must restate the situation of interest, namely, the union of two arbitrary sets, in terms of disjoint sets. Note that

$$A = (A - B) \cup (A \cap B), \quad B = (B - A) \cup (A \cap B),$$

and

$$A \cup B = (A - B) \cup (A \cap B) \cup (B - A),$$

where  $A - B$ ,  $A \cap B$  and  $B - A$  are disjoint. Thus using Condition 3 twice in the first equality below, and once in the last, we obtain

$$\begin{aligned} \Pr(A) + \Pr(B) &= [\Pr(A - B) + \Pr(A \cap B)] + [\Pr(B - A) + \Pr(A \cap B)] \\ &= [\Pr(A - B) + \Pr(A \cap B) + \Pr(B - A)] + \Pr(A \cap B) \\ &= \Pr(A \cup B) + \Pr(A \cap B). \end{aligned}$$

Subtracting  $\Pr(A \cap B)$  from both sides gives us the theorem. ■

## Types of Probability Spaces

All our examples so far have been of finite probability spaces. That is,  $S$  has been finite. There are, however, many types of infinite probability spaces. The two simplest types are discrete infinite spaces and continuous spaces. For instance, suppose we wish to study the number of telephone calls passing through a phone company switching center each day. In practice this number is limited by the capacity of the switching network. But in analyzing such telephone traffic, it is simpler — and sacrifices little accuracy — to assume that the number of calls can be any nonnegative integer. Thus, the sample space is the set of natural numbers. (The phone company needs to know the probability measure on this space when building

or expanding a switching center. Why?) This is a discrete infinite probability space.

As a second example of an infinite probability space, suppose we pick a number from 0 to 1 at random. Since every number is eligible (theoretically), the sample space is the interval  $0 \leq x \leq 1$ . This is a continuous space. Spaces can also be partly discrete and partly continuous.

Going to infinite sample spaces also imposes some surprising restrictions. You can't always assign probabilities the way you want. In Example 4, we give a simple situation in which there *cannot* be an equiprobable measure. In fact, for many sample spaces, some subsets cannot be assigned *any* probability value. But this is a very subtle matter, studied in the advanced subject called measure theory. Fortunately, it need not concern us further. All "nice" sets (events people typically consider) turn out to be "measurable" (probabilities can be assigned to them in reasonable ways), and the few times we slip over into infinite probability spaces we will always look at nice sets.

## EXAMPLE 4

Show that there is no equiprobable measure on the positive integers.

**Solution** An equiprobable measure would require each integer to have the same probability. What could this probability be? It can't be positive, no matter how small. For suppose that for any integer  $k$ ,  $\Pr(k) = \epsilon$ . ( $\epsilon$ , the Greek lower case epsilon, is the traditional symbol in mathematics for a very small but arbitrary positive number.) Since for some positive integer  $n$ ,  $n\epsilon > 1$ , by equiprobability and Condition 3,  $\Pr(\{1, 2, \dots, n\}) = n\epsilon > 1$ , contradicting Condition 1 of Definition 1. On the other hand, we can't have  $\Pr(k) = 0$  for, by the extension of Eq. (4) to infinite unions,

$$\Pr(S) = \sum_{k=1}^{\infty} \Pr(k) = 0 + 0 + \dots = 0,$$

when  $\Pr(S)$  should be 1. ■

Example 4 should bother you because, offhand, the notion of picking a positive integer at random makes sense. But think about it. If you were told to pick a positive integer at random, what would you actually do [24]?

## Defining Probability Spaces

Picking an appropriate probability space is the key to a good start on a probability problem. The examples we have given are reasonably easy. In a real-world example it's a lot harder; you need the sort of judgment that one develops with experience. Here are two more (still not very hard) examples.

## EXAMPLE 5

### Example 1, Section 6.1, Continued: Guessing on the College Boards

Devise a probability space for this problem. Recall that there are five choices for each question and you want to know whether to guess.

**Solution** What hasn't been said is how many of the 30 questions you guess on randomly. Let's call that number  $n$ . Then we can make the elements of the sample space the  $n$ -tuples of possible guesses. For example, the  $n$ -tuple (A,A,B,E,D,...,C) represents the atomic event that you guess answer A on the first guessed-at question, A again on the second, B on the third, and so on with C the guess on the thirtieth question. Because you guess randomly, the appropriate probability measure on such  $n$ -tuples is the equiprobable measure. However, all that really matters is whether you guess right or wrong. So a simpler space is the set of "words" of length  $n$  on the two symbols R and W. Thus the word RWWRW... represents the event that you guess right on the first guessed-at question, wrong on the second, and so on. This simplification causes no complication in figuring out the probability measure, even though it is no longer equiprobable: Because you guess randomly, each R has probability  $1/5$  and each W probability  $4/5$ . Each of the repeated, "independent" choices of R or W is called a "Bernoulli trial". We'll discuss Bernoulli trials in Section 6.4. ■

## EXAMPLE 6

### Example 4, Section 6.1, Continued: Mail-in contest

Devise a probability space. Recall that you might win a \$10,000 trip, but you have to pay for a stamp to mail in your lottery number.

**Solution** We assume that the winning number is chosen randomly (i.e., with equiprobable measure); that's what the cover letter usually says. So a natural choice of  $S$  is the set of all ticket numbers. Unfortunately, you don't know how many there are. Also, in most cases the contest promises that all prizes will be awarded. In other words,  $S$  really consists only of *returned* ticket numbers. Again, it's best to simplify. Let  $S = \{\text{Win}, \text{Lose}\}$ , and evaluate the probability that you win by estimating the number of returned tickets. If your estimate is  $N$  tickets, then your estimate of your chance of winning is  $\Pr(\text{Win}) = 1/N$ . ■

The last example shows that determining the probability space can be hard — how do you obtain  $N$ ? At least once you've done that, clearly you should use the equiprobable measure. What about problems such as tossing a warped coin, where you don't have equiprobability (or some other form of symmetry) to guide you? The standard approach is to assign probabilities by the **ratio interpretation**. The probability of an event  $E$  is defined as the fraction of the time  $E$  happens if we repeat the situation being modeled without limit. For instance,  $E$  might be the event that a head shows up if we toss a particular warped coin once. Imagine repeatedly tossing the coin forever. We expect that the fraction of the times that  $E$  occurs would approach some limiting value. We define  $\Pr(E)$  to be that value.

This approach is fraught with difficulties. First, you can't repeat an experiment forever. Consequently, even if the fraction of times  $E$  occurs appears to settle down very nicely after a large number of repetitions, you can't be absolutely certain what the limiting value at infinity is, or even that there *is* such a value. Second, some events are repeatable, but it's hard to get the data. For instance, there are lots of mail-in contests, but it may be hard to find out from the various sponsors what fraction of the tickets are returned (and hardly worth the effort when your

maximum loss is the cost of postage!). Finally, some events are not repeatable even once. What sense, for instance, does the ratio interpretation make for getting even a rough idea of the probability that the world will end in nuclear war?

These remarks could lead us into a discussion of the *philosophy* of probability, a fascinating but difficult subject. Fortunately, we can pull back. Mathematically, we do not need to justify where a probability measure comes from. So long as the measure satisfies Definition 1, the results in this chapter will hold. Also, so long as we are dealing with a repeatable situation, then in some sense the ratio interpretation is forced on us, at least as a conceptual way of understanding what probability means. For so long as we intend the statement “ $\Pr(E) = 1$ ” to mean that  $E$  surely happens, there is a theorem which says: If an experiment is repeated independently and infinitely many times, it surely happens that the fraction of times that any event  $A$  occurs goes to a limit, and the limit is  $\Pr(A)$ . In other words, if we are talking about a repeatable event, the ratio interpretation must be right — either that or our whole mathematical formulation of probability is not a good model of reality. The theorem just referred to is called the **Law of Large Numbers** (see Theorem 4, Section 6.6).

## Problems: Section 6.2

---

1.
  1. Using the sample space of Example 1, describe in set notation the event that the number which comes up is
    - a) Even.
    - b) A perfect square.
    - c) Divisible by 2 or 3.
2.
  2. Consider the three events listed near the start of the solution of Example 2.
    - a) Using the preferred sample space (2) for that example, describe each of these events as a set of elements. Also describe the event “the sum of the dots is 5” and “the difference of the dots is 3”.
    - b) Using the equiprobable measure on this sample space, determine the probability for each event in a).
3.
  2. For each atomic event in the first sample space suggested in Example 2, express that event as a subset of the preferred sample space.
4.
  2. Suppose that a penny is tossed three times. Let  $S$  be the sample space. The event  $E$  = “the second toss is an H” is an event in the space, but it is not an atomic event. Why not? Express  $E$  as a set of atomic events.
5.
  2. Let  $A, B, C$  be events in some sample space  $S$ . Using set notation, express the event
    - a) At least one of  $A, B$ , and  $C$  happens.
    - b) At least two of  $A, B$ , and  $C$  happen.
    - c) At most two of  $A, B$ , and  $C$  happen.
    - d) At most one of them happens.
    - e) Exactly one of them happens.
6.
  1. For Example 1, verify that  $\Pr(A) = |A|/6$  meets the definition of a probability measure. Verify that the probability of  $\{1, 3, 5\}$  is  $1/2$ .
7.
  2. What probabilities should be assigned to each of the atomic events in sample space (3)?
8.
  2. Prove that the equiprobable measure really is a probability measure. That is, prove that the function  $\Pr(A) = |A|/|S|$  satisfies Definition 1.
9.
  2. In the definition of probability measure (Definition 1), the requirement that  $\Pr(\emptyset) = 0$  is redundant: It follows from Condition 3. Show this. Hint:  $\emptyset$  is disjoint from every set.
10.
  2. How many atomic events are there in the sample space (3) of Example 2? What type of ball and urn problem (Section 4.7) is this?

11.  $\langle 2 \rangle$  Returning to the Monty Hall problem ([5], Section 6.1), let us call the door the car is behind door 1, and call the other doors 2 and 3.

- a) Define an appropriate sample space whose atomic events consist of all possible choices of a door by you and all possible choices of a door opened by Monty Hall.
- b) Using the rules of the game as stated in [5], Section 6.1, give a probability measure on this sample space.
- c) Using your sample space, what is the probability of the event that you win if you stick to your original choice? What is the probability of the event that you win if you switch?
- d) Can you think of an explanation of the answer in c) that does not explicitly use sample spaces?

*Note:* The version of the problem we have used is not actually what happened on the quiz show. Consequently, there are many variants of this problem, some with even more subtle and surprising results. To learn more, search on “Monty Hall” to check out various websites.

12.  $\langle 2 \rangle$  What, if anything, is wrong with this analysis of the Monty Hall problem? After you and Hall have picked, there are only two choices: your original choice has the car or the other remaining door has the car. So the chances are 50–50 and it doesn’t matter if you switch.

13.  $\langle 2 \rangle$  Suppose that in the Monty Hall problem there are  $2n + 1$  doors with a car behind just one of them. You choose  $n$  doors, Monty Hall opens  $n$  doors with goats behind them and then you may stick with your choice (getting the car if it is behind one of the  $n$  doors) or switch to the one door not opened by Monty Hall or chosen initially by you. What should you do? Why?

14.  $\langle 2 \rangle$  Prove Eq. (4) by induction from Condition 3 in Definition 1.

15.  $\langle 2 \rangle$  Prove that Eq. (5) is equivalent to Condition 3 of Definition 1. That is, in any finite sample space, any function  $\Pr$  which satisfies Condition 3 satisfies Eq. (5), and vice versa.

16.  $\langle 2 \rangle$  Prove that  $\Pr(A - B) = \Pr(A) - \Pr(A \cap B)$ .

17.  $\langle 2 \rangle$  Prove that

$$\Pr(\sim A \cap \sim B) = 1 - \Pr(A) - \Pr(B) + \Pr(A \cap B).$$

18.  $\langle 2 \rangle$  Show that if  $A_1, A_2, \dots, A_n$  are disjoint and  $A$  is their union, then

$$\Pr(A \cap B) = \sum_{i=1}^n \Pr(A_i \cap B).$$

19.  $\langle 2 \rangle$  Use [16] to give another proof of Theorem 1. Hint: Write  $A \cup B$  using  $A - B$ .

20.  $\langle 2 \rangle$  Prove: If  $A \subset B$  then  $\Pr(A) \leq \Pr(B)$ .

21.  $\langle 2 \rangle$  Odds. When one says that the **odds** against event  $E$  are  $s : t$  (pronounced “ $s$  to  $t$ ”), that means  $E$  will fail to occur with probability  $s/(s+t)$ . In this problem, assume that exactly one of three events,  $A$ ,  $B$ , and  $C$ , must happen.

a) If the odds against  $A$  are 1:1 and the odds against  $B$  are 2:1, what are the odds against  $C$ ?

b) If the odds against  $A$  are instead 3:2 and against  $B$  3:1, what are the odds against  $C$ ?

c) If the odds against  $A$  are  $p : q$  and those against  $B$  are  $r : s$ , what are the odds against  $C$ ?

22.  $\langle 2 \rangle$  Prove an inequality relating

$$\Pr\left(\bigcup A_i\right) \quad \text{and} \quad \sum \Pr(A_i).$$

23.  $\langle 3 \rangle$  Let function  $F$  be defined on all subsets  $A$  of  $R$  by

$$F(A) = \begin{cases} 0 & \text{if } |A| \text{ is finite,} \\ 1 & \text{if } |A| \text{ is infinite.} \end{cases}$$

a) Prove: For all  $A, B \subset R$ ,  $F(A \cup B) \leq F(A) + F(B)$ .

b) By induction, prove for all positive integers  $n$  that

$$F\left(\bigcup_{i=1}^n A_i\right) \leq \sum_{i=1}^n F(A_i).$$

c) Nonetheless, find an infinite sequence of sets  $A_1, A_2, \dots$  such that

$$F\left(\bigcup_{i=1}^{\infty} A_i\right) \not\leq \sum_{i=1}^{\infty} F(A_i). \tag{6}$$

Moral: Proving something for an arbitrarily large positive integer is not the same as proving it for infinity. Consequently, for sample spaces with infinitely many atoms, Condition 3 of Definition 1 needs to be replaced with an infinite sum version that disallows (6).

24. {3} (Example 4 continued) To pick a positive integer at random, we think you would do the following. You would pick some large positive integer  $N$  and pick an integer at random *between* 1 and  $N$ . (This is what a computer must do, since each computer has a largest integer known to it.) Or you might pick a string of *digits* at random, continuing until you got tired. (But does this make each integer equally likely?) In any event, so long as there is some largest  $N$  that you will consider, you have a finite space and there can be an equiprobable measure on it.

But in some ways this begs the question. It

seems perfectly reasonable, for instance, to contend that the probability of picking an even integer by random choice from among *all* positive integers is  $1/2$ . Fortunately, we can justify such statements by using the idea in the preceding paragraph, if we also use limits. Namely, let  $P_N$  be the probability that, when an integer from 1 to  $N$  is picked at random, it is even. Then our contention may be restated this way: The limit of  $P_N$  as  $N \rightarrow \infty$  is  $1/2$ .

For those who already have studied limits, show that this contention is correct.

## 6.3 Conditional Probability, Independence, and Bayes' Theorem

---

Since probabilities measure the likelihood of events, probabilities may change as new information is discovered. Example 2, Section 6.1 (tuberculosis), provides a good example. Once you are tested, the probability that you are a carrier changes; it goes up if you test positive, down if you test negative. On the other hand, sometimes new information is irrelevant. If you find out that the previous toss of a coin was heads, that doesn't affect the probability that the next toss will be heads.

In this section we show how to fit such considerations into the probability space formulation of the previous section. We do so using Definitions 1 and 2. First we state these definitions, then we motivate them and show how to use them. We conclude this section with a result called Bayes' Theorem. This theorem is really nothing more than a restatement of Definition 1, but a particularly useful one.

In many places in this section we divide by a probability. Needless to say, that probability must be nonzero. We make this explicit in Definitions 1 and 2, but thereafter it will be implicit.

---

**Definition 1.** Suppose that  $\Pr(A) \neq 0$ . The **conditional probability of  $B$  given  $A$** , denoted  $\Pr(B|A)$ , is defined to be

$$\Pr(B|A) = \frac{\Pr(A \cap B)}{\Pr(A)}. \quad (1)$$

The quantity  $\Pr(B|A)$  is usually read just as “the probability of  $B$  given  $A$ ”.

---

**Definition 2.** Events  $A$  and  $B$  are **independent** if

$$\Pr(A \cap B) = \Pr(A) \cdot \Pr(B), \quad (2)$$

or equivalently (when  $\Pr(A) \neq 0$ ), if

$$\Pr(B|A) = \Pr(B). \quad (3)$$

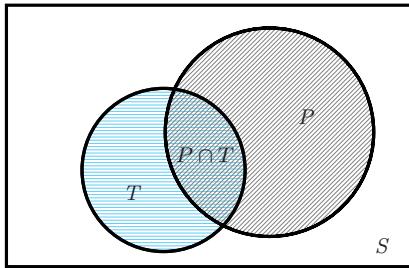
Equation (2) is the primary display in Definition 2 because it is slightly more general than Eq. (3) since  $\Pr(A) = 0$  is allowed in the former and because the roles of  $A$  and  $B$  are symmetric in Eq. (2). Mathematicians love symmetry.

In any event, it's easy to show that the two equations are equivalent when  $\Pr(A) \neq 0$ . Assuming Eq. (3) and substituting Eq. (1) into it, we get

$$\frac{\Pr(A \cap B)}{\Pr(A)} = \Pr(B).$$

Multiplying by  $\Pr(A)$  gives Eq. (2). Reversing the steps gets us back to Eq. (3).

We motivate Definition 1 by continuing with the tuberculosis example. Consider Fig. 6.2. The universe  $S$  is the set of all people to whom the statistics apply (the U.S. population, say),  $T$  is the set of all tuberculosis carriers, and  $P$  represents the set of people who test positive (or would if they were tested). We were asked: If all we know about you is that you are in  $P$ , what is the probability that you are also in  $T$ ? This is what  $\Pr(T|P)$  is supposed to mean.



**FIGURE 6.2**

Tuberculosis testing.

We need to show that Eq. (1) captures that intended meaning. The example asks “How probable is it that someone tests positive and is a carrier (event  $P \cap T$ ) relative to all those who test positive (event  $P$ )?” This relative probability might be close to 1 even if  $\Pr(P \cap T)$  is small (i.e., few people might test positive but almost all of those might be carriers). So what we need is a *change of scale*. Since only  $P$  “counts” any more, we want  $P$  effectively to have probability 1. So for every  $X \subset P$ , we multiply  $\Pr(X)$  by  $1/\Pr(P)$ . In short,

$$\Pr(T|P) = \frac{\Pr(P \cap T)}{\Pr(P)}.$$

This is precisely what Eq. (1) says, with  $A = P$  and  $B = T$ .

We now give some numerical examples of the use of Definition 1. The key to a correct application of the definition is a careful identification of the sample space and the sets  $A$  and  $B$ .

### EXAMPLE 1

An urn contained two black balls and two white balls. Balls were removed one at a time, randomly, and not replaced. You were not present. (a) What is the probability (from your viewpoint, based on just this information) that the first ball removed was black? (b) You are now informed that the second ball removed was black. How should you change your answer?

**Solution** Surely your answer to (a) should be  $1/2$ , since half the balls available for the first draw were black. You do not need to think consciously in terms of a sample space to arrive at this conclusion. However, in order to answer (b), conscious thought about a sample space is helpful. (Before reading on, think about what sample space you would use and see if your answer is the same as we get below.) Let  $F$  (for first) be the event that the first ball was black;  $N$  (for next) the event that the second ball was black. Let the four balls be  $\{b_1, b_2, w_1, w_2\}$ . Then the sample space we use has at its atoms the ordered pairs corresponding to the possible first and second choices for removed balls. Since there are four ways the first ball may be chosen and three ways the second ball may be chosen, there are  $4 \times 3 = 12$  atoms. The measure is equiprobable; i.e., each element, such as  $(b_2, w_1)$  has the same probability as any other. Then  $F$  is the subset of all ordered pairs with the first entry black. There are  $2 \times 3$  of these, since there are two black balls for the first draw and three remaining balls for the second draw:

$$F = \{(b_1, b_2), (b_1, w_1), (b_1, w_2), (b_2, b_1), (b_2, w_1), (b_2, w_2)\}.$$

Similarly,  $N$  is a subset with six ordered pairs:

$$N = \{(b_2, b_1), (w_1, b_1), (w_2, b_1), (b_1, b_2), (w_1, b_2), (w_2, b_2)\}.$$

Clearly,  $F \cap N = \{(b_1, b_2), (b_2, b_1)\}$ . Consequently,

$$\Pr(F \cap N) = \frac{2}{12}, \quad \Pr(N) = \frac{6}{12}, \quad \text{and} \quad \Pr(F|N) = \frac{2/12}{6/12} = \frac{1}{3}.$$

Thus the information that the second ball was black *lowers* the probability that the first ball was black. Is this intuitively reasonable [22]?

*Note:* We could have used for our sample space the set of all  $4!$  ways in which all 4 balls could be removed in order, but this is a bigger space than we need for either part (a) or (b). For (a) but not (b) we could use the simpler space  $\{b_1, b_2, w_1, w_2\}$  of individual balls, since each may be drawn first. Indeed, it is this space that one is implicitly using when one sees right away that the answer to (a) is  $1/2$ . ■

### EXAMPLE 2

A fair coin was tossed twice while you weren't present. (a) What is the probability that the first toss was a head? (b) What is the probability that the first toss was a head, given that the second toss was a head?

**Solution** Again, the first answer is  $1/2$ . We hope that you feel very strongly that the second answer is also  $1/2$ , but let's check it out as carefully as in the previous example. We may take  $S$  to be the four-element set  $\{(H, H), (H, T), (T, H), (T, T)\}$ . Taking  $F$  to be the event that the first toss is a head and  $N$  that the second is a head, we have

$$F = \{(H, H), (H, T)\}, \quad N = \{(H, H), (T, H)\}$$

so that  $F \cap N = \{(H, H)\}$  and thus

$$\Pr(F \cap N) = \frac{1}{4}, \quad \Pr(N) = \frac{2}{4} \quad \Pr(F|N) = \frac{1/4}{2/4} = \frac{1}{2}. \blacksquare$$

If you felt at the outset that both answers in Example 2 should be the same, probably you knew intuitively that one coin toss is “independent” of another in the sense that knowledge of one shouldn't affect probabilities for the other.

*Caution.* Many people think that Eq. (2) in Definition 2 is true for *any* events  $A$  and  $B$ . It's not. It is true if and only if the events are independent. For example, in Example 1,  $\Pr(F) = \Pr(N) = 1/2$  but  $\Pr(F \cap N) = 1/6$ ; see also [25, 27]. The multiplication fact that *is* true for all events  $A, B$  is

$$\Pr(A \cap B) = \Pr(A) \Pr(B|A). \tag{4}$$

This is obtained by multiplying Eq. (1) in Definition 1 by  $\Pr(A)$ . (Actually, Definition 1 assumes that  $\Pr(A) \neq 0$ , but (4) is true even when  $\Pr(A) = 0$ ; see [6].)

*Another caution.* Many people (often the same ones) also have the exact opposite misconception: They don't believe events are independent even when they are! Some years ago, one of us volunteered for an experiment at a leading business school. The “experiment” consisted of our being shown a penny (we were allowed to examine it to see that it was ordinary), being told that in a sequence of  $N$  tosses it had come up heads  $m$  times, and being asked what we then expected the next toss would be. The past results were always very unusual, for instance, in 10 tosses there had been 8 heads. Every time, we shrugged in disbelief at the question and said “It's 50–50.” Finally, the experimenter said, “Why don't you think the past history has an effect?” We explained our mathematical background and that we knew coin tosses are independent events. “Well, in that case we won't count your answers in our data; you'll bias our study.” We still wonder sometimes what this study claimed to prove.

### EXAMPLE 3

At No Nonsense College, a first-year student must take a foreign language (Chinese, German, or Spanish), a science (Biology, Chemistry, or Physics), English (Composition or Literature), social science (Economics or Politics), and Discrete Math. What are the atoms of the sample space?

**Solution** The basic units we tend to think of are the subjects, e.g., German or Economics. However, these cannot be the atomic events. Atomic events should be indivisible; nothing that we want to consider can be a special case of an atom.

Since taking German and Economics is a subcase of taking German (taking both has lower probability than taking German), German is not an atom.

In fact, the atoms in this example are 5-element sets, containing one course from each subject area. For instance, one atom is

$$\{\text{Chinese, Chemistry, Literature, Economics, Discrete Math}\} \quad (5)$$

and the sample space contains  $3 \cdot 3 \cdot 2 \cdot 2 \cdot 1 = 36$  atomic events. The event “German” is the compound event consisting of the 12 5-sets that contain German as one of the 5 courses.

In short, “simplest to state” and “indivisible” are not the same. Specifically, on many occasions we will consider sequences of  $n$  events. In all such cases, each possible sequence is an atom. In fact, the atoms in Examples 1 and 2 were of this form with  $n = 2$ . If order does not matter, as for the choice of courses, we may let the atoms be sets. ■

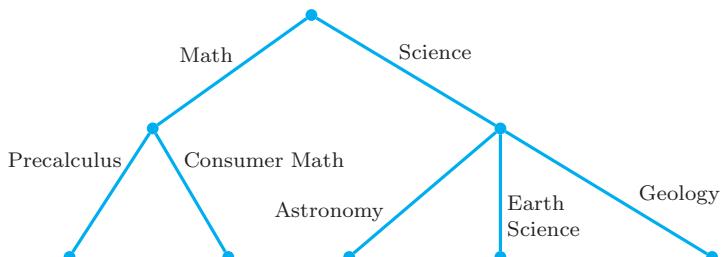
#### EXAMPLE 4

At Soft Touch College, the one supposedly tough requirement is that you must take one of Math and Science (not both!). For math you have a choice between Precalculus and Consumer Math, and for science you pick one of Astronomy, Earth Science, and Geology. Ima Student decides to meet this requirement by first picking randomly between the fields math or science, and then picking randomly among the courses in the chosen area. What is the probability she takes Geology?

**Solution** In this case the atoms of the sample space really are the individual courses (why?). We want  $\Pr(G)$ , where  $G$  is the atom (taking) Geology. Let  $Sc = \{A, E, G\}$  be the compound event “taking a science”.

Ima’s method is an example of *sequential decision making*. First she chooses randomly between science and math, so  $\Pr(Sc) = 1/2$ . Then, given that she has chosen science, she chooses randomly among the 3 science courses, so  $\Pr(G|Sc) = 1/3$ . Therefore, by Eq. (4) with  $B = G$  and  $A = Sc$ , we get  $\Pr(Sc \cap G) = \Pr(Sc)\Pr(G|Sc) = 1/6$ . Since  $\Pr(G) = \Pr(Sc \cap G)$  (why?), we conclude that  $\Pr(G) = 1/6$ . In sequential problems it is more common to use conditional probabilities to compute regular probabilities rather than vice versa.

By the way, sequential decision problems are conveniently represented by trees. The decision tree for Soft Touch College is shown in Fig. 6.3. ■



**FIGURE 6.3**

A decision tree for Soft Touch College courses.

## Bayes' Theorem

Let  $A_1, A_2, \dots, A_n$  be a set of **mutually exclusive and exhaustive** events. This means that every atomic event of the sample space  $S$  must be in exactly one of the  $A_i$ . (“Mutually exclusive” means disjoint; “exhaustive” means the union of all the events  $A_i$  is  $S$ .) Let  $B$  be any event on  $S$ . Bayes’ Theorem is a formula for  $\Pr(A_i|B)$ , one that’s especially useful when  $B$  is an observable consequence of unobservable  $A$ ’s. Since the idea of an observable consequence of unobservable events may seem obscure, we first illustrate the sort of situation in which it can be used and then state Bayes’ Theorem.

### EXAMPLE 5

In a certain county, 60% of the registered voters are Republicans, 30% are Democrats, and 10% are Independents. 40% of the Republicans oppose increased military spending, while 65% of the Democrats and 55% of the Independents oppose it. A voter writes a letter to the county paper, arguing against increased military spending. Assuming nothing is known about the voter except what is in the letter, what is the probability that this voter is a Democrat?

We won’t answer this until after we state and prove Bayes’ Theorem, but let us set up the sample space  $S$  now. Let the atoms be the individual voters in the county. Let  $A_1$  be the event “is a Republican”. This event is the set of all voters in the county who are Republicans, however they may feel about military spending.  $\Pr(A_1) = |A_1|/|S|$  is the probability a voter in the county is a Republican. Similarly, for  $A_2$  substitute Democrats and for  $A_3$  Independents. Clearly these are mutually exclusive events; according to the data given they are also exhaustive. Let  $B$  be the event “is against increased military spending”.  $\Pr(B)$  is the probability that a random voter is against increased military spending. It is given that the letter writer is in  $B$  (i.e., it is observable), but we can’t observe his/her political party. Therefore, we wish to know  $\Pr(A_2|B)$ , or generalizing,  $\Pr(A_i|B)$  for  $i = 1, 2, 3$ . We know the reverse probabilities  $\Pr(B|A_i)$  as well as the  $\Pr(A_i)$ . Bayes’ Theorem will tell us how to find  $\Pr(A_i|B)$  from what we know.

Now let’s state Bayes’ Theorem, then prove it and apply it to Example 5.

---

**Theorem 1, Bayes’ Theorem.** Let  $A_1, \dots, A_n$  be mutually exclusive and exhaustive events in the sample space  $S$ . Let  $B$  be any event on  $S$ . Then for each  $i$  from 1 to  $n$ ,

$$\Pr(A_i|B) = \frac{\Pr(A_i) \Pr(B|A_i)}{\sum_{j=1}^n \Pr(A_j) \Pr(B|A_j)}. \quad (6)$$

**PROOF** The set manipulations of this proof are best understood by looking at Fig. 6.4, which illustrates the case  $n = 4$ . Since  $S = \bigcup A_j$ , we have

$$B = \bigcup_{j=1}^n (A_j \cap B).$$

As all the sets  $A_1 \cap B, \dots, A_n \cap B$  are disjoint,

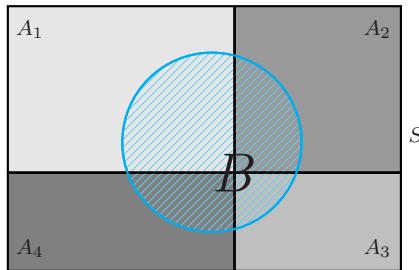
$$\Pr(B) = \sum_{j=1}^n \Pr(A_j \cap B).$$

Furthermore, by Eq. (4), for any particular  $i$ ,

$$\Pr(A_i \cap B) = \Pr(A_i) \Pr(B|A_i).$$

Thus, using first Definition 1 and then the preceding two equations

$$\begin{aligned}\Pr(A_i|B) &= \frac{\Pr(B \cap A_i)}{\Pr(B)} \\ &= \frac{\Pr(A_i) \Pr(B|A_i)}{\sum_{j=1}^n \Pr(A_j \cap B)} \\ &= \frac{\Pr(A_i) \Pr(B|A_i)}{\sum_{j=1}^n \Pr(A_j) \Pr(B|A_j)}. \blacksquare\end{aligned}$$



**FIGURE 6.4**

Diagram for proof of Bayes' Theorem.

**Solution to Example 5** Using the notation and data in the example, we have:

$$\begin{aligned}\Pr(A_1) &= .6, & \Pr(A_2) &= .3, & \Pr(A_3) &= .1 \\ \Pr(B|A_1) &= .4, & \Pr(B|A_2) &= .65, & \Pr(B|A_3) &= .55.\end{aligned}$$

Plugging into Bayes' formula with  $i = 2$ , we find that

$$\begin{aligned}\Pr(A_2|B) &= \frac{(.3)(.65)}{(.6)(.4) + (.3)(.65) + (.1)(.55)} \\ &= \frac{.195}{.49} \approx .398.\end{aligned}$$

In short, knowledge from the letter increases the probability that the voter is a Democrat from 30% to about 40%. ■

**Solution to the tuberculosis problem** (finally!) This problem (Example 2, Section 6.1) asked: What is  $\Pr(T|P)$ , the probability you are a carrier given that you have tested positive? Plugging into Bayes' Theorem, using  $T$  for  $A_1$ ,  $\sim T$  for  $A_2$  and  $P$  for  $B$ , we get

$$\Pr(T|P) = \frac{\Pr(T)\Pr(P|T)}{\Pr(T)\Pr(P|T) + \Pr(\sim T)\Pr(P|\sim T)}.$$

Using the data from the original statement of the problem that 8% of the carriers test negative and that 4% of the noncarriers test positive, this equation becomes

$$\Pr(T|P) = \frac{.92\Pr(T)}{.92\Pr(T) + .04\Pr(\sim T)}.$$

Now, at least, it is clear what we still need to know. We need to know what fraction of the total population is tuberculosis carriers. For the United States the answer seems to be about  $\frac{3}{4}\%$  so that  $\Pr(T) = .0075$  and  $\Pr(\sim T) = .9925$ . Thus

$$\Pr(T|P) = \frac{(.92)(.0075)}{(.92)(.0075) + (.04)(.9925)} \approx .148,$$

so that only about 15% of those who test positive are carriers. ■

Does it surprise you that the answer is so low? One's reaction to such a fact might be to question the value of the tine test and to demand a better test. But this reaction would show a lack of understanding of what the tine test is all about. Its real purpose is to show that a large number of people *don't* have TB. As [35] shows, it does this very well: Anyone from the general population who tests negative has probability .9994 of not having TB. Those people who test positive are probably OK too, but they need to have another, more precise test (the Mantoux test or a chest X-ray). Why don't doctors simply give better tests to begin with? Because they are much more time-consuming and costly. The tine test drastically reduces the number of people who need more elaborate tests. For more on tuberculosis, see [36–38].

## Problems: Section 6.3

---

1.  $\langle 2 \rangle$  Suppose that

$$\Pr(A) = .6, \Pr(B) = .5, \text{ and } \Pr(A \cap B) = .4.$$

- a) Are  $A$  and  $B$  independent?
  - b) Determine  $\Pr(A|B)$  and  $\Pr(B|A)$ .
2.  $\langle 2 \rangle$  Repeat [1], except  $\Pr(A \cap B) = .3$ .
3.  $\langle 2 \rangle$  Repeat [1], except instead of information about  $A \cap B$ , you are given that  $\Pr(A \cup B) = .8$ .
4.  $\langle 2 \rangle$  What is  $\Pr(A|B)$  if  $\Pr(B|A) = .8$ ,  $\Pr(B|\sim A) = .3$ , and  $\Pr(A) = .2$ ?

5.  $\langle 2 \rangle$  What is  $\Pr(A|Z)$  if events  $A, B, C$  are mutually exclusive and

$$\Pr(A) = .2, \quad \Pr(B) = .3, \quad \Pr(C) = .5 \\ \Pr(Z|A) = .9, \quad \Pr(Z|B) = .5, \quad \Pr(Z|C) = .1 ?$$

6.  $\langle 1 \rangle$  We defined  $\Pr(B|A)$  only when  $\Pr(A) \neq 0$ , but in fact it is often possible to give  $\Pr(B|A)$  a useful definition even when  $\Pr(A) = 0$ . (This happens most often for continuous probability spaces.) In any event, this fact is one reason why we rarely state  $\Pr(A) = 0$ .) Show that Eq. (4) is true when  $\Pr(A) = 0$  no matter how  $\Pr(B|A)$  is defined.

7. (3) Weather on consecutive days is not independent. Let us suppose that all days can be classified as Good or Bad as far as weather is concerned. Suppose also that the probability of weather today conditioned on weather yesterday is as follows:

|            | Good<br>yesterday | Bad<br>yesterday |
|------------|-------------------|------------------|
| Good today | .7                | .4               |
| Bad today  | .3                | .6               |

Finally, suppose the probability that any given day is Good is  $4/7$ .

- a) If today is Good, what is the probability that yesterday was Good?
- b) If today is Bad, what is the probability that yesterday was Good?

Note: It turns out that the probability that any given day is Good is redundant; it can be deduced from the table by a clever argument. Can you figure it out?

8. (2) From the definition of conditional probability, check that

$$\Pr(B|A) = P(B) \iff \Pr(A|B) = \Pr(A).$$

9. (3) You are given that  $\Pr(B) = .5$  and  $\Pr(A|B) = .4$ . For each of the following probabilities, either determine its value or show that its value is not unique. (Do the latter by giving two different “weighted” Venn diagrams, each of which is consistent with the given information. By a weighted Venn Diagram involving sets  $A$  and  $B$ , we mean one in which all four events,  $A \cap B$ ,  $A \cap \sim B$ ,  $\sim A \cap B$ , and  $\sim A \cap \sim B$ , have been assigned probabilities which add to 1.)

- a)  $\Pr(A \cap B)$
- b)  $\Pr(\sim A|B)$
- c)  $\Pr(A|\sim B)$
- d)  $\Pr(\sim A)$

10. (2) In Example 5, what is the probability that the letter writer is a Republican? An Independent?

11. (2) In Example 5, what percent of the registered voters are against increased military spending?

12. (3) In Example 5, not only do we know that the letter-writing voter is opposed to increased military spending, but we also know that he or she feels strongly enough about it to write to the newspaper. Suppose we somehow know that 5% of Democrats in that county oppose military spending strongly enough to write (at some point) to the

newspaper about it and that 2% of Republicans and 3% of Independents also feel that strongly. Does this change the answer to the question?

13. (2) A penny is tossed until either a head comes up or three tosses have been made, whichever comes first. Assume the coin is fair.

- a) List all the atomic events in the sample space and give the probability for each.
- b) What is the probability that the game ends with a head? With the third toss? With the second toss?

14. (3) One penny in a million has two heads. A friend tosses the same penny 10 times and reports that he got 10 heads. Based on this information alone, what probability should you give to the possibility that his penny has two heads? (Assume that all pennies other than two-headed pennies are fair coins with heads and tails.)

15. (2)

- a) How does the answer to [14] change if your friend got all heads on 20 tosses?
- b) How does it change if there were still 10 tosses but there are also two-tailed pennies, again one in a million?

16. (2) The following questions may shed some light on why many people believe that past performance of coin-tossing (or repeatable random events in general) affects future performance. In your answers, be explicit about your sample space and probability measure.

- a) A fair coin was tossed 10 times and heads came up 8 times. If it is tossed again, what is the probability of getting a head?
- b) A fair coin was tossed 10 times and heads came up 8 times. What is the probability that the tenth toss was a head?
- c) A fair coin was tossed 10 times and heads came up 6 times. However, 4 of the first 5 tosses were heads. What is the probability that the tenth toss was a head?

17. (2) The following list gives some information about students who take introductory calculus at Podunk U. Each row lists a department, then the percentage of calculus students who end up majoring in that department, and finally the percentage of those majors who got an A in calculus. For instance, the first row says that 25% of calculus

students eventually major in math, and of those math majors, 40% got an A in calculus.

|                |    |    |
|----------------|----|----|
| Math           | 25 | 40 |
| Science        | 20 | 30 |
| Engineering    | 40 | 25 |
| Social science | 10 | 20 |
| Humanities     | 5  | 15 |

a) What fraction of the students enrolled in calculus get A's?

b) John Doe gets an A. What is the probability he will major in Math? In Engineering?

18. (2) A mediocre student gets a perfect paper on a 20-question multiple choice test. The teacher is suspicious and speaks to the student. The student acknowledges that he didn't know the answers to the last 10 questions, but claims that he simply guessed at random and was lucky. Each question had 5 answers listed. The teacher estimates that 1 student in 1000 cheats. Should she believe this student? What if her estimate is wrong and only 1 student in 10,000 cheats? Should she believe the student then?

19. (2) Do part (a) of Example 1 again carefully, using the sample space used for part (b). Verify that the answer is the same as was gotten with the simplified sample space  $\{b_1, b_2, w_1, w_2\}$  in the note at the end of the solution.

20. (3) For Example 1, here are three other sample spaces we might have chosen. (They are stated in the parts below.) Which ones are appropriate with an equiprobable measure? What are  $F$  and  $N$  in each? Check that you get the same conditional probability  $\Pr(F|N)$  as in the text for each sample space that you declare appropriate.

a) The set of all  $4!$  permutations of  $\{b_1, b_2, w_1, w_2\}$ .

b) The set of all  $4!/2!2!$  permutations of the multiset  $\{b, b, w, w\}$ .

c) The set of all ordered pairs each of whose elements is either  $b$  or  $w$ .

21. (3) *The second-child problem.* In the following parts, you are asked probability questions about the sex of children. In each case, assume that any child is equally likely to be male or female and that the sex of any one child in a family is independent of the sex of any other children in that family. The questions are quite tricky, but only because your intuition may mislead you; no linguistic or other

cheap tricks are intended. The way to keep from going wrong is to be very careful about defining your probability space and the particular events that you consider in it. State your space and your events carefully.

a) You visit the home of an acquaintance, who says, "I have two kids." A boy walks into the room. The acquaintance says, "That's my older child." What is the probability that the younger one is a boy?

b) You visit the home of an acquaintance, who says, "I have two kids." A boy walks into the room. The acquaintance says, "That's one of my kids." What is the probability that the other one is a boy?

c) You live in a culture where, when children are introduced, male children are always introduced first, in descending order of age, and then female children, also in descending age order. You visit the home of an acquaintance, who says, "I have two kids, let me introduce them." He yells, "John come here." (John is a boy's name.) What is the probability that the other child is a boy?

d) You go to a parent-teacher meeting. The principal is sitting in the first row. You've heard that the principal has two children. The teacher in charge asks everyone who has a son (meaning at least one) to raise a hand. The principal raises her hand. What is the probability that the principal has two sons?

22. (2) (Philosophical question) In Example 1, we said the first draw depended on the second. But this is absurd. Something can never depend on something else that happens later. Which ball shows up second can never affect which ball showed up first. Thus the conditional probability should equal the unconditional probability (as in Example 2). Respond to this objection.

23. (2) At one Chinese restaurant, you have a choice of two soups — eggdrop or Wonton — and then you have a choice of three main dishes — pork, chicken, or beef. Assume that there's nothing else on the menu, that you make each choice randomly, and that you don't skip any choices.

a) What are the atoms in the sample space?

b) Draw a decision tree and determine the probability of your choosing each possible meal.

- c) What is the probability that you'll eat beef?  
d) What is the conditional probability you chose Wonton soup given that you choose pork?
24. ⟨2⟩ At another Chinese restaurant, you have the same choice of soups, but if you choose eggdrop soup, then you must choose pork or beef, and if you choose Wonton soup, then you must choose either beef, chicken, or Peking duck. Again assume that you choose randomly between the soups and then randomly among the main course entrees available to you.
- What are the atoms in the sample space?
  - Draw a decision tree and determine the probability of your choosing each possible meal.
  - What is the probability that you'll eat beef?
  - What is the conditional probability that you chose Wonton soup given that you choose pork?
  - What is the conditional probability that you chose Wonton soup given that you choose beef?
25. ⟨2⟩ Let  $\Pr(A) = 2/3$ ,  $\Pr(B) = 3/4$ . What is the largest that  $\Pr(A \cap B)$  can be? The smallest? Is every number in between also possible?
26. ⟨2⟩ If  $\Pr(A) = 2/3$  and  $\Pr(B) = 3/4$ , what is the largest and smallest  $\Pr(B|A)$  can be? Answer the same question for  $\Pr(A|B)$ .
27. ⟨2⟩ Do [25] again but this time in general. If  $\Pr(A) = p$  and  $\Pr(B) = q$ , what is the range of possible values for  $\Pr(A \cap B)$ ?
28. ⟨2⟩ Ima Student transfers to No Nonsense College (Example 3). To choose her schedule, she decides to independently pick at random in each of the subject areas where she has a choice. For instance, in social science she picks either Economics or Politics with probability  $1/2$ . Show that each schedule (i.e., each atom) is equally likely. While this is pretty obvious, show it using appropriate definitions from the section.
29. ⟨2⟩ Traditional College requires each first-year student to either take mathematics or start a foreign language. If a student chooses mathematics, it can be either continuous or discrete. The language can be French, German, or Russian. For French there is only one starting course, but in German and Russian one may choose between regular and scientific versions. Heesa Student decides to make sequential decisions, first math or language, then which course. If he takes a language, he only decides whether to take regular or scientific after choosing which language. At each stage he decides to choose randomly.
- Draw the decision tree.
  - What are the atoms of the sample space for this problem? Are they equiprobable?
  - What is the probability Heesa takes a scientific language course?
  - What is the probability he takes a scientific language course, given that he takes a language?
  - What is the probability he takes Russian, given that he takes a scientific language course?
30. ⟨3⟩ Actuaries (life insurance mathematicians) have collected various sorts of statistics about mortality. One important statistic is  $p_n$ , the probability that a person who lives to age  $n$  years will continue to live until at least age  $n + 1$ . (Actually, there are separate statistics for men, women, smokers, etc., but let's ignore that.) Note that the  $p_n$  are actually conditional probabilities; in what probability space? Express the following probabilities in terms of the  $p_n$ 's. The answers may be complicated expressions involving  $\sum$  and  $\prod$  notation.
- The probability that a person age 30 lives to at least age 50
  - The probability that a person age 30 dies before age 31
  - The probability that a person age 30 dies at age 65
  - The probability that a person age 30 dies before age 65
  - The probability that a person age 60 dies between 65 and 70
  - The probability that a newborn baby lives to age 65
  - The probability that a person who died before age 40 died after age 30
31. ⟨3⟩ Express your answers to [30] using algorithmic language rather than formulas (which is what actuaries do today anyway — computers have liberated them from many tedious calculations).
32. ⟨3⟩ Another important actuarial statistic is  $l_n$ , the probability that someone just born will live to at least age  $n$  years. Express all the parts of [30] again, using the  $l_n$ 's.

- 33.**  $\langle 3 \rangle$  Express the answers to [32] using algorithmic language.
- 34.**  $\langle 3 \rangle$  In the proof of Bayes' Theorem, we expanded  $\Pr(B \cap A_i)$  as  $\Pr(A_i) \Pr(B|A_i)$ . Why didn't we expand it as  $\Pr(B) \Pr(A_i|B)$ ? Isn't the second way correct, too? Wouldn't it give an equally true theorem?
- 35.**  $\langle 2 \rangle$  (Tuberculosis testing) Using  $\Pr(T) = .0075$  and the data of Example 2, Section 6.1, verify the claim on p. 534 that  $\Pr(\sim T|\sim P) \approx .9994$ .
- 36.**  $\langle 3 \rangle$  The ideal in medical tests is to find a cheap, quick test which is so precise that no further tests are needed. However, it is unlikely one can find such tests when the condition being tested for is rare — because the test must be *extremely* precise, more precise than most things in medicine are. Suppose we found a test such that every TB carrier tests positive. Let  $p$  be the probability that someone who is not a carrier also tests positive on this test. How small does  $p$  have to be in order for  $\Pr(T|P)$  to equal .95? Assume that  $\Pr(T) = .0075$ . (*Note:* For a test to be completely accurate for those people who really are carriers, the test probably has to err in favor of positive results. Is a  $p$  this low likely, given such “bias”?)
- 37.**  $\langle 2 \rangle$  The prevalence of tuberculosis infection in the United States is generally low, but it is far from uniform. In certain immigrant or minority neighborhoods it is as high as 15%. That is, if  $S$  is restricted to certain subpopulations,  $\Pr(T) = .15$ . Using this value for  $\Pr(T)$ , and otherwise continuing to use the probabilities given in Example 2, Section 6.1, recompute  $\Pr(T|P)$  and  $\Pr(\sim T|\sim P)$ . (Doctors call  $\Pr(T|P)$  the “sensitivity” of the test and  $\Pr(\sim T|\sim P)$  its “specificity”.)
- 38.**  $\langle 2 \rangle$  For various good reasons, the only group in the general U.S. population that currently receives widespread TB testing is preschool children. For this group  $\Pr(T) = .003$ . Do [37] again using this value.
- 39.**  $\langle 2 \rangle$  “Being independent is like being perpendicular.” This admittedly fuzzy assertion can be made precise in a surprising number of situations in mathematics — situations that on the face of it have nothing to do with either probabilistic independence or geometric perpendicularity! Here
- we give one simple way to connect these two notions: rectangular Venn diagrams. Represent sample space  $S$  by a rectangle. For all events  $E \subset S$ , let  $\Pr(E)$  be represented by the relative area we assign to  $E$  in the diagram, i.e.,
- $$\Pr(E) = \frac{\text{Area } E}{\text{Area } S}.$$
- Now, if independence is like perpendicularity, let's see if we can make events  $A$  and  $B$  independent by representing them as perpendicular rectangles. That is, let  $A$  be horizontal, stretching all the way across  $S$ . Let  $B$  be vertical, stretching from top to bottom of  $S$ . Now verify that
- $$\Pr(A \cap B) = \Pr(A) \cdot \Pr(B).$$
- 40.**  $\langle 3 \rangle$  It is a fact that
- $$\Pr(B \cap C|A) = \Pr(C|A \cap B) \Pr(B|A).$$
- a)** Give an intuitive explanation. *Hint:* Imagine that  $A, B, C$  are events which, if they happen at all, must happen in that order.
- b)** Of course, the events don't have to happen in the order  $A, B, C$ , so, if you used the hint in a), your proof is not really general. Give a “real” proof, using Definition 1.
- 41.**  $\langle 2 \rangle$  Prove: If  $A$  and  $B$  are independent, then so are  $A$  and  $\sim B$ , so are  $\sim A$  and  $B$ , and so are  $\sim A$  and  $\sim B$ .
- 42.**  $\langle 3 \rangle$  Definition.  $A_1, \dots, A_n$  are **pairwise independent** if each pair of them is independent (as in Definition 2).  $A_1, \dots, A_n$  are **(mutually) independent** if for all  $I \subset \{1, 2, \dots, n\}$ ,
- $$\Pr\left(\bigcap_{i \in I} A_i\right) = \prod_{i \in I} \Pr(A_i).$$
- In other words, the “product rule” for probabilities holds for any subset of these events, not just subsets of two events.
- a)** Assume  $A_1, A_2, A_3$  are mutually independent. Show that
- $\Pr(A_1 \cap A_2 \cap \sim A_3) = \Pr(A_1) \Pr(A_2) \Pr(\sim A_3)$ .
  - $A_1, A_2$ , and  $\sim A_3$  are pairwise independent.
  - $A_1, A_2$ , and  $\sim A_3$  are mutually independent.
- b)** Generalize part a).

- c) If  $A_1, A_2, A_3$  are pairwise independent does this imply that the three events are *mutually independent*?

Mutual independence is the more natural and valuable concept, so for 3 or more variables *independence* is taken to mean *mutual independence*.

43. (2) Definition. Let  $S$  be a sample space with probability measure  $\Pr$ . Let  $A$  be any subset with  $\Pr(A) \neq 0$ . Then the **induced probability measure**  $\Pr_A$  on  $A$  is defined on subsets  $C$  of  $A$  by

$$\Pr_A(C) = \frac{\Pr(C)}{\Pr(A)}.$$

Show that this definition does indeed define a

probability measure with  $A$  as its sample space. (That is, show that  $\Pr_A$  meets the conditions in Definition 1, Section 6.2.)

44. (2) Show that if  $S$  is a finite set and  $\Pr$  is the equiprobable measure on  $S$ , then any induced measure on a subset of  $S$  is also an equiprobable measure.

45. (2) The fact that we have called  $\Pr(B|A)$  a conditional *probability*, not just a conditional number, suggests that it really ought to satisfy the requirements for probability in its own right. Show that this is correct.

## 6.4 Random Variables and Distributions

---

Random variables are a device for transferring probabilities from complicated sample spaces to simple spaces whose atomic events are numbers. Once such a transfer is made, the new probability space is usually called a **distribution** — intuitively the probability is distributed among points on the real line. More important, once such transfers are made, different-looking probability spaces often turn out to be the same. In fact, a small number of distributions cover most situations. We explain these things in this section and introduce the most important discrete distributions as well as the notion of a continuous distribution.

The definition of a random variable is quite simple; only the name is mystifying. So we go right to the definition and then explain the name.

---

**Definition 1.** A **random variable** is any real-valued function on a sample space.

---

### EXAMPLE 1

Consider the usual sample space for tossing two dice:

$$\{ (i, j) \mid 1 \leq i, j \leq 6 \}.$$

Let  $X(i, j)$  be the sum of the dots when toss  $(i, j)$  occurs. That is,  $X(i, j) = i + j$ . Then  $X$  is a random variable. ■

### EXAMPLE 2

Let  $Y$  be a man's shoe size (length only, American sizes). So  $Y$  is really a function whose domain  $S$  is the set of American men. For instance,  $Y(\text{Stephen Maurer}) = 12\frac{1}{2}$ . When  $S$  is viewed as a sample space,  $Y$  is a random variable. ■

The reason for the name “variable” is simple: The quantity varies. It's just like calling  $y$  both a variable and a function of  $x$  when we consider  $y = x^2$ . The

name “random”, though unnecessary, reminds us of the probability context. If we are viewing the set of men as a sample space, presumably we have some chance events to consider. Perhaps we run a shoe store and want to know how many shoes of different sizes to stock to meet customer demand. The variable is random in that we don’t know what its value will be when the next atomic event occurs, i.e., another man enters our store.

The set of all American men is an unwieldy sample space. The elements have long names, they can’t all be represented easily in a graph, etc. Besides, we don’t particularly care *who* buys our shoes; we care about their shoe sizes. The set of shoe sizes is finite and has a natural order as points on the  $x$ -axis. It is much simpler to associate a probability with each shoe size than with each person. For instance, if size 10 has the highest probability, then we will want to stock more size 10 shoes than other sizes.

We started with the idea of a random variable, but in the previous paragraph we have gone on to a second important idea: probability *on* a random variable. In effect, a random variable from a sample space  $S$  to the real line  $R$  transfers the probabilities from  $S$  to a new sample space consisting of a subset of  $R$ . For instance, in Example 2 the new sample space consists of all those integers and half integers that are American men’s shoe sizes. The number 10 in the new space gets assigned the probability of the event in the original space consisting of all men who wear size 10 shoes. Put another way, the probability of size 10 is the probability that the next man who enters the store wears size 10.

We may formalize this transfer with two definitions.

---

**Definition 2.** Let  $X$  be a random variable on sample space  $S$  with probability measure  $\Pr$ . Then the probability that  $X$  takes on the value  $c$ , written  $\Pr(X=c)$  or  $\Pr_X(c)$ , is defined by

$$\Pr(X=c) = \Pr(\{s \in S \mid X(s)=c\}).$$

Similarly,

$$\Pr(X \leq c) = \Pr(\{s \in S \mid X(s) \leq c\}),$$

and more generally, if  $C$  is a set of real numbers, then

$$\Pr(X \in C) = \Pr_X(C) = \Pr(\{s \in S \mid X(s) \in C\}).$$


---

In the spirit of this definition, we may also use, for instance, the notation  $\{X = c\}$  to represent the event  $\{s \in S \mid X(s)=c\}$ . See Example 3.

We have used the random variable  $X$  to transfer probability from  $S$  to values  $c$  in the codomain  $C(X)$  of  $X$ . These values of the random variable do indeed give us a probability measure on  $C(X)$  (why? [11]). When this codomain is a discrete set of points, it is customary to describe this new measure using the terminology and notation of the next definition. We use  $x$  instead of  $c$  to emphasize that we have a variable.

**Definition 3.** Let  $X$  be a random variable on a sample space  $S$  with probability measure  $\Pr$  and with the codomain  $C(X)$  of  $X$  being a discrete subset of the real numbers. Then the function  $f$  defined on  $C(X)$  by

$$f(x) = \Pr(X=x) \quad (1)$$

is called the **probability distribution function** associated with  $X$ . Another name is **point distribution function**. The function

$$F(x) = \Pr(X \leq x) \quad (2)$$

is called the **cumulative distribution function** for  $X$  (or for  $f$ ). For both the word “function” may be dropped.

The letters  $f$  and  $F$  in Eqs. (1) and (2) are customary, to distinguish these probability functions on the real numbers from the underlying probability measure on  $S$ . The cumulative distribution function is just the “running sum” of the point distribution function. Note that the underlying sample space  $S$  could itself be a discrete subset of  $R$ , in which case a probability measure on it is already a distribution. In this case the transfer from  $S$  to a new subset of  $R$  may just make a problem easier to deal with.

### EXAMPLE 3

Let  $S$  be the usual equiprobable, ordered-pair sample space for tossing two dice, and let  $X(i, j) = i+j$ . (See Example 1.) Determine the point distribution and the cumulative distribution of the real-number sample space associated to  $S$  by  $X$ .

**Solution** The new sample space is the codomain of  $X$ , namely,  $\{2, 3, \dots, 12\}$ . The point distribution function is

$$\begin{aligned} f(2) &= \Pr(X=2) = \Pr(\{(1, 1)\}) = \frac{1}{36}, \\ f(3) &= \Pr(X=3) = \Pr(\{(1, 2), (2, 1)\}) = \frac{2}{36}, \\ &\vdots \\ f(7) &= \Pr(X=7) = \Pr(\{(1, 6), (2, 5), \dots, (6, 1)\}) = \frac{6}{36}, \\ &\vdots \\ f(12) &= \Pr(X=12) = \Pr(\{(6, 6)\}) = \frac{1}{36}. \end{aligned} \quad (3)$$

Thus the cumulative distribution is

$$F(2) = f(2) = \frac{1}{36},$$

$$F(3) = f(2) + f(3) = \frac{3}{36},$$

⋮

$$F(12) = f(2) + \cdots + f(12) = 1.$$

Go back to Example 3, Section 6.2. Do you see that, in computing  $f(x)$  in display (3), we have repeated exactly what we did there? Thus the explicit formula for  $\Pr(n)$  there is also a formula for  $f(n)$  here. By using formulas for summing arithmetic series, we could also obtain an explicit two-case formula for  $F(n)$  [12]. ■

Logically speaking, it is not necessary to define both a probability distribution and a cumulative distribution; each can be derived from the other. For instance, if the elements of the codomain of  $X$  are  $x_1 < x_2 < \cdots < x_n$ , then for each  $k$  it follows from (2) that

$$F(x_k) = \sum_{i=1}^k f(x_i)$$

and

$$f(x_k) = \sum_{i=1}^k f(x_i) - \sum_{i=1}^{k-1} f(x_i) = F(x_k) - F(x_{k-1}).$$

However, we have defined both concepts because they are both convenient to use. For instance, probabilities that are very natural to ask about, e.g.,  $\Pr(a < X < b)$ , are easily described in terms of  $F$ ; see [7].

*A word on terminology.* When we call something simply a “distribution” (e.g., “the uniform distribution” below), we are referring to the underlying probability space associated with some random variable (i.e., the subset of numbers on the real line and the probabilities associated with each of these). This space is fully specified by either the probability distribution function  $f$  or the cumulative distribution function  $F$  as defined in Definition 3. Therefore we (and other authors) will sometimes identify a distribution by naming one of these functions. For instance, after Example 5 below we will sometimes say “the distribution  $B_{n,p}$ ” as shorthand for “the binomial distribution, which is specified by the probability function  $f(k) = B_{n,p}(k)$ ”.

We now turn to four important (classes of) distributions. All four show up frequently in applications, and the first three show up repeatedly in other sections of this chapter.

## EXAMPLE 4

### The Finite Uniform Distributions

In this example  $S$  refers to the derived sample space, that is the set of values in the codomain of  $X$  after the random variable  $X$  has been applied to the underlying sample space. For the uniform distribution, if  $n = |S|$ , then for all  $x \in S$ , we define

$$f(x) = \frac{1}{n}.$$

In other words, the finite uniform distributions are precisely the finite distributions in which atomic events are equiprobable.

As for the cumulative distribution function, if we again label the elements of  $S$  as  $x_1 < x_2 < \dots < x_n$ , then

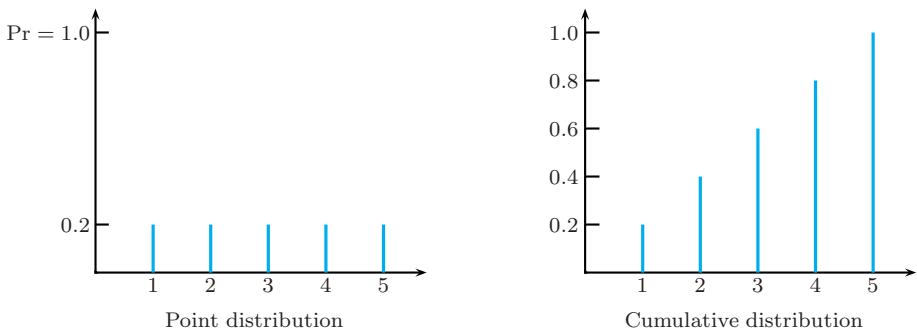
$$F(x_k) = \frac{k}{n}.$$

In some cases, the notation is even simpler. For instance, if  $S = \{1, 2, \dots, n\}$ , then  $F(k) = k/n$ .

One advantage of distributions over arbitrary probability measures is that the order on the real line makes a graphical display easy and informative. In Fig. 6.5, we show the point and cumulative distributions for the uniform measure on  $\{1, 2, 3, 4, 5\}$ . ■

**FIGURE 6.5**

The uniform distribution on  $\{1, 2, 3, 4, 5\}$ , shown through point and cumulative distribution functions with bar graphs.



## EXAMPLE 5

### The Binomial Distribution

An experiment, with probability  $p$  of success and, thus,  $1 - p$  of failure, is repeated independently  $n$  times. (We may or may not know  $p$  when we perform the experiment – see Section 6.7.) Such an experiment is called a sequence of **Bernoulli trials**. We seek the point distribution,  $B_{n,p}(k)$ , which gives the probability that exactly  $k$  trials are successes.

In the natural sample space for Bernoulli trials the atoms are the  $n$ -tuples like SSFSFFF..., where S means success and F means failure. We map this space over to the real numbers  $\{0, 1, 2, \dots, n\}$  by grouping together all  $n$ -tuples that have the same number of successes. Since the trials are assumed to be independent, the appropriate probability model involves multiplication. That is, if an  $n$ -tuple contains  $k$  S's and  $(n-k)$  F's, then the probability of the tuple is  $p^k(1-p)^{n-k}$ . Finally, there are  $C(n, k)$   $n$ -tuples with  $k$  S's (why?), and they represent disjoint events. Thus the point distribution is

$$B_{n,p}(k) = \binom{n}{k} p^k (1-p)^{n-k}. \quad (4)$$

The distribution for Bernoulli trials is called the **binomial distribution** because

of Eq. (4), which looks like a term out of the Binomial Theorem. Only occasionally is it called the Bernoulli distribution.

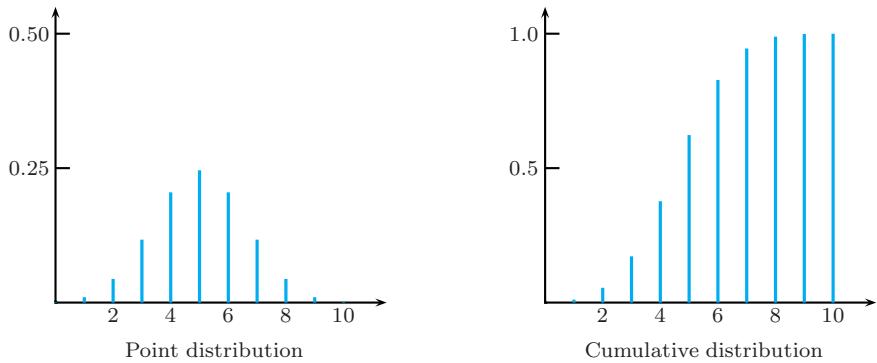
There's no simple exact formula for the cumulative binomial distribution; from the definition, we simply have to say

$$F(k) = \sum_{j=0}^k B_{n,p}(j) = \sum_{j=0}^k \binom{n}{j} p^j (1-p)^{n-j}. \quad (5)$$

Note, in particular, that it follows from this equation that

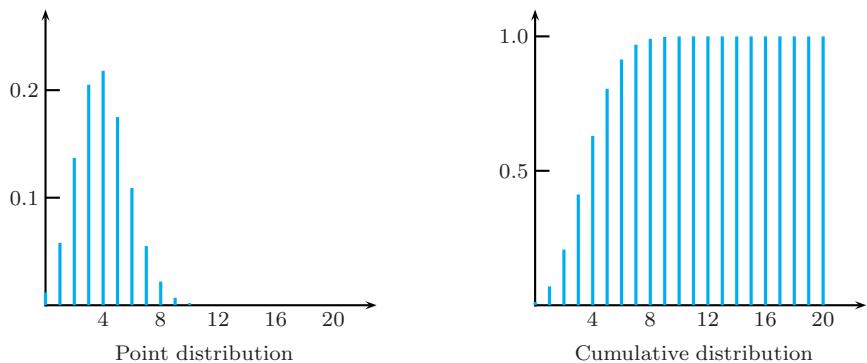
$$F(n) = [p + (1 - p)]^n = 1$$

as it should. Fig. 6.6 shows the binomial distribution in terms of  $B_{n,p}$  for  $n = 10$  and  $p = .5$ . Fig. 6.7 shows it for  $n = 20$  and  $p = .2$ . Note that the vertical scales on the point distributions have been stretched to improve readability. From these point distributions you can easily see which values of  $k$  are most likely and which are very unlikely. Are the results what you would expect? ■



**FIGURE 6.6**

The binomial point distribution for  $n = 10$  and  $p = .5$ . Note the different vertical scales.



**FIGURE 6.7**

The binomial point distribution for  $n = 20$  and  $p = .2$ .

*Note.* Throughout the study of probability, many formulas involve both  $p$  and  $(1-p)$ . To shorten things, the following notation has become the

$$\text{Probabilist's tradition: } q = 1 - p.$$

For instance, the binomial point distribution may be written as

$$B_{n,p}(k) = \binom{n}{k} p^k q^{n-k}.$$

### Solution to Example 1, Section 6.1 (Guessing on the College Boards)

The question was: What is the probability that you will improve your score? The example didn't say how many problems you guess on. Let's say it's 10. Your score on these problems is  $r - (w/4)$ , where  $r$  is your number of right answers on guesses and  $w = 10 - r$  is your number wrong. Thus in order to come out ahead, you must have

$$r - \frac{10 - r}{4} > 0 \iff r > 2.$$

Since there are 10 independent guesses, each with probability .2 of being right, we are talking about the binomial distribution  $B_{10,.2}(r)$ . Therefore the probability of improving your score is

$$\sum_{r=3}^{10} B_{10,.2}(r).$$

This expression involves eight terms. To simplify it in a way that allows us to use the cumulative distribution, we apply a little algebra:

$$\begin{aligned} \sum_{r=3}^{10} B_{10,.2}(r) &= \sum_{r=0}^{10} B_{10,.2}(r) - \sum_{r=0}^2 B_{10,.2}(r) \\ &= 1 - \sum_{r=0}^2 B_{10,.2}(r) \tag{6} \\ &= 1 - F(2). \tag{7} \end{aligned}$$

Tables have been compiled for all the common cumulative distributions, and they are built in to some calculators and most CASs. If we have access to such resources, Eq. (7) will be easy to evaluate. Even if we don't, the right-hand side of Eq. (6) is not hard to calculate. Specifically,

$$F(2) = \left(\frac{4}{5}\right)^{10} + 10\left(\frac{1}{5}\right)\left(\frac{4}{5}\right)^9 + 45\left(\frac{1}{5}\right)^2\left(\frac{4}{5}\right)^8 \approx .6778.$$

Thus the probability of improving your score by pure guessing on 10 questions is

$$1 - F(2) \approx 1 - .6778 = .3222.$$

This may seem like a pretty good chance of success for no mental effort, but before you go out and try it, also compute the probability of decreasing your score [24]. In fact, for an informed judgment, you really need to take into account the probabilities of various *amounts* of increase and decrease. In the next section, we show how to summarize such information. ■

### EXAMPLE 6

#### The Negative Binomial Distribution

Again suppose you run an experiment repeatedly and independently, but this time you keep going until you obtain your  $n$ th success. What is the probability that you need exactly  $k$  trials?

**Solution** The underlying sample space now consists of S/F-tuples of varying length. For instance, if  $n = 2$ , some of them are:

SS, FSFFFS, and FFFFSSS.

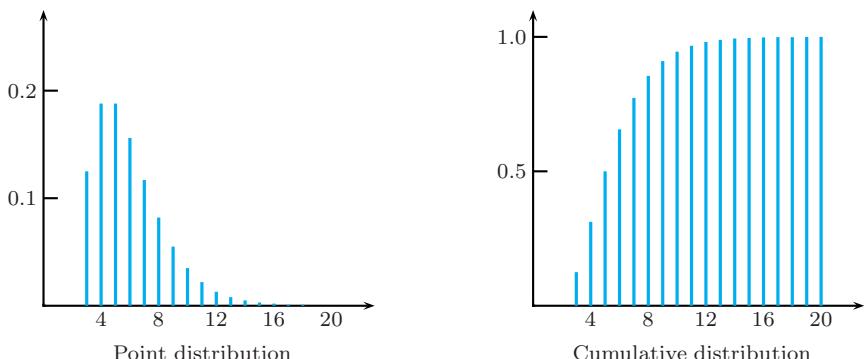
For any  $n$  there are also infinite-tuples in the space — all F's except for  $i$  S's,  $i = 0, 1, \dots, n - 1$ . (What's the probability of any one of the infinite-tuples?) Any one sequence which takes  $k$  trials to obtain  $n$  successes will have probability  $p^n q^{k-n}$ . There will be many such sequences; the final S must go at the end, but the  $n - 1$  others may go in any of the other  $k - 1$  slots. Thus the probability of achieving the  $n$ th success on the  $k$ th trial is

$$f(k) = \binom{k-1}{n-1} p^n q^{k-n}, \quad k \geq n.$$

This is called the probability distribution function for the **negative binomial of order  $n$** . (If one refers simply to the negative binomial, that means the negative binomial of order 1.) Note that the domain of the random variable  $k$  of this distribution is an infinite discrete set — all integers  $k \geq n$ . Note also that  $k \geq n$  is the reverse inequality from that in the binomial distribution; hence  $n$  and  $k$  switch roles in all the formulas. (Why don't we switch the meanings of  $n$  and  $k$  in the definition of the negative binomial and thus make their roles in the formulas the same in both distributions? Because the usage we have chosen is consistent in a more important way: In both distributions  $n$  is a *parameter*, that is, an index which indicates which of several related distributions we are talking about; and in both  $k$  is the random variable.)

Again, the cumulative distribution is obtained by summing.

Fig. 6.8 shows the negative binomial of order 3 with  $p = .5$ . We can't show the whole domain, so we have cut it off at  $k = 20$ .



**FIGURE 6.8**

The negative binomial of order 3 with  $p = .5$ .

Why is this distribution called the negative binomial? Because its terms arise in the Binomial Series Theorem (Theorem 2, Section 4.6) when  $-n$  is the exponent [29].

Note that Example 3, Section 6.1 (constructing a permutation by random selections), is really about the negative binomial distribution. We repeatedly search for another unused number, until the first success. Once the  $k$ th new number has been

selected, the probability in each succeeding trial of successfully picking a  $(k+1)$ st unused number is  $p = (n-k)/n$ . We are interested in the total number of trials before all numbers from 1 to  $n$  have been used. ■

## EXAMPLE 7

### The Poisson Distribution

A large computer receives, on average,  $\lambda$  new jobs a minute. What is the probability that exactly  $k$  new jobs will come in during a given minute?

**Solution** There are many problems of this sort. Another is: If the telephone company handles, on average,  $\lambda$  calls a minute, what is the probability that, in the course of a day, at some point they will need  $k$  circuits (if no caller is to receive an “all circuits busy” signal)?

Problems of this sort are usually modeled very well by the following probability distribution:

$$f_{\lambda}(k) = e^{-\lambda} \frac{\lambda^k}{k!}, \quad k = 0, 1, 2, \dots \quad (8)$$

The distribution with this function is called the **Poisson distribution**. Sorry about the  $\lambda$  (a Greek lambda); a roman letter would do fine (so long as it does not make you think the value must be an integer, or make you think that it varies), but  $\lambda$  is traditional. Also in Eq. (8),  $e$  is the famous base of the natural logarithms, 2.71828 . . .

Where does Eq. (8) come from? It is the limit of a Bernoulli trials model. To see this, assume that at most one new computer job comes in per second. If, on average,  $\lambda$  new jobs come in per minute, that suggests that the probability of receiving a new job in any given second is  $\lambda/60$ . From this we can calculate the probability that new jobs begin during exactly  $k$  of the 60 seconds — it's just the probability of  $k$  successes (job starts) in 60 Bernoulli trials (the seconds).

The reason this Bernoulli model isn't good enough is that it doesn't allow more than one new job to come in each second. So, let's divide seconds into tenths, and model the situation again with 600 segments in a minute instead of 60. If we keep increasing the number of segments, in the limit there is no bound on the number of jobs that could come in per minute (or per any nonzero time period). The formula we get in the limit is Eq. (8).

To show that we get Eq. (8), we need a fact from calculus about  $e$  and some familiarity with limit calculations. The fact from calculus is:

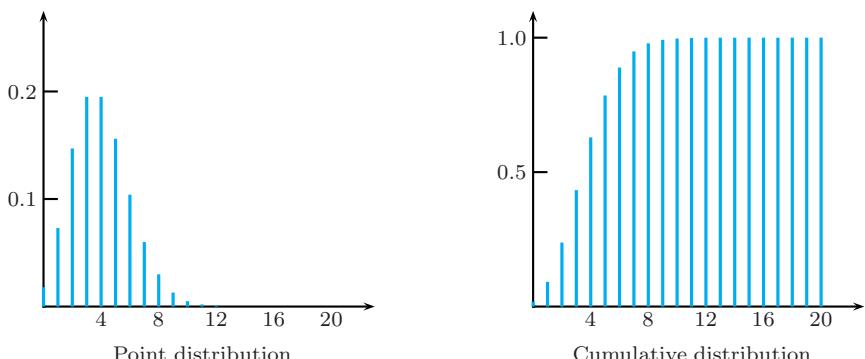
$$e^x = \lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n.$$

We use it as follows. Fix  $\lambda$  and let  $n$  be the number of intervals into which the single Poisson period is to be divided in a Bernoulli-trials approximation. Then the probability of “success” in each interval is  $p = \lambda/n$ . Hence, the probability of  $k$  successes in the  $n$  intervals in the binomial approximation is

$$\begin{aligned} \binom{n}{k} p^k q^{n-k} &= \binom{n}{k} \left(\frac{\lambda}{n}\right)^k \left(1 - \frac{\lambda}{n}\right)^{n-k} \\ &= \binom{n}{k} \left(\frac{\lambda/n}{1 - \frac{\lambda}{n}}\right)^k \left(1 - \frac{\lambda}{n}\right)^n. \end{aligned}$$

Now we factor out the terms independent of  $n$ , namely  $\lambda^k/k!$ , let  $n$  go to infinity, use the fact above from calculus, and simplify — in the limit many of the quantities in the first two factors cancel [27]. (In the last part of this section and then later in Section 6.7, we consider another way of looking at the binomial distribution as the number of trials  $n$  goes to infinity.)

Fig. 6.9 shows the Poisson point distribution for  $\lambda = 4$ . If we were to approximate this by a binomial distribution with  $n = 20$ , then  $p$  would be  $4/20 = .2$ . So the approximation is  $B_{20,.2}$ , exactly the binomial distribution in Fig. 6.7. Compare! ■



**FIGURE 6.9**

The Poisson probability distribution for  $\lambda = 4$  ( $k$  shown up to 20).

## New Distributions from Old

There are lots of ways to make new distributions directly from previous distributions. We discuss here the most useful way, addition. If  $X$  and  $Y$  are random variables on the same space  $S$ , then so is  $X + Y$ , for is it not a real-valued function? Therefore  $X + Y$  has a distribution.

But why should this interest us? Because mathematicians prefer thinking to remembering. There are a lot of distributions out there. We can either try to catalog and remember them all (ugh!), or we can seek methods of generating many from a few and remember only those few (yeah!). Reducing something to a small number of simple base cases is a general principle of good mathematics.

In fact, we will use our ability to relate random variables through addition in two ways. One is to add the variables we have already discussed and see what we get. The other is to see if we can represent the variables we have already discussed as sums of still simpler variables. The latter approach will be very helpful in Sections 6.5 and 6.8.

### EXAMPLE 8

Two distinct dice are tossed. Let  $X$  be the number of dots which come up on the first die and  $Y$  the number of dots on the second. Then  $X + Y$  is the total number of dots. We found the distribution for this sum in Example 3. It is not one of our basic distributions — it would be Bernoulli trials with  $n = 2$  except that every trial has six possible outcomes instead of two. On the other hand, the distributions for  $X$  and  $Y$  are standard distributions, namely, uniform distributions. Thus, once

we have a general method for calculating sum distributions from the summands (see the analysis after Example 10), and once we apply this method to standard distributions, then we will not need to remember the distribution of  $X + Y$  as an isolated case. ■

### EXAMPLE 9

An experiment is run  $n$  times. Let  $X_k$  be a random variable which can take on just two values as follows:

$$X_k = \begin{cases} 1 & \text{if experiment } k \text{ is a success,} \\ 0 & \text{if experiment } k \text{ is a failure.} \end{cases}$$

Then  $\sum_{k=1}^n X_k$  is the total number of successes. We already know the distribution for this sum — the binomial distribution. In other words, we have shown how to represent every binomial distribution as a sum of extremely simple distributions in which the sample space has two atomic events — success and failure — where the probability measure of one is 1 and of the other is 0. A random variable which equals 1 or 0, such as  $X_k$ , is called a **characteristic function** or **binary random variable**. ■

### EXAMPLE 10

Let  $X_n$  be the number of calls entering a central phone office during minute  $n$ . Then  $X_n + X_{n+1}$  is the number of calls entering during the two-minute stretch starting with minute  $n$ . If each  $X_i$  is Poisson with parameter  $\lambda$ , do you have any hunch about the distribution  $X_n + X_{n+1}$  will have [40]? ■

We now develop the general formula for the point distribution function of the sum of two random variables  $X$  and  $Y$ . (For sums of more variables, one proceeds by induction.) We assume the variables are discrete and that  $X$  can take on values  $x_1, \dots, x_n$ . Then in order for  $X + Y$  to take on value  $z$ ,

$$\begin{aligned} &\text{either } X = x_1 \quad \text{and} \quad Y = z - x_1, \\ &\text{or} \quad X = x_2 \quad \text{and} \quad Y = z - x_2, \\ &\quad \vdots \qquad \qquad \vdots \\ &\text{or} \quad X = x_n \quad \text{and} \quad Y = z - x_n. \end{aligned}$$

Since each pair of  $X$  and  $Y$  values are a disjoint event from any other pair, we can add probabilities and obtain

$$\Pr(X+Y=z) = \sum_{i=1}^n \Pr(X=x_i \text{ and } Y=z-x_i). \tag{9}$$

This is as far as we can go, unless we assume that  $X$  and  $Y$  are independent random variables [43], in which case we get

$$\Pr(X+Y=z) = \sum_{i=1}^n \Pr(X=x_i) \Pr(Y=z-x_i). \tag{10}$$

Finally, it's a nuisance to have all these subscripts, and to have to declare how many points are in the codomain  $C(X)$  of  $X$ . So we simply write

$$\Pr(X+Y=z) = \sum_x \Pr(X=x) \Pr(Y=z-x). \quad (11)$$

We can think of this sum as being over all real numbers  $x$ ; if  $x \notin C(X)$ , then  $\Pr(X=x) = 0$ , so the sum reduces to the same finite sum as in Eq. (10). Similarly, even when  $\Pr(X=x) \neq 0$ , it may be that  $\Pr(Y=z-x) = 0$ , which reduces the number of terms even further.

### EXAMPLE 11

Find the distribution of  $X + Y$  if  $X$  has distribution  $B_{n,p}$ ,  $Y$  has distribution  $B_{m,p}$ , and  $X$  and  $Y$  are independent.

**Solution** From Eqs. (4) and (11), we obtain

$$\begin{aligned} \Pr(X+Y=k) &= \sum_j \left[ \binom{n}{j} p^j q^{n-j} \right] \left[ \binom{m}{k-j} p^{k-j} q^{m-(k-j)} \right] \\ &= \left[ \sum_j \binom{n}{j} \binom{m}{k-j} \right] p^k q^{m+n-k}. \end{aligned}$$

The sum in the last line runs over integers  $j$  from 0 to  $n$  (why?). We saw in [19] of Section 4.5 that

$$\sum_j \binom{n}{j} \binom{m}{k-j} = \binom{m+n}{k}.$$

We conclude that

$$\Pr(X+Y=k) = \binom{m+n}{k} p^k q^{m+n-k}.$$

In short,  $X + Y$  also has a binomial distribution, namely, the probability function is  $B_{m+n,p}$ . With hindsight, we can justify this conclusion without a single computation. Let  $X$  and  $Y$  be defined on the sample space of  $m+n$  Bernoulli trials with success probability  $p$ . Let  $X$  be the number of successes in the first  $n$  trials and  $Y$  the number in the last  $m$ . They are certainly independent. In this context, what does  $X + Y$  measure? ■

## Continuous Distributions

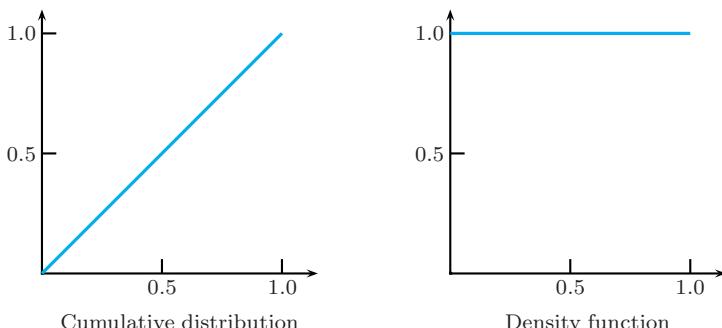
Why introduce some continuous mathematics in a book on discrete mathematics? One reason is to illustrate that there is no clear dichotomy between discrete and continuous mathematics. Indeed, discrete mathematics often provides good approximations to continuous mathematics when the latter is intractable. Examples are approximating differential equations by difference equations and using sums to approximate integrals. But sometimes it works the other way, as here: Continuous distributions provide good approximations to discrete distributions when the size  $n$  of the sample space becomes too large to deal with conveniently. We shall use this approximation fact in Section 6.7 where a continuous distribution will play an important role.

Like discrete distributions, continuous distributions have two functions associated with them, one called as before the cumulative distribution function, and the other called the **probability density function**. The cumulative distribution function  $F(x)$  has the same meaning as in the discrete case:  $F(x)$  is the probability that the random variable has a value  $\leq x$ . However, the value of the density function at  $x$ ,  $f(x)$ , does *not* mean the probability of getting the value  $x$ , because for continuous distributions the probability of getting any one value is 0 (see [18] for a careful explanation). In continuous distributions only intervals of  $x$ -values can have positive probability, and the interpretation of the density function is that  $\Pr(a \leq x \leq b)$  is the area under the graph of  $f$  between the vertical lines at  $x = a$  and  $x = b$ .

(What is the relation between  $F(x)$  and  $f(x)$ ? We have just said that probability is area under  $f(x)$ . So readers who know some calculus will note that  $F(x)$  is the integral of  $f(x)$  over all values  $\leq x$ . We noted earlier that, in the discrete case,  $F(x)$  is the running sum of  $f(x)$ . Well, integrals are the continuous analog of sums. So there really is a complete parallel between  $f(x)$  in the discrete and continuous cases, even though in the continuous case  $f(x)$  is not the probability at  $x$ .)

As our first illustration, consider the uniform distribution with sample space the interval  $[0, 1]$ . The density and cumulative functions are shown in Fig. 6.10. The density function is the constant function  $f(x) = 1$  on the domain  $[0, 1]$ . This works because it makes the probability of each interval  $[a, b]$  equal to  $b - a$ , the area of the height 1 rectangle over that interval and under  $f$ . And sure enough, in a continuous uniform distribution the probability that  $x$  is in an interval should be proportional to its length.

The cumulative distribution function for this uniform distribution is  $F(x) = x$ , since, for instance, the chance of getting  $x \leq \frac{3}{4}$  is  $\frac{3}{4}$ . (Note to those who know calculus: Sure enough,  $F(x)$  is the integral of  $f(x)$  [19].)

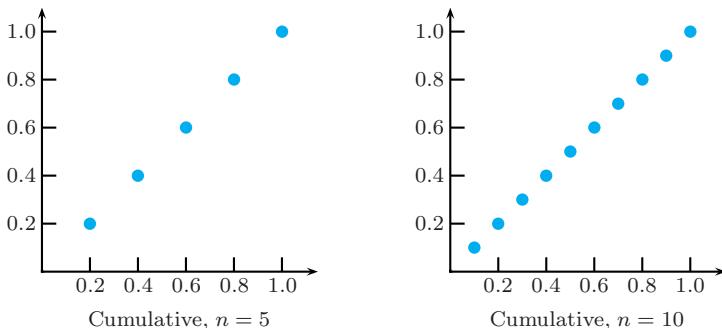


**FIGURE 6.10**

The continuous uniform distribution on the interval  $[0, 1]$ : cumulative distribution and density function.

We said that continuous distributions can approximate discrete ones and vice versa, so look also at Fig. 6.11. Here we show the cumulative distribution for two discrete uniform distributions with  $n$  points ( $n = 5, 10$ ), where we have made the sample space the set of points  $j/n$ ,  $j = 1, 2, \dots, n$ , instead of just  $j = 1, 2, \dots, n$  as in our earlier discussion, and we have shown the cumulatives as point graphs instead of bar graphs, to parallel the presentation for continuous distributions. As

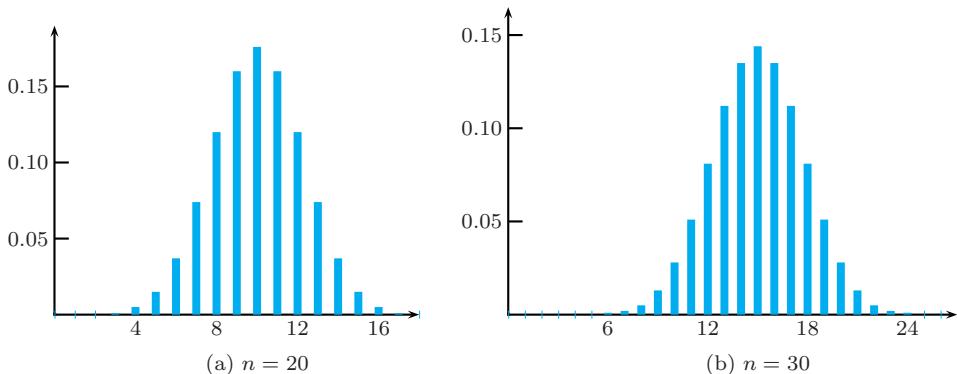
$n \rightarrow \infty$ , these discrete cumulative distributions approach the continuous cumulative distribution in Fig. 6.10.



**FIGURE 6.11**

The discrete uniform cumulative distribution on the interval  $[0, 1]$  with  $n = 5$ , then  $n = 10$ .

We just illustrated a nice relationship between discrete and continuous uniform cumulative distribution functions. Is there also a limit relationship between discrete probability distribution functions and continuous density functions? Yes, but only if you scale the discrete functions appropriately. We don't want to get into any details until Section 6.7, but we show a picture. Consider the binomial distribution  $B_{n,.5}$ . As  $n \rightarrow \infty$ , the probability function  $f(x)$  looks more and more like a nice smooth hump. See Fig. 6.12, where we show  $B_{n,.5}$  for  $n = 20, 30$ . In fact, the limit here is the famous **normal density function**, which, together with its associated cumulative distribution, is generally referred to as the normal distribution.



**FIGURE 6.12**

The binomial probability distribution for  $p = .5$  and (a)  $n = 20$ , (b)  $n = 30$ , scaled to indicate convergence to the normal distribution.

## Problems: Section 6.4

1. {1} For the uniform distribution on

$$S = \{10, 11, \dots, 99\},$$

what is

- a)  $\Pr(25)?$  b)  $\Pr(30 \leq x \leq 40)?$

2. {2} Consider the uniform continuous probability distribution on the interval  $[1, 9]$ .

- a) What is the cumulative probability function  $F(x)$ ?  
b) What is the probability that  $x \in [2, 4]$ ?

3. ⟨2⟩ Consider the binomial distribution with  $n = 3$  and  $p = .4$ . Make a complete table of values for its point and cumulative distribution functions.
4. ⟨2⟩ Consider the Poisson distribution with  $\lambda = 2$ . What is the probability that the random variable of this distribution, call it  $k$ , satisfies
- $k = 2?$
  - $k \leq 2?$
  - $k \geq 2?$
5. ⟨2⟩ Let sample space  $S$  be the set of 6 points on the plane with  $x$ -coordinate in  $\{-1, 0, 1\}$  and  $y$ -coordinate in  $\{0, 1\}$ . Let the probability measure on  $S$  be equiprobable. Let  $X$  be the random variable that gives the distance of points in  $S$  from the origin. What is the sample space  $S'$  to which  $X$  transfers the probabilities, and what is the probability distribution on  $S'$ ?
6. ⟨2⟩ Let  $S$  be the sample space of all sequences of 3 tosses of a coin, so the 8 atoms are  $(H, H, H)$ , etc. Assume the coin is biased, and each toss results in a head with probability  $.6$ . Let  $X$  be the number of heads in 3 tosses. What is the associated sample space associated with  $X$ ? That is, what are the atoms in the space and what are their probabilities? Is this one of the standard sample spaces discussed in the section?
7. ⟨2⟩ The cumulative distribution function tells us  $\Pr(X \leq x)$ , but this may not be very interesting: It's more likely that you would want to know the probability that  $X$  is in some interval. However, every such question can be answered by combining values of the cumulative function — hence it's importance. Let  $x_1 < x_2 < \dots < x_n$  be the values of random variable  $X$ . Express each of the following using the cumulative function  $F(x) = \Pr(X \leq x)$ .
- $\Pr(X > x)$
  - $\Pr(X \geq x_i)$
  - $\Pr(x_i < X \leq x_j)$
  - $\Pr(x_i < X < x_j)$
  - $\Pr(x_i \leq X \leq x_j)$
  - $\Pr(x_i \leq X < x_j)$
8. ⟨3⟩ What is the best probability distribution to use to study each of the questions a)–g)? Be as precise as you can; if the distribution you choose has parameters, and you can give their specific values, do so. On the other hand, some of these situations are vaguely stated, and there isn't necessarily one right distribution to use. Maybe no distribution we have studied is appropriate. Be prepared to defend your choice.
- a) How many 6's will show up in ten throws of a fair die?
- b) How many 7's will show up in ten throws of two fair dice?
- c) How many customers will arrive at the First National Bank between 10 and 11 A.M.?
- d) How many alpha particles will be emitted in the next minute from a 1-kg piece of radioactive uranium?
- e) How many throws of two fair dice will be necessary to get a 7?
- f) How many throws of one fair die will be necessary to get 6 twice?
9. ⟨2⟩ Suppose that our sample space is the set of all possible infinite sequences of tosses of a fair coin. Let the random variable  $X$  give, for each infinite sequence, the number of the trial when the first H appears. What distribution does  $X$  have?
10. ⟨2⟩ Let us go back to the weather situation in [7, Section 6.3] Let our sample space  $S$  consist of the 4 possible ordered pairs for the weather on two consecutive days:  $(\text{good}, \text{good})$ ,  $(\text{good}, \text{bad})$ , etc.
- Figure out the probability for each atom of  $S$ . Remember, weather on consecutive days is not independent.
  - Let  $X$  count the number of good days in our 2-day period.  $X$  transfers the problem onto the sample space  $S' = \{0, 1, 2\}$ . Figure out the probabilities of the atoms in  $S'$ .
11. ⟨2⟩ Let  $X$  be a random variable with domain some sample space  $S$  and codomain some finite set  $R'$  within the real numbers  $R$ . Show that Definition 2 does, indeed, define a probability measure on  $R'$  according to the definition in Section 6.2. We say that  $X$  **induces** a probability measure on  $R'$  from the original measure on  $S$ .
12. ⟨2⟩ Find an explicit formula for  $F(n)$  in Example 3. Your formula will have two cases:  $1 \leq n \leq 7$  and  $7 < n \leq 12$ .
- Problems 13–16 are best done with a computer having graphics capabilities, or a graphing calculator. At the very least, use a calculator with a few scientific function keys. *Caution:* For some of the problems, you will have to think hard about how to compute the binomial coefficients. If you just use the definition directly, the factorials will cause overflows or underflows.
13. ⟨2⟩ Compute, or plot, the Poisson probability distribution, for  $k = 0, 1, \dots, 10$ , in the cases  $\lambda = 1, 2, 3, 4$ .

14.  $\langle 2 \rangle$  Compute, or plot, the point distribution function for the negative binomial distribution (order 1) for  $k = 0, 1, \dots, 10$ , and  $p = .3, .5$ , and  $.7$ .

15.  $\langle 2 \rangle$  Repeat [14], but for the negative binomial of order 2.

16.  $\langle 2 \rangle$  Repeat [14], but for order 3.

17.  $\langle 2 \rangle$  A probability distribution is **symmetric** if  $f(x) = f(-x)$  for all real numbers  $x$ . If  $F(x)$  is the cumulative function for a symmetric distribution, what is the probabilistic interpretation of  $G(x) = 2F(x)-1$ ? Over what domain for  $x$  does this interpretation hold?

18.  $\langle 2 \rangle$  Why does the probability of any one point in a continuous distribution equal 0? This problem considers two answers to this question.

a) Intuitively, what continuous means is that what happens at one point cannot be much different from what happens at nearby points. So suppose in a continuous distribution some point  $a$  has probability  $\epsilon > 0$ . Then at each of the infinitely many points sufficiently nearby, the probability is at least  $\epsilon/2$ . But this is impossible; why? We have reached a contradiction, so no point has positive probability.

b) More precisely, for a distribution to be continuous we mean that there is a continuous function  $f$  so that for every interval  $[a, b]$ ,  $\Pr(a \leq x \leq b)$  is the area under  $f(x)$  over  $[a, b]$ . Explain how this forces  $\Pr(a) = 0$  for every  $a$ .

By the way, distributions don't have to be purely discrete or purely continuous. There are distributions where some points have positive probability and the rest of the probability is spread continuously.

19.  $\langle 2 \rangle$  (Assumes calculus) Formally, for continuous distributions, the cumulative  $F(x)$  is defined from the density  $f(x)$  by

$$F(x) = \int_{-\infty}^x f(t) dt. \quad (12)$$

The lower bound of  $-\infty$  allows us to not worry about just what part of the real line is in the sample space. For instance, if  $S = [0, 1]$  as in Fig. 6.10, then Eq. (12) means  $\int_0^x f(t) dt$ . The two integrals are equal because  $f(x)$  is always understood to be 0 outside of  $S$ .

With this understanding, verify that Fig. 6.10 is correct: if  $f(x) = 1$  on  $[0, 1]$  as shown there, then  $F(x) = x$  for  $x \in [0, 1]$ .

20.  $\langle 2 \rangle$  An urn contains  $r$  red balls and  $b$  black balls. A ball is removed at random, the color is noted, and the ball is returned to the urn. This procedure is called **selection with replacement**. Suppose that it is repeated  $n$  times. Let  $X$  be the number of times a red ball is selected. Determine the probability distribution of the random variable  $X$ . It's one you've seen before.

21.  $\langle 3 \rangle$  An urn contains  $r$  red balls and  $b$  black balls. A ball is removed at random, the color is noted, and the ball is thrown away. This procedure is called **selection without replacement**. Suppose that it is repeated  $n$  times. (Note that  $n \leq r+b$ .) Let  $Y$  be the number of times a red ball is selected. Determine the probability distribution of the random variable  $Y$ . This is called the **hypergeometric distribution**.

22.  $\langle 3 \rangle$  A widgit manufacturer has trouble with her production line. A certain machine malfunctions one quarter of the time, but in a way that cannot be directly detected. She finds out about it only because more widgits than usual are defective when tested. If the machine is running properly, only 1 widgit in each batch of 10 is defective. If the machine malfunctions, 3 out of 10 are defective. A batch of 10 is produced, and the manufacturer has her quality control specialist choose two of the widgits (without replacement) and test them. One of the two is defective.

a) What is the probability of this outcome (one of two defective) if the machine is running properly?

b) What is the probability of this outcome if the machine is malfunctioning?

c) What is the probability that the machine is malfunctioning?

23.  $\langle 3 \rangle$  Do [22] again, but now the troublesome machine is running properly if, for each widgit independently, the probability that it is defective is .1; and the machine is malfunctioning if, for each widgit independently, the probability that it is defective is .3.

24.  $\langle 2 \rangle$  Consider Example 1, Section 6.1 (College Boards) again.

a) As in the solution in the text, assume that you guess on 10 questions. What is the probability that your score goes down? That it stays the same?

b) Now assume that you guess on 20 questions. Determine the probabilities that your score goes up, stays the same, or goes down. Compare these numbers to those in part a) and the text. Do you have an intuitive explanation for what's going on?

25. (2) Some multiple choice exams with five answers per question only take off 1/5 point for a wrong answer. If you guess at random on 10 questions on such an exam, what is the probability that you improve your score?

26. (2) Back to the College Boards, with 1/4 off for each wrong answer. Suppose that you can eliminate one answer on each of 10 questions, and guess randomly among the remaining answers. Now what is the probability that you increase your score?

27. (2) Complete the derivation of the Poisson distribution from the binomial distribution, as outlined at the end of Example 7.

28. (3) We have used the fact that the binomial distribution approaches the Poisson distribution to motivate the claim that the Poisson distribution is appropriate for situations like telephone calls. But as with all limit relationships, it cuts both ways. Suppose that a problem satisfies the binomial distribution exactly, but  $n$  is large and  $p$  is small. Then it is easier (and usually sufficient) to find approximate probabilities using the Poisson distribution than to get exact values using the binomial distribution.

a) Approximate  $B_{100,01}(k)$  for  $k = 0, 1, 2$ . Compare your Poisson approximation with the actual binomial value.

b) Approximate  $B_{50,.98}(k)$  for  $k = 48, 49, 50$ . *Caution:*  $p$  is not small, so you have to rewrite the binomial probabilities before you can approximate them with the Poisson distribution.

29. (2) Use the Binomial Series Theorem (Theorem 2, Section 4.6) to show that

$$(1-q)^{-n} = \sum_{k=n}^{\infty} \binom{k-1}{n-1} q^{k-n}$$

and conclude that

$$\sum_{k=n}^{\infty} \binom{k-1}{n-1} p^n q^{k-n} = 1.$$

What is the probabilistic interpretation of this equality using the negative binomial distribution?

30. (3) *The Trinomial Distribution.* An experiment has three possible outcomes,  $a$ ,  $b$ , and  $c$ , with probabilities  $p$ ,  $q$ , and  $r$ . The experiment is repeated independently  $n$  times. Let  $f_n(k, j)$  be the probability that event  $a$  happens  $k$  times and  $b$  happens  $j$  times. Find a formula for  $f_n$ . Note:  $f_n$  is not a distribution function in the sense of Definition 3 because its domain is a subset of  $\mathbb{R} \times \mathbb{R}$ , not  $\mathbb{R}$ . In many books the definition of distribution is broader to include functions like  $f_n$ .

31. (2) *The Multinomial Distribution.* Figure out what this is — what situation it models and what the formula for it must be. See Theorem 3, Section 4.6.

32. (3) For the uniform distribution, clearly each value of the variable is equally likely. For most distributions, some one value is most likely. It is called the **maximum likelihood value**. What is the maximum likelihood value for each of the other discrete distributions we have introduced? Try to guess the answer; then use the method described next to find out for sure.

Let  $a_1, a_2, \dots$  be a sequence of positive numbers. Define the ratio  $r_k = a_{k+1}/a_k$ . Suppose that the sequence  $\{r_k\}$  is decreasing and that  $n$  is the first index for which  $r_n < 1$ .

- Show that  $a_n$  is the maximum term in  $\{a_k\}$ .
- Suppose, in addition, that  $r_{n-1} = 1$ . What does this say about the maximum of  $\{a_k\}$ ?
- Apply your results in parts a) and b) to the binomial distribution  $B_{n,p}$ .
- Apply it to the negative binomial.
- Apply it to the Poisson distribution.

33. (3) People often believe that if you toss a coin 100 times, the most likely outcome is 50 heads. That's true, but they often believe even more: That this event is *quite* likely, or at least, almost certainly the number of heads will fall within a few of 50. Furthermore, they believe that if 100 is replaced by, say, 1000, and 50 by 500, then the certainty of getting 500 heads, or being within a few of 500, is

even greater. This problem asks you to check out these beliefs.

a) With  $p$  fixed and  $n$  increasing, what happens to  $B_{n,p}(k)$ , where  $k$  is the maximum likelihood value? (See [32].) Base your answer on computations, using  $p = .5$  and  $n = 10, 20, 50$ .

b) Repeat part a) with  $p = .7$ .

c) With  $p$  fixed and  $n$  increasing, what happens to the probability that  $k$  is within 2 of the maximum likelihood value (see [32])? That is, if the maximum likelihood value is 10,  $k$  should be from 8 to 12. Use the same values for  $p$  and  $n$  as in parts a) and b), but if you aren't certain of the pattern, try some larger  $n$ 's.

d) With  $p$  fixed and  $n$  increasing, what happens to the probability that  $k$  is within  $.1n$  of the maximum likelihood value? That is, if that value is 10,  $k$  should be within  $10 \pm .1n$ . The advice about  $p$  and  $n$  given in part c) holds.

34. ⟨3⟩ Repeat [33] for the Poisson distribution: that is, check if the most likely value gets more likely. You might think this problem doesn't make any sense: There is no  $n$ , representing a number of trials, to let grow. But reason as follows. If  $\lambda$  is a measure of the number of phone calls expected per time period, then if we take longer and longer time periods, we should use larger and larger values for  $\lambda$ . Furthermore, since a longer time period should mean a more definitive sample, for large  $\lambda$  we should expect the probability to be concentrated around the most likely value. So when we say repeat [33], we mean use increasing values of  $\lambda$ . Find out what happens to the probability of the most likely value — by itself,  $\pm$  a constant amount, and  $\pm$  a percentage amount.

35. ⟨3⟩ Do the analysis of [33] for the negative binomial distribution.

36. ⟨3⟩ [33] and its variants can, of course, be answered analytically, but you need some knowledge of limits of binomial coefficients to do it. The fundamental tool for this sort of thing is **Stirling's Formula**:

$$n! \sim \sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n}. \quad (13)$$

The  $\sim$  (read “is asymptotic to”) means that, as  $n \rightarrow \infty$  the ratio of the right side to the left side goes to 1. This means the right side can be substituted for the left side in approximating probabilities. (Compare with [26, Section 0.3].)

Use Eq. (13) to show that the probability of the most likely value of  $B_{n,p}$  is approximately  $1/\sqrt{2\pi np(1-p)}$ . So what is the answer to [33], parts a) and b)?

37. ⟨2⟩ Compute the point distribution for the number of dots on two dice using Eq. (11) and Example 8.

38. ⟨2⟩ Derive a formula for the point distribution of  $X - Y$ ,

- a) without assuming  $X$  and  $Y$  are independent;
- b) assuming they are independent.

You may assume in both cases that both variables have finite codomains.

39. ⟨4⟩ Let  $X_{n,p}$  be the number of the trial in which the  $n$ th success occurs for Bernoulli trials with probability of success  $p$ . It is a fact that, if  $X_{n,p}$  and  $X_{m,p}$  are independent, then their sum has the distribution of  $X_{m+n,p}$ .

a) Justify this fact conceptually as follows. Let the sample space for both  $X_{n,p}$  and  $X_{m,p}$  be the set of all possible infinite Bernoulli trials. Interpret  $X_{n,p}$  in the usual way, but reinterpret  $X_{m,p}$  to be the number of additional trials needed to obtain  $m$  more successes after the  $n$ th success. This reinterpretation doesn't change the distribution of  $X_{m,p}$ . Why? Now, what does  $X_{n,p} + X_{m,p}$  count?

b) Prove the fact algebraically in the special case  $n = m = 1$ , using Eq. (10).

c) Prove it algebraically in general. Hint: Mimic Example 11. As in that example, the powers of  $p$  and  $q$  will combine nicely, and you will be left with a sum of products of binomial coefficients — but a different product than before. Use a combinatorial argument to reduce that sum to a single binomial; you already know what binomial coefficient that must be, if the fact is to be true.

40. ⟨3⟩ Let  $X_\lambda$  have the Poisson distribution with parameter  $\lambda$ . If  $X_\lambda$  and  $X_\mu$  are independent, guess and prove the distribution of their sum.

41. ⟨3⟩ Consider a dart board of radius 1 unit. Assume a dart is equally likely to hit any point on the board. What is the probability  $F(x)$  that the dart hits  $\leq x$  units from the center. Note: this distance is a continuous random variable, and you are computing its cumulative distribution function.

42. ⟨3⟩ Let  $X$  be a continuous uniform random variable on the interval  $[0, 1]$  and consider the new random variable  $Y = X^2$ . Find the cumulative distribution function for  $Y$ .

43. ⟨3⟩ *Independent variables.* We talked about independent variables, and multiplied their probabilities together, as if we had defined what we were talking about long ago. However, we have defined independence only for *events*, never for *variables*. To put our talk on a firm foundation, we have to explain independence of variables in terms of independence of events.

**Definition 4.** Let  $X$  and  $Y$  be random variables on space  $S$ , with codomains  $C$  and  $C'$ , respectively. Then  $X$  and  $Y$  are **independent** if for any  $A \subset C$  and  $B \subset C'$  the events  $\{X \in A\}$  and  $\{Y \in B\}$  are independent; in other words, if

$$\Pr(X \in A \text{ and } Y \in B) = \Pr(X \in A) \Pr(Y \in B). \quad (14)$$

Recall that  $\{X \in A\}$  really is an event since it is shorthand for  $\{s \in S \mid X(s) \in A\}$ . The most useful case of Eq. (14) is when  $A = \{x\}$  and  $B = \{y\}$ . We get

$$\Pr(X=x \text{ and } Y=y) = \Pr(X=x) \Pr(Y=y), \quad (15)$$

which is what we used in the text.

Prove: For random variables with finite codomains, Eqs. (14) and (15) are equivalent. That is, Eq. (14) is true for all  $A$  and  $B$  iff Eq. (15) is true for all  $x$  and  $y$ .

44. ⟨3⟩ Review the definition of  $n$  events being pairwise or mutually independent from [42, Section 6.3]. In light of [43] above, define what it means for 3 or more random variables to be pairwise independent; to be mutually independent. As with events, the more useful concept is mutual independence, so when several random variables are said to be independent, that means they are mutually independent.

## 6.5 Expected Value

Now that we can associate numbers with a probability space (via random variables or distributions), we can combine those numbers into a single summarizing number. Such a number is called a **statistic**. The most important statistic is the average or expected value. Recall that the average of the numbers  $x_1, x_2, \dots, x_n$  (no probability space yet) is defined by

$$\text{average} = \frac{1}{n} \sum_{i=1}^n x_i. \quad (1)$$

Formula (1) may also be applied to an equiprobable sample space, but when different sample space points have different probabilities a modified definition is necessary.

In this section we define and motivate the general definition of average and develop its key mathematical properties. Later in Section 6.8 we use our knowledge of expected value to determine the average run time of various algorithms.

The average run time is important because it serves as a *predictor* of which of two (or more) algorithms for the same task may be expected to be most efficient. Suppose then that you have two different algorithms for solving the same problem. They may take different numbers of steps, depending on the data; sometimes the first algorithm may be faster, at other times the second. How do you choose between them? Generally you should pick the one with the smaller average or *expected* value. It will generally finish the job sooner.

The usefulness of expected value is not limited to analyzing algorithms. Suppose you have the choice of several investments, each of which involves some risk. You can choose among them using their expected values [17–20].

Even when you have no choices, an expected value can be useful as a *descriptor*. Suppose there is only one section of the math course you plan to take next term. Suppose the average grade given last year by the professor who teaches that course was low. You better plan to work hard! Or consider that average from the professor’s point of view. Suppose it is lower than the average grade for the same course two years ago. Then the professor can conclude that last year’s class wasn’t so smart — either that or she’s getting tougher.

To see the need to generalize formula (1), imagine you are in a contest that offers a very large first prize, to be given to one person, and a modest second prize, to be given to 100 people. Clearly, if you want to summarize your expectations, you should take a weighted average, with the weights determined by the different probabilities of winning first prize, second prize, or nothing. This leads to the following definition. It has two parts because we use both random variables and distributions.

---

**Definition 1.** The **expected value** of a discrete random variable  $X$ , written  $E(X)$ , is

$$E(X) = \sum_x x \Pr(X=x), \quad (2)$$

where  $x$  runs over all values in the codomain of  $X$ . Expected value is also called **expectation**, **mean** or **average** and is often denoted by  $\bar{X}$ , or even  $\overline{x}$ . Similarly, the expected value of a discrete probability distribution  $f$  is

$$E(f) = \sum_x xf(x), \quad (3)$$

where  $x$  runs over the domain of  $f$ .

---

The two parts of the definition are two sides of the same coin. For a random variable  $X$ , the associated distribution,  $f(x) = \Pr(X=x)$ , has the same expectation. When discussing expectations, we will move freely between random variable and distribution language, using whatever seems most natural at the time.

In Definition 1,  $x$  in the sums may take on a finite or at most a countably infinite number of values. (“Countably infinite” means that these values can be labeled  $x_1, x_2, x_3, \dots$ ) In the finite case, Eqs. (2) and (3) are ordinary finite sums. In the countably infinite case, they are the limits of finite sums — something we shall continue to deal with informally.

## EXAMPLE 1

Show that the expectation of an equiprobable variable (i.e., uniform distribution) is the same as the average of its values.

**Solution** Let  $x_1, \dots, x_n$  be the values of  $X$ . We are given that  $\Pr(X=x_i) = 1/n$  for each  $i$ . Hence

$$E(X) = \sum_{i=1}^n x_i \frac{1}{n} = \frac{1}{n} \sum_{i=1}^n x_i.$$

The last expression is just the average. ■

**Solution to Example 4, Section 6.1 (Should you mail in your lottery number?)** Recall that there is one prize, worth \$10,000, but you have to pay the postage (\$.37 in 2004) if you participate. Suppose you do participate. Let  $X$  be your net gain.  $X$  is a random variable which takes on one of two values: \$9999.63 if you win,  $-\$.37$  if you lose. Let  $p$  be the probability of winning. Thus

$$E(X) = p(9999.63) + (1-p)(-.37) = 10000p - .37.$$

Since the net gain if you don't participate is 0, it is worth participating if

$$10000p - .37 > 0, \quad \text{that is,} \quad p > .000037.$$

Presumably, every participant has an equal chance of winning, so we set  $p = 1/n$ , where  $n$  is the number of participants. Thus we conclude that

$$E(X) > 0 \iff n < \frac{1}{.000037} \approx 27,000.$$

Of course, you don't know what  $n$  is! But you can estimate it. Presumably, the lottery company doesn't waste numbers. They probably sent out a number of invitations not many fewer than the number on your ticket. The lottery tickets of this sort that we have received have numbers in the tens of millions. Even if only 1 person in a 100 replies (a low estimate, we think), it's not worth it to play. (For a more subtle approach to this problem, see [6] and [8, Supplement]). ■

## EXAMPLE 2

What is the expected count when two dice are tossed?

**Solution** Let  $X$  be the count. Then

$$E(X) = \sum_{i=2}^{12} i \Pr(X=i). \tag{4}$$

Assuming as always that the 36 pairs  $(i, j)$  are equiprobable and using the values given in (3), Section 6.4, Eq. (4) becomes

$$\begin{aligned} E(X) &= 2 \cdot \frac{1}{36} + 3 \cdot \frac{2}{36} + 4 \cdot \frac{3}{36} + \cdots + 7 \cdot \frac{6}{36} + 8 \cdot \frac{5}{36} + \cdots + 11 \cdot \frac{2}{36} + 12 \cdot \frac{1}{36} \\ &= \frac{252}{36} = 7. \end{aligned}$$

Such a simple answer (7 is halfway between the minimum number of dots, 2, and the maximum, 12) should have a simpler computation. It does. You will be able to do the simpler computation later in this section, once you learn the various methods for simplifying expectation computations [4]. ■

We now compute the expectation for some of the standard distributions.

### EXAMPLE 3

What is  $E(B_{n,p})$ , the expectation for the binomial distribution  $B_{n,p}(k)$ ?

**Solution** By definition, the answer is

$$\sum_{k=0}^n k \binom{n}{k} p^k q^{n-k} \quad (q = 1-p). \quad (5)$$

Can this be simplified? It looks tough, but there are lots of successful ways to attack it. We will show two in the text and leave others for the problems [22, 23].

To begin with, let's conjecture what the answer is on purely conceptual grounds. There are  $n$  trials and each has probability  $p$  of success. Therefore we should expect  $np$  successes. The only question is: How do we make Eq. (5) simplify to this?

*Method 1: Direct algebraic attack.* Does Eq. (5) look like anything you've seen before? Yes: If we throw out the initial factor  $k$ , we have the expression from the Binomial Theorem with  $x = p$  and  $y = q$ . This realization suggests that our attack may be more successful if we generalize by ignoring the known relationship of  $p$  and  $q$ . So, to use the more common notation of the Binomial Theorem, we replace  $p$  by  $x$  and  $q$  by  $y$  (where now  $y$  varies independently of  $x$ ) and try to simplify

$$\sum_{k=0}^n k \binom{n}{k} x^k y^{n-k}. \quad (6)$$

What we somehow need to do is get rid of the initial factor  $k$  in expression (6), thus reducing it to the Binomial Theorem. We use the identity

$$k \binom{n}{k} = n \binom{n-1}{k-1}, \quad (7)$$

which is easily proved either algebraically or combinatorially; see [1, Section 4.5]. Substituting this identity into (6), we obtain

$$\sum_{k=1}^n n \binom{n-1}{k-1} x^{1+(k-1)} y^{(n-1)-(k-1)},$$

where we omitted the term in  $k = 0$ . (Why could we do this?) The multiple appearance of  $k - 1$  suggests making  $k - 1$  the basic variable and then giving it a new, single-letter name, say  $j$ . Doing this, the expression above may be rewritten

$$nx \sum_{j=0}^{n-1} \binom{n-1}{j} x^j y^{(n-1)-j}. \quad [j = k-1]$$

Then applying the Binomial Theorem to the sum, we get

$$\sum_{k=0}^n k \binom{n}{k} x^k y^{n-k} = nx(x+y)^{n-1}.$$

Finally, setting  $x = p$  and  $y = q$ , we get that  $E(B_{n,p}) = np$ .

*Method 2: Recursion.* As usual, recursion is shorter and sweeter, but more subtle. Let  $E_n = E(B_{n,p})$ . The key idea is: If we are running  $n$  trials, then after we run the first we are back to the situation for  $B_{n-1,p}$  and can use knowledge of  $E_{n-1}$ . Specifically, with probability  $q$ , the first trial is a failure. Hence the only successes will be those in the remaining trials, so we can expect  $E_{n-1}$  successes. On the other hand, if the first trial is a success, then all told we expect  $1 + E_{n-1}$  successes. In short,

$$E_n = q E_{n-1} + p(1+E_{n-1}) = E_{n-1} + p. \quad (8)$$

Clearly this holds for  $n > 1$ . Furthermore, it's easy to compute  $E_1$  directly from the definition:

$$E_1 = 0q + 1p = p.$$

The explicit solution to this recurrence and initial condition is immediate [24]:

$$E_n = np.$$

Note how the algebra in the recursive solution is much closer to our intuition — each trial increases the expected number of successes by  $p$  — than the direct algebraic attack. ■

## EXAMPLE 4

Find the mean of the negative binomial of order 1. That is, find the expected number of Bernoulli trials to get the first success, given that the chance of success each time is  $p$ .

**Solution** The negative binomial distribution of order 1 has  $f(k) = q^{k-1}p$ . So by Definition 1, the answer is

$$\sum_{k \geq 1} kq^{k-1}p = p \sum_{k \geq 1} kq^{k-1}. \quad (9)$$

Let's try to guess the simplification first. Since each trial achieves on average the fraction  $p$  of successes, it should take  $1/p$  trials to get 1 full success. Of course, successes don't come in fractional parts, so this is just a plausibility argument, but here it will stand us in good stead.

Equation (9) is tricky for us to attack directly, because it involves an infinite sum. Nevertheless, let's plunge right in.

*Method 1: Direct attack.* We compute the sum from  $k = 1$  to an arbitrary  $n$ , look for a formula, and then decide intuitively whether the formula approaches a limit as  $n \rightarrow \infty$ . The summand is of the form  $Akq^{k-1}$ , so we can find a formula by the methods of Section 5.7. One finds after some simplification [25, 26] that

$$p \sum_{k=1}^n kq^{k-1} = \frac{1}{p} - \left(n + \frac{1}{p}\right)q^n. \quad (10)$$

So long as  $q < 1$  (i.e.,  $p > 0$ ), the term  $q^n/p$  goes to 0 since  $q^n$  goes to 0. The tricky term is  $nq^n$ , as the growth in  $n$  might conceivably compensate for the decline in  $q^n$ ; however, exponentials (such as  $q^n$ ) always overpower polynomials (like  $n$ );

see the subsection ‘‘Exponentials and Logarithms’’ in Section 0.2. In conclusion,  $E(X) = 1/p$ , as anticipated.

*Method 2: Recursion.* Let  $X$  be the random variable for the number of the trial with the first success. We separate the initial trial from the rest. With probability  $p$  we succeed in the first trial and quit, so  $X = 1$ . With probability  $q$  we fail and must go on to further trials. That is, it is just as if we are starting from scratch, so we can expect  $E(X)$  more trials. But we have already done one trial. Thus

$$E(X) = p \cdot 1 + q(E(X)+1).$$

Solving for  $E(X)$ , we find again that  $E(X) = 1/p$ . Note that this time there is no subscript on  $E$ ; there is no parameter  $n$  because the sequence of trials has no predetermined stopping point. When a random variable (here the number of trials) has an infinite range, the direct calculation of  $E$  from the definition gets harder. However recursion gets easier! (We will give more examples of this type in Section 6.9.)

*Method 3: Generating functions.* Look at Eq (14), Section 4.8. The LHS is very close to the LHS of (9) above. Indeed, substitute  $k+1$  for  $k$  in (9), and  $q$  for  $s$  in Eq. (14), Section 4.8, and they become

$$\begin{aligned} E(X) &= p \sum_{k=0}^{\infty} (k+1)q^k, \\ \frac{1}{(1-q)^2} &= \sum_{k=0}^{\infty} (k+1)q^k. \end{aligned}$$

Multiply the second equation by  $p$  and the right-hand sides are the same! Thus

$$E(X) = \frac{p}{(1-q)^2} = \frac{p}{p^2} = \frac{1}{p}$$

as before.

This is perhaps the first time that the  $s$  in a generating function has been more than a placeholder; we set it equal to the probability  $q$ . But in fact, this usage is quite common. Consider such a substitution whenever you want to evaluate an infinite sum involving a sequence for which you have the generating function. As we have just seen, we *do* want to evaluate an infinite sum whenever we want the expected value of a random variable defined on  $0, 1, \dots$ . For instance, the same generating function technique is available to find the expected value of the negative binomial of order  $n$  [29]. ■

**Solution to Example 3, Section 6.1 (picking the next item for a permutation)** Recall the issue: Given  $n$  objects, and having already picked  $k$  for a permutation, how long will it take to find a new  $(k+1)$ st object if we repeatedly pick from all  $n$  at random? If there are  $n-k$  objects left, the chance of success on any one random pick is  $p = (n-k)/n$ . Hence by Example 4, the expected number of additional trials to get the  $(k+1)$ st object is  $1/p = n/(n-k)$ . ■

## Expectations of Composite Variables

At the end of Section 6.4, we showed that random variables can often be expressed in terms of two or more simpler random variables. We now develop some theorems which show how to compute the expectation of such composite variables in terms of the expectations of the simpler variables. This process often provides a particularly easy way to compute expectations of complicated variables.

---

**Theorem 1.** Let  $X$  and  $Y$  be random variables on sample space  $S$ . ( $X$  and  $Y$  need *not* be independent.) Then

$$E(X+Y) = E(X) + E(Y). \quad (11)$$

Similarly,

$$E\left(\sum X_i\right) = \sum E(X_i). \quad (12)$$

---

Before proving this theorem, we give several examples of its use.

### EXAMPLE 5

#### Example 3 Continued: Expected Number of Successes in $n$ Bernoulli Trials

Let  $X$  be the total number of successes. Let  $X_i$  be a binary random variable defined by

$$X_i = \begin{cases} 1 & \text{if trial } i \text{ is a success,} \\ 0 & \text{if trial } i \text{ is a failure.} \end{cases}$$

Note that

$$X = \sum_{i=1}^n X_i.$$

The whole point of introducing the  $X_i$  is that they are simple to analyze yet they can be summed to get  $X$ . For each  $i$ ,  $E(X_i) = p$ . (We computed this before as  $E_1$  after Eq. (8).) Thus, by Eq. (12),  $E(X) = np$ . ■

### EXAMPLE 6

#### Example 4 Continued: Expected Value of the Negative Binomial of Order 1

Let  $X$  be the number of trials through the first success. Define  $X_i$  by

$$X_i = \begin{cases} 1 & \text{if the } i\text{th trial takes place} \\ 0 & \text{if the } i\text{th trial does not take place.} \end{cases}$$

Then  $X = \sum_{i=1}^{\infty} X_i$ . (Why?) Furthermore,  $\Pr(X_i=1) = q^{i-1}$ . (Why?) Thus

$$E(X_i) = 1 \Pr(X_i=1) + 0 \Pr(X_i=0) = q^{i-1},$$

and

$$E(X) = \sum_{i \geq 1} q^{i-1} = \sum_{j=0}^{\infty} q^j = \frac{1}{1-q} = \frac{1}{p},$$

as anticipated. ■

*Note.* The  $X_i$  variables are definitely not independent [35].

**PROOF OF THEOREM 1** Let  $z = x + y$  run over Codomain( $X+Y$ ) and let  $x$  and  $y$  run over all pairs with  $x \in \text{Codomain}(X)$  and  $y \in \text{Codomain}(Y)$ . By definition,

$$\begin{aligned} E(X+Y) &= \sum_z z \Pr(X+Y=z) \\ &= \sum_{x,y} (x+y) \Pr(X=x, Y=y) \\ &= \sum_{x,y} x \Pr(X=x, Y=y) + \sum_{x,y} y \Pr(X=x, Y=y). \end{aligned} \quad (13)$$

We now analyze the first sum in the last line of (13). We rewrite it using a double sum:

$$\sum_x \sum_y x \Pr(X=x, Y=y) = \sum_x \left( x \sum_y \Pr(X=x, Y=y) \right). \quad (14)$$

For any particular  $x$ , call it  $x_0$ , the inner sum on the right divides the ways  $X$  can equal  $x_0$  into mutually exclusive and exhaustive cases, one for each possible value of  $Y$ . Thus

$$\sum_y \Pr(X=x_0, Y=y) = \Pr(X=x_0).$$

Therefore, the RHS of Eq. (14) becomes

$$\sum_x x \Pr(X=x) = E(X).$$

Similarly,  $\sum_{x,y} y \Pr(X=x, Y=y)$  in line (13) is  $E(Y)$ , and we have proved Eq. (11). The generalized form, Eq. (12), follows for finite sums by induction [34]. ■

By essentially the same method, one proves

**Theorem 2.** Let  $X$  and  $Y$  be random variables, and let  $a$  and  $b$  be constants. Then

$$E(aX + bY) = aE(X) + bE(Y).$$

In particular, for any constant  $c$ ,

$$E(cX) = cE(X) \quad (\text{set } a = c, b = 0),$$

a result we shall use in the next section. Also,

$$E(X - Y) = E(X) - E(Y) \quad (\text{set } a = 1, b = -1).$$

There is also a theorem about  $E(XY)$ . Sure enough, it equals  $E(X)E(Y)$ , but only if  $X$  and  $Y$  are *independent* [44].

## Problems: Section 6.5

---

1.  $\langle 2 \rangle$  Let  $X_A$  be the characteristic function for event  $A$ :  $X_A(s) = 1$  for  $s \in A$  and  $X_A(s) = 0$  otherwise. Show that  $E(X_A) = \Pr(A)$ .
2.  $\langle 1 \rangle$  From the definition of mean value, find the mean of the discrete uniform distribution with values  $1, 2, \dots, n$ .
3.  $\langle 2 \rangle$  Find the mean of the discrete uniform distribution whose  $n$  values are evenly spaced from  $a$  to  $b$ .
4.  $\langle 2 \rangle$  We claimed in Example 2 (average number of dots when two dice are thrown) that there are easier ways to get the answer. Using Theorem 1, show an easier way.
5.  $\langle 2 \rangle$  You receive a lottery ticket in the mail. It says that the first prize is \$10,000 and that there are 1000 runner-up prizes of a \$25 bond. Naturally, only those who mail back their stubs can receive prizes. There are indications that 10 million people were sent tickets. If it costs you 37 cents to mail back your ticket to participate, what is your expected net gain?
  - a) assuming one person in 10 mails back the stub;
  - b) assuming one person in 100 mails it back?
6.  $\langle 2 \rangle$  At the end of the solution to Example 4, Section 6.1 (on p. 559), we assumed that the lottery company doesn't waste numbers and must have sent out at least as many tickets as the number on our ticket. Actually, this assumption is suspect. First, all numbers sent out probably have equally many digits — What would you think if your number was 37? Second, if every number was given out, it would be easy for people to make and submit fake lottery tickets. Probably most numbers are not valid lottery numbers (just like most credit

card numbers and driver's license numbers are not valid) so that if one fakes, one is likely to pick an invalid number.

You may think of still other considerations. Give your reasons and come up with what you think is a reasonable guess for the number of numbers if your number is  $N$ .

7.  $\langle 2 \rangle$  A common sort of slot machine has three wheels which spin, each showing a single entry when it comes to rest. Suppose that each wheel has the same 10 distinct entries. Suppose that it costs 25¢ to play, you win 50¢ if exactly two entries are the same, and you win \$10 if all three entries are the same. Assume that all possible combinations are equally likely. What is your expected net gain?
8.  $\langle 2 \rangle$  A horse has probability  $p$  of winning a race. If you bet \$1 on the horse and it wins, you get  $\$w$  back; if it loses, you get 0. At what value should racetrack officials set  $w$  so that your expected return is 0?
9.  $\langle 2 \rangle$  Show that, if racetrack officials set the winning amounts according to [8], then no matter how many people bet whatever amounts on whichever horses, the expected profit to the track (the amount the track keeps after paying off bets) is 0. Assume that the probabilities on the horses add to 1. Give simple examples (say with just two horses in the race) to show that the actual profit can be positive; that it can be negative.
10.  $\langle 3 \rangle$  Four horses,  $A$ ,  $B$ ,  $C$  and  $D$ , are in a race. The probabilities that each horse will win are .4, .3, .2 and .1, respectively. The race track has announced that if you bet  $\$x$  on a given horse and it

wins, you will receive

- \$  $2x$  if it is horse *A*,
- \$  $3x$  if it is horse *B*,
- \$  $4x$  if it is horse *C*,
- \$  $9.5x$  if it is horse *D*.

(If your horse loses, you get nothing back.)

- For each horse, compute the expected monetary outcome from betting \$1 on that horse.
- Suppose you spread your risk by betting  $25\%$  on each horse. Now what is your expected outcome?
- There is a way to split up your \$1, betting different amounts on each horse, so that you achieve *exactly* your expected outcome no matter which horse wins. What is this split? Show that you are right.

*Note:* This part assumes the race track accepts fractional bets. Even if the track only accepts bets in multiples of \$2, as is customary, you can approximate the needed fractions very closely if your total bet is large. Also, there is a method behind this example. No matter what probabilities the horses have of winning, and no matter how much the track will pay for winning tickets, you can always split your bet so that you know the outcome in advance. Unfortunately, tracks arrange things so that this outcome is negative!

11. (2) Racetracks don't like to run the risk of losing money on a race. They avoid this risk by adjusting the amount that you win from a \$1 bet as betting proceeds, based on how much has been bet on each horse. (What they announce is changes in odds.) Show how a racetrack can set the winning amounts when the betting closes, based on the total amount bet on each horse, so that the track makes a profit of *exactly* 0 no matter which horse wins. (Tracks use this system, but they modify it slightly so that they are guaranteed a profit of, say, 5% of all money bet.)
12. (2) A betting game is said to be **fair** if the expected monetary return to each player is 0. For instance, suppose Ann and Bill agree to flip a good penny, with Ann paying Bill \$1 if they get heads, and Bill paying Ann \$1 if they get tails. This game is fair.

  - Suppose instead they agree to throw a good die, with Ann paying Bill \$1 if 1 dot comes up,

and Bill paying Ann \$1 otherwise. Obviously, this is not fair. What are the expected payoffs to each player?

- Unfair games can be made fair by having initial payments. In this case, Ann should pay Bill a certain amount before each flip to make it fair. How much?
- Yvonne and Zack agree to play the following game. They will throw two fair dice once. If a sum of 2 or 11 comes up, Yvonne pays Zack \$2. If 7 comes up, Zack pays Yvonne \$1. Otherwise the game is a draw. Who should pay whom how much before each throw to make the game fair?
- (3) Solve [2] again, but this time do it recursively. (In this case, a recursive solution is much more work. Even finding the recurrence is work. However, it's a good review.)
- (2) A friend picks an integer from 1 to 3 at random, and you get three guesses to figure out which one she picked. We know you can do it, but think about the average number of guesses you will need. Find this average using the definition of expected value.
- (3) College Boards again. You guess at random on  $n$  questions. Recall that there are five answers per question and you get  $1/4$  off for each wrong answer.
  - For each individual question on which you guess, what is the expected difference in your score from what it would be if you had left that question blank? (Here the difference, if nonzero, is a signed number; if your score goes down, the difference is negative.)
  - What is the expected difference in your score for guessing on all  $n$  questions?
  - For each individual question, what is the expected absolute value of the difference in your score from guessing?
  - Consider the question: What is the expected absolute value of the difference in your score from guessing on all  $n$  problems? This is much harder to answer than part b). Why? Answer it in the specific case  $n = 5$ . (*Note:* Absolute difference is what the College Board is talking about when it makes statements like "If the same student takes an equivalent test on the

next day, his or her score will typically differ by as much as 30 points from the previous score.”)

- e) Answer parts a) and b) again on the assumption that you were able to eliminate one answer on each question before guessing randomly among the remaining answers.

- f) With the assumption in part e) and  $n = 5$ , determine the probability that guessing increases your score. (This is [26, Section 6.4] with  $n = 5$  instead of  $n = 10$ .)

- g) How do you reconcile the fact that the answer to part f) was less than .5 with the fact that the expected difference from part e) is positive? Does this same situation arise when  $n = 10$ ?

16. ⟨3⟩ (Example 3, Section 6.1, extended) What is the expected number of attempts until one finally gets a complete permutation of the  $n$  objects?

17. ⟨2⟩ You want to invest some money in the stock market for a year. A broker suggests the stock of XYZ Company. He believes there is a 40% chance the price will double in a year, a 40% chance it will halve, and a 20% chance it will stay the same. Assuming he’s right, what is your expected gain if you buy \$5000 of XYZ stock now and sell it in a year?

18. ⟨2⟩ The answer to [17] is positive, but that doesn’t mean you should buy the stock, even assuming the broker is right and even if there are no other options. It all depends on what the money is worth to you. Suppose you are just making ends meet. While it would be nice to gain \$5000, it would be a *disaster* to lose \$2500. Economists like to measure the worth of money (and everything else) to individuals in hypothetical units called “utils”. Suppose gaining \$5000 is worth 1000 utils to you, but losing \$2500 is worth -3000 utils. Assume that just breaking even on the stock purchase is worth 0 utils. Your gain in utils is a random variable. It has an expected value. Should you buy the stock?

19. ⟨2⟩ You have \$10,000 to invest. You can either put it in a bank certificate of deposit, at 5% interest per year, or you can buy a bond which returns 6% a year. Whereas the face value of the certificate never varies from \$10,000, the selling price of the bond can go up or down, depending on prevailing interest rates. (This has no effect on the interest you receive, which is fixed.) Your broker estimates

that, one year from now, there is a 30% chance the bond will be worth 10% more, a 30% chance that its price will be unchanged, and a 40% chance its price will be 10% less. You intend to cash in your investment in one year. Which investment has the greater expected return?

20. ⟨2⟩ It may be that expected monetary value is not the right measure in [19]. Just as in [18], you should probably consider the utility of money. Assume that simply preserving your \$10,000 is worth 0 additional utils to you; that each dollar lost at the end of the year is worth -1 util; and that each additional dollar gained at the end of the year is worth +1 util, up to \$1000, but that thereafter each additional dollar is worth only +1/2 util. Which investment strategy, bank certificate or bond, has the greater expected utility?

21. ⟨2⟩ Let  $X$  be any random variable. What is

$$E(X - E(X))?$$

22. ⟨2⟩ Again let  $B_{n,p}(k)$  be the point distribution for the number of successes in  $n$  Bernoulli trials with probability of success  $p$  on each trial. Prove that  $E(B_{n,p}) = np$  by induction. Hint: Start with a recurrence for combinations of  $n$  things in terms of combinations of  $n - 1$  things.

23. ⟨3⟩ (Assumes calculus) Prove that  $E(B_{n,p}) = np$  by starting with the Binomial Theorem, regarding  $x$  as the variable and  $y$  as a constant, and differentiating.

24. ⟨2⟩ Use the method of Section 5.7 to solve the difference equation (8).

25. ⟨3⟩ Verify Eq. (10) by the methods of Sections 5.7 and 5.9.

26. ⟨3⟩ Here is a special, shorter method for verifying Eq. (10).

- a) Explain why

$$\sum_{k=1}^n kf(k) = \sum_{j=1}^n \sum_{k=j}^n f(k). \quad (15)$$

- b) Now substitute  $q^{k-1}$  for  $f(k)$  and on the right of (15) and use the formula for the sum of a geometric series twice.

27. ⟨2⟩ (Assumes calculus) Derive Eq. (10) by differentiating something.

28.  $\langle 3 \rangle$  Let  $N_{n,p}$  be the negative binomial distribution of order  $n$ . Thus  $N_{n,p}(k)$  is the probability of obtaining the  $n$ th success on the  $k$ th trial, if each trial has success probability  $p$ . Let  $X_i$  be the number of additional trials after the  $(i-1)$ st success up to and including the  $i$ th success. Use the  $X_i$ 's to derive  $E(N_{n,p})$ .

29.  $\langle 3 \rangle$  Derive  $E(N_{n,p})$  by generating functions. Hint: Rewrite  $k C(k-1, n-1)$  so that there is no  $k$  in front; then rewrite it further so that Eq. (13), Section 4.8 applies.

30.  $\langle 2 \rangle$  Find  $E(N_{n,p})$  by recursion on  $n$ .

31.  $\langle 2 \rangle$  Let  $X_\lambda$  be the Poisson random variable. Find  $E(X_\lambda)$  directly from the definition of expected value. You need just one fact from calculus:

$$\sum_{k \geq 0} \frac{x^k}{k!} = e^x.$$

If you regroup and make the right change of index, the answer falls out.

32.  $\langle 2 \rangle$  (Assumes calculus) Find  $E(X_\lambda)$  using differentiation.

33.  $\langle 3 \rangle$  Balls labeled 1 through  $n$  are put in an urn. A ball is picked at random, its number is recorded, and it is put back in the urn. Let  $X_k$  be the sum of the numbers recorded if this is done  $k$  times. Now suppose a ball is picked at random, its number is recorded, and it is thrown away. Let  $Y_k$  be the sum of the numbers recorded if this is done  $k$  times. Compare  $E(X_k)$  and  $E(Y_k)$ .

34.  $\langle 2 \rangle$  Verify that Eq. (12) follows by induction from Eq. (11) if the number of variables is finite. (In Example 6 there are infinitely many variables. Eq. (12) is usually true for infinitely many variables; it is always true if the variables only take on nonnegative values, as in Example 4. However, this infinite case cannot be proved by induction. Its proof requires the careful use of limits.)

35.  $\langle 2 \rangle$  Verify that  $X_i$  and  $X_j$  of Example 6 are not independent.

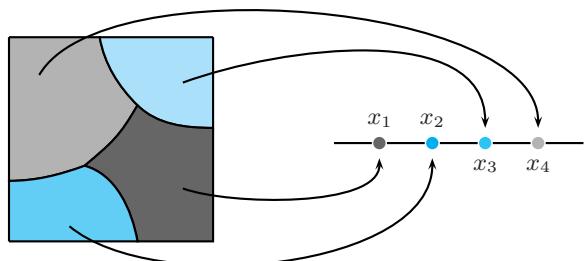
36.  $\langle 2 \rangle$  Prove Theorem 2. Hint: Mimic the proof of Theorem 1.

37.  $\langle 2 \rangle$  We defined  $E(X)$  in terms of probabilities of codomain values  $x$  of  $X$ , but these in turn are defined in terms of probabilities of events in  $S$ . Thus it is possible to express  $E(X)$  directly in terms of

such sample space probabilities. Fig. 6.13 illustrates what is going on: The probability  $\Pr(X=x)$  which “weights”  $x$  is merely the sum of the probabilities of the elements  $s$  which  $X$  maps to  $x$ . It follows that

$$E(X) = \sum_{e \in S} X(e) \Pr(e). \quad (16)$$

Now let  $S$  be the sample space whose atoms are all triple tosses of a fair coin. Let  $X$  be the total number of heads obtained. Compute  $E(X)$  using Definition 1. Compute it again using Eq. (16).



**FIGURE 6.13**

38.  $\langle 2 \rangle$  Let  $g$  be any real-valued function. Let  $X$  be any random variable on some sample space  $S$ . Then  $g(X)$  is another random variable — for any  $s \in S$ ,  $g(X(s))$  is a real number, which is all that is required of a random variable. Thus by definition,

$$E(g(X)) = \sum_{z \in \text{Codomain}(g)} z \Pr(g(X)=z). \quad (17)$$

Explain why this is equivalent to the following, often more convenient formula:

$$E(g(X)) = \sum_{x \in \text{Codomain}(X)} g(x) \Pr(X=x). \quad (18)$$

39.  $\langle 2 \rangle$  Let  $X$  be the number of dots when you toss one die. Use Eq. (17) to compute  $E((X-4)^2)$ . Check your work by using Eq. (18).

40.  $\langle 2 \rangle$  Let  $X$  be a random variable and let  $a$  and  $b$  be constants. Show that  $E(aX+b) = aE(X)+b$ . Use this to answer [3] again.

41.  $\langle 3 \rangle$  Let  $X$  be a random variable on  $S$ , and let  $A \subset S$ . Then  $E(X|A)$ , the expectation of  $X$  given  $A$ , is the expectation of  $X$  on the induced probability space  $A$ . (That is,  $A$  is the sample

space and  $\Pr_A(C) = \Pr(C)/\Pr(A)$ . See [43, Section 6.3].) Find a formula for  $E(X|A)$  in terms of probabilities on the original space  $S$ .

42. ⟨2⟩ Prove that

$$E(X) = \Pr(A)E(X|A) + \Pr(\sim A)E(X|\sim A).$$

We have already used this fact many times in the text; where?

43. ⟨2⟩ You toss two dice. Let  $X$  be the number of dots showing on the first and  $Y$  the number on

the second. Using the definition of expectation, compute

- a)  $E(X)$ ,
- b)  $E(X^2)$ ,
- c)  $E(XY)$ .

Confirm that your results are consistent with the remark after Theorem 2 about expectations of products.

44. ⟨3⟩ Prove: If  $X$  and  $Y$  are independent random variables, then  $E(XY) = E(X)E(Y)$ .

## 6.6 Variance

---

Expected value is a measure of the *center*. But no one number can tell all we need to know about a random variable. In this section we consider the second most important statistic, **variance**, which gives us a measure of *dispersion* from the center.

Why do we need a measure of dispersion? Well, did you ever hear about the statistician who drowned trying to wade across a stream whose average depth was known to be 2 feet? If you don't get the joke, consider the following two random variables:  $X$  such that

$$X = 1, 2, 3, \quad \text{each with probability } 1/3, \tag{1}$$

and  $Y$  such that

$$Y = 0, 6 \quad \text{with} \quad \Pr(0) = 2/3, \quad \Pr(6) = 1/3. \tag{2}$$

You can easily see that both  $X$  and  $Y$  have expected value 2. Look at Fig. 6.14. However, the values of  $X$  are much closer to 2 than the values of  $Y$  are. (And 6 feet is deeper than the statistician could wade through!)

One way to get a measure of dispersion would be to consider the average of distances from the mean of a random variable  $X$ , that is

$$E(|X - E(X)|).$$

But absolute values are hard to deal with computationally, so instead we use

$$E([X - E(X)]^2) \tag{3}$$

since the square, like the absolute value, makes everything nonnegative but is much easier to deal with ([21] in Section 6.5). The quantity in (3) is traditionally denoted by  $\text{Var}(X)$ . Formally:

**Definition 1.** Let  $X$  be a random variable on some probability space. The **variance** of  $X$ , denoted  $\text{Var}(X)$ , is defined by

$$\text{Var}(X) = E([X - E(X)]^2) = \sum_x [x - \sum_x x \Pr(X=x)]^2 \Pr(X=x).$$

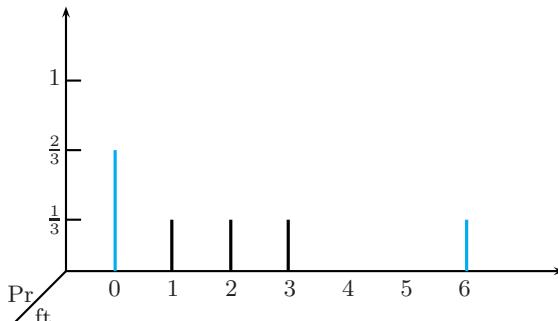
Similarly the variance of a discrete probability distribution  $f$  is given by

$$\text{Var}(f) = \sum_x [x - E(f)]^2 f(x) = \sum_x [x - \sum_x x f(x)]^2 f(x).$$

The **standard deviation** of  $X$ , denoted  $\sigma_X$ , is defined by

$$\sigma_X = \sqrt{\text{Var}(X)}.$$

Consequently,  $\text{Var}(X)$  is also denoted by  $\sigma_X^2$ . Similarly, the standard deviation of  $f$ , denoted by  $\sigma_f$ , is  $\sqrt{\text{Var}(f)}$ .



**FIGURE 6.14**

Two distributions (one black, one in color) with the same mean ( $=2$ ) but different variances.

### EXAMPLE 1

Compute the variance and standard deviation for  $X$  and  $Y$  as defined in displays (1) and (2).

**Solution** We have

$$\begin{aligned} \text{Var}(X) &= \sum_k \Pr(X=k)[k-E(X)]^2 \\ &= \frac{1}{3}(1-2)^2 + \frac{1}{3}(2-2)^2 + \frac{1}{3}(3-2)^2 = \frac{2}{3}. \end{aligned}$$

Similarly,

$$\begin{aligned} \text{Var}(Y) &= \sum_k \Pr(Y=k)[k-E(Y)]^2 \\ &= \frac{2}{3}(0-2)^2 + \frac{1}{3}(6-2)^2 = \frac{24}{3} = 8, \end{aligned}$$

which is much bigger. Taking positive square roots, we obtain the standard deviations:

$$\sigma_X = \sqrt{2/3} \approx .816, \quad \text{and} \quad \sigma_Y = \sqrt{8} \approx 2.828.$$

Again the value for  $Y$  is much bigger. ■

Based on Definition 1 and Example 1, the following fact should be intuitively reasonable: If the variance of a random variable is very small, then with near certainty a random value of that variable will be very close to the mean. (See also Theorem 3 below.) It further turns out that, for any random variable  $X$ , if values are sampled repeatedly and independently and a new variable  $Y$  is defined as the average of the sample values of  $X$ , then the variance of  $Y$  will be very small. (More formally, if we start with a random variable  $X$ , then even if the variance of  $X$  is large, the variance of the new variable  $Y = (1/n) \sum_{i=1}^n X_i$  will be small, if  $n$  is large enough.) Thus the *average* of a *sample* is a good estimate of the *mean* of the random variable. This fact is of fundamental importance in **statistics**, the science of determining underlying structure from sample data.

## EXAMPLE 2

Compute the variance for the number of dots which appear when one fair die is tossed.

**Solution** Let  $X$  be the random variable on the set  $\{1, 2, 3, 4, 5, 6\}$ . Then, since each value is equiprobable,  $E(X) = \frac{1}{6}(21) = \frac{7}{2}$ . Then the variance is

$$\sum_{i=1}^6 \frac{1}{6} \left( i - \frac{7}{2} \right)^2 = \frac{1}{3} \left[ \left( \frac{5}{2} \right)^2 + \left( \frac{3}{2} \right)^2 + \left( \frac{1}{2} \right)^2 \right] = \frac{35}{12},$$

where we have used the fact that the terms in the sum for  $i$  and  $6 - i$  are the same. ■

## EXAMPLE 3

Determine  $\text{Var}(B_{n,p})$ , the variance of the binomial distribution.

**Solution** We outline two approaches.

*Method 1: Direct algebraic attack.* We know from Example 3 in Section 6.5 that the expectation,  $E(B_{n,p})$ , of the binomial distribution is  $np$ . Therefore, letting  $k$  run over the domain of  $B_{n,p}$ , namely  $\{0, 1, 2, \dots, n\}$ , and using the definition of the variance of a distribution,

$$\text{Var}(B_{n,p}) = \sum_{k=0}^n (k-np)^2 B_{n,p}(k) = \sum_{k=0}^n \binom{n}{k} p^k q^{n-k} (k-np)^2. \quad (4)$$

This sum is a bit trickier than the one in Example 3 in Section 6.5 because of the  $k^2$  which results when  $(k - np)^2$  is expanded. Nevertheless, essentially the same approach as in that example works using the binomial coefficient identity given there. We'll leave the gory details to a problem [14]. The result is that  $\text{Var}(B_{n,p}) = npq$ .

*Method 2: Decompose the variable.* We noted in Example 5, Section 6.5, that if  $X$  has distribution  $B_{n,p}$ , then  $X = \sum_{i=1}^n X_i$  where each  $X_i$  is a 0–1 random variable with  $\Pr(X_i=1) = p$ . If variances add just as expected values add,  $\text{Var}(B_{n,p})$  is just

$n$  times  $\text{Var}(X_1)$ , the latter being easy to compute. Unfortunately, variances do *not* generally add, but they do for independent variables, which these  $X_i$  are. These facts, plus the conclusion of this example, are worked out in [1, 5, 6, 15]. ■

Now let us try to justify our earlier intuition that for a random variable  $X$ , if  $\text{Var}(X)$  is small, then values of  $X$  not close to  $E(X)$  are highly unlikely. More generally, whatever  $\text{Var}(X)$  is, we shall show that it is unlikely that  $X - E(X)$  will be large *relative* to the size of  $\text{Var}(X)$ . This is the result of Theorem 3 below but first we start with a simple scaling result.

Recall that we showed in Section 6.5 that, if  $c$  is a constant, then  $E(cX) = cE(X)$ . Here is the corresponding result for variance.

---

**Theorem 1.** For any random variable  $X$  and constant  $c$ ,

$$\text{Var}(cX) = c^2\text{Var}(X).$$


---

**PROOF** We just use the definition of variance in Definition 1 and the expectation of  $cX$  given just above to calculate

$$\begin{aligned}\text{Var}(cX) &= E([cX - E(cX)]^2) = E([c(X - E(X))]^2) \\ &= c^2 E([X - E(X)]^2) = c^2 \text{Var}(X),\end{aligned}$$

as desired. ■

A corollary of this that we shall use in the next section is

---

**Corollary 2.** Let  $Y$  be the fraction of successes in  $n$  Bernoulli trials with probability of success  $p$  on each trial. Then

$$E(Y) = p \quad \text{and} \quad \text{Var}(Y) = pq/n.$$

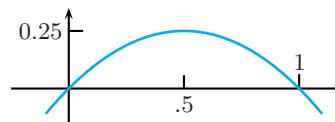

---

**PROOF** Let  $X$  be the random variable that denotes the number of successes in  $n$  trials. Then  $X$  is binomially distributed and  $Y = X/n$ . Since we know that  $E(X) = np$ , it follows that  $E(Y) = p$  and, since we know that  $\text{Var}(X) = npq$ , it follows from Theorem 1 that  $\text{Var}(Y) = (1/n^2)\text{Var}(X) = pq/n$ . ■

In Fig. 6.15 we have plotted  $pq = p(1-p)$ . Note that  $pq < 1/4$  for all  $p$  in  $[0, 1]$  so that from Corollary 2 we know that  $\text{Var}(Y)$  is very small when  $n$  is large and so, therefore, the standard deviation  $\sigma_Y$  is also very small. As we are about to show (Theorem 3), every random variable  $X$  must almost certainly be fairly close to its mean if distance is measured in  $\sigma_X$  units (that is, in multiples of  $\sigma_X$ ). Since  $Y$  in the Corollary has a very small  $\sigma$  if  $n$  is large, it will follow (Theorem 4) that  $Y$  is almost certainly *very close* to its mean if  $n$  is large.

**FIGURE 6.15**

Graph of  $pq = p(1-p)$ .



**Theorem 3, Chebyshev's Inequality.** Let  $X$  be a random variable with standard deviation  $\sigma_X$  and let  $\alpha$  be any positive constant. Let  $A$  be the event that  $|X - E(X)| > \alpha\sigma_X$ . Then  $\Pr(A) < 1/\alpha^2$ . Equivalently,

$$\Pr(|X - E(X)| \leq \alpha\sigma_X) > 1 - 1/\alpha^2. \quad (5)$$

The event  $A$  just amalgamates all events in the sample space of  $X$  that have values more than  $\alpha$  standard deviations from the mean. The conclusion says that the sum of the probabilities of all these events must be small if  $\alpha$  is large. For example, Chebyshev's Inequality says that  $X$  is more than 2 standard deviations away from the mean (i.e.,  $\alpha > 2$ ) with probability less than  $1/4$ .

**PROOF** Let  $Y = [X - E(X)]^2$  so that  $E(Y) = \text{Var}(X) = \sigma_X^2$ . Now for any random variable  $Y$  the definition of its expectation is (with  $e$  denoting any atomic event in  $S$ , the sample space for  $Y$ )

$$E(Y) = \sum_{e \in S} \Pr(e)Y(e).$$

For the  $Y$  just defined and  $A$  as defined by the theorem, we may rewrite this as

$$E(Y) = \sigma_X^2 = \sum_{e \in A} \Pr(e)Y(e) + \sum_{e \notin A} \Pr(e)Y(e). \quad (6)$$

By the definition of the event  $A$ ,  $Y(e) > (\alpha\sigma_X)^2 = \alpha^2\sigma_X^2$  for all  $e \in A$ . Thus so long as  $\Pr(A) > 0$  this strict inequality is preserved when we weight by the probability of elements  $e \in A$ :

$$\sum_{e \in A} \Pr(e)Y(e) > \alpha^2\sigma_X^2 \sum_{e \in A} \Pr(e) = \alpha^2\sigma_X^2 \Pr(A). \quad (7)$$

(What if  $\Pr(A) = 0$ ? Then the conclusion of the theorem, that  $\Pr(A) < 1/\alpha^2$ , is certainly true anyway.) Next, since for any  $e$ ,  $Y(e) \geq 0$ ,

$$\sum_{e \notin A} \Pr(e)Y(e) \geq \sum_{e \notin A} \Pr(e) \cdot 0 = 0. \quad (8)$$

Combining (6–8), we obtain

$$\sigma_X^2 > \alpha^2\sigma_X^2 \Pr(A) + 0 = \alpha^2\sigma_X^2 \Pr(A).$$

Dividing by  $\alpha^2\sigma_X^2$  (which is nonzero if  $\Pr(A) > 0$  – why?), we obtain  $\Pr(A) < 1/\alpha^2$ .

As for the final claim (5), the event  $|X - E(X)| \leq \alpha\sigma_X$  is  $\sim A$ . Since  $\Pr(A) + \Pr(\sim A) = 1$ , therefore  $\Pr(A) < 1/\alpha^2$  iff  $\Pr(\sim A) > 1 - 1/\alpha^2$ . ■

**EXAMPLE 4**

A fair coin is flipped 100 times. Bound the probability that the actual fraction of heads is (a) more than .1 away from the mean .5; (b) more than .2 away from .5; (c) within .2 of the mean.

**Solution** From Corollary 2 we know that the standard deviation of the fraction of heads is  $\sqrt{(.5)(.5)/100} = .05$ . Thus .1 is 2 standard deviations from the mean and from Theorem 3 the probability of being more than 2 standard deviations away is less than 1/4. Similarly, the probability of being more than .2 (= 4 standard deviations) from the mean is less than 1/16. Consequently, the probability of being at most .2 from the mean is more than 15/16. ■

Actually, the probabilities of being 2 or 4 standard deviations away from the mean for Bernoulli trials is *much* less than 1/4 or 1/16, respectively. Chebyshev's Inequality holds for *any* distribution, no matter how bizarre. But the Binomial Distribution is particularly well behaved. In fact the actual probabilities are about 1/20 and 1/10000. We say more about this in the next section.

Chebyshev's Inequality is all we need to prove the following famous theorem.

**Theorem 4, The Law of Large Numbers .** Consider a sequence of  $n$  Bernoulli trials with an (unknown) individual success probability  $p$ . Let  $\bar{p}$  be the *observed* fraction of successful trials. Then for any positive numbers  $\epsilon$  and  $\delta$ , each of which can be arbitrarily small, if  $n$  is large enough

$$\Pr(|p - \bar{p}| \leq \epsilon) > 1 - \delta. \quad (9)$$

Thus, the theorem says that the observed value  $\bar{p}$  of the average number of successes almost certainly gets *arbitrarily* close to the true probability  $p$  of success if you do enough trials.

**PROOF** Let  $Y$  be the random variable that counts the fraction of successes in  $n$  trials, namely  $\bar{p}$ . Then  $|\bar{p} - p|$  is a particular value of  $|Y - E(Y)|$ . By Chebyshev's Inequality (5), for any  $\alpha > 0$

$$\Pr(|\bar{p} - p| \leq \alpha \sigma_Y) > 1 - 1/\alpha^2. \quad (10)$$

To prove (9) from (10), we need to justify replacing  $1/\alpha^2$  by  $\delta$  and  $\alpha \sigma_Y$  by  $\epsilon$ . As for  $\delta$ , in Bernoulli trials we have seen that  $\sigma_Y = \sqrt{pq/n} \leq 1/(2\sqrt{n})$  because  $pq \leq 1/4$ . So, given any  $\delta$ , we can choose  $\alpha$  such that  $1/\alpha^2 \leq \delta$ ; this makes  $1 - 1/\alpha^2 \geq 1 - \delta$  and thus from (10) we get

$$\Pr(|\bar{p} - p| \leq \alpha \sigma_Y) > 1 - \delta. \quad (11)$$

As for  $\epsilon$ , choose  $n$  such that

$$\alpha \sigma_Y \leq \alpha / (2\sqrt{n}) \leq \epsilon \quad (\text{i.e., } n \geq \alpha^2 / 4\epsilon^2).$$

Thus

$$\Pr(|\bar{p}-p| \leq \epsilon) \geq \Pr(|\bar{p}-p| \leq \alpha\sigma_Y). \quad (12)$$

Combining inequalities (12) and (11) gives (9). ■

One word of caution. Given  $\epsilon$  and  $\delta$ , the Law of Large Numbers tells you only that a large enough  $n$  will give you a  $\bar{p}$  within  $\epsilon$  of  $p$  with probability arbitrarily close to 1 (i.e., within  $\delta$  of 1). It does not guarantee that any value of  $n$  will actually get you an observed value of  $\bar{p}$  within  $\epsilon$  of the true value of  $p$ .

Our intuition suggests, in fact, a stronger result, namely that if we continue Bernoulli trials indefinitely and compute  $\bar{p}$  after each trial, than after some point  $\bar{p}$  stays near the true value  $p$  and, indeed, goes to  $p$  in the limit as  $n$  goes to infinity. This intuition is true; it is called the *Strong Law of Large Numbers*. The proof, however, is beyond us here.

## Problems: Section 6.6

---

1.  $\langle 1 \rangle$  Let  $X$  be a binary random variable with

$$\Pr(X=1) = p.$$

Find  $\text{Var}(X)$  and  $\sigma_X$  directly from the definitions.

2.  $\langle 2 \rangle$  Let  $Z$  be the number of heads when a fair coin is tossed twice. Find  $\text{Var}(Z)$  and  $\sigma_Z$  directly from the definitions. Compare this result with that obtained using Corollary 2 to compute  $\text{Var}(Z)$  and  $\sigma_Z$ .

3.  $\langle 2 \rangle$  A biased coin has probability  $p \neq 1/2$  of coming up heads on each toss. From the definition of variance compute  $\text{Var}(X)$  and  $\sigma_X$  if  $X$  is the number of heads if this coin is tossed twice. Compare with Corollary 2.

4.  $\langle 1 \rangle$  Show that  $\text{Var}(X+k) = \text{Var}(X)$ . That is, if all values of a random variable are increased by  $k$ , the variance is unchanged.

5.  $\langle 3 \rangle$  We know that  $E(X+Y) = E(X) + E(Y)$  for any random variables  $X$  and  $Y$  on the same sample space (Theorem 1, Section 6.5). There is a similar theorem for variance,

$\text{Var}(X+Y) = \text{Var}(X) + \text{Var}(Y)$ , but it only applies when  $X$  and  $Y$  are independent. The following steps provide a proof.

- a) Begin by using the definition of variance and Theorem 1 from Section 6.5 to show that

$$\begin{aligned} \text{Var}(X+Y) &= \text{Var}(X) + \text{Var}(Y) + \\ &\quad 2E[(X-E(X))(Y-E(Y))]. \end{aligned} \quad (13)$$

- b) So it remains to show that the last term in

part a) is zero. One fact you will need is that terms of the form  $E[XE(Y)]$  may be written as  $E(X)E(Y)$ . Why is this correct?

- c) Use the result of part b) and the result of [44, Section 6.5] to show that the last term in (13) is, indeed, 0, thus completing the proof.
6.  $\langle 2 \rangle$  Problem [5] provides the basis case of the following.

---

**Theorem 5.** If  $X_1, \dots, X_n$  are mutually independent random variables, then

$$\text{Var}\left(\sum X_i\right) = \sum \text{Var}(X_i).$$


---

The inductive step is not hard given the following fact:

If  $X_1, \dots, X_n$  are mutually independent, then  $X_1$  and  $\sum_{i=2}^n X_i$  are independent.

We hope this fact is plausible, perhaps even obvious, but the proof is technically involved, so we don't ask you to tackle it. Prove Theorem 5 assuming this fact. (If you are curious how the fact is proved, you must start with the definition of mutual independence for random variables; see [44, Section 6.4].)

7.  $\langle 2 \rangle$  Let  $X_1, \dots, X_n$  be independent copies of a random variable  $X$  (such as you get by sampling

from  $X$   $n$  times). Suppose  $X$  has variance  $V$  and standard deviation  $\sigma$ . Use the result of [5] to calculate the variance and standard deviation of  $(1/n) \sum_{i=1}^n X_i$ ?

8. (2) A fair die is tossed 6 times.

a) Use Chebyshev's Inequality to bound the probability that no 6 appears on any throw.

b) What is the exact probability that no 6 appears on any throw?

9. (2) A fair die is tossed 600 times.

a) Bound the probability that the actual frequency of 4 dots is more than 10 away from 100.

b) Bound the probability that the frequency of an odd number of dots is more than 20 away from 300.

10. (2) If  $X$  and  $Y$  are independent, show that

$$\sigma_{X+Y} \leq \sigma_X + \sigma_Y.$$

One says therefore that standard deviation is **subadditive**.

11. (2) Let  $X$  be any random variable. Using the result in [5], we find  $\text{Var}(X+X) = \text{Var}(2X) = \text{Var}(X) + \text{Var}(X) = 2\text{Var}(X)$ . But by Theorem 1  $\text{Var}(2X) = 4\text{Var}(X)$  which implies that  $\text{Var}(X) = 0$ . Where did we go wrong?

12. (2) Let  $X$  be the sum of the dots when two fair dice are tossed. Compute  $\text{Var}(X)$

a) directly from the definition,  
b) using [5].

13. (2) What is the variance in the sum of the dots if  $n$  fair dice are tossed?

14. (3) Evaluate the sum in Eq. (4). First expand  $(k - np)^2$ . After you do that, the hard part is evaluating the  $k^2$  term. Do this using an identity like that in Eq. (7) in Section 6.5. But be careful. You can't just use  $k^2$  in place of  $k$  on the left-hand side.

15. (2) Use [1] and [6] to find the variance of the binomial distribution  $B_{n,p}$  relatively painlessly.

16. (2) Although it is usually wise to choose between options using the expected value, sometimes that's not a sure thing. Some years ago one of us commuted to New York City by train to work for a foundation. To get from the foundation's office to the train station we could either walk or take

the subway. A few weeks' experience showed that, from office desk to train, (1) walking took an average of 25 minutes with a standard deviation of 1 minute; and (2) the subway took an average of 20 minutes with a standard deviation of 10 minutes. The disutility of missing the train was extremely high and the disutility of waiting around at the station was also high. Which do you think was better, walking or subway? (You don't have enough information to do a thorough analysis, but give your intuitive reasons.)

17. (3) There is another version of Chebyshev's Inequality with the strict inequalities ( $<$  and  $>$ , as opposed to  $\leq$  and  $\geq$ ) appearing in different places. Let  $A$  be the event that  $|X - E(X)| \geq \alpha\sigma_X$ . Then  $\Pr(A) \leq 1/\alpha^2$ . Equivalently,

$$\Pr(|X - E(X)| < \alpha\sigma_X) \geq 1 - 1/\alpha^2.$$

a) Identify the differences between this statement and Theorem 3.

b) Prove this version.

In fact, for continuous distributions the probabilities of, say,  $|X - E(X)| \geq \alpha\sigma_X$  and  $|X - E(X)| > \alpha\sigma_X$  are the same, and so often one doesn't pay attention to the distinctions between these two versions of the theorem.

18. (2) Here is a fact that is often helpful in simplifying the computation of variances:

$$\text{Var}(X) = E(X^2) - [E(X)]^2.$$

That is, the variance is the expected value of the square minus the square of the expected value. Prove this identity. *Hint:* Start with  $E([X - E(X)]^2)$ , expand the square, and use the various properties of  $E$ .

19. (2) Use the result of [18] to compute

a) The variance in the number of heads when a fair coin is tossed twice.  
b) The variance in the number of heads when the biased coin of [3] is tossed twice.  
c) The variance of the number of dots when one fair die is tossed.  
d) The variance of the number of dots when two fair dice are tossed.

20. (3) The following equation is a generalization of the result in [18]:

$$E[(X-k)^2] = \text{Var}(X) + [E(X)-k]^2, \quad (14)$$

where  $k$  is any constant.

- a) In what sense is this a generalization of the previous result?
- b) Prove that the equation above is correct.
- c) The expression  $E[(X-k)^2]$  may be interpreted as measuring the dispersion of  $X$  around a base point  $k$ . Viewed in this way, what does the equation above tell you about dispersion as a function of  $k$ ?
21. (2) For any constant  $k$ , use (14) (or otherwise, if you wish) to prove
- $$E([(X+k) - E(X)]^2) = \text{Var}(X) + k^2.$$
22. (2) Let  $Y$  have a uniform distribution over the integers  $1, 2, \dots, n$ .
- Compute  $\text{Var}(Y)$  directly.
  - Compute  $\text{Var}(Y)$  using [18].

$$\text{Var}(X) = \left( \sum_{k=1}^{\infty} pk^2 q^{k-1} \right) - 1/p^2. \quad (15)$$

- b) Use the methods of Section 5.7 to evaluate the sum in Eq. (15) and thus find the variance of the negative binomial.
- c) Evaluate Eq. (15) with a CAS.

24. (3) (Assumes calculus) Starting with the fact that, for  $|x| < 1$ ,

$$\sum_{k=0}^{\infty} x^k = 1/(1-x),$$

use differentiation (and some algebra) to evaluate the sum in (15).

25. (3) Let  $X$  be a negative binomial variable ( $n = 1$ ) and let  $V$  be its variance. Determine  $V$  by breaking the outcomes into two cases as follows and applying [42, Section 6.5].

The two cases are: either we succeed on the first try or we don't.

If we don't, use recursion. It is as if we are starting all over with a new negative binomial experiment, except that we add 1 to the number of trials. That is, if we don't succeed on the first trial, the conditional variance is  $E([(X+1) - E(X)]^2)$ , which simplifies by [21]. Use these two observations to find an equation that has  $V$  on both sides, and solve for  $V$ .

23. (3)

- Use the result of [18] to show that if  $X$  has a negative binomial distribution, then

## 6.7 Statistical Estimation

This section provides an introduction to statistics. Whereas probability theory predicts the values of variables given knowledge of the underlying distribution, statistics does the reverse: given knowledge of sample values of a variable, statistics predicts the values of the underlying distribution. Thus, for example, given the results of a poll about which of two candidates is preferred by those polled, statistical estimation is concerned with the relation of the poll results to the preferences of the electorate as a whole.

Recall that in Example 7 in Section 6.1 we were concerned with the interpretation of the results of a poll where the voter preferences for candidates B and G are given and are claimed to be correct to, say, within  $\pm 3\%$ . We noted in Section 6.1 that it couldn't possibly be correct that the error in the result is *certainly* in error by no more than  $\pm 3\%$ . Using the language we have developed in this chapter, we have a sample space consisting of all potential voters with a random variable on that sample space having the value 1 if the voter favors candidate G and 0 if

the voter favors candidate B. If we sample  $n$  voters and  $\bar{p}$  is the fraction (i.e., the percentage) favoring G, then we would like to be able to estimate the underlying fraction  $p$  of all voters who favor G. Of course, we can't do this with certainty. But perhaps we can draw conclusions that have a high probability of being correct. So to speak then, we try to answer a question about a probability  $p$  with a probability about that probability. Statistics is the art of devising reliable tests of this sort and the science of justifying the reliability of the tests.

In polling voters we perform a sequence of  $n$  Bernoulli trials in which each trial consists of picking a voter at random and asking which candidate the voter favors. Then we compute the fraction of the sample who favor each candidate. (For the purposes of this section we assume there are only two candidates who we'll designate, as before, by G and B.) Let's fix some notation. Let  $X$  be a random variable which counts the number of voters favoring G in the polling sample and let  $Y = X/n = \bar{p}$  be the fraction of voters in our sample who favor G.<sup>†</sup> As before let  $p$  be the (unknown) fraction of all the voters favoring G. The Law of Large Numbers (Theorem 4 in Section 6.6) tells us that, if  $n$  is large enough,  $|p - \bar{p}|$  can be made as small as we wish with probability nearly 1. But how small do we wish  $|\bar{p} - p|$  to be and how large should  $n$  be to achieve this? Answering this question is the subject of this section.

$Y$  is a binomial variable scaled by  $1/n$ . In Corollary 2, Section 6.6, we showed that  $Y$  has mean  $p$  and variance  $pq/n$ . Now let's shift and scale again. Consider the random variable  $Z = (Y-p)/\sqrt{pq/n}$ .  $Z$  has mean 0 and variance 1 (why? [2]). In Fig. 6.16 we show a sequence of point distributions of  $Z$  for increasing values of  $n$  and various values of  $p$ . Also in this figure we show the **normal density function**, which was mentioned in Section 6.4. The former are approaching the latter.

The normal density function has the specific form

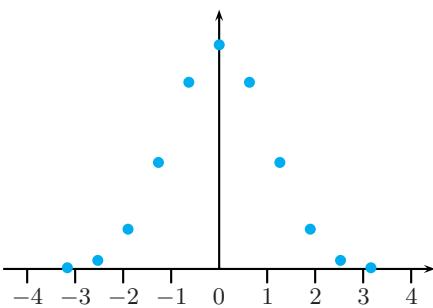
$$f(x) = \frac{1}{\sqrt{2\pi}} \exp^{-x^2/2}, \quad (1)$$

where the variable  $x$  is called a (standard) **normal variable**.

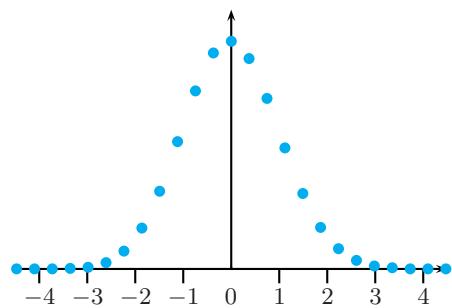
The mean of a density function, like the mean of a probability distribution function in Section 6.5, is found by averaging the product of the density function  $f(x)$  and  $x$ . So, just as the mean of a discrete distribution is the weighted sum  $\sum xf(x)$ , the mean of a continuous variable is the integral (area under the curve) of  $xf(x)$ . For the normal density function, you can see that  $f(x)$  is symmetric about the vertical axis (i.e.,  $f(x) = f(-x)$ ) so that each pair  $xf(x)$  and  $(-x)f(-x)$  cancel out in calculating the area. Thus the mean of the normal density function is 0. The variance is found by integrating  $x^2f(x)$  (i.e., finding the area under the curve  $x^2f(x)$ ). Doing this is quite a bit harder than for the mean but the result is that the normal density function has variance 1. Thus,  $f(x)$  and  $Z$  have the same mean and variance — more evidence for the convergence suggested by Fig. 6.16.

---

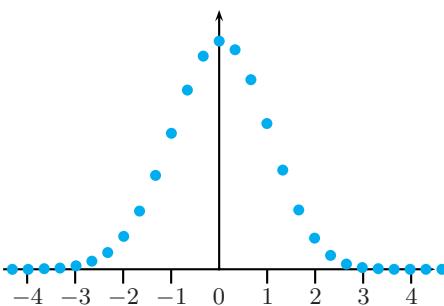
<sup>†</sup>Why call it both  $Y$  and  $\bar{p}$ ? Well,  $Y$  is used when we wish to emphasize that the fraction of voters is a variable that will have different values for different possible polls while  $\bar{p}$  is used when we are thinking of the actual result of a poll. But in fact they are the same thing. If we didn't think of  $\bar{p}$  as varying over possible polls, we couldn't do a probability analysis.



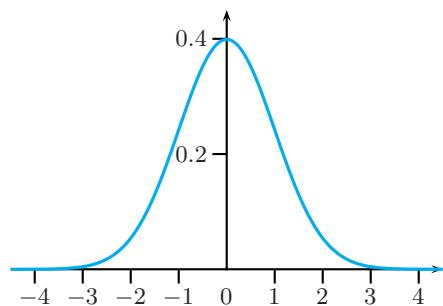
(a):  $n = 10, p = .5$



(b):  $n = 30, p = .4$



(c):  $n = 40, p = .65$



(d): The Normal Density

Graphs (a), (b) and (c) show point distributions of  $Z = (Y-p)/\sqrt{pq/n}$ , where  $Y$  has a binomial distribution with the values of  $n$  and  $p$  shown under each graph. Graph (d) is the normal density function. As  $n \rightarrow \infty$  the distributions of  $Z$  approach the normal density.

The probability that the normal variable  $x$  lies between any two real numbers  $c$  and  $d$  equals the area under the curve  $f(x)$  between  $c$  and  $d$ . This probability is just  $F(d) - F(c)$ , where  $F$  is the cumulative distribution function derived from the normal density function. Because probabilities for  $x$  are so easy to compute from the cumulative distribution, for the rest of this section it will be best to think of the normal distribution (and also the  $Z$  distribution) in terms of the cumulative distribution function instead of the density function (1). We will be able to look up cumulative distribution values, as discussed below.

The convergence of the  $Z$  distribution to the normal distribution is in fact very fast: for  $n$  of the size that occurs in surveys of voters, the difference (in probabilities over the same interval) is so small that it can be ignored. Thus, no matter what  $p$  and  $n$  are, we can use the normal distribution as a very good approximation to the distribution of  $Z$ . And this is what we shall do since calculating with the normal distribution is much easier than calculating with a binomial distribution when  $n$  is large. This is what we were referring to in Section 6.4 when we bruited the notion of using continuous mathematics to approximate discrete mathematics.

**FIGURE 6.16**

We won't attempt to prove the claim in the previous paragraph about the convergence of the scaled binomial  $Z$  to the normal distribution. But for the record we state the famous theorem about this convergence.

---

**Theorem 1, The Central Limit Theorem for Binomial Distributions.** Let  $X$  be a random variable with distribution  $B_{n,p}$ , let

$$Z = \frac{(X/n) - p}{\sqrt{pq/n}} = \frac{Y - p}{\sqrt{pq/n}}, \quad (2)$$

and let  $x$  be a normal variable distributed as in (1). Finally, let  $c$  and  $d$  be any real numbers. Then as  $n \rightarrow \infty$

$$\Pr(c \leq Z \leq d) \quad \text{approaches} \quad \Pr(c \leq x \leq d).$$


---

So far in this section, we have used  $Z$  for a modified binomial variable, and  $x$  for a normal variable. We have used different letters so that we could state the theorem that says how close they are. Given that they are so close, from now on we will use  $Z$  for both. The letter  $x$  will reappear later, in a different but also very traditional role, as a bound on values of  $Z$ .

For the normal distribution, we will want to know things like the probability that  $|Z| \leq 2$ , that is, that  $Z$  is within 2 standard deviations of the mean (since the mean is 0 and  $\sigma_Z = 1$ ). In fact, using (1), it can be shown that  $\Pr(|Z| \leq 2) = .9544$ . In other words, 95% of the time a normal variable is within 2 s.d.'s (standard deviations) of the mean. Tables of the normal distribution appear in every book on statistics, and now as public domain pages and interactive forms on the Web. Also, normal functions are built into most CASs and some calculators. For reference purposes, we provide Table 6.1. In line with our earlier remarks, this table gives the normal distribution in terms of its cumulative distribution, not its density function.

Now let's get down to some specifics that will end with a discussion of Example 7 in Section 6.1.

## EXAMPLE 1

A fair die is tossed 600 times. What is the probability that one dot comes up between 90 and 110 times, inclusive? (Compare this example with [9, Section 6.6].)

**Solution** Let random variable  $X$  count the number of occurrences of one dot. Then setting  $n = 600$  and  $p = 1/6$  in (2), we find that

$$Z = \frac{(X/600) - 1/6}{\sqrt{(1/6)(5/6)/600}} = \sqrt{\frac{6}{5}} \left( \frac{X - 100}{10} \right) \quad (3)$$

may be assumed to have a normal distribution. If  $90 \leq X \leq 110$ , then from Eq. (3),  $|Z| \leq \sqrt{6/5} \approx 1.095$ . From Table 6.1 we find (by interpolating between the values at 1.09 and 1.10) that

**TABLE 6.1**  
**Table of the Cumulative Function for the Normal Distribution**

| Z   | .00   | .01   | .02   | .03   | .04   | .05   | .06   | .07   | .08   | .09   |
|-----|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0.0 | .5000 | .5040 | .5080 | .5120 | .5160 | .5199 | .5239 | .5279 | .5319 | .5359 |
| 0.1 | .5398 | .5438 | .5478 | .5517 | .5557 | .5596 | .5636 | .5675 | .5714 | .5753 |
| 0.2 | .5793 | .5832 | .5871 | .5910 | .5948 | .5987 | .6026 | .6064 | .6103 | .6141 |
| 0.3 | .6179 | .6217 | .6255 | .6293 | .6331 | .6368 | .6406 | .6443 | .6480 | .6517 |
| 0.4 | .6554 | .6591 | .6628 | .6664 | .6700 | .6736 | .6772 | .6808 | .6844 | .6879 |
| 0.5 | .6915 | .6950 | .6985 | .7019 | .7054 | .7088 | .7123 | .7157 | .7190 | .7224 |
| 0.6 | .7257 | .7291 | .7324 | .7357 | .7389 | .7422 | .7454 | .7486 | .7517 | .7549 |
| 0.7 | .7580 | .7611 | .7642 | .7673 | .7704 | .7734 | .7764 | .7794 | .7823 | .7852 |
| 0.8 | .7881 | .7910 | .7939 | .7967 | .7995 | .8023 | .8051 | .8078 | .8106 | .8133 |
| 0.9 | .8159 | .8186 | .8212 | .8238 | .8264 | .8289 | .8315 | .8340 | .8365 | .8389 |
| 1.0 | .8413 | .8438 | .8461 | .8485 | .8508 | .8531 | .8554 | .8577 | .8599 | .8621 |
| 1.1 | .8643 | .8665 | .8686 | .8708 | .8729 | .8749 | .8770 | .8790 | .8810 | .8830 |
| 1.2 | .8849 | .8869 | .8888 | .8907 | .8925 | .8944 | .8962 | .8980 | .8997 | .9015 |
| 1.3 | .9032 | .9049 | .9066 | .9082 | .9099 | .9115 | .9131 | .9147 | .9162 | .9177 |
| 1.4 | .9192 | .9207 | .9222 | .9236 | .9251 | .9265 | .9279 | .9292 | .9306 | .9319 |
| 1.5 | .9332 | .9345 | .9357 | .9370 | .9382 | .9394 | .9406 | .9418 | .9429 | .9441 |
| 1.6 | .9452 | .9463 | .9474 | .9484 | .9495 | .9505 | .9515 | .9525 | .9535 | .9545 |
| 1.7 | .9554 | .9564 | .9573 | .9582 | .9591 | .9599 | .9608 | .9616 | .9625 | .9633 |
| 1.8 | .9641 | .9649 | .9656 | .9664 | .9671 | .9678 | .9686 | .9693 | .9699 | .9706 |
| 1.9 | .9713 | .9719 | .9726 | .9732 | .9738 | .9744 | .9750 | .9756 | .9761 | .9767 |
| 2.0 | .9772 | .9778 | .9783 | .9788 | .9793 | .9798 | .9803 | .9808 | .9812 | .9817 |
| 2.1 | .9821 | .9826 | .9830 | .9834 | .9838 | .9842 | .9846 | .9850 | .9854 | .9857 |
| 2.2 | .9861 | .9864 | .9868 | .9871 | .9875 | .9878 | .9881 | .9884 | .9887 | .9890 |
| 2.3 | .9893 | .9896 | .9898 | .9901 | .9904 | .9906 | .9909 | .9911 | .9913 | .9916 |
| 2.4 | .9918 | .9920 | .9922 | .9925 | .9927 | .9929 | .9931 | .9932 | .9934 | .9936 |
| 2.5 | .9938 | .9940 | .9941 | .9943 | .9945 | .9946 | .9948 | .9949 | .9951 | .9952 |
| 2.6 | .9953 | .9955 | .9956 | .9957 | .9959 | .9960 | .9961 | .9962 | .9963 | .9964 |
| 2.7 | .9965 | .9966 | .9967 | .9968 | .9969 | .9970 | .9971 | .9972 | .9973 | .9974 |
| 2.8 | .9974 | .9975 | .9976 | .9977 | .9977 | .9978 | .9979 | .9979 | .9980 | .9981 |
| 2.9 | .9981 | .9982 | .9982 | .9983 | .9984 | .9984 | .9985 | .9985 | .9986 | .9986 |
| 3.0 | .9987 | .9987 | .9987 | .9988 | .9988 | .9989 | .9989 | .9989 | .9990 | .9990 |

This table gives the cumulative distribution function  $F(x)$  of a normal variable  $Z$ . That is,  $F(x) = \Pr(Z \leq x)$ . The table specifies  $x$  values to two decimal places. You find  $x$  through its first decimal place by row, and then find the second decimal place by column. For instance, if  $x = 1.63$ , you look in the row labeled 1.6 on the left and in the column labeled .03 on top. Thus  $F(1.63) = .9484$ . To determine  $F(x)$  for  $x < 0$  you use the fact that the normal density function is symmetric about 0.

$$\Pr(Z \leq 1.095) = .8632.$$

But what we want is  $\Pr(|Z| \leq 1.095)$  which we can't look up directly in a cumulative table like Table 6.1. How do you get from the probability above to the one we want? Well, since the normal is symmetric about 0, you know that  $\Pr(Z \leq 0) = .5$ . Therefore,  $\Pr(0 \leq Z \leq 1.095) = .8632 - .5 = .3632$ . By symmetry again, the probability we wish must be twice this or .7264. Thus a result within 10 of the expected value is likely, but far from a sure bet.

Finally, how good in this case is the assumption that  $Z$  is normal? The exact probability that one dot comes up between 90 and 110 times, now computed from the (discrete) Binomial distribution with a CAS, is  $\sum_{k=90}^{110} \binom{600}{k} (1/6)^k (5/6)^{600-k} = .7501$  so, in fact, our approximate value of .7264 is only moderately close. (In fact, the normal approximation is actually much better than that, because we really should pick the bounds on  $Z$  not to match  $90 \leq X \leq 110$  but rather to match  $89.5 \leq X \leq 110.5$ . See [3], where you will find that, with this change, the approximation is extremely close.) ■

## EXAMPLE 2

One hundred people are chosen at random and asked if they prefer G or B. Suppose 62 say B. Let  $p$  be the fraction of all the people who support B. What is the probability that  $|.62 - p| \leq .1$ ? That is, what is the probability that B is supported by between 52% and 72% of the people?

**Solution** If  $X$  is the random variable that counts the number of people out of 100 who support B, then what the poll gives is a value of .62 for  $Y = X/100$ . What we seek is the probability that

$$|(X/100) - p| \leq .1. \tag{4}$$

As in the previous example we may assume that

$$Z = \frac{(X/100) - p}{\sqrt{pq/100}} = \frac{10}{\sqrt{pq}} \left( \frac{X}{100} - p \right) \tag{5}$$

is distributed like a standard normal variable. Therefore, using (4) and (5) we get as an equivalent statement to (4)

$$|Z| \leq \frac{10}{\sqrt{pq}} (.1) = \frac{1}{\sqrt{pq}}. \tag{6}$$

What now? We would like to look up the right side of (6) in a table but we don't know  $p$ .

Here's a nifty way around this. We've noted before that  $pq \leq 1/4$ . Therefore,  $1/\sqrt{pq} \geq 2$ . So  $\Pr(|Z| \leq 2) \leq \Pr(|Z| \leq 1/\sqrt{pq})$  (why?) and we can evaluate  $\Pr(|Z| \leq 2)$  easily from our table and obtain .9544 (see the discussion in the solution to Example 1). Summarizing,

$$\Pr(|(X/100) - p| \leq .1) = \Pr(|Z| \leq 1/\sqrt{pq}) \geq \Pr(|Z| \leq 2) = .9544.$$

The value we got for  $X$  was .62. Therefore we conclude that  $|.62 - p| \leq .1$  with probability *at least* .9544. (It is, by the way, a little misleading to give all four digits

of .9544 because  $Z$  isn't *exactly* a normal variable although it's very close to being one. We should really be more modest; .95 is enough.) ■

Note this general rule: Whenever you wish to estimate an unknown probability  $p$  of a result being within some range around a sample probability  $\bar{p}$ , you will have the problem that  $p$  will be on the "wrong" side of the equation as in Eq. (6). The solution always is to replace  $pq$  with  $1/4$  and plough ahead but remember that the result you announce will be an understatement.

We'll now apply this rule to (finally!) get an answer to Example 7 in Section 6.1.

### EXAMPLE 3

A pollster wishes to determine the fraction  $p$  of voters who favor G. A sample of the voters will be polled and the sample fraction  $\bar{p}$  determined. How many voters should be polled in order to assert  $|\bar{p} - p| \leq .03$  (i.e., that the actual percentage of voters who favor G is within 3% of the percentage of those polled who favor G) with a 95% confidence level? ("Confidence level" is the conventional phrase for the probability that an assertion about an underlying probability  $p$  is correct.)

Notice the difference between this example and the previous one. There the poll was already taken and an after-the-fact confidence level was desired. Here we know how certain we wish to be of the closeness of  $\bar{p}$  to  $p$  and we wish to determine the size  $n$  of the sample to be polled.

**Solution** In light of Theorem 1 we assume that

$$Z = \frac{\bar{p} - p}{\sqrt{pq/n}}$$

is a standard normal variable. Note the use of  $\bar{p}$  instead of  $X/n$  as justified by the footnote on p. 578. Thus

$$\Pr(|\bar{p} - p| \leq .03) = \Pr(|Z| \leq .03/\sqrt{pq/n}) \geq \Pr(|Z| \leq 2\sqrt{n}(.03)), \quad (7)$$

where, once again, we have used the fact that  $1/\sqrt{pq} \geq 2$  to obtain the inequality. From (7) we see that we wish  $.06\sqrt{n}$  to be that number  $c$  such that  $\Pr(|Z| \leq c) \geq .95$ . From Table 6.1 one finds that  $c = 1.96$  (why?). So

$$.06\sqrt{n} = 1.96,$$

from which it is easily calculated that  $n = 1067$ . ■

Wait a minute. Do you believe this result? It says that no matter how large the underlying population, such as the approximately 200,000,000 Americans of voting age, if you poll 1067 people about which candidate they prefer, the result you get will be within 3% of the actual  $p$  with a 95% confidence level. Yes, that's what it says and that's what it means. When you see a poll of, typically, about 1100 people (1100 rather than precisely 1067 to give a bit of a cushion), the resulting candidate preferences are, indeed, within 3% of the true value with a 95% confidence level. Well, it doesn't *quite* mean that. Our assumption that  $Z$  is a standard normal variable is only an approximation but it's such a good one that the error introduced by this assumption is negligible. A more serious problem is the assumption that the

1100 people sampled constitute a *random* sample. Of course, the sample isn't truly random because it is always very hard to make a sample truly random. Thus there is no way that 1100 voters out of a population of 200,000,000 can really be a random sample. But modern polling techniques do allow a pretty good approximation to randomness so that the claim of  $\pm 3\%$  error with confidence level 95% is a pretty good one.

But why settle for a 95% confidence level? Why not aim for 99%? Indeed, 99% and sometimes 90% confidence do occasionally show up in the literature. If you do require 99%, the result corresponding to 1067 is that  $n$  should be 1842 [8]. So such a poll would be about twice as expensive as for a 95% confidence level. You might also ask: Why settle for a 3% error? Why not, for example, aim for 1%? We leave the calculations related to these questions to a problem [8].

*A Cautionary Tale.* We have stressed the need for the items sampled (voters, dice, whatever) to be *randomly* selected because if we stray too far from randomness, the results obtained can be badly wrong.

A famous example of biased (i.e., nonrandom) sampling was a poll by *The Literary Digest* magazine in 1936 about the presidential race between Landon and Roosevelt. The poll predicted a landslide for Landon with Roosevelt getting only 40.9% of the vote. In fact, the landslide was the other way with Roosevelt getting 60.7% of the vote. The problem was not the size of the sample because 10 million people were sent postcards to be returned. The problem was that the names of the people polled were taken from telephone books. In 1936 you had to be relatively rich to own a phone. But wealth and politics then (and now!) are not independent. One fallout from this poll was that shortly afterwards *The Literary Digest* went out of business!

## Problems: Section 6.7

---

For many of the problems of this section you will need to use Table 6.1 of the cumulative normal distribution function, or an appropriate calculator or software.

1.  $\langle 1 \rangle$  Is Example 1 probability or statistics (as differentiated in the first paragraph of this section)?
2.  $\langle 2 \rangle$  Let  $Y$  be a binomially distributed random variable based on a probability  $p$  of success scaled by  $1/n$  where  $n$  is the number of trials. Define

$$Z = \frac{Y - p}{\sqrt{pq/n}}.$$

Show that  $Z$  has mean 0 and variance 1.

3.  $\langle 2 \rangle$  When approximating a binomial distribution by a normal distribution, the accuracy is improved substantially by extending the bounds out to the next half integer. For instance, in Example 1,  $X$

was between 90 and 110, but, to find appropriate bounds on the continuous normal variable  $Z$ , it would be better to assume  $89.5 \leq X \leq 110.5$ , because to make the comparison we should now think of  $X$  as continuous and so we shouldn't ignore values of  $X$  closer to 90 than to 89 (or closer to 110 than 111). Do the calculations of Example 1 over again with this correction. How much difference does it make in the answer? How close are we now to the exact binomial value of .7501?

4.  $\langle 2 \rangle$  Let  $Y$  be a random variable and let  $F(y)$  be its cumulative distribution function. Assume  $Y$  is a continuous variable, which means  $\Pr(x < c) = \Pr(x \leq c)$  for all numbers  $c$ .
  - a) Express  $\Pr(|Y| \leq c)$  in terms of  $F(c)$  and at least one other value of  $F$ .

- b) Now assume  $Y$  is symmetric around 0. Express  $\Pr(|Y| \leq c)$  solely in terms of  $F(c)$  and constants. (One consequence of the answer is that if you are only interested in events like  $|Y| \leq c$  or  $Y \geq c \geq 0$ , then, as in Table 6.1, you only need half the cumulative distribution table — the half with  $c \geq 0$ .)
5. ⟨2⟩ A fair coin is to be tossed 10,000 times. Let  $h$  be the number of heads. Which is more likely, that  $4990 \leq h \leq 5010$  or that  $5020 \leq h \leq 5100$ ?
6. ⟨3⟩ Two airlines compete on the same route, for which demand is 1000 seats a day. Airline A believes it has a 50% chance of attracting any given customer. How much capacity must Airline A offer per day if it does not want to turn away customers more than 10% of the days?
7. ⟨2⟩
- In the second paragraph after Theorem 1, we noted that, for a normal variable  $Z$ , it can be shown that  $\Pr(|Z| \leq 2) = .9544$ . Using Chebyshев's Inequality (Theorem 3 in Section 6.6), what is the bound you get on  $\Pr(|Z| \leq 2)$ ?
  - Why do you think it is the case that Chebyshев's Inequality gives a much poorer bound on the probability that  $|Z| \leq 2$  than can actually be obtained for a normal variable?
8. ⟨2⟩ Example 3 is concerned with determining how many people should be polled to get within a fraction  $F = .03$  of the true fraction supporting a candidate with a confidence level  $C = 95\%$ . Repeat this calculation for the following values of  $F$  and  $C$ :
- $F = .03, C = 99\%$ . (The correct answer is given in the text but do the calculation to verify this answer.)
  - $F = .03, C = 90\%$ .
  - $F = .01, C = 95\%$ .
  - $F = .01, C = 99\%$ .
9. ⟨2⟩ A pollster polls 500 people on their preference between two candidates. She wishes to claim that her poll is correct within a fraction  $F$  with a confidence level  $C$ . Give three examples of  $C, F$  pairs that would be appropriate for this sample size.
10. ⟨2⟩ 1000 people are asked to reply Yes or No on a question of national importance. 52% of those surveyed answer Yes. What is the probability that a majority of the nation would answer Yes?
11. ⟨3⟩ The dean wants to know what percentage of first-year students are using cocaine. He decides to send out a confidential questionnaire (i.e., names not to be included on replies). But realizing that many people doubt the confidentiality of even supposedly confidential questionnaires, he uses the following technique to encourage students to reply. Instead of simply asking if you use cocaine, the questionnaire is worded as follows:
- Flip a coin two times. If it comes up heads twice, answer the following question falsely. Otherwise answer it truthfully. The question is: Do you use cocaine? Please note: with this rule, even if someone somehow figures out who you are from this questionnaire, that person will have no way to know whether your answer is true or false and therefore cannot use it against you in any way.
- Suppose 400 first-year students answer the question and 26% say Yes. Give a 95% confidence interval for the percentage of freshmen who use cocaine. *Hint:* Let  $p$  be the probability that a random freshman will answer Yes. This you can estimate by the methods of this section. Now let  $p'$  be the probability that the Dean is actually interested in, namely that a random freshman uses cocaine. Relate the two probabilities.
12. ⟨3⟩ The binomial model of sampling assumes that each sample point is picked at random *and* independently from the entire population. In particular, the same point *could* be chosen more than once. In polling this is not done — you poll each person at most once. Thus polling should be modeled by sampling *without replacement* (see also Example 1 in Section 6.3). That is, once a point is sampled, it is discarded so that it cannot be sampled again.
- When the number of sample points  $n$  is small compared to the size of the total population, the probability of picking the same point more than once at random is so small that the error in the analysis due to assuming the binomial distribution model is insignificant. This is good because the mathematics of sampling *without replacement* is more complicated so it's nice to avoid it.

To get a feel for the difference, assume that the entire population consists of 5 people, 2 of whom favor candidate B. Assume that we take a sample of 2 from this population without replacement. Let  $X$  be the number of people in the sample who favor B. Determine the distribution of  $X$  completely, and also its mean and variance. Compare these with the analogous values for sampling with replacement (the binomial model) for this population.

13. {2} Suppose the entire population has  $N$  people,

and we poll them *all* on some Yes-No question. Suppose the sample frequency of a Yes answer is  $\bar{p}$ . According to the theory developed in this section and the previous one, for this  $N$  we can find some tolerance  $\epsilon$  and come confidence level  $C$  so that  $|\bar{p} - p| \leq \epsilon$  with confidence  $C$  but we cannot make  $\epsilon = 0$  and  $C = 1$ . Yet obviously it's absolutely certain that  $\bar{p} = p$  in this case. So what's wrong with our theory? (Answer: nothing. See [12] and then explain.)

## 6.8 Applications to Algorithms: Proofs of Prior Claims

At various points earlier in this book, we gave intuitive probabilistic arguments about algorithms. In many cases we made claims we said we would justify in Chapter 6. Now's the time!

It's time because this chapter has built a lot of muscles, and you should be ready to use your new strength. We don't need any new example algorithms because the old ones provide quite enough opportunity for a workout. In each of the following subsections, we take an algorithm discussed earlier, review how we argued its analysis before, and show how this argument is justified with our new concepts and theorems (i.e., muscles). The subsections are ordered from easier to harder, but none is trivial.

### Average Case Analysis of Algorithm MAXNUMBER

MAXNUMBER, which we introduced in Example 1, Section 1.5, is repeated as Fig. 6.17. It finds the maximum of distinct numbers  $x_1, x_2, \dots, x_n$  by comparing each element  $x_k$  in turn with  $m$ , the maximum of the previous elements, and updating  $m$  if  $x_k$  is larger. The hard question is: On average, how many times is  $m$  assigned? (Actually, the question asked in Section 1.5 was: How many times is  $m$  reassigned? The difference in the number of assignments is 1, as there is one initial assignment,  $m \leftarrow x_1$ . Henceforth we discuss the number of assignments, because the results are a little prettier.)

We argued in Section 1.5 that it is sufficient to consider the case where the  $x$ 's are a random permutation of  $[n] = \{1, \dots, n\}$ . For a given permutation, the number of assignments of  $m$  equals the number of times, as  $k$  goes from 1 to  $n$ , that  $x_k$  is the largest  $x$  so far. However, instead of looking at the number of assignments per permutation, we looked at *positions*. We computed the probability, over all permutations of  $[n]$ , that the  $k$ th entry is the largest so far. This probability is clearly  $1/k$ , because for any set of  $k$  distinct integers in the first  $k$  entries,  $1/k$  of them have the largest of these  $k$  integers in the  $k$ th spot. Thus we claimed that the expected number of assignments was  $\sum_{k=1}^n 1/k$ . (For reassessments the sum starts at  $k = 2$ .) Why is this right?

**Input**  $n, x_1, x_2, \dots, x_n$   
**Output**  $m$

**Algorithm** MAXNUMBER

```

 $m \leftarrow x_1$ 
for  $k = 2$  to  $n$ 
    if  $x_k > m$  then  $m \leftarrow x_k$ 
endfor

```

**FIGURE 6.17**

Algorithm  
MAXNUMBER

First of all, you can now recognize that we are talking about random variables. Let  $Y$  be the random variable with domain (i.e., sample space) the set of permutations of  $[n]$ , which gives for each permutation the number of times the variable  $m$  is assigned by the algorithm. We are doing an average case analysis, so we want  $E(Y)$ . We obtain it by writing  $Y$  as a sum of simpler variables. Specifically, let  $X_k$  be the binary random variable which is 1 when a permutation causes MAXNUMBER to make an assignment of  $m$  at the  $k$ th spot, and 0 if it doesn't. Clearly,

$$Y = \sum_{k=1}^n X_k. \quad (1)$$

By the sum theorem for expectations (Theorem 1, Section 6.5), it suffices to find  $E(X_k)$  for all  $k$ . Since each  $X_k$  is a 0–1 variable, we know that

$$E(X_k) = \Pr(X_k=1) = \frac{1}{k}.$$

(This equality is correct even for  $X_1$ , for a special reason. Why?) Thus

$$E(Y) = \sum_{k=1}^n E(X_k) = \sum_{k=1}^n \frac{1}{k},$$

and the justification is done.

Notice that we did *not* compute  $E(Y)$  directly from its definition:

$$E(Y) = \sum_k k \Pr(Y=k). \quad (2)$$

We didn't because it looked hard. If we could count the *number* of  $n$ -permutations that require  $k$  assignments of  $m$ , then we would know that  $\Pr(Y=k)$  is this number divided by  $n!$ , the total number of permutations. But short of brute force, how to do this count isn't obvious – and after doing it, we would still have to evaluate the sum in Eq. (2). That's why we turned from looking at assignments per permutation (i.e.,  $Y$ ) to assignments per position ( $X_k$ ). This problem provides an excellent example of why decompositions like Eq. (1), coupled with the sum theorem for expectations, are so valuable.

Actually, the number of  $n$ -permutations requiring  $k$  assignments of  $m$  *can* be counted, and Eq. (2) *can* be evaluated directly. One way is to use doubly indexed

recurrences [18]. Indeed, the counts of permutations turn out to be the absolute values of the **Stirling numbers of the first kind**, important numbers that appear in other contexts as well [21].

*One caution.* Although decompositions like Eq. (1) provide a powerful solution technique, they can't do everything. At the end of Section 6.5 we mentioned that expected value is not the whole story and that you ought at least to know the variance, too. Unfortunately, the sum theorem for variances requires independent variables. For Algorithm MAXNUMBER, we luck out: The  $X_k$ 's are independent and so decomposition leads to an easy computation of the variance [2]. In general it is not possible to obtain an independent decomposition and the variance must be obtained by more advanced methods.

## Average Case Analysis of Sequential Search

We considered this average case problem in Section 1.5, Example 4, and again in Section 5.2, Example 3. It is the more sophisticated recursive approach in Section 5.2 that we return to now.

With  $E_n$  as the average number of comparisons for a list of  $n$  words (when the search word is guaranteed to be on the list), we concluded in Section 5.2 that

$$E_n = \frac{k}{n} E_k + \frac{n-k}{n} (k + E_{n-k}). \quad (3)$$

We obtained Eq. (3) by first claiming that the number of comparisons with the first  $k$  elements of the list is, on average,

$$\frac{k}{n} E_k + \frac{n-k}{n} k. \quad (4)$$

Then we claimed that the average number of comparisons with the remaining  $n - k$  words is

$$\frac{n-k}{n} E_{n-k}. \quad (5)$$

Finally, we added these expressions to get Eq. (3). Why is this right?

First of all, once again we are dealing with a random variable, namely, the number of comparisons during the search, and we are seeking its expected value. Call that variable  $X$ , so that  $E_n = E(X)$ . Once again we use a sum decomposition:  $X = Y + Z$ , where  $Y$  is the number of comparisons with the first  $k$  words, and  $Z$  is the number with the remaining  $n - k$  words. Thus so long as expression (4) is  $E(Y)$  and expression (5) is  $E(Z)$ , then adding them to obtain Eq. (3) is correct by the sum theorem for expectations.

So why is expression (4) equal to  $E(Y)$ ? The argument we gave at the time broke the analysis into two cases: Either we found the search word among the first  $k$  words (call this event  $A$ ), for an average of  $E_k$  over these cases only, or we didn't find the search word among the first  $k$  (event  $\sim A$ ), in which case the average value of  $Y$  is  $k$  because the value of  $Y$  is always  $k$  for this event. Finally, we weighted these two events according to their probabilities ( $k/n$  for the first case and  $(n-k)/n$  for the second).

In short, we computed  $E(Y)$  by computing  $Y$ 's expected values when restricted to two complementary events and then taking a weighted average of these values. That is, define

$$E(Y|A) = \text{Expected value of } Y \text{ given that event } A \text{ occurs.}$$

and let  $A$  be the event “search word found among the first  $k$  words”. Then we are asserting  $E(Y)$  equals expression (4) because this assertion is of the form

$$E(Y) = \Pr(A)E(Y|A) + \Pr(\sim A)E(Y|\sim A). \quad (6)$$

(That is,  $\Pr(A) = k/n$ ,  $E(Y|A) = E_k$ , and so on.) However, Eq. (6) is not one that we've discussed before (though we've used it implicitly several times, and it appears in some problems; see [42, Section 6.5] and [12, 13]). It deserves a proof.

---

**Theorem 1.** For any event  $A$  and random variable  $Y$ , Eq. (6) is true.

---

**PROOF** First, you must understand what the **conditional expectation**  $E(Y|A)$  means. It means that we restrict our attention to event  $A$ , making it the whole sample space, and weight the values of  $Y$  by their conditional probabilities in this space. Thus

$$E(Y|A) = \sum_y y \Pr(\{Y=y\}|A) = \sum_y y \frac{\Pr(A \cap \{Y=y\})}{\Pr(A)}$$

using Eq. (1) in Section 6.3. Now to the proof proper. By definition

$$E(Y) = \sum_y y \Pr(Y=y).$$

So using the analogous equation to the one just above for  $E(Y|\sim A)$

$$\begin{aligned} & \Pr(A)E(Y|A) + \Pr(\sim A)E(Y|\sim A) \\ &= \Pr(A) \sum_y y \frac{\Pr(A \cap \{Y=y\})}{\Pr(A)} + \Pr(\sim A) \sum_y \frac{\Pr(\sim A \cap \{Y=y\})}{\Pr(\sim A)} \\ &= \sum_y y \Pr(A \cap \{Y=y\}) + \sum_y y \Pr(\sim A \cap \{Y=y\}) \\ &= \sum_y y [\Pr(A \cap \{Y=y\}) + \Pr(\sim A \cap \{Y=y\})] \\ &= \sum_y y \Pr(Y=y) = E(Y), \end{aligned}$$

the last line following because the sum in the previous line includes all values of  $y$  and the sum in the last line is  $E(Y)$  by definition. ■

Returning to sequential search, we have now justified that  $E(Y)$  is given by expression (4). How come  $E(Z)$  is given by expression (5)? For the same reason —

just substitute  $Z$  for  $Y$  in the theorem. Recall that  $Z$  is the number of comparisons on the back  $n - k$  words, and  $A$  is the event that we succeed in finding the search word before getting there. Thus  $E(Z|A) = 0$ , because  $Z$  is constantly 0 for event  $A$ , and  $E(Z|\sim A) = E_{n-k}$ , because once the search of the first  $k$  words fails, we are back to the case of searching  $n - k$  words from scratch (with success guaranteed since success on the whole list of  $n$  was assumed).

This finishes the justification of Eq. (3). For an alternative derivation, without recourse to summing expressions (4) and (5), see [9]. ■

## Analysis of Algorithm PERMUTE-1

We reprint PERMUTE-1 (from Section 4.9) as Fig. 6.18. It repeatedly picks a random integer from 1 to  $n$ , putting each into the permutation if it hasn't appeared before and throwing it out otherwise.

|                                            |                                                 |
|--------------------------------------------|-------------------------------------------------|
| <b>Input</b> $n$                           | [Length of permutation]                         |
| <b>Output</b> Perm                         | [A random permutation given as a sequence]      |
| <b>Algorithm</b> PERMUTE-1                 |                                                 |
| Flag $\leftarrow 0$                        | [Set <i>all</i> components of Flag to 0]        |
| <b>for</b> $i = 1$ <b>to</b> $n$           | [ $i$ th pass will determine $\text{Perm}(i)$ ] |
| <b>repeat</b>                              |                                                 |
| $r \leftarrow \text{RAND}[1, n]$           | [Pick a random number]                          |
| <b>endrepeat when</b> $\text{Flag}(r) = 0$ | until an unused one found]                      |
| $\text{Perm}(i) \leftarrow r$              |                                                 |
| $\text{Flag}(r) \leftarrow 1$              | [ $r$ has been used]                            |
| <b>endfor</b>                              |                                                 |

**FIGURE 6.18**

Algorithm  
PERMUTE-1

In Section 4.9 we had to answer several questions about PERMUTE-1 informally. The first was: Does PERMUTE-1 really pick a *random* permutation, that is, does each permutation of  $1, \dots, n$  have an *equal* chance of being selected? This would not be a question if Algorithm PERMUTE-1

- immediately picked any integer from 1 to  $n$  with probability  $1/n$ ;
- then immediately picked any unpicked integer with probability  $1/(n-1)$ ;
- then immediately picked a third unpicked integer with probability  $1/(n-2)$ ;
- and so on;

as do Algorithms PERMUTE-2 and PERMUTE-3.

For then each permutation would have probability  $1/n!$ . However, at each stage PERMUTE-1 picks *any* integer from 1 to  $n$  with probability  $1/n$ , but then throws it out if it's been picked before. Let  $k$  be the number of entries *already* put into the permutation when we enter the for-loop. (In terms of the variable  $i$  in

the algorithm,  $k = i - 1$ .) We must show that, for each  $k$  from 0 to  $n - 1$ , each of the  $n - k$  so far unpicked integers has probability  $1/(n - k)$  of being the next integer picked and retained. (We introduce  $k$  because we often need to refer to the number of so-far unpicked integers, and  $n - k$  is more pleasant than the equivalent expression  $n - i + 1$ .)

Actually, all we need to show is that each so-far unpicked integer has an *equal* chance of being the next new number picked, for then each of the  $n - k$  possible outcomes would have probability  $1/(n - k)$ . (The only other possible outcome — previously picked integers keep coming up forever — has probability 0.) But that each unpicked integer has an equal chance is obvious since, when an integer different from one already picked does appear (i.e.,  $\text{Flag}(r) = 0$ ), each unpicked integer has an equal probability of being that one.

The other thing we claimed in a hand-waving way about PERMUTE-1 was that the average number of RAND calls is

$$n \sum_{j=1}^n \frac{1}{j},$$

i.e.,  $nH(n)$  where  $H(n)$  is the  $n$ th harmonic number. But this expression is easy to justify: From the solution on p. 562 to Example 3, Section 6.1, we know that the expected number of RAND calls to pick the  $(k+1)$ st number is  $n/(n-k)$ , and thus by the additivity of expectations, the expected number of calls to pick all  $n$  numbers is

$$\sum_{k=0}^{n-1} \frac{n}{n-k} = n \sum_{j=1}^n \frac{1}{j}.$$

## Average Case Analysis of Algorithm PERMUTE-2

PERMUTE-2 (from Section 4.9) is repeated as Fig. 6.19. When  $k$  numbers have already been picked on entry to the for-loop (again  $k = i - 1$ ), the algorithm picks a random integer  $r$  from 1 to  $n - k$  and then finds the value  $j$  of the  $r$ th so-far unused integer by checking the flags. So there is no problem about the permutation being random. The hard question was: What is the expected number of Flag checks?

In each iteration of the  $i$ -loop, there is one flag check for each value of  $j$  from 1 to the final value assigned to  $\text{Perm}(i)$ . In Section 4.9 we said it was plausible that this final value of  $j$  is uniformly distributed from 1 to  $n$ , because the numbers previously put in the permutation are randomly distributed, and the next assigned number is randomly distributed among the unassigned (via  $r$  being random).

Let's firm this up. Let  $J$  be a random variable, with values in  $[n] = \{1, 2, \dots, n\}$ , for the final value of  $j$  in the  $i$ th pass of the outer loop. We want to show that, for each integer  $m \in [n]$ ,  $\Pr(J=m) = 1/n$ . The previous paragraph breaks the evaluation of  $J$  into two parts — what happened in previous passes of the outer loop and what happens in the current pass. Since the second part depends on the first, we are talking about the product of probabilities where one is conditional on the other. Specifically, let  $U_m$  be the event that integer  $m$  is not picked in the  $k$

**Input**  $n$

**Output** Perm

**Algorithm** PERMUTE-2

```

Flag ← 0
for  $i = 1$  to  $n$                                 [Pick  $i$ th entry in Perm]
     $r \leftarrow \text{RAND}[1, n-i+1]$ 
     $c \leftarrow 0$                                [Number of unchosen values passed over]
     $j \leftarrow 0$                                [Initializes number considered for  $\text{Perm}(i)$ ]
    repeat while  $c < r$ 
         $j \leftarrow j + 1$ 
        if  $\text{Flag}(j) = 0$  then  $c \leftarrow c + 1$ 
   [Another unchosen number found]
    endrepeat
     $\text{Perm}(i) \leftarrow j$ 
     $\text{Flag}(j) \leftarrow 1$ 
endfor

```

**FIGURE 6.19**

Algorithm  
PERMUTE-2

previous passes of the outer loop ( $U$  for unpicked). Then

$$\Pr(J=m) = \Pr(U_m) \Pr(J=m | U_m). \quad (7)$$

The value of  $\Pr(J=m | U_m)$  is definitely  $1/(n-k)$ , because  $m$  is previously unchosen,  $r$  is chosen uniformly from set  $[n-k]$ , and *only one* value of  $r$  results in  $J = m$ . (We don't know what that value of  $r$  is — it depends in a complicated way on the numbers already chosen for  $\text{Perm}$  — but we don't need to know it.)

Next,  $\Pr(U_m)$  is not hard to determine. Either  $m$  is in the first  $k$  numbers picked or it is in the latter  $n - k$  still to be picked.  $\Pr(U_m)$  is just the probability of the second of these two possibilities, namely  $(n - k)/n$ . Therefore, Eq. (7) becomes

$$\Pr(J=m) = \frac{n-k}{n} \frac{1}{n-k} = \frac{1}{n}.$$

Finishing up the average case analysis of PERMUTE-2 is easy. First,

$$E(J) = \sum_{m=1}^n m \Pr(J=m) = \sum_{m=1}^n \frac{1}{n} m = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2}.$$

This equality gives the expected number of flag checks per pass. Since PERMUTE-2 consists of  $n$  passes, and since expectations add, we get that the total number of expected flag checks is  $n(n+1)/2$ , as claimed in Section 4.9.

# Problems: Section 6.8

1. ⟨1⟩ In the analysis of Algorithm MAXNUMBER, what is the special reason that makes  $E(X_1) = 1$ ?

2. ⟨3⟩ Consider the variables  $X_1, X_2, \dots, X_n$  in the analysis of Algorithm MAXNUMBER.

- a) Show that they are pairwise independent. It suffices to show (See [43, Section 6.4]) that for  $i \neq j$

$$\Pr(X_i=a, X_j=b) = \Pr(X_i=a) \Pr(X_j=b),$$

for  $a = 0, 1$  and  $b = 0, 1$ . In fact,  $X_1, X_2, \dots, X_n$  are mutually independent, which is what we need for part b). However, we do not ask you to show this, since we have barely touched on the definition of mutual independence for variables (see [44, Section 6.4]) and the demonstration gets involved.

- b) Recall that  $Y = \sum_{k=1}^n X_k$  is the random variable for the number of assignments in MAXNUMBER. Using the fact that variances add for mutually independent variables (see [6, Section 6.6]), show that

$$\text{Var}(Y) = \sum_{k=1}^n \frac{1}{k} - \sum_{k=1}^n \frac{1}{k^2}.$$

3. ⟨1⟩ Write an algorithm MINNUMBER, similar to MAXNUMBER, to find the minimum of  $n$  inputted numbers. Determine the expected number of assignments. (The answer is the same as for MAXNUMBER. The point is for you to go through the steps yourself.)

## Algorithm 6.1 BIGNEXT

**Input**  $n, x_1, \dots, x_n$

$[n \geq 2]$

**Output**  $B, N$

### Algorithm BIGNEXT

```
if  $x_1 > x_2$  then  $B \leftarrow x_1; N \leftarrow x_2$ 
else  $B \leftarrow x_2; N \leftarrow x_1$ 
endif [Initialize  $B$  and  $N$ ]
for  $k = 3$  to  $n$ 
if  $x_k > B$  then  $N \leftarrow B; B \leftarrow x_k$ 
 $x_k > N$  then  $N \leftarrow x_k$ 
endif [3 cases; if  $x_k < N$  do nothing]
endfor
```

4. ⟨3⟩ Consider Algorithm 6.1, BIGNEXT, which finds the Biggest and Next-biggest of  $n$  distinct numbers. Do a best, worst, and average case analysis for the

- a) number of assignments;  
b) number of comparisons.

5. ⟨2⟩ In Algorithm BIGNEXT of [4], suppose we replace the for-loop with

```
for  $k = 3$  to  $n$ 
if  $x_k \leq N$  then do nothing
 $x_k \leq B$  then  $N \leftarrow x_k$ 
else  $N \leftarrow B; B \leftarrow x_k$ 
endif
endfor
```

Now what is the average number of assignments?  
The average number of comparisons?

6. ⟨2⟩ On a given day in Miami, the probability that it will rain is  $1/3$ . If the average amount of rain on a rainy day is 1 cm, what is the expected amount of rain per day? What is the expected amount of rain per year?

7. ⟨2⟩ On a typical January day in Buffalo, the probability of no precipitation is 20%, the probability of rain is 10%, and the probability of snow is 70%. When it rains, the average amount is .5 cm. When it snows, the average amount (in melted equivalent) is 1.5 cm. What is the expected amount of precipitation (in liquid measure) on a typical January day?

8. ⟨2⟩ Redo the proof of Theorem 1 using the alternative definition of  $E(X)$  in Eq. (16) of [37, Section 6.5], where  $E(X)$  is expressed as a weighted sum over events, not variable values. (The algebra of the proof is unchanged, but the notation is perhaps a little simpler.)

9. ⟨2⟩ Justify Eq. (3) as follows. Let  $X$  be the number of comparisons with the whole list of  $n$  words. Let  $A$  be the event “search word is among the first  $k$  words on the list”. Apply Theorem 1 directly to  $X$ . You must show that  $E(X|A) = k + E_{n-k}$ .

10. ⟨2⟩ For the proof of Theorem 1, give the reason for each equality in the final, multiline display.

11. (2) Let  $X$  be a random variable over a probability space and let  $A_1, A_2, \dots, A_n$  be a set of mutually exclusive and exhaustive events in the space. Prove

$$E(X) = \sum_{i=1}^n \Pr(A_i) E(X|A_i).$$

12. (2) Justify Eq. (8), Section 6.5, which gave a recursive relation among expected values for the binomial distribution. The justification depends on a result only really proved in the current section.

13. (2) Example 4, Section 6.5, gave two methods for finding the expected value of the negative binomial distribution with  $n = 1$ . The second method depended on the recursive equation

$$E(X) = p \cdot 1 + (1-p)(E(X)+1).$$

Justify this equation.

14. (2) Consider Algorithm BINSEARCH of Example 5, Section 1.5. Let  $A_n$  be the average number of comparisons when there are  $n$  words on the list and the search word is known to be on the list. Argue carefully that

$$\begin{aligned} A_n &= 1 + \frac{\lfloor \frac{n-1}{2} \rfloor}{n} A_{\lfloor \frac{n-1}{2} \rfloor} + \\ &\quad \frac{\lceil \frac{n-1}{2} \rceil}{n} A_{\lceil \frac{n-1}{2} \rceil}, \end{aligned} \tag{8}$$

$$A_1 = 1.$$

*Hint:* Use the three-term generalization of (6).

15. (3) Using the procedure for solving Divide and Conquer recurrences in Section 5.8, you can make some progress towards solving Eq. (8).

- a) What difference equation do you get if you use that procedure. What is its solution? Is that solution the exact solution to Eq. (8) for any values of  $n$ ?

- b) Following the statement of the procedure in Section 5.8, there was a brief discussion of a more refined approach. Using that approach, obtain a slightly different difference equation from Eq. (8).

- c) Solve the difference equation from part b). *Caution:* This is tough.

16. (2) We wish to produce a random multiset of size two from the integers 1 to 3. What's wrong with the following algorithm?

**Output**  $x, y$

**Algorithm** MULTISSET

$x \leftarrow \text{RAND}[1,3]$

$y \leftarrow \text{RAND}[1,3]$

17. (2) Use the formula for the sum of an infinite geometric series to show that

$$\sum_{m=0}^{\infty} \left(\frac{k}{n}\right)^m \frac{1}{n} = \frac{1}{n-k}.$$

What's this got to do with Algorithm PERMUTE-1?

In problems 18-21 below, let  $P_{n,k}$  be the number of permutations of  $[n]$  in which exactly  $k$  of the numbers  $x_1, \dots, x_n$  are the largest of the  $x$ 's up to that point. Let us say that such a permutation has  $k$  *temporary maxes*.

18. (4) We seek a recurrence for  $P_{n,k}$ . A natural way to start is with a recursive procedure for generating the  $n$ -permutations from the  $(n-1)$ -permutations.

Perhaps the most natural method is: For each permutation  $P$  of set  $[n-1]$ , create  $n$   $n$ -permutations by inserting the number  $n$  at each of the  $n$  possible slots before, in between, and after the  $n-1$  numbers in  $P$ . Call these  $n$   $n$ -permutations the permutations *derived* from  $P$ .

- a) Illustrate this method by using it to generate all the permutations of  $\{1, 2, 3\}$  from the two permutations of  $\{1, 2\}$ . Indicate which 3-permutations are derived from which 2-permutation.

- b) This method will lead easily to a recurrence for  $P_{n,k}$  if there is a simple relationship between the number of temporary maxes of the derived permutations and the number of temporary maxes of the starting permutation. Alas, there is not a simple relationship. Why?

Here is a perhaps less natural method for generating  $n$ -permutations from  $(n-1)$ -permutations. For a given  $(n-1)$ -permutation  $P$  of  $[n-1]$ , create  $n$  sequences by appending at the end, in turn,  $1, 2, \dots, n$ . Only the last of these is an  $n$ -permutation. (Why?) So modify the others as follows: If the number appended at the end is  $k$ , then for all  $j$  from  $k$  to  $n-1$ , replace  $j$  in  $P$  by  $j+1$ .

- c) Illustrate this method by using it to generate all the permutations of  $\{1, 2, 3\}$  from the

two permutations of  $\{1, 2\}$ . Indicate which 3-permutations are derived from which 2-permutation.

- d) Justify this method in general. Each  $(n-1)$ -permutation  $P$  leads to  $n$  different sequences of length  $n$ . Show that they are all permutations, and that each  $n$ -permutation is generated in this way exactly once.
- e) There is a simple relationship between the number of temporary maxes of  $P$  and the number in the  $n$ -permutations derived from it in this second way. The reason is that the only change in order is at the end. Find this relationship and use it to write a recurrence for  $P_{n,k}$ .

The numbers  $s_{n,k} = (-1)^{n+k} P_{n,k}$  are the Stirling numbers of the first kind. They satisfy a recurrence very similar to the recurrence of part e). The “temporary max” numbers  $P_{n,k}$  are thus the absolute values of the Stirling numbers of the first kind; sometimes the temporary max numbers are themselves called Stirling numbers.

19. ⟨3⟩ Let  $A_n$  be the expected number of assignments of  $m$  in Algorithm MAXNUMBER when  $n$  numbers are input.

- a) Show that  $A_n$  satisfies the recurrence

$$A_n = A_{n-1} + \frac{1}{n}.$$

*Hint:* Plug the recurrence from [18e] into the definition of  $E(A_n)$ . You will then have to do a fair amount of juggling of sums.

- b) To verify that  $A_n = \sum_{k=1}^n 1/k$ , it now suffices to check that the sequence  $\{A_n\}$  satisfies the right initial condition. Check this.

20. ⟨3⟩ Let  $p_{n,k} = P_{n,k}/n!$  be the probability that a random  $n$ -permutation has  $k$  temporary maxes. Develop a recurrence for  $p_{n,k}$  two ways:

- a) By substituting  $(n!)p_{n,k}$  for  $P_{n,k}$  in the recurrence of [18e].

- b) By arguing directly using the idea behind the permutation generation scheme preceding [18c].

21. ⟨4⟩ Permutations in cycle form. We have thought of permutations as a reordering of set  $[n]$ . One can also think of permutations as one-to-one functions from  $[n]$  to itself. For instance, the permutation 45132 is the function  $P$  which maps 1 to 4 (i.e., the first position holds 4), 2 to 5, 3 to 1, 4 to 3, and 5 to 2.

There is a third representation which is often useful. Starting with 1, look at  $P(1)$ ,  $P(P(1))$ , and so on, until you get back to 1. For the permutation 45132,

$$P(1) = 4, \quad P(P(1)) = P(4) = 3, \quad \text{and } P(3) = 1.$$

Thus 1 generates the cycle 143. Similarly, 2 generates the cycle 25, and in this case, every number in the permutation is now accounted for. The permutation as a whole is represented as (143)(25). (Does the order in which we consider numbers make any difference? Not really. Had we started with 5 and then gone to 4, we would have come up with the representation (52)(431), which is considered the same, since each parenthesized string represents a cycle with no natural beginning and end.)

Let  $C_{n,k}$  be the number of  $n$ -permutations whose cycle representation involves  $k$  cycles. Show two ways that  $C_{n,k} = P_{n,k}$ :

- a) By a one-to-one correspondence. That is, show that for each  $n$ -permutation with  $k$  cycles there is a (usually different)  $n$ -permutation with  $k$  temporary maxes.
- b) By showing that  $C_{n,k}$  satisfies the same recurrence and same initial conditions as  $P_{n,k}$ .

## 6.9 Recursive Methods in Probability

From time to time we have shown that recursive techniques often make the solution of a probability problem much easier. In this section we give several more examples of increasing difficulty, all couched in terms of various games. Among the examples is a complete solution to the game Gambler’s Ruin, a particular case of which is the one remaining unanswered problem from Section 6.1 (Example 5).

## EXAMPLE 1

Two people play the following game. They alternate tossing a single fair die. The first person to throw a 6 wins. What is the probability that the first player wins?

**Solution** This problem is not that hard to solve directly, using an infinite sum [2]. But it is even easier using recursion. In fact, we show two different solutions using recursion.

*Method 1.* Let  $p_1$  denote the probability of a win by player 1. Either she wins on the first toss (probability  $\frac{1}{6}$ ), or winning takes more tosses. If it takes more tosses for player 1 to win, this means that neither player tosses a 6 on the first try (probability  $\frac{5}{6} \cdot \frac{5}{6}$ ), after which it is as if the game is starting over. Thus

$$p_1 = \frac{1}{6} + \left(\frac{5}{6}\right)\left(\frac{5}{6}\right)p_1. \quad (1)$$

Solving Eq. (1), we find that  $p_1 = 6/11$ .

*Method 2.* Let  $p_2$  be the probability that player 2 wins. In order for him to win, it is necessary that player 1 not toss a 6 on her first throw. Thereafter, it is as if player 2 were the first player. Thus

$$p_2 = \frac{5}{6} p_1.$$

Combining this equality with

$$p_1 + p_2 = 1,$$

we readily solve and find again that  $p_1 = 6/11$ . ■

Example 1 is typical of a type of problem where, after certain sequences of events, it is as if you have started all over again. You can always solve such a problem recursively, by assuming you know the answer in the case that the problem starts over again inside itself. You don't even get a difference equation to solve — just an ordinary algebraic equation in the unknown probability.

## EXAMPLE 2

Find the expected number of tosses in Example 1, (a) in general; (b) in the case that player 1 wins; (c) in the case that player 2 wins.

**Solution** We solved (a) in Section 6.5 when we computed the expectation of the negative binomial. In fact, the one simple solution there was by recursion. If we don't care who wins, this game is just Bernoulli trials until the first 6 is tossed, so the answer, call it  $E$ , is  $1/p = 6$ .

To solve (b) and (c), let  $E_i$  be the expected number of tosses if player  $i$  wins. From Theorem 1, Section 6.8, and using the notation and result of Example 1, we know that

$$E = p_1 E_1 + p_2 E_2 = \frac{6}{11} E_1 + \frac{5}{11} E_2. \quad (2)$$

Now, since every win by player 2 looks just like a win by player 1 with an extra toss at the front,

$$E_2 = 1 + E_1.$$

Thus Eq. (2) becomes  $6 = (6/11)E_1 + (5/11)(1 + E_1) = E_1 + (5/11)$ , from which it follows that

$$E_1 = \frac{61}{11} \quad \text{and} \quad E_2 = \frac{72}{11}. \blacksquare$$

### EXAMPLE 3

#### Winning a Game of Tennis

Tennis consists of many *rallies*, each of which ends in a point. A *game* in tennis is won by the player who first gets 4 points, except that one must always win by 2 points. By tradition, the score is recorded in a funny way, but we won't worry about that and will use the obvious recording method, e.g., 4–2. Also, a game is just part of a set, which in turn is part of a match, but we won't worry about that either, yet.

So,  $A$  and  $B$  play tennis. If  $A$  has probability  $p$  of winning any given point, what is the probability that  $A$  will win a game?

**Solution** If we temporarily ignore cases where the game goes into “overtime” due to the win-by-2 rule, the problem is not hard. Player  $A$  can win 4–0, 4–1, or 4–2. The probability of winning 4–0 is  $p^4$ . The probability of winning 4–1 is the number of sequences of 5 points ending in a win times the probability,  $p^4q$ , of any such sequence. (Recall that  $q = 1 - p$ .) The number of such 5-point sequences is  $C(4, 1)$ , because we can allow the one lost point to be any one of the first four. Similarly,

$$\Pr(A \text{ wins } 4-2) = \binom{5}{2} p^4 q^2.$$

So far there's no recursion in sight, but let's get back to the win-by-2 rule. This rule makes a difference only if play reaches 3–3. To concentrate on the essentials, let's figure out the probability that  $A$  wins *given* that 3–3 has been reached. Call this  $p^*$ . To win,  $A$  must either win two points in a row, or else  $A$  and  $B$  must split points and then  $A$  has probability  $p^*$  of winning from there — because of the win-by-2 rule it is as if they were back at 3–3. Therefore

$$p^* = p^2 + (2pq)p^*, \tag{3}$$

from which simple algebra gives

$$p^* = \frac{p^2}{1 - 2pq}.$$

(Why is it  $2pq$  in Eq. (3) and not just  $pq$  for the probability of winning 1 point each?) Thus the probability that  $A$  wins in overtime, starting from the beginning, is  $p^*$  times the probability of reaching 3–3, that is,  $C(6, 3)p^3q^3p^*$ . In conclusion,

$$p_G = p^4 + \binom{4}{1} p^4 q + \binom{5}{2} p^4 q^2 + \binom{6}{3} \frac{p^5 q^3}{1 - 2pq} \tag{4}$$

is the total probability that  $A$  wins.  $\blacksquare$

## EXAMPLE 4

### Winning a Point in Volleyball or Racquetball

In either of these games, who is serving makes a difference in scoring. If *you* are serving and win the rally, you win a point and keep the serve. If the other side is serving and you win the rally, no point is scored but you get the serve. Suppose that you have probability  $p$  of winning a rally, independent of who is serving. What is the probability that you will win the next point if (a) you have the serve; (b) the other side has the serve? *Note:* The assumption that  $p$  is independent of who serves is not really correct, but let's go one step at a time in building more complex models. Later tackle [19–20].

#### Solution

- a) You can win the next point by either winning a rally right away or by trading rally wins back and forth with your opponent for a while before winning a rally on your serve. Thinking recursively, either you win a point right away, or your opponent wins the first rally, you win the next, and then it is as if the game is just starting with you serving. So, letting  $p_a$  be the desired probability, we have

$$p_a = p + (pq)p_a.$$

Thus

$$p_a = \frac{p}{1-pq}.$$

- b) Think recursively again. Your opponent is serving, so you must win the first rally. But then it is exactly as if you were starting the game by serving, case a). Thus

$$p_b = pp_a = \frac{p^2}{1-pq}.$$
 ■

A more natural question about volleyball or racquetball is: Given the probability of winning a rally, what's the probability of winning a game? (Both games are of the first-to-21 type.) We return to this question later. But there are already interesting special cases you can answer [17].

## EXAMPLE 5

### Gambler's Ruin

The general situation is this. Player *A* starts with \$ $m$  and player *B* with \$ $n$ . They repeatedly play some game where player *A* has probability  $p$  of winning each time. At the end of each game, the loser pays the winner \$1. Play continues until one of the players is broke (ruined). The question is: What is the probability that *A* will go broke?

**Solution** Your first thought might be to approach this question directly, counting the number of ways *A* could go broke and multiplying each by its probability. By a “way” we mean a sequence of games, for instance, *BBABB*, meaning five games were played and *B* won all but game 3. If *A* starts with \$3 and *B* with \$5, as in Example 5, Section 6.1, then *BBABB* is one way for *A* to go broke. Its probability is  $pq^4$ .

Of course, there are infinitely many ways to go broke because sequences can go on arbitrarily long, with  $A$  and  $B$  trading wins back and forth before  $B$  pulls ahead. So we would have to break this approach into pieces: For each  $k$ , find the number of sequences of length  $k$  which result in  $A$ 's ruin. A necessary condition on such a  $k$ -sequence is that  $A$  lose  $m$  more times than  $B$  does. Furthermore, it is not hard to count the number of such sequences using combinations [21]. However, this does not help us. Again considering Example 5, Section 6.1, look at the sequence of length 13 in which  $A$  wins the first 5 and loses the remaining 8. This does not result in  $A$ 's ruin, because  $B$  is ruined after the fifth game and play stops.

In other words, the sequences we want to count must not only have a certain overall frequency of losses by  $A$ , but also the losses must not bunch up towards the end. Whenever there are such internal restrictions, counting sequences or probabilities by direct methods is usually very hard. With experience, you would know right away to try recursion.

So, let  $p_{m,n}$  be the probability that  $A$  goes broke, given that  $A$  starts with  $\$m$  and  $B$  with  $\$n$ , and let  $p$  be the probability that  $A$  wins any particular game. With probability  $p$ , after one game the distribution of wealth is  $(m+1, n-1)$ ; otherwise it is  $(m-1, n+1)$ . Thus we already have our recurrence:

$$p_{m,n} = p p_{m+1,n-1} + q p_{m-1,n+1}. \quad (5)$$

The initial conditions are easy because generalizing the problem to all starting amounts allows us to consider the degenerate case when one player starts with 0: If  $A$  starts with 0, she's already ruined; if  $B$  starts with 0, he's already ruined, so  $A$  will never be. In other words,

$$p_{m,0} = 0, \quad m > 0, \quad \text{and} \quad p_{0,n} = 1, \quad n > 0. \quad (6)$$

Unfortunately, we have a recurrence with two subscripts. Although we have treated some **partial difference equations** before, we have done so by guessing. We haven't introduced any general methods. (There are some but they're difficult.) If we can reduce the two subscripts to a single index, we'll be much better off.

We can! Fix the total amount of money in the game at  $m+n$ ; then the second subscript is unnecessary — it is always  $m+n$  minus the first subscript. Therefore our problem becomes

$$p_m = p p_{m+1} + q p_{m-1}, \quad p_0 = 1, \quad p_{m+n} = 0. \quad (7)$$

(This recurrence is just Eq. (5) with the second subscripts removed. The initial conditions correspond, respectively, to  $A$  having none of the  $m+n$  dollars and  $A$  having them all.) Eq. (7) is easy to solve by the methods of Sections 5.5. Rewrite the recurrence as

$$p p_{m+1} - p_m + q p_{m-1} = 0.$$

The characteristic equation is

$$p r^2 - r + q = 0.$$

(Remember:  $p$  and  $q = 1-p$  are constants;  $r$  is the unknown.) By the quadratic formula, the roots are

$$\frac{1 \pm \sqrt{1 - 4p(1-p)}}{2p} = \frac{1 \pm (1 - 2p)}{2p} = \left(\frac{1-p}{p}, 1\right) = \left(\frac{q}{p}, 1\right).$$

Thus the general solution is

$$p_m = C(q/p)^m + D,$$

except when  $q = p$ , which gives a double root. (This important special case of “fair-per-game Gambler’s Ruin” is left to the problems; see [22, 25, 26].) To find  $C$  and  $D$ , solve from the boundary conditions:

$$C + D = 1, \quad C(q/p)^{m+n} + D = 0.$$

We find

$$C = \frac{1}{1 - (q/p)^{m+n}}, \quad D = 1 - C.$$

For convenience in writing, set

$$r_* = (q/p).$$

Then, after a little more algebra, we have

$$p_m = \frac{r_*^m - r_*^{m+n}}{1 - r_*^{m+n}} = \frac{r_*^m (1 - r_*^n)}{1 - r_*^{m+n}}. \quad (8)$$

We are done. We have the probability of  $A$ ’s ruin in explicit form, and it’s even fairly simple.

Of course, we are not really done until we interpret what this formula means. For instance, how do the sizes of the initial “pots”,  $m$  and  $n$ , affect the outcome? Such questions are usually answered by considering what happens as  $m$  or  $n$  gets large. These quantities appear as powers in Eq. (8), and thus that equation is easiest to interpret when  $r_* < 1$ , because

$$\text{if } 0 < r_* < 1, \text{ then } r_*^t \rightarrow 0 \text{ as } t \rightarrow \infty,$$

even if  $r_*$  is just slightly less than 1. So, to make  $r_* < 1$ , consider the case  $p > q$ . That is,  $A$  has the advantage play by play. If  $m$  is large, it follows that Eq. (8) is essentially  $(0(1-r_*^n))/(1-0) = 0$ . In other words, player  $A$  has very little chance of being ruined. This conclusion is true even if  $n$  is much larger than  $m$  and if  $p$  is only slightly greater than 1/2. For instance, if  $p = .51$  and  $m = 100$ ,  $A$ ’s chance of ruin is 1.8% (to one decimal place), whether  $B$  has \$200 or \$200 million [23]. If  $m = 200$ ,  $A$ ’s chance of ruin is merely .03%.

What if  $n$  is large but  $m$  is small? Then Eq. (8) is essentially  $(r_*^m \times 1)/(1-0) = r_*^m$ . In other words,  $A$ ’s chance of ruin goes down geometrically as  $m$  increases.

Now, what if  $A$  has the disadvantage play by play? There are two ways to study this situation. One is to rewrite Eq. (8) in terms of  $\bar{r} = 1/r_*$  (since now  $\bar{r} < 1$ ), and consider large  $m$  and  $n$  again [24]. However, it is simpler to use symmetry. To find out what happens when a player has the disadvantage, simply restate the previous results from  $B$ ’s point of view.

The result is: Any player with even a slight disadvantage play by play is in big trouble. For instance, suppose  $B$  has probability .49 (almost even) of winning

any given play. Then if  $A$  starts with a mere \$100, the probability that  $B$  will be ruined is *at least* 98.2% no matter how much money  $B$  starts with. And if  $A$  starts with \$200,  $B$  will go broke at least 99.97% of the time!

This analysis should tell you something about betting against the house on casino trips. Set  $B = \text{you}$ ,  $A = \text{the casino}$ , and consider that  $A$  starts with a lot of money and  $p$ , the probability that the house wins, is always greater than .5 (though only slightly, so as to entice people who haven't read our book). ■

## EXAMPLE 6

### First-to- $N$ Games

Suppose  $A$  and  $B$  play a game where  $A$  has probability  $p$  of winning the next point, whatever has happened so far. Suppose the winner of the game is the player who gets  $N$  points first. What is the probability that  $A$  wins the game?

There are many games of this sort. For instance, except for win-by-2 rules, in tennis the winner of a set is the first to 6 games; in racquetball, the first to 21 points; and in squash, the first to 9 points. To keep things simple, this example does not include a win-by-2 rule, so it provides only a first pass at a model for these games. Also, in some of these games the probability of winning a point is heavily dependent on who serves; see [36, 37] for further discussion of that case.

**Solution** To solve this problem we need only generalize the analysis of Example 3. The probability that  $A$  gets to  $N$  when  $B$  has  $i < N$  points is

$$\binom{N-1+i}{i} p^N q^i$$

so the probability that  $A$  wins is

$$\sum_{i=0}^{N-1} \binom{N-1+i}{i} p^N q^i. \tag{9}$$

However, expression (9) can be unwieldy to evaluate. Is there an easier equivalent, either in the form of a more efficient algorithmic evaluation or another formula?

The answer is Yes to both. We will show you a recurrence for the probability of winning the game (this time with two indices that cannot be reduced to one). Like other recurrences, this one leads to an algorithm. You can then compute with the algorithm, look at the output, and conjecture a simple formula. In fact, this is the way the authors (re)discovered a simpler formula for this problem. You can then use mathematical induction to prove the formula. Or, you can look for a simpler, entirely different combinatorial argument to justify the formula. Such a combinatorial argument happens to exist — it almost always does if the formula is simple enough. Progress would be faster if we could hit upon combinatorial arguments in the first place (before trying recursion), but life rarely works that way.

On to a recurrence! If  $A$  wins the first point, then thereafter she has to win  $N - 1$  points, whereas  $B$  has to win  $N$ ; if  $B$  wins the first point, it's the reverse. So to create a recurrence, we have to consider games in which one player needs to win more points than the other. Therefore we generalize the problem to ask: What is

$p_{m,n}$ , the probability that  $A$  gets  $m$  points *before*  $B$  gets  $n$  points? The recurrence is then

$$p_{m,n} = p p_{m-1,n} + q p_{m,n-1}, \quad (10)$$

with the first term being the probability that  $A$  wins after winning the first point and the second term being the probability that  $A$  wins after  $B$  wins the first point. The initial conditions are

$$p_{m,0} = 0, \quad m > 0; \quad \text{and} \quad p_{0,n} = 1, \quad n > 0. \quad (11)$$

(Why?) Anyway, algorithmically speaking, the problem is done. The answer to the original question is  $p_{N,N}$ , and you can write a program [32] to compute this for any initial  $p$  and  $N$ . However, much more can be done; see [33–35]. ■

## Problems: Section 6.9

---

1.  $\langle 2 \rangle$  Two people alternate flipping a fair coin until it comes up heads. Player 1 goes first, and the person who gets the head is the winner. What is the probability that player 1 wins?
2.  $\langle 2 \rangle$  Solve Example 1 directly by summing an infinite geometric series.
3.  $\langle 2 \rangle$  Modify Example 1 by letting the second player win if he tosses either a 6 or a 5. Again find the probabilities  $p_1$  and  $p_2$ 
  - a) directly via infinite sums,
  - b) via recurrence.
4.  $\langle 2 \rangle$  If a problem can be solved by a recurrence involving a single unknown instead of a sequence, e.g., Eq. (1), then the underlying process must have the possibility of going on forever. Explain why. For instance, in Example 1, it is logically possible that a 6 might never show up (although the probability is 0).
5.  $\langle 2 \rangle$  Three players rotate tossing a fair die until one of them wins by throwing a 6. (“Rotate” means the order in which they toss is always the same: player 1, player 2, player 3.) Use recursion to determine the probabilities of a win by each of the players.
6.  $\langle 2 \rangle$  Change [5] so that, after each toss, which of the other two players tosses next is chosen at random. Determine the winning probabilities.
7.  $\langle 2 \rangle$  For Example 2, here is another recursive approach to finding  $E_1$ , the expected number of tosses if player 1 wins. If she wins, she either wins in one toss, or each player tosses once and then it is as if the game starts over. Use Theorem 1, Section 6.8, to obtain an equation for  $E_1$ . *Caution:* When using this theorem, you should weight the event that player 1 wins on the first toss by  $11/36$ , not  $1/6$ . Why?
  - a) using the method presented in Example 2,
  - b) using the method of [7].
8.  $\langle 2 \rangle$  For the game with three players in [5], determine  $E_i$ , the expected number of tosses if player  $i$  wins
  - a) using the method presented in Example 2,
  - b) using the method of [7].
9.  $\langle 2 \rangle$  Players 1 and 2 alternate flipping a fair coin until heads comes up twice in a row. The player who flips the second of these heads is the winner. Determine the probability that player 1 wins. What if the coin is not fair?
10.  $\langle 1 \rangle$  In Example 3 (tennis), if  $A$  wins 4–2 there were 6 points. So why isn’t the probability of this sort of win  $C(6,2)p^4q^2$ ?
11.  $\langle 2 \rangle$  The scoring system of a game is “good” if it magnifies the difference in ability between players. That is, if player  $A$  is only slightly better than player  $B$  on a per-point basis, player  $A$  should nonetheless have a strong likelihood of winning the game.
  - a) Show that tennis (Example 3) is good by making a table of values of  $p$  and  $p_G$  from Eq. (4).
  - b) Continuing with tennis, also verify that

$$p=.5 \implies p_G=.5.$$

(Since tennis treats the players symmetrically, if  $p_G$  did not turn out to be .5, we would rightly suspect that we made a mistake in our analysis.)

You might also like to make point/game probability tables for other games discussed in this section. As for tennis, it should be no surprise that the goodness is magnified further once one goes on to consider sets and matches.

- 12.** (2) In most women's tennis tournaments, a match consists of two out of three sets. That is, as soon as one player wins two sets, the match is over. If player  $A$  has probability  $p$  of winning any given set against  $B$ , what is the probability that  $A$  wins the match?

- 13.** (2) In some men's tennis tournaments, a match consists of three out of five sets. If player  $A$  has probability  $p$  of winning any given set against  $B$ , what is the probability that  $A$  wins the match?

- 14.** (2) In the old days, to win a set in tennis a player had to reach six games first, except that one always had to win by two games. (Now, when the game score is 6–6, there is usually a tie-breaker.) Assume that  $A$  has probability  $p$  of winning any given game against  $B$ . Under the old rules, what is the probability that  $A$  wins the set?

- 15.** (3) Players  $A$  and  $B$  reach a 3–3 tie in a tennis game ("game" is defined in Example 3).

- a) Find the expected number of additional points to end the game.
- b) Find the expected number of additional points to end the game with  $A$  the winner.
- c) Answer b) again with  $B$  the winner.

- 16.** (3) For Example 3 (tennis) compute the expected number of points per game.

- 17.** (2) Players  $A$  and  $B$  play racquetball. Player  $A$  has probability  $p$  of winning any given rally. What is the probability that  $A$  shuts out player  $B$ ? This means  $A$  gets 21 points and  $B$  gets 0. Recall that a player wins a point only by winning a rally while serving. Who serves first is chosen at random.

- 18.** (2) In Example 4 (racquetball), assume that player  $A$  is about to serve.

- a) What is the expected number of rallies for someone to win a point?
- b) What is the expected number of rallies for  $A$  to win the first point?

- c) What is the expected number of rallies for  $B$  to win the first point?

- 19.** (2) Modify Example 4 by assuming that player  $A$  has probability  $p$  of winning a rally if  $A$  serves and probability  $p'$  of winning if player  $B$  serves. Determine the probability that  $A$  wins the next point if:

- a)  $A$  has the serve,
- b)  $B$  has the serve.

- 20.** (2) Do [18] again under the revised probability assumptions of [19].

- 21.** (2) Assuming that players  $A$  and  $B$  play  $k$  games, what is the number of ways  $B$  could win  $m$  more games than  $A$  does? What is the probability that  $B$  wins  $m$  more games than  $A$ , if  $A$  has probability  $p$  of winning any given game?

- 22.** (2) Solve Gambler's Ruin when  $p = .5$ . In particular, answer the question in Example 5, Section 6.1 ( $m = 3$ ,  $n = 5$ ).

- 23.** (2) Use Eq. (8) to show that if  $B$  has  $n > n_0$  dollars and  $p > q$ , then  $A$ 's chance of ruin is between

$$r_*^m - r_*^{m+n_0} \quad \text{and} \quad \frac{r_*^m}{1 - r_*^{m+n_0}}.$$

Use this range to verify the claim in the text that "if  $p = .51$  and  $m = 100$ ,  $A$ 's chance of ruin is 1.8% (to one decimal place) whether  $B$  has \$200 or \$200 million".

- 24.** (3) Analyze Gambler's Ruin in the case  $p < q$  by rewriting Eq. (8) with  $\bar{r} = 1/r_* = p/q$ .

- a) Show that Eq. (8) becomes

$$P_m = \frac{1 - \bar{r}^n}{1 - \bar{r}^{m+n}}. \quad (12)$$

- b) What does Eq. (12) say if  $m$  is large but  $n$  is not?

- c) What does Eq. (12) say if  $n$  is large?

- d) Use Eq. (12) to show that if  $B$  has  $n \geq n_0$  dollars, then player  $A$ 's chance of ruin is at least  $1 - \bar{r}^{n_0}$ . Hence show that if  $p = .49$  and  $B$  starts with a mere \$100, then the probability that  $A$  will be ruined is *at least* 98.2%, no matter how much she starts with.

- e) Suppose that player  $A$  has .499 probability of winning an individual game. What is the least amount of money that player  $B$  needs so that  $A$ 's chance of ruin is at least .99, no matter how much money  $A$  starts with?

25. ⟨2⟩ In Gambler's Ruin, suppose one player has the edge per game, even just a slight edge. The analysis in the text (and also in [23–24]) shows that there is some fixed amount of money which protects that player against the other, no matter how big the other's pot, in the sense that the first player has, say, at most a 1% chance of ruin. Is the same true when each game is 50–50?
26. ⟨3⟩ In Gambler's Ruin with  $p = q$ , what is the expected number of games until somebody is ruined if player A starts with \$ $m$  and player B starts with \$ $n$ ?
27. ⟨3⟩ In Gambler's Ruin with  $p \neq q$ , what is the expected number of games until somebody is ruined? As usual, player A starts with \$ $m$  and B starts with \$ $n$ .
28. ⟨2⟩ Show that winning by 2 (e.g., winning a tennis game after a 3–3 tie) is a special case of Gambler's Ruin. Reconcile the formulas we obtained for  $p^*$  in Example 3 with Eq. (8) in Example 5.
29. ⟨3⟩ In Gambler's Ruin, suppose both players start with \$ $n$ .
- a) Show from one of the formulas in this section that the probability player 1 gets ruined is
- $$\frac{q^n}{p^n + q^n}.$$
- b) A result this simple deserves a simple, direct derivation. With hindsight, can you find one? That is, don't use recursion or difference equations; argue from first principles. If you don't see a general argument, try to give an argument in the case  $n = 2$ .
30. ⟨2⟩ Solve Gambler's Ruin in the case both players start with \$2. That is, relate  $P_{2,2}$  to other  $P_{i,j}$  just as before, but now there are so few equations that you can solve them directly. Verify that your answer agrees with the answer we got in Example 5.
31. ⟨3⟩ In Gambler's Ruin, let  $N_{k,m,n}$  be the number of sequences of exactly  $k$  games that begin with player A having \$ $m$  and player B having \$ $n$ , and end with A going broke.
- a) Write a recurrence and boundary conditions for  $N_{k,m,n}$ .
  - b) Check your recurrence by using it to compute  $N_{5,3,5}$ , a value that is easy to determine directly.
32. ⟨3⟩ Solve the first-to- $N$  game by writing a program to compute  $p_{m,n}$  using Eqs. (10) and (11). An iterative algorithm is much wiser than a recursive one. (Why?) Run the program for squash ( $N=9$ ), for  $p = .51, .55$  and  $.60$ .
33. ⟨3⟩ This problem is about a simpler formula than Eq. (9) for the probability that player A gets  $N$  points first.
- a) Start by picking the simplest interesting value for  $p$ , namely,  $1/2$ , and compute some values of  $p_{m,n}$  using Eqs. (10) and (11). You should recognize a pattern. The formula we have in mind still involves a sum, but it is more familiar than Eq. (9) and easy to approximate using the continuous normal distribution. *Hint:* Except for the boundary conditions and the division by 2, Eq. (10) with  $p = 1/2$  is the recurrence for combinations with repetitions, the table of which contains Pascal's triangle along the diagonals. See Example 5, Section 5.3.
  - b) Now guess a formula for arbitrary  $p$ .
  - c) Prove your general formula by double induction.
34. ⟨3⟩ Here is a combinatorial argument for the formula we had in mind in [33]. Consider the following two games,  $G$  and  $G'$ . In  $G$ , player A wins if she gets  $m$  points before player B gets  $n$  points. In  $G'$ ,  $m+n-1$  points are played no matter what, and at the end A is declared the winner if she got  $m$  or more; otherwise B is the winner. Assume that A has probability  $p$  of winning any given point.
- a) Show that A has the same probability of winning  $G$  as  $G'$ . Show this with a many-to-one correspondence between possible scorecards for  $G'$  and scorecards for  $G$ . By a scorecard we mean a listing  $AABABBA\dots$  that indicates who won each point.
  - b) It's easy to write a binomial sum for the probability A wins  $G'$ . Do it. By part a), this is also a formula for  $G$ .
35. ⟨3⟩ Many first-to- $N$  games have an additional win-by-2 rule. You already know how to compute the probability of a win by player A, given that

the game has reached a stage at which the latter rule is invoked.

- a) Modify a single case of the recurrence in Eq. (10) so that the values  $p_{m,n}$  are correct for such games.
- b) Modify the argument and the summation in [34] to accomplish the same.

36. ⟨2⟩ The first-to- $N$  model developed so far is inappropriate for *sets* of tennis, because serving makes a tremendous difference in tennis, and the server changes with each game. (The model *is* appropriate for *games* of tennis, for the same person serves the entire game.)

A more appropriate model for tennis sets is: Player  $A$  has probability  $p$  of winning a game when she serves and  $p'$  when her opponent serves.

Assume that a set of tennis is a first-to-6 game without a win-by-2 or a tie-breaker rule.

- a) Revise the recursive method of Eq. (10), assuming that  $A$  serves first.
- b) Revise it again, assuming that  $B$  serves first.
- c) The first serve is determined at random. In terms of the probabilities determined in parts a) and b), what is the probability that  $A$  wins the set?

37. ⟨3⟩ The model developed in [32–35] is also inappropriate in any game in which a player must have the serve to win a point — even if the serve itself has no effect on the probability of winning a rally. The reason is that the probability of winning the next point (as opposed to the next rally) is not independent of who won the previous point.

- a) Illustrate this dependence. Example 4 should help.
- b) Explain how both the recursive and the direct approaches we have taken to our previous model assume independence.
- c) Modify the recurrence Eq. (9), which is about winning points, to account for this dependence. Assume as the basic building block that player  $A$  has probability  $p$  of winning any given rally and that rallies are independent. *Hint:* You will now need three indices — the number of points  $A$  must win, the number  $B$  must win, and a 0–1 index indicating who serves first (0 means  $A$  serves first). We do not know any combinatorial argument, or similarly direct approach, for treating this revised model.

38. ⟨2⟩ A drunkard leaves an inn located at position  $m$  on a straight line between the police station, at 0, and his home, at  $n$ . Each minute, he randomly staggers either one unit towards home or one unit towards the police station. Determine the probability he gets home first.

39. ⟨3⟩ Consider another drunkard who does his drinking at home. He staggers out his front door and, each second, randomly goes one unit to the left (say, west) or one unit to the right (east).

- a) Show that he returns home at some point with probability 1.
- b) Determine the expected number of seconds until he first returns home.

40. ⟨3⟩ Take the model of a tennis game in Example 3 and corroborate the results by a simulation. That is, write the scoring rules into a computer program that “plays” a tennis game with player  $A$  having some probability  $p$  of winning any given point. Then run the program many times and compute the fraction of games that  $A$  won. (See below for exactly how many times to run it.) Compare this to the probability of winning computed in Example 3. You wouldn’t expect the two numbers to be identical, but they should be close. Based on Section 6.7 you can even say how close.

- a) Show that, if you run the game 100 times, with 95% certainty your fraction of wins should differ from the answer in Example 3 by at most .1.
- b) Show that, if you run your simulation 1000 times, your fraction should differ from the theoretical result by at most .03, also with confidence level 95%.
- c) Now run one simulation 100 times, and a separate one 1000 times. Were you accurate to the bounds described in a) and b)?
- d) What is the probability that both your simulated probabilities were accurate to the given bounds? That at least one of them was?

41. ⟨3⟩ We attempted to make our models more and more realistic as we went along in this problem set, but surely you can think of ways in which we could go further. However, we have gone about as far as we think we should by theoretical methods. Make up a more detailed model for one of the examples which interests you, and use simulation to estimate the probability of winning the game.

## Supplementary Problems: Chapter 6

1. ⟨2⟩ (Philosophy question) Consider Example 2, Section 6.1, but forget about the tine test. Just consider the question, “What is the probability I have tuberculosis?” One argument goes: The answer is 0 or 1; either I have it or I don’t. Claims that the probability should be some other number, say .001, are nonsense, because my tuberculosis status is not a repeatable experiment. It’s a fact.

Argue against this claim. Come up with an interpretation of the statement “I have tuberculosis with probability .001” which allows it to have a ratio interpretation.

2. ⟨3⟩ Two fair dice are tossed repeatedly.

- a) Let  $\Pr(m, n)$  be the probability that the first 2 or 11 appears on toss  $m$  and the first 7 appears on toss  $n$ . Find a formula for  $\Pr(m, n)$ . *Hint:* Consider three cases,  $m > n$ ,  $m = n$ , and  $m < n$ .

Let  $f(k)$  be the probability that the first 7 shows up on the  $k$ th toss, without either 2 or 11 appearing earlier.

- b) Find a formula for  $f(k)$ .

- c) Why is  $f(k)$  not a point distribution? This is a conceptual question; it has nothing to do with the form of the answer to part b).

3. ⟨2⟩ A fair coin is tossed four times.

- a) What is the probability that the third toss is a head?
- b) What is the probability that the third toss is the last head?
- c) What is the probability that the third toss is a head, given that exactly two of the tosses are heads?
- d) What is the probability that the third toss is the last head, given that exactly two of the tosses are heads?

4. ⟨2⟩ In analyzing a game of tennis (Example 3, Section 6.9), we noted that the win-by-2 rule goes into effect only if the game reaches 3–3. That’s the way tennis players think of it (3–3 is the first score called “deuce”), but in fact the win-by-2 rule goes into effect at 2–2 as well, because you can’t win 4–3.

Once again, suppose player  $A$  beats player  $B$  on a given point with probability  $p$  and let  $p_G$  be the probability  $A$  wins the game. Use the observation about win-by-2 at 2–2 to come up with a different-looking formula for  $p_G$  than in Example 3, Section 6.9. Verify that these two formulas are equal.

5. ⟨3⟩ Suppose that the rules of a tiebreaker are that the winner is the first player to win 2 points in a row.

- a) Show by example that this is different from winning by 2 points, as discussed in Section 6.9.
- b) If the first player has probability  $p$  of winning a point, find the probability that she wins the tiebreaker. *Hint:* Write down an infinite sequence which represents all possible winning sequences for player 1.
- c) Answer part b) again, recursively. We do not see how to do this directly, but it can be done if you consider two cases and use co-recursion. Namely, let  $P_p$  be the probability that player 1 wins starting with winning the first point of the tiebreaker. Let  $P_q$  be the probability that player 1 wins starting with losing the first point.
- d) Which tiebreaker rule, win-by-2 or 2-in-a-row, is better in the sense that it more greatly magnifies small differences in players on a point-by-point basis? See the discussion in [11, Section 6.9].
6. ⟨3⟩ The negative binomial distribution is often useful with the roles of success and failure exchanged. We ask for the probability  $f_n(k)$  that the  $n$ th failure happens on the  $k$ th trial. For instance, if each trial consists of seeing whether a light bulb lasts for another day, then to say that the first failure occurs on day  $k$  is to say that the light bulb lasts  $k$  days.

The negative binomial, viewed as a model of failure probabilities, has an interesting property: The chance of future failure is independent of the past record. We illustrate this broad claim with a specific instance. Let  $X$  be the trial in which the first failure occurs. Consider the statement

$$\Pr(X=k|X>k_0) = \Pr(X=k-k_0). \quad (1)$$

In what sense does this statement say that the distribution of  $X$  forgets its past? Prove that the negative binomial (with  $n = 1$ ) satisfies Eq. (1).

7.  $\langle 3 \rangle$  Consider a lottery where six numbers are chosen randomly from  $\{1, 2, \dots, M\}$ . What is the largest value of  $M$  so that it is more likely than not that the chosen set has two consecutive hbox-integers?

You may have noticed this consecutive-number phenomenon in the winning 6-tuples of real state lotteries. The numbers are chosen without replacement; they are presented in sorted order (i.e., order is irrelevant).

8.  $\langle 4 \rangle$  In [18, 20, Section 6.5], we introduced utility as a preferred measure to money. We also followed the assumption about utility usually made by economists: The more money you have, the fewer utils the next dollar is worth. (Another way to say this is: The utility function, the function  $U(x)$  that tells you how many utils \$ $x$  is worth to you, is concave down.) After all, at some point there is little more you will want to buy—how much caviar can you eat?

However, you can argue that this assumption is not always correct. Sure, you get tired of little things like caviar after awhile, but maybe what you really want is the sense of power one gets from owning Microsoft, or the instant fame from winning a million-dollar state lottery. In other words, the utility of more money may be small until you get to very large amounts of money, at which point the utility may suddenly become large. We can summarize this situation by saying the utility function is concave up.

A third possibility is that your utility function is linear:  $U(x) = ax + b$ , with  $a > 0$ .

- a) Show that if you have a linear utility function, you can work with money and forget utility. That is, you will make the same decisions by choosing the actions which maximize your expected monetary return as you will if you maximize your expected util return.
- b) In the solution to Example 4, Section 6.1 (which appears in Section 6.5), would the answer change if we analyzed the problem using a concave-down utility function in which \$1 extra is worth at most 1 util?

c) Complete Example 4, Section 6.1, using the following concave-up utility function: Additional dollars are worth 1 util up to \$5000 (and losses are worth  $-1$  util each), but gains above \$5000 are worth 10 utils per dollar. Assume that 250,000 people return their tickets and are thus eligible for prizes.

9.  $\langle 3 \rangle$  Review the conditional life probabilities  $p_n$  of [30, Section 6.3]. Using mathematical expressions, and then using algorithmic language, find expressions for

- a) the expected age at which a newborn infant will die;
- b) the expected age at which a person who has just turned 50 will die.

*Note:* For the purposes of such computations, actuaries assume that no one lives beyond 100, i.e.,  $p_{100} = 0$ .

10.  $\langle 3 \rangle$  *Life annuities.* Insurance companies offer a contract where you pay them a lump sum, and then they pay you a fixed amount at regular intervals until you die. (Usually the interval is monthly, but for the purposes of this problem we assume that it is yearly.) The lump sum depends on the amount you want to receive in each payment, and how old you are when you start. Suppose that you want \$5000 a year, that you just turned 50, and that you want your first payment when you turn 51. Using the  $p_n$ 's (see [30, Section 6.3]), what is the expected amount the company will pay you during your lifetime? Answer using formulas and algorithms.

11.  $\langle 3 \rangle$  The answer to [10] is *not* the amount insurance companies charge you for the annuity. They charge less because they can earn money on the part of the lump sum they haven't paid you back yet. Assume that they can earn 10% per year. How much money do they need now to cover the expected amount they will pay you during your lifetime? Answer using formulas and algorithms.

*Note:* The answer to this problem is a good first approximation of what insurance companies will charge, but there is much more to it. For instance, they must charge some extra in order to set up reserves in case they have bad luck — you and the other people they sell annuities to may live longer than you're supposed to! To plan for this

type of situation and others (e.g., there is the little matter of profit), more complicated probability analysis is called for.

Problems 12–15 ask you to find several ways to compute  $E(B_{n,p}^2)$ , the expectation of the square of the binomial distribution.

- 12.** ⟨2⟩ Let  $X_i$  be the characteristic function of success on the  $i$ th trial from a sequence of Bernoulli trials. Then  $B_{n,p} = \sum_i X_i$ , and thus

$$B_{n,p}^2 = \sum_i X_i^2 + \sum_{i \neq j} X_i X_j.$$

Find  $E(B_{n,p}^2)$  from this equality.

- 13.** ⟨2⟩ Find  $E(B_{n,p}^2)$  directly from the definition. Method 1 of Example 3, Section 6.5, might help.

- 14.** ⟨2⟩ Find  $E(B_{n,p}^2)$  by recursion.

- 15.** ⟨2⟩ (Assumes calculus) Find  $E(B_{n,p}^2)$  by differentiating something.

The following five problems are an introduction to “hypothesis testing”, an important subfield of statistics. Contrast this with the “parameter estimation” we did in Section 6.7.

- 16.** ⟨3⟩ You are given a coin and told that, with 50–50 odds, it is either a fair coin or one that has been jimmied so that it comes up heads with probability .7. You flip it 25 times and find that it comes up heads 15 times. What is the probability that it is a fair coin? *Caution:* The answer is not .5.

- 17.** ⟨3⟩ As in [16] you are given a coin and told that it is either fair or has probability .7 of coming up heads. Now, however, you are given no odds concerning which sort you have, so you can’t use Bayes’ Theorem. Nonetheless, you are asked to determine with at least 95% confidence which type you have.

Here’s a way to do it. You will do  $n$  tosses with  $n$  to be determined. If the sample frequency  $\bar{p}$  is  $> .6$ , you will declare that you have the biased coin; otherwise you will declare for the fair coin. The number  $n$  must satisfy two conditions. If the coin is biased (i.e.,  $p = .7$ ), the probability that  $\bar{p} > .6$  must be at least .95; if the coin is fair, the probability that  $\bar{p} \leq .6$  must be at least .95. Since fair and biased at .7 are the only possibilities, your total probability of a successful declaration is at

least .95 in any event. Find the least  $n$  that does the trick.

- 18.** ⟨3⟩ In [17] the decision to let .6 be the division point was actually not the smartest, in that a different point would allow a lower  $n$  to achieve the .95 confidence. Find the best division point and the associated lowest  $n$ .

- 19.** ⟨3⟩ You are given a coin and told that maybe it is fair, maybe it’s not. You are asked to make a judgment.

In contrast to [16–18], we now have infinitely many choices for  $p$ , not just .5 and .7. We cannot optimize for distinguishing among all of them at one time, as we did between .5 and .7 in [18]. So once again we need a different approach.

Fortunately, one of the choices stands out, the “null hypothesis” that the coin is fair. So we would like our test at least to have the property that, if the null hypothesis is true, we are unlikely to reject it. (This is called avoiding **Type I error**. A **Type II error** would be to accept the null hypothesis when it is false.)

If you allowed 100 tosses, name a test that has 95% probability of accepting the null hypothesis if it is true. By a test we mean a condition on  $\bar{p}$ , stated in advance, that will cause you to declare for the null hypothesis. The condition ought to be symmetric around  $p = .5$ , since we are given no reason to suppose that, if the coin is unfair, it is unfair in one direction (heads) or the other (tails).

- 20.** ⟨2⟩ As in [19], you are given a coin that may or may not be fair, but now you are told that if the coin is not fair, then it has a *higher* than 1/2 probability of heads. What do you think is the best test with 100 tosses now? (There is a whole theory of “most powerful tests” that explains how to choose, but we won’t get into that. Here we just want your intuition.)

- 21.** ⟨2⟩ What is the range of values for  $\Pr(A \cap B \cap C)$  if  $\Pr(A) = p$ ,  $\Pr(B) = q$  and  $\Pr(C) = r$ ?

- 22.** ⟨3⟩ A simple card game is played as follows. There are two cards, High and Low. One of the cards is dealt, face down, to the first player, who then gets to look at it. At this point, he can either *fold*, paying the second player \$1 and ending the game, or *play* (by simply announcing “I play”). If he plays, the second player must either *fold*, paying the first player \$1, or *see*. If she sees, then the

first player reveals his card; if it is High, he wins \$2 from the second player; if it is Low, he pays \$2 to the second player.

It turns out that the optimal strategies for this game are the following. For the first player: If he is dealt High, he should always play; if he gets Low, he should fold with probability  $2/3$  and play the rest of the time. (In other words, he should bluff  $1/3$  of the time.) For the second player: In those games where the first player plays, she should fold  $1/3$  the time and see the rest. (How do you determine that these are the best strategies? Take a mathematics course called “Game Theory”.)

Assuming that the card is dealt randomly and both players play optimally, find the probability that, in a given game:

- a) The first player folds.
- b) The second player folds.
- c) A bluff is called.
- d) The first player makes money.
- e) The second player sees, but she would have been better off to fold.
- f) The first player folded, given that you know he lost.
- g) If the first player loses, it’s because his bluff is called.

23. (2) Suppose that players  $A$  and  $B$  play a first-to-21 game in which one must be serving in order to receive a point for winning a rally, and in which the game must be won by 2 points. Player  $A$  has probability  $p$  of winning any given rally, and  $A$  and  $B$  have battled to a 20–20 tie. Let  $p_A^*$  be the probability that  $A$  goes on to win the game if it is now her serve. Let  $p_B^*$  be the probability that  $A$  goes on to win if it is now  $B$ ’s serve. Determine  $p_A^*$  and  $p_B^*$  by “co-recursion”, that is, by establishing and solving two simultaneous equations in these two unknowns.

*Suggestion:* Life will be simpler if you first define  $p_A$  (respectively,  $p_B$ ) as the probability  $A$  wins a point given that  $A$  (respectively,  $B$ ) now has the serve.

24. (3) Refer to Example 5, Section 6.1, and suppose you are told that player  $A$  got ruined. With this additional information, what is the probability that  $A$  lost the first game?

25. (3) Suppose that we redefine the rules of Gambler’s Ruin as follows. In each round, the loser

gives \$1 to a referee, instead of directly to the winner. When someone goes broke, the referee gives everything he has collected to the remaining player. Write a recurrence and boundary conditions for this modified game. What is the solution?

26. (3) A bug moves from vertex to vertex of square  $ABCD$  at random. That is, whatever vertex it is currently at, in one step it moves with equal probability to either adjacent vertex. Assume that it starts at  $A$ .
- a) What is the probability that the bug reaches  $C$  before it visits  $B$ ?
  - b) What is the expected number of steps for it to reach  $C$  (whether or not it passes through  $B$ )?
  - c) What is the expected number of steps to reach  $B$ ?
27. (3) The eight vertices of a cube are at the points  $(a_1, a_2, a_3)$ , where each  $a_i$  is either 0 or 1. A bug moves at random from vertex to adjacent vertex. Assume it starts at  $(0,0,0)$ . For each other vertex, what is the expected number of steps for the bug to reach that vertex for the first time?
28. (4) Consider Algorithm BINSEARCH from Example 5, Section 1.5. We will now determine  $A_n$ , the average number of comparisons when there are  $n$  words on the list and the search word is known to be on the list. Consider the binary tree of possible searches through the list. That is, the root is the word the algorithm checks first. The left child is the word on the list that the algorithm checks if the root word was after the search word in the alphabet, etc. For the purposes of this problem, we say that the root is at level 1, the children of the root are at level 2, and so on. If the search word  $w$  is on the list, the number of comparisons done to find  $w$  is the level of  $w$  in the tree. Therefore, to determine the average amount of work, you merely have to evaluate  $\sum_{i=1}^k i n_i$ , where  $n_i$  is the number of words at level  $i$ , and then divide by the total number of words,  $n$ .
- a) Show that the tree has  $k = \lceil \log_2(n+1) \rceil$  levels, and that each level except perhaps level  $k$  is complete.
  - b) Compute  $A_n$  when  $n = 2^k - 1$ , that is, when the binary tree is complete. You can get an answer in terms of  $k$  or  $n$ .
  - c) Compute  $A_n$  again in general. You may leave your answer in terms of both  $k$  and  $n$ .

# An Introduction to Mathematical Logic

## 7.1 Introduction and Terminology

---

Logic, the systematic study of reasoning, is one of the oldest branches of mathematics. For example, the notion of a *syllogism* was important to the ancient Greeks, and the phrase “Aristotelian logic” is still commonly used. But the flowering of logic in modern mathematics did not begin until the mid-nineteenth century, particularly with the work of George Boole (1815–1864). In the twentieth century, particularly with the growing importance of computers, logic became one of the most important and useful branches of modern mathematics.

In Section 0.6 we introduced various terminology and notation of mathematical logic. In particular in that section, we discussed at some length *implication* and *quantification*, two topics that will also play a major role in this chapter. (You might want to review the relevant portions of Section 0.6 now, particularly if Chapter 0 seems like something from the distant past.) In Section 0.6 we also introduced various ideas related to theorems and their proofs. One purpose of this chapter is to recapitulate somewhat more formally what we’ve said earlier about proofs in order to stress the importance of logical thinking in mathematics. Another purpose is to introduce you to a variety of areas of mathematical logic which are finding increasing application in various disciplines, especially computer science. In particular, since the study of algorithms has been an important theme throughout this book, we want to show you how to use logic in the formal study of algorithms as mathematical objects. And last — but certainly not least — we want to provide you with some tools for organizing your thoughts about mathematics and about reasoning generally.

We’ll begin by introducing the **propositional calculus**. This is the first layer of an algebra for logical thinking, and is useful in everyday life as well as mathematics. We’ll then use the propositional calculus to discuss some formal aspects of mathematical proofs and to introduce the notion of **algorithm verification**.

Then we'll discuss **Boolean algebra** — an adaptation of the propositional calculus — which has many applications, a noteworthy one being in the design of digital circuits. Next we'll introduce some of the basic ideas of the **predicate calculus** — an extension of the propositional calculus in which quantification plays a major role. Then we'll use the predicate calculus to discuss some further aspects of algorithm verification. Finally, we'll introduce briefly the notion of a **formal deductive system** in the contexts of both the propositional and the predicate calculus.

Let's begin as usual with some examples illustrating the variety of problems to which the subject of this chapter can be applied.

## EXAMPLE 1

### Logic and Ordinary Language

Consider the following *argument* (that is, deductive reasoning) in English:

If it is cloudy in the morning, then it is going to rain in the afternoon.

If it is going to rain in the afternoon, I should take my raincoat with me.

Therefore, if it is cloudy in the morning, I should take my raincoat with me.

Can the methods of mathematical logic help us evaluate the merits of this argument? Yes. First, it involves various propositions, which we may represent by letters:

$P$ : It is cloudy in the morning.

$Q$ : It is going to rain in the afternoon.

$R$ : I should take my raincoat with me.

Using these letters we may formalize the argument as

$$P \implies Q$$

$$Q \implies R$$

$$\frac{}{P \implies R},$$

where  $\implies$  should be read as “implies” (cf. Section 0.6). The two propositions above the dashed line are the **premises** (or **hypotheses**) and the one below the dotted line is the **conclusion**. Do you find this argument convincing? We hope you do, and we hope you find it just as convincing in the symbolic form, for logic is about determining what arguments are correct even when we strip away the specific information. If the argument is correct, it should be correct with any propositions for  $P, Q, R$ . For instance, suppose we changed “cloudy” to “sunny” everywhere in the preceding argument; in other words, we change  $P$ . How would you feel about the argument then? Both the first premise and the conclusion are probably false, but would the conclusion be true if the premises were true? That is, is the

argument *valid*? Determining whether an argument is valid is one of the main tasks of mathematical logic.

Now consider the following.

If astrology is a true science, then the economy is improving.

The economy is improving.

Therefore, astrology is a true science.

Again, we may represent propositions by letters:

$P$ : Astrology is a true science.

$Q$ : The economy is improving.

The argument may be represented as

$$P \implies Q$$

$$\begin{array}{c} Q \\ \hline \end{array}$$

$$\begin{array}{c} P. \\ \hline \end{array}$$

How do you feel about this argument? Is your opinion any clearer when you look at the symbolic form of the argument, where  $P$  and  $Q$  can be any propositions? Whatever you think now, we will discuss systematic methods for determining the validity of such arguments. ■

## EXAMPLE 2

You are designing a computer and want to build a circuit whose input will be a decimal digit expressed in binary form and whose output will be the **complement** of this digit (where the complement of a decimal digit  $d$  is defined to be  $9 - d$ ). Table 7.1 expresses what you want to accomplish. The circuits available to you and their properties are shown in Fig. 7.1. How should you design the circuit? We'll solve this problem in Section 7.5. ■

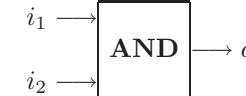
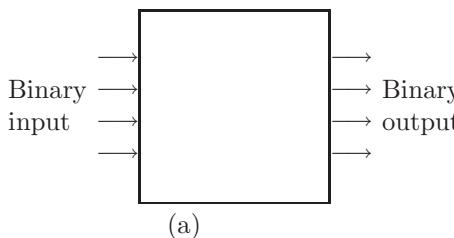
## EXAMPLE 3

Algorithm 7.1 MULT purports to multiply a positive integer by another integer. Here is an instance of how it works. Suppose  $x = 4$  and  $y = 11$ . Then the values of *prod* and *u* each time we enter the repeat-loop are:

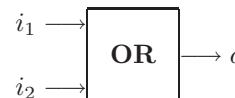
| <i>prod</i> | <i>u</i> |
|-------------|----------|
| 0           | 0        |
| 11          | 1        |
| 22          | 2        |
| 33          | 3        |
| 44          | 4        |

**TABLE 7.1**  
Complementing a Decimal Digit

| Input Digit |        | Output Digit |         |
|-------------|--------|--------------|---------|
| Decimal     | Binary | Binary       | Decimal |
| 0           | 0000   | 1001         | 9       |
| 1           | 0001   | 1000         | 8       |
| 2           | 0010   | 0111         | 7       |
| 3           | 0011   | 0110         | 6       |
| 4           | 0100   | 0101         | 5       |
| 5           | 0101   | 0100         | 4       |
| 6           | 0110   | 0011         | 3       |
| 7           | 0111   | 0010         | 2       |
| 8           | 1000   | 0001         | 1       |
| 9           | 1001   | 0000         | 0       |



Output is 1 if both inputs are 1 and 0 otherwise.



Output is 1 if either input is 1 and 0 otherwise.



Output is 1 if input is 0 and vice versa.

(b)

**FIGURE 7.1**

- (a) Designing a completer;
- (b) available logic circuits

Do you see why we have required  $x$  to be nonnegative but allowed  $y$  to be any integer [1]?

Of course, the example above is not a proof that Algorithm 7.1 works for all  $y$  and all nonnegative  $x$ . But you should be able to give a persuasive argument that this algorithm is, indeed, correct [2]. Such an argument would be akin to the proofs we have presented at various places earlier in this book for algorithms and which

## Algorithm 7.1

Mult

|                                     |                                   |
|-------------------------------------|-----------------------------------|
| <b>Input</b> $x$                    | [Integer $\geq 0$ ]               |
| <b>Input</b> $y$                    | [Integer]                         |
| <b>Output</b> $prod$                | [ $= xy$ ]                        |
| <b>Algorithm</b> MULT               |                                   |
| $prod \leftarrow 0; u \leftarrow 0$ |                                   |
| <b>repeat until</b> $u = x$         | [When $u = x$ , stop computation] |
| $prod \leftarrow prod + y$          |                                   |
| $u \leftarrow u + 1$                |                                   |
| <b>endrepeat</b>                    |                                   |

use a loop invariant or some other form of induction. But it would probably not be like the more *formal* proof that we shall present in Section 7.4. ■

Much of this chapter is concerned with systematic ways to describe and identify valid arguments. Validity is concerned with whether the conclusions follow from the premises. That is, an argument is **valid** if the conclusion is true whenever the hypotheses are true, even if the individual statements within the argument are changed, as we did in considering different possibilities for  $P$  in Example 1. Thus, validity is *not* about whether the conclusions are true but only about whether the conclusion would follow if the hypotheses are assumed true.

But isn't this emphasis wrong? Aren't we in fact interested in telling truth from falsehood? Of course we are. But there are two aspects to how an argument can lead to false conclusions: the facts can be wrong or the logic can be wrong (or both). Mathematical logic, the subject of this chapter, is concerned only with **deduction**, the process by which we get to conclusions from premises. Once you know that an argument is valid, you had better make sure that the premises are true (often an empirical matter) before you act on the conclusions of the argument. Only if both the premises are true *and* the argument is valid can you be sure that your conclusions are true. If some premise is false or the argument is invalid, you will be in danger of accepting false conclusions (although, by luck, they may turn out to be true [3]).

In all the foregoing examples, we gave only the premises and conclusions of arguments. Normally, however, a valid argument begins with its premises and then has various intermediate steps before reaching the conclusion. In other words, in Example 1 we stated logical “theorems”, but did not give the proof (if an argument is valid) or disproof (if it is not). Much of this chapter is about how to find the reasoning process (i.e., the deduction, or logic proof) which leads from premises to a conclusion.

Our description of a valid argument has a flavor much like that of algorithmic thinking. We begin an algorithm with a specification of the allowable input (in a sense, the premises of the algorithm) and proceed through a series of steps (analogous to a reasoning process) to obtain some desired output (similar to a true

conclusion). Thus the structures of valid arguments and correct algorithms are quite similar. One result of this similarity is that computers can be programmed to produce valid arguments. Another is that valid arguments can be used to explore whether an algorithm is correct. In the remainder of this chapter we shall keep reminding you of this connection between logic and algorithms.

## Problems: Section 7.1

---

1.
  - 1) Explain why Algorithm 7.1 does not work when  $x$  is a negative integer.
2.
  - 2) Give an informal (but mathematical) proof that Algorithm 7.1 correctly multiplies a nonnegative integer  $x$  by any integer  $y$ .
3.
  - 2) In each of the following cases, give reasons why the argument given is valid or not.
    - a) Russia and USA are countries. Therefore Russia is a country.
    - b) If it is sunny, then it must be very hot. It is not very hot. Therefore it is not sunny.
    - c) If Bob lives in Chicago, then he lives in Illinois. If Bob lives in Chicago, then he lives in the USA. Bob lives in Illinois. Therefore Bob lives
- in the USA. (Compare with [10b, Section 7.3].)
- d) All Americans are rich. Bill Gates is an American. Therefore Bill Gates is rich.
4.
  - 2) An argument is **sound** if it is valid and the hypotheses are true. Thus, when an argument is sound we can be sure that the conclusion is true. For each of the following arguments, determine if it is sound. If it is not, why not?
    - a) London is in England. England is in the UK. Therefore London is in the UK.
    - b) London is in Scotland. Scotland is in the UK. Therefore, London is in the UK.
    - c) London is in England. London is also in the UK. Therefore England is in the UK.

## 7.2 The Propositional Calculus

---

As its name implies, this section is about the propositions we introduced in Section 7.1 — how to build up more complicated ones from simpler ones, and how to calculate with them. Hence the use of “calculus”. The propositional calculus is concerned with reasoning about propositions.

As in the previous section, for now we'll use capital letters,  $P, Q, R, \dots$ , to denote propositions. These letters are really **propositional variables** since the domain of  $P$  or  $Q$  etc. may generally be some set of propositions, that is, some set of declarative statements. These statements might be ordinary sentences such as “I am not a mathematician yet but I will be soon” or “The moon is blue” or mathematical statements such as “ $5 + 3 \geq 7$ ”. Because propositions may be just English sentences — as in some of our examples — the propositional calculus is sometimes called the **sentential calculus**.

A proposition may be viewed as an assertion that something is true. Since, in fact, a particular proposition may not be true (i.e., we may have made an incorrect assertion), a proposition is said to have a **truth value**, which is either True (T) or False (F). Some statements such as “Jane is nice” may be a matter of opinion, true to some but false to others. In the remainder of this chapter we'll restrict ourselves to propositions whose truth values are clearly either true or false. And generally

we'll just say that a proposition *is* true (or false), meaning that its truth value is T (or F).

Later in this chapter we will wish to consider expressions of the form

$$x > 2, \quad prod = uy, \quad \text{and} \quad p_n < \frac{r^2}{4}, \quad (1)$$

which are not propositions because their truth or falsity depends upon the values of the variables in them. Such expressions are called **predicates**, a term we first introduced in Section 0.4. Note that when all the variables in expressions like those in (1) are given values, then these expressions are either true or false and thus become propositions. Predicates are the subject of Section 7.6.

## The Algebra of the Propositional Calculus

Propositions that express a single thought and thus cannot be subdivided into two or more simpler propositions are the **atoms** of the propositional calculus. We will now put atoms together to get **molecular** or **compound** propositions.

Consider the propositions

$P$ : Jill is 13,

$Q$ : Andrew is 12,

$R$ : Jill is 13 and Andrew is 12.

$R$  is called the **conjunction** of  $P$  and  $Q$ . For now we shall denote  $R$  by  $P$  **and**  $Q$ .

$P$  **and**  $Q$  is indeed a proposition, in that it is either true or false. Moreover, its truth value depends only on the truth values of  $P$  and  $Q$ , not on the actual propositions themselves. Thus  $P$  **and**  $Q$  is true if and only both  $P$  and  $Q$  are true.  $R$  is thus a compound proposition since it is formed from two atomic propositions,  $P$  and  $Q$ . (Careful, though. Just because there is an “and” in a proposition doesn’t mean that it’s not atomic. For example, “Jill and Andrew are siblings” is an atomic proposition because it expresses a single thought.)

Operators on propositions like **and** that produce other propositions, whose truth value depends only on the truth values of the components, are called **logical operators**. If we restrict attention to propositional variables like  $P, Q, R, \dots$  and logical operators, as the propositional calculus does, then we shall be able to get at the form of correct arguments, because the truth or falsity of our statements does not depend on what the particular propositions  $P, Q, R, \dots$  stand for but only on their truth or falsity.

The things an operator combines are called its **operands**. Thus, in  $P$  **and**  $Q$  the operands are  $P$  and  $Q$  (just as  $a$  and  $b$  are the summands in  $a + b$ ).

The meaning of any logical operator can be completely described by a **truth table** in which there is one column corresponding to each operand and one row for each choice of the truth values of the operands. Then there is one final column on the right that shows the truth value of the compound proposition. The truth table of **and** is shown in Table 7.2a.

**TABLE 7.2**  
**Truth Tables for**  
**(a) “and”, (b) “or”,**  
**and (c) “not”**

| (a) | <u><math>P</math></u> | <u><math>Q</math></u> | <u><math>P</math> and <math>Q</math></u> |
|-----|-----------------------|-----------------------|------------------------------------------|
|     | T                     | T                     | T                                        |
|     | T                     | F                     | F                                        |
|     | F                     | T                     | F                                        |
|     | F                     | F                     | F                                        |

| (b) | <u><math>P</math></u> | <u><math>Q</math></u> | <u><math>P</math> or <math>Q</math></u> |
|-----|-----------------------|-----------------------|-----------------------------------------|
|     | T                     | T                     | T                                       |
|     | T                     | F                     | T                                       |
|     | F                     | T                     | T                                       |
|     | F                     | F                     | F                                       |

| (c) | <u><math>P</math></u> | <u>not <math>P</math></u> |
|-----|-----------------------|---------------------------|
|     | T                     | F                         |
|     | F                     | T                         |

You should be aware that in ordinary English we have a variety of ways of expressing the *thought* “and” without necessarily using the *word* “and”. For example:

Mathematics is hard and computer science is useful.

Mathematics is hard; computer science is useful.

Whereas mathematics is hard, computer science is useful.

Mathematics is hard, but computer science is useful.

Although mathematics is hard, computer science is useful.

Or, using examples containing mathematical notation:

$$x < 0 \text{ and } y > 14.$$

$$x < 0 \text{ whereas } y > 14.$$

$$x < 0 \text{ but } y > 14.$$

$$\text{Although } x < 0, y > 14.$$

In both of the preceding groups, each sentence has the same meaning (i.e., the same truth value), except perhaps for nuance. However, in each group you would probably prefer one (which?) rather than the others in ordinary discourse. In our algorithmic notation, we have allowed more than one way of expressing the same thing (can you think of some examples?) because, when thinking algorithmically,

expressing a thought in one way rather than another is often more conducive to accuracy. But in writing mathematical expressions there must be some very good reason to allow more than one way to express the same thing because precision is so important. Put another way, if you consistently say the same thing the same way in a poem, it would probably be boring, but in mathematics consistency of language is a good way to avoid ambiguity. Therefore we allow only the “and” form in the mathematical expressions above.

### EXAMPLE 1

Use “and” and propositional variables to express the sentence:

I am going to the store; then I am going to the movies.

**Solution** With

$P$ : I am going to the store.

$Q$ : I am going to the movies.

the sentence becomes just  $P$  and  $Q$ . Note two things:

1. The propositional notation does not capture the temporal aspect of the English sentence, namely, that first you are going to the store and then you are going to the movies. Although there are systems of logic which can handle temporal ideas, propositional logic cannot.
2. If someone tells you that “I am going to the store”, you tend to accept the statement as implicitly true. But in the propositional calculus you must remember that any proposition can be true or false; in other words, we consider all possible values of our variables even-handedly. ■

Our discussion of **and** suggests the question: What other logical operators should we allow? Aside from **and**, two other logical operators,

**or**,

**not**,

are of fundamental importance because, with **and**, they allow expression of the kinds of propositions familiar in everyday discourse as well as in mathematics. Like **and**, **or** is a **binary operator** because it has two operands but **not** is a **unary operator** because it has only one operand.

Table 7.2b,c gives the truth tables for “or” and “not”. Thus  $P$  or  $Q$  is the assertion that at least one of  $P$  and  $Q$  is true, and **not**  $P$  is the assertion of the negation of  $P$  (compare to the discussion of negation in Section 0.6). You should be able to think of other ways to express the concept of “or” or “not” in English [1, 2]. Note that “or” as used in logic is the so-called **inclusive or**. It is *not* the assertion that “either  $P$  or  $Q$  but not both” is true (**exclusive or**). This is shown by the first row of the truth table: when both  $P$  and  $Q$  are true, then so is “ $P$  or  $Q$ ”. (What would be the truth table for the “exclusive or” [3a]?)

### EXAMPLE 2

Use **and**, **or**, and **not** and propositional variables to express each of the following:

- a) If my mother doesn’t meet my plane, my father will.

- b)** If I go to eat at The Fisherman tonight, I'll eat lobster, but otherwise I'll have steak.

We will introduce notation for direct translation of if-then statements like these shortly, but the point here is that they can be expressed with **and**, **or**, and **not**.

### Solution

- a)** With

$P$ : My mother will meet my plane.

$Q$ : My father will meet my plane.

this sentence becomes  $P$  **or**  $Q$ . Or does it? Should it be the “exclusive or” instead of the “inclusive or”? Does the English sentence allow the possibility that both parents will meet the plane? The two of us disagree about the answer to this question, one believing that  $P$  “exclusive or”  $Q$  would be a better answer. But such is the inherent ambiguity of natural language that must be avoided in mathematical language. We’ll return below to the question of how to express the “exclusive or” using mathematical notation (see also [3]).

- b)** With

$P$ : I am going to eat at The Fisherman.

$Q$ : I’m going to eat lobster.

$R$ : I’m going to eat steak.

this sentence becomes

$$(P \text{ and } Q) \text{ or } ((\text{not } P) \text{ and } R).$$

Note the use of parentheses to associate operands (i.e., propositions) with operators (i.e., **and**, **or**, and **not**). Later in this section we’ll return to how and when to use parentheses in forming compound propositions. ■

Just about now we’ve gotten tired of typing all those “and”s and “or”s and “not”s. This tedium is a sure sign that we should introduce some mathematical notation as a shorthand for these three words. Be warned that there are a variety of such notations in the mathematical literature. The notation displayed in Fig. 7.2 is, however, quite common. It bears a striking resemblance to the notation we introduced in Section 0.1 for set intersection, union, and complement. This similarity should not greatly surprise you. There is a close analogy between the intersection of two sets (which is the set with elements in one set *and* the other) and the “and” of two propositions, and likewise for union and “or”. The analogy is not quite so close between set complement and “not” because the “universe” in the proposition case consists only of the proposition and its negation. Nevertheless, you will sometimes find an overbar ( $\overline{P}$ ) used for “not”  $P$  instead of  $\neg P$ . We have chosen to use the latter in this section to emphasize the notion of an operator ( $\neg$ ) operating on an operand ( $P$ ). However, in Section 7.5 we’ll use the overbar notation. Also shown in Fig. 7.2 are mathematical terms for the three operators, which we shall use from time to time in what follows.

|            |          |             |
|------------|----------|-------------|
| <b>and</b> | $\wedge$ | conjunction |
| <b>or</b>  | $\vee$   | disjunction |
| <b>not</b> | $\neg$   | negation    |

**FIGURE 7.2**

Notation for logical operators.

**Implication.** If you look at Table 7.2a,b, it will probably occur to you that, by filling in the right-most column with a different sequence of T's and F's you would define another logical operator. Are any of these important? In fact, many of them are important. One of particular importance to us in this chapter is the **implication operator** that is defined in Table 7.3. It corresponds in English to the proposition “if  $P$ , then  $Q$ ” which is close to a statement in our Algorithmic Language. This proposition is written as  $P \Rightarrow Q$  as we already did in Section 7.1.

**TABLE 7.3**  
**Truth Table for**  
**Implication**

| $P$ | $Q$ | $P \Rightarrow Q$ |
|-----|-----|-------------------|
| T   | T   | T                 |
| T   | F   | F                 |
| F   | T   | T                 |
| F   | F   | T                 |

We introduced and discussed implication in Section 0.6. This might be a good time to review that discussion. One aspect of the truth table in Table 7.3 requires some explanation here. Does it bother you that  $P \Rightarrow Q$  is defined to be true when  $P$  is false? Many people find this objectionable at first. You might want to argue that  $P \Rightarrow Q$  should be false when  $P$  is false because then the hypothesis on which it is based is false. But to so argue would be to confuse the conditional statement,  $P \Rightarrow Q$ , with its hypothesis,  $P$ . They are *not* the same thing. Why? Because  $P \Rightarrow Q$  doesn't purport to say anything about  $P$  but only that something else,  $Q$ , is true *if*  $P$  is true. For instance, if someone says “If John has arrived then we are ready to go”, that person is indicating that John is the only thing holding things up. If John has not arrived yet (which the speaker may not know because John might have entered another part of the building) that does not make the claim false or nonsensical or irrelevant. The statement is about the connection between  $P$ , John arriving, and  $Q$ , being ready to leave.

Still, why not just give  $P \Rightarrow Q$  no truth value when  $P$  is false? After all, it's implicit when you say **if  $P$  then  $Q$**  that you are not making a claim about  $Q$  when  $P$  is false. So why assign a truth value to a nonclaim? Well, in mathematics, leaving things undefined is not just untidy; it often makes subsequent analysis and reasoning much more difficult. If we want — and we do — to use  $P \Rightarrow Q$  as a part

of larger logical constructs, we would quickly find ourselves drowning in a sea of complication if we had to take care of undefined cases. So even though we may not care about  $P \Rightarrow Q$  when  $P$  is false, giving it a value is still useful in this case.

But which value? Why did we choose to define  $P \Rightarrow Q$  as true in the “don’t care” cases when  $P$  is false? We gave one reason for this in the previous section where we argued that an argument is valid so long as its conclusion follows from its premises even if one or more of the premises is false. The example above about waiting for John gives another reason but perhaps the best answer is illustrated by the following example. Many mathematical theorems are stated in the form “if  $P$ , then  $Q$ ”. For example, in Section 4.5 we stated the theorem: For any nonnegative integer  $n$

$$\sum_{k=0}^n kC(n, k) = n2^{n-1}, \quad (2)$$

which has the “if  $P$ , then  $Q$ ” form with  $P$ :  $n \in N$ , and  $Q$  the assertion in Eq. (2). Now using our convention about the implication operator when  $P$  is false, by proving  $Q$  assuming that  $P$  is true, we are allowed to assert that  $P \Rightarrow Q$  *always*. There’s even more to it.  $P$  and  $Q$  are actually predicates, with variable  $n$ , so the theorem is really about  $P \Rightarrow Q$  being true for all  $n$ . All  $n$  in what set? With our convention about the truth of  $P \Rightarrow Q$  it doesn’t matter! We may state the theorem as “For all  $n$ ,  $P \Rightarrow Q$ ” and not bother with the domain of  $n$ . It could be all integers (the usual domain for the letter  $n$ ), all real numbers, or whatever mathematical domain we happen to be in at the time. The statement will be true for all these  $n$ , because it will be true by convention when  $P$  is false, that is, when  $n$  is not a nonnegative integer.

Here is another example.

### EXAMPLE 3

State the theorem, “For all even integers  $n$ ,  $n^2$  is also an even integer” as an implication.

**Solution** With  $P$  and  $Q$  the predicates

$P$ :  $n$  is an even integer.

$Q$ :  $n^2$  is an even integer.

this theorem becomes  $P \Rightarrow Q$  since the phrase “for all even integers  $n$ ” means the same as “if  $n$  is an even integer”. As in the previous example,  $P$  and  $Q$  are predicates but that doesn’t matter because even if  $n$  is odd or comes from some other domain, the implication is still true. ■

**Equivalence.** One other important operator that we shall use later in this chapter is the equivalence operator (sometimes called the **biconditional**) defined in Table 7.4. The usual notation for equivalence is  $\Leftrightarrow$  but another sometimes used is  $\equiv$ . The notation  $P \Leftrightarrow Q$  asserts that  $P$  and  $Q$  have the same truth value. For this reason it is sometimes called the “if and only if” or the “necessary and sufficient” operator (compare to Section 0.6). The latter phrase makes sense because, for

$P \iff Q$  to be true when  $Q$  is true, it is not just sufficient that  $P$  be true (as with  $P \implies Q$ ); it is also necessary that  $P$  be true. If  $P$  and  $Q$  are compound propositions such that  $P$  and  $Q$  have the same truth values for all possible combinations of values of the atoms in  $P$  and  $Q$ , we say that  $P$  and  $Q$  are **logically equivalent**. In other words,  $P$  and  $Q$  are logically equivalent if  $P \iff Q$  has value T for all values of the atoms. Analogously, if  $P \implies Q$  has the truth value T for all possible combinations of the atoms in  $P$  and  $Q$ , we say that  $P$  **logically implies**  $Q$ .

---

**TABLE 7.4**  
**Truth Table for**  
**Equivalence**

---

| $P$ | $Q$ | $P \iff Q$ |
|-----|-----|------------|
| T   | T   | T          |
| T   | F   | F          |
| F   | T   | F          |
| F   | F   | T          |

---

## EXAMPLE 4

What other binary operators can be defined besides the four ( $\wedge$ ,  $\vee$ ,  $\implies$  and  $\iff$ ) that we have already discussed?

**Solution** Since the truth table for any binary operator has four rows (one for each possible pair of values of true and false for the operands) and since for each row the entries in the other columns can have one of two values (T or F), the Product Rule of Section 4.2 says that there are  $2^4 = 16$  possible binary truth tables or **truth functions**. These are shown in Fig. 7.3. The notation in the various columns is that commonly used for these operators. Only 10 of the 16 columns correspond to useful, nontrivial binary operators. In the other six, those headed “True”,  $P$ ,  $Q$ ,  $\neg Q$ ,  $\neg P$ , and “False”, the value of the operator depends upon neither of the propositions  $P$  and  $Q$  or on only one of them. ■

Is Example 4 just an academic exercise? The four operators we’ve already discussed correspond to columns 2, 5, 7 and 8. Are any of the six operators in columns 3, 9, 10, 12, 14 or 15 useful? Well, three of them (in columns 3, 12 and 14) are just variations on implication and a fourth (column 10) represents what we called earlier the exclusive or. The other two, the **Sheffer stroke** | (column 9) and **Peirce’s arrow** ↓ (column 15), are very important, indeed, in the design of digital circuits. Discussion of these is beyond us here but see [4–8, Supplement].

The binary operators we’ve discussed in this section are not independent in the sense that various of them can be expressed in terms of the others. One instance of this is the replacement of  $P \iff Q$  by  $\neg(P \implies Q)$ , which is hardly surprising since nonequivalence is just the negation of equivalence. In [6] and [3 and 6–8, Supplement], we explore further the relationships among various logical operators.

| <i>P</i> | <i>Q</i> | 1<br>“True”  | 2<br>$P \vee Q$      | 3<br>$P \iff Q$ | 4<br>$P$                 | 5<br>$P \implies Q$ | 6<br>$Q$             | 7<br>$P \iff Q$        | 8<br>$P \wedge Q$ |
|----------|----------|--------------|----------------------|-----------------|--------------------------|---------------------|----------------------|------------------------|-------------------|
| <i>P</i> | <i>Q</i> | T            | T                    | T               | T                        | T                   | T                    | T                      | T                 |
| T        | F        | T            | T                    | T               | T                        | F                   | F                    | F                      | F                 |
| F        | T        | T            | T                    | F               | F                        | T                   | T                    | F                      | F                 |
| F        | F        | T            | F                    | T               | F                        | T                   | F                    | T                      | F                 |
| <i>P</i> | <i>Q</i> | 9<br>$P   Q$ | 10<br>$P \not\iff Q$ | 11<br>$\neg Q$  | 12<br>$P \not\implies Q$ | 13<br>$\neg P$      | 14<br>$P \not\iff Q$ | 15<br>$P \downarrow Q$ | 16<br>“False”     |
| <i>P</i> | <i>Q</i> | F            | F                    | F               | F                        | F                   | F                    | F                      | F                 |
| T        | F        | T            | T                    | T               | T                        | F                   | F                    | F                      | F                 |
| F        | T        | T            | F                    | F               | T                        | T                   | F                    | F                      | F                 |
| F        | F        | T            | F                    | T               | F                        | T                   | F                    | T                      | F                 |

Names of Operators

|                                    |                                   |
|------------------------------------|-----------------------------------|
| 2 — inclusive or, disjunction      | 10 — nonequivalence, exclusive or |
| 3, 5 — conditional, implication    | 11, 13 — not, negation            |
| 7 — equivalence                    | 12, 14 — nonimplication           |
| 8 — and, conjunction               | 15 — Peirce’s arrow, nor (not or) |
| 9 — Sheffer stroke, nand (not and) |                                   |

### FIGURE 7.3

The 16 binary truth functions.

To conclude this subsection and provide a bridge to the next subsection, here is an example of an application of the propositional calculus.

### EXAMPLE 5

Suppose you have three kitchen appliances all plugged into the same circuit. You know from experience that if all three are on at the same time, your circuit breaker will switch them all off. You don’t like resetting your circuit breaker, so you wish to build a new switch in which to plug the three appliances that will not allow all three appliances to be on at once. Express this requirement using the propositional calculus.

**Solution** Call the three appliances  $A_1$ ,  $A_2$  and  $A_3$ . Then defining

$P$ :  $A_1$  is switched on.

$Q$ :  $A_2$  is switched on.

$R$ :  $A_3$  is switched on.

we need to express the idea that at most two of  $P$ ,  $Q$ ,  $R$  can be true. One answer is that  $P \wedge Q \wedge \neg R$  will be true if  $A_1$  and  $A_2$  are on and  $A_3$  is off and similarly we can write expressions for each of the other six allowable states [14a] and then “or” these together. Pretty tedious. ■

There is, in fact, a simpler solution to the problem in Example 5 than the tedious one just outlined [14b]. But suppose there wasn’t. Then somehow we

would have to write a complex expression to encompass the answer. The next section is about the rules by which we construct such expressions.

## Constructing and Evaluating Compound Propositions

In the foregoing we wrote some compound propositions, that is, expressions containing one or more propositional variables and at least one logical operator. But we did this informally without specifying the rules that must be followed to form *legal* compound propositions. Such rules and their application are the subject of this subsection.

We surely can't write just any old string of propositions and operators. It should be obvious, for example, that  $\wedge PQ$  or  $P \vee \iff Q$  can have no reasonable interpretation. But how about

$$P \vee Q \wedge R \implies S \iff T$$

or

$$P \wedge Q \vee \neg\neg R ?$$

These may make sense but, if so, what is their appropriate interpretation? To answer this question we need rules about forming compound propositions from atomic propositions and logical operators.

In what follows we use  $P, Q, R, \dots$  for atomic propositions and  $A, B, C, \dots$  for general propositions that may be atomic or compound. Thus we might denote  $P \wedge \neg Q \vee R$  by  $A$ .

An allowable expression consisting of propositions and logical operators is called a **well-formed formula**, commonly abbreviated **wff** and pronounced “woof” (or, in some precincts, “wiff”). Logicians often just call a wff a formula.

The rules for forming wffs from atomic propositions and operators are simple and intuitive:

1. An individual atomic proposition  $P, Q, R, \dots$ , is a wff.
2. If  $A$  is a wff, then  $\neg A$ , the negation of  $A$ , is a wff.
3. If  $A$  and  $B$  are wffs, then  $A \circ B$  is a wff, where  $\circ$  is any of the binary operators that we have defined ( $\wedge, \vee, \implies, \iff$ ).

You should recognize these as **syntactic rules**, which define the *structure* of wffs. They enable you to determine whether a particular string is a wff. One problem [17] is devoted to asking you to determine if particular strings of operators and propositions are wffs. You should also recognize that these rules provide a recursive definition of the set of all wffs: Given any two wffs, Rules 2 and 3 allow you to construct new wffs ad infinitum. See the discussion of recursively defined sets in Section 2.7.

But our real interest is not in the (rather tedious) business of determining syntactic correctness or in generating new wffs from old ones. Rather we want to know about the **semantics**, or meaning, of wffs. That is, given any truth values of the atomic propositions in a wff, we'd like to know the truth value of the wff. But

now we have some problems, for the meaning of a wff is determined by *how* it was generated by the rules, yet many wffs can be generated in more than one way. For example, what is the interpretation of

$$P \wedge Q \implies R? \quad (3)$$

We can generate this wff from the atomic propositions  $P, Q$  and  $R$  by (a) using Rule 3 to generate  $P \wedge Q$  and then using it again to generate (3), or (b) by using Rule 3 to generate  $Q \implies R$  and then using this rule again to generate (3). Sure enough, if we evaluate (3) by first evaluating  $P \wedge Q$  and then applying the implication, we get a different result than if we first evaluate  $Q \implies R$  and then “and” this with  $P$ . By “different result” we mean the expression is true in different circumstances and thus must be said to have a different meaning. (See Table 7.5, where we have used parentheses to indicate the order in which the operators are applied and have used color to indicate where the truth values of the two parenthesizations differ.)

---

**TABLE 7.5**

**Truth Tables for  $(P \wedge Q) \implies R$  and  $P \wedge (Q \implies R)$**

---

| $P$ | $Q$ | $R$ | $P \wedge Q$ | $(P \wedge Q) \implies R$           | $Q \implies R$ | $P \wedge (Q \implies R)$           |
|-----|-----|-----|--------------|-------------------------------------|----------------|-------------------------------------|
| T   | T   | T   | T            | T                                   | T              | T                                   |
| T   | T   | F   | T            | F                                   | F              | F                                   |
| T   | F   | T   | F            | T                                   | T              | T                                   |
| T   | F   | F   | F            | T                                   | T              | T                                   |
| F   | T   | T   | F            | <span style="color: blue;">T</span> | T              | <span style="color: blue;">F</span> |
| F   | T   | F   | F            | <span style="color: blue;">T</span> | F              | <span style="color: blue;">F</span> |
| F   | F   | T   | F            | <span style="color: blue;">T</span> | T              | <span style="color: blue;">F</span> |
| F   | F   | F   | F            | <span style="color: blue;">T</span> | T              | <span style="color: blue;">F</span> |

---

One way to resolve the ambiguity in (3) would be to define a precedence rule for logical operators, as we did in Section 0.1 for set operators. Recall that a precedence rule says evaluate all of one operator before any of another, e.g., all  $\wedge$  before all  $\implies$ . Indeed, this is often done but we shall explore this approach for binary operators only in the problems [18]. The only precedence rule we will assume (because everyone does) is that all nots ( $\neg$ ) are evaluated first. That is,  $\neg P \wedge Q$  means  $(\neg P) \wedge Q$ , not  $\neg(P \wedge Q)$ . (See [17e].)

Such precedence rules would not be enough by themselves to resolve all possible ambiguities. Consider the interpretation of

$$P \implies Q \implies R. \quad (4)$$

Precedence rules could not tell us which of the implications in (4) to apply first. And, as Table 7.6 illustrates, the interpretation  $(P \implies Q) \implies R$  is different from  $P \implies (Q \implies R)$ , where again we have used parentheses to indicate the order of application of the operators. How can we avoid the ambiguity illustrated by Table 7.6? One way would be to define a rule that requires that identical operators

be evaluated in a left-to-right (or right-to-left) order. However as with precedence rules, we explore this idea only in the problems [19].

**TABLE 7.6**

**Truth Tables for  $(P \Rightarrow Q) \Rightarrow R$  and  $P \Rightarrow (Q \Rightarrow R)$**

| $P$ | $Q$ | $R$ | $P \Rightarrow Q$ | $(P \Rightarrow Q) \Rightarrow R$ | $Q \Rightarrow R$ | $P \Rightarrow (Q \Rightarrow R)$ |
|-----|-----|-----|-------------------|-----------------------------------|-------------------|-----------------------------------|
| T   | T   | T   | T                 | T                                 | T                 | T                                 |
| T   | T   | F   | T                 | F                                 | F                 | F                                 |
| T   | F   | T   | F                 | T                                 | T                 | T                                 |
| T   | F   | F   | F                 | T                                 | T                 | T                                 |
| F   | T   | T   | T                 | T                                 | T                 | T                                 |
| F   | T   | F   | T                 | F                                 | F                 | T                                 |
| F   | F   | T   | T                 | T                                 | T                 | T                                 |
| F   | F   | F   | T                 | F                                 | T                 | T                                 |

Our discussion of Tables 7.5 and 7.6 implies that ambiguities can be avoided by the use of parentheses. We could do this formally by requiring the use of a **fully parenthesized notation** in which each binary operator and its operands are enclosed in parentheses [20]. No similar rule is required with a unary operator such as  $\neg$  because these operators cannot cause an ambiguity if all binary operators are fully parenthesized [20a].

However, instead of fully parenthesizing, we shall proceed informally as we did with arithmetic expressions. First, we will parenthesize anywhere where it might seem that the lack of a precedence rule or left-to-right rule leads to an ambiguity. Thus, for the expression in (3) we would write either  $(P \wedge Q) \Rightarrow R$  or  $P \wedge (Q \Rightarrow R)$  depending upon which of these we really wanted. And similarly for (4) we would write either  $(P \Rightarrow Q) \Rightarrow R$  or  $P \Rightarrow (Q \Rightarrow R)$ . Second, we will add additional parentheses if they make an expression more readable. Yes, if we're not careful, this informal approach could still leave ambiguities. But *we* shall be careful and *you* should be, too. In any case, if an ambiguity does crop up, you'll just add some parentheses to remove it. So our Rule 4 for forming wffs, which we add to the three at the beginning of this subsection, is:

4. Any wff formed by Rules 1–3 *may* have parentheses around any binary operator and its operands and *should* have such parentheses inserted if ambiguity would result in their absence.

Rules 1–4 enable us to construct and evaluate any wff given only that we apply Rule 4 carefully enough to avoid any ambiguities.

We have now developed all of the paraphernalia of the propositional calculus that we need. In the next section we'll apply it.

## Problems: Section 7.2

1. ⟨1⟩ Give two — more if you can — words or phrases which, as far as their truth value is concerned, are equivalent to “or”. Use each in a sentence to show what you mean.

2. ⟨1⟩ Same as [1] except with “not” instead of “or”.

3. ⟨1⟩

a) Give the truth table for the “exclusive or”.

b) By constructing the truth table for

$$\neg(P \iff Q)$$

show that this wff is equivalent to ( $P$  “exclusive or”  $Q$ ).

4. ⟨1⟩ Same as [1] except with “exclusive or” instead of “or”.

5. ⟨1⟩ For Fig. 7.3 express in words the meaning of the operators heading

a) columns 9 and 15

b) columns 12 and 14

c) column 10.

6. ⟨2⟩ Each of the binary operators in columns 9, 10, 12, 14, and 15 of Fig. 7.3 can be expressed in terms of the logical operators,  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\implies$ , and  $\iff$ . For instance, in the text we noted that  $P \iff Q$  (column 10) can be expressed as  $\neg(P \iff Q)$  because both have the same truth tables. Show how to use  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\implies$ , and  $\iff$  to express the operators heading

a) columns 9 and 15

b) columns 12 and 14.

7. ⟨2⟩ Each of the binary operators in columns 9, 10, 12, 14, and 15 of Fig. 7.3 can be expressed just using the logical operators,  $\wedge$ ,  $\vee$ , and  $\neg$ . Show how to do this for the operators for

a) columns 9 and 15

b) columns 12 and 14

c) column 10.

8. ⟨2⟩ Same as [7] except you can only use  $\vee$ , and  $\neg$ .

9. ⟨3⟩ Not every way to combine two propositions is a logical operator.

a) Consider the propositional form “ $P$  because  $Q$ ”. Find true propositions for  $P$  and  $Q$  which make “ $P$  because  $Q$ ” true. Find two other true

propositions for  $P$  and  $Q$  which make “ $P$  because  $Q$ ” false. Why is “because” therefore not a logical operator?

b) Consider the propositional form “ $P$  after  $Q$ ”. Find true propositions for  $P$  and  $Q$  which make “ $P$  after  $Q$ ” true. Find two other true propositions for  $P$  and  $Q$  which make “ $P$  after  $Q$ ” false, or even nonsensical.

10. ⟨2⟩ Using  $J$  for “John is here” and  $M$  for “Mary is here”, and using  $\wedge$ ,  $\vee$  and  $\neg$ , write an expression for each of the following sentences. (Sometimes even simple English sentences can be ambiguous, especially if they involve “not”. If you think one of these sentences is ambiguous, state in your own words the different things you think it can mean and translate each meaning into symbols.)

a) John is not here.

b) John isn’t here.

c) John is here and Mary is here.

d) John and Mary are here.

e) John and Mary are not here.

f) John and Mary are not both here.

g) Neither John nor Mary is here.

h) John or Mary is here.

i) John is here but Mary isn’t.

j) One of John or Mary isn’t here.

k) At least one of John or Mary isn’t here.

11. ⟨2⟩ Let  $R$  stand for “It is raining” and  $C$  stand for “It is cloudy”. Now using implication ( $\implies$ ) as well as the  $\wedge$ ,  $\vee$  and  $\neg$ , write an expression for each of the following sentences. The instructions about ambiguity in [10] still apply.

a) If it’s raining, then it’s cloudy.

b) If it is raining, there are clouds.

c) Only if it’s raining are there clouds.

d) For rain it is necessary that there be clouds.

e) For rain it is sufficient that there are clouds.

f) It isn’t raining unless it is cloudy.

g) It rains just in case it’s cloudy.

12. ⟨2⟩ Write each of the following sentences using logic and math notation. For instance, “if  $x$  is 2, then  $x$  squared is 4” becomes  $(x = 2) \implies (x^2 = 4)$ .

Also, indicate whether each assertion is true or false.

- a) Whenever  $x$  is 2, then  $x$  squared is 4.
- b) Given that  $x$  squared is 4, it follows that  $x$  is 2.
- c) Assuming that  $x$  is 2, we have  $x$  cubed is 8.
- d) If  $x$  squared is 4 and  $x$  is positive, then we can conclude that  $x$  is 2.
- e) If  $x$  is positive, then  $x$  squared being 4 implies that  $x$  is 2.
- f) From the assumption that  $x$  cubed is 8, it is necessary that  $x$  be 2.
- g) It is not true that  $x$  squared being 4 forces  $x$  to be 2.
- h) It is not true that if  $x$  squared is 4, then  $x$  is 2.
- i) If  $x$  squared is not 4, then  $x$  is 2.
- j) If  $x$  squared is 4, then  $x$  is 2 or  $x$  is -2.
- k) Either  $x$  squared is 4 implies that  $x$  is 2, or  $x$  squared is 4 implies that  $x$  is -2.
- l) 4 is greater than 6 just in case 0 is greater than 2.

13.  $\langle 2 \rangle$  Write each of the following sentences using math and logic notation. Use  $x$  and  $y$  for arbitrary numbers. For instance, the statement “the sum of two numbers is 6” should be translated as “ $x + y = 6$ ”.

- a) If the sum of two numbers is negative, then at least one of the numbers is negative.
- b) The product of two numbers is 0 if and only if at least one of them is 0.
- c) The sum of two positive numbers is positive.
- d) A number isn't 3 unless it's positive.
- e) When the absolute values of two numbers are the same, the numbers are themselves the same or one is the negative of the other.

14.  $\langle 2 \rangle$

- a) Write the six other expressions besides the one given in Example 5 that might be true if at most two of the appliances are on.
- b) You could write an expression for the idea that only two of the appliances are allowed to be on at any one time by “or”ing together the six expressions found in a) together with the one given in Example 5. Can you find a simpler expression (still using  $P, Q, R$  as defined in the Example) to accomplish the same thing?

15.  $\langle 2 \rangle$  Find all wffs without parentheses, using just the atom  $P$  and the operands  $\neg$  and  $\wedge$ , that can be obtained by at most three applications of Rules 1–3 defining wffs. For instance,  $P \wedge \neg P$  is one such wff. The first rule (atoms are wffs) was used to get  $P$ . Then one application of the rule about “nots” gives us  $\neg P$ . Then an application of the rule about  $A \circ B$  gives us  $P \wedge \neg P$ . Thus we used three applications all told.

16.  $\langle 2 \rangle$  Use Rules 1–3 to prove that no wff has two consecutive binary operators. Begin by proving that no wff begins or ends with a binary operator.

17.  $\langle 2 \rangle$  Use Rules 1–4 to determine whether each of the following is an unambiguous wff.

You must show how application of the rules leads to your conclusion and you must show that there is only one possible interpretation of any string that you claim is a wff.

a)  $P \wedge Q \vee R$

b)  $P \wedge \neg \neg Q$

c)  $P \neg \wedge Q$

d)  $P \vee (Q \wedge R)$

- e)  $\neg P \wedge Q$  (Assume for this part that there are no precedence rules, not even precedence of  $\neg$  over everything else.)

f)  $P \implies Q \iff R$

18.  $\langle 2 \rangle$  A *precedence rule* states in which order two or more operators inside the same set of parentheses must be evaluated. So consider the operators  $\wedge, \vee, \implies$  and  $\iff$ . (By convention, all  $\neg$ 's are applied before any binary operators.) If all the following wffs are to have the same interpretation if the parentheses are removed, what can you infer about the precedence of the operators (i.e., in what order must the operators with their arguments be evaluated)?

1)  $(P \wedge Q) \vee R$ .

2)  $(P \vee Q) \implies (R \wedge S)$ .

3)  $(P \implies Q) \iff (R \implies S)$ .

19.  $\langle 2 \rangle$  Precedence rules among the operators are not by themselves enough to disambiguate any logical expression without parentheses.

- a) For each of  $\wedge, \vee, \implies$  and  $\iff$ , consider expressions of the form

$$P \circ Q \circ R \tag{5}$$

where each  $\circ$  represents the same one of the

four operators listed above. For each of the four operators, show whether or not the expression (5) is ambiguous without parentheses. This is equivalent to answering the question: Which of the four operators satisfy an associative law?

- b) For those operators in part a) for which there is ambiguity, suppose there is a rule that requires them to be evaluated in a left to right order. Is this sufficient to remove any ambiguity in the absence of parentheses? Why?

20. ⟨3⟩ In fully parenthesized notation (FPN) each binary operator and its associated operands *must* be enclosed in parentheses. Thus, for example, we cannot write  $P \wedge Q \vee R$  but must write instead  $((P \wedge Q) \vee R)$  or  $(P \wedge (Q \vee R))$ .

- a) No parenthesizing rule is needed for  $\neg$  and its associated operand since there can never be any ambiguity without parentheses, so long as binary operators are fully parenthesized. Explain why.
- b) In a wff, count each left parenthesis as +1 and each right parenthesis as -1. Prove that, as one reads any FPN wff from left to right, the parenthesis count is always nonnegative, and is 0 at the end.
- c) Suppose you have a string of operators, operands (consisting of one letter each) and

parentheses that purports to be an FPN expression. Outline an algorithm that scans across this string and determines if it is in FPN or not.

21. ⟨3⟩ Any FPN expression may be displayed as a tree using the following rules:
- If the string has no binary operators and no negations, the tree has only a root which contains the entire string.
  - If there are any leading (i.e., not enclosed in parentheses) negations, we place them, successively, at the root of the tree and at the roots of successive right subtrees until all are used up.
  - Place the main binary operator (the one enclosed in only a single pair of parentheses) if there is one, at the root of the entire tree or at the root of the next right subtree if there are leading negations.
  - Then consider the two operands of the main binary operator after stripping off any leading negations and the outer pair of parentheses; we then find the left and right subtrees of the node for the main binary operator by applying rules i)-iii) recursively.

Fully parenthesize the expressions in [17a,b,f] each allowable way and then use these rules to display each expression as a tree.

## 7.3 Validity and Tautology

In the previous section we developed the paraphernalia of the propositional calculus. In this section we discuss a very important application of the propositional calculus, namely testing the validity of arguments. Recall from Section 7.1 that a valid argument is one in which the conclusion must be true whenever the premises are true. First, we'll return to two examples of Section 7.1.

In Example 1 in Section 7.1 an argument about when to wear a raincoat was expressed in words as

If  $(P)$  it is cloudy in the morning, then  $(Q)$  it is going to rain in the afternoon.

If  $(Q)$  it is going to rain in the afternoon, then  $(R)$  I should take my raincoat.

Therefore, if  $(P)$  it is cloudy in the morning, then  $(R)$  I should take my raincoat with me.

More formally, we expressed this argument as

$$P \implies Q$$

$$Q \implies R$$

$$\hline$$

$$P \implies R.$$

This argument can be associated with a single wff:

$$((P \implies Q) \wedge (Q \implies R)) \implies (P \implies R). \quad (1)$$

We may use this wff to restate the claim that the argument is valid: the conclusion of this wff is true whenever its premise  $((P \implies Q) \wedge (Q \implies R))$  is true.

Similarly, the associated wff for the argument in the second part of Example 1 in Section 7.1, about astrology, is

$$((P \implies Q) \wedge Q) \implies P, \quad (2)$$

and we need to check if the conclusion is true whenever the hypothesis is true.

How can we test the validity of the arguments that have been summarized in (1) and (2)? That is, how can we check that the conclusions of these implications are true whenever their premises are true? In light of the discussion in Section 7.2 about implications being true automatically when their premises are false, there is a simple answer: An argument is valid if its associated wff is true for *all* values of the atoms in it. Furthermore, in Section 7.2 we discussed an infallible procedure for calculating the truth value of a wff for all values of its atoms, namely *truth tables*. Therefore, the arguments we have just discussed are valid if and only if when we construct the truth tables for (1) and (2), the column for the entire wff consists of all T's. Thus truth tables are said to constitute a **decision procedure**, really just an *algorithm*, to determine the validity of arguments. The decision procedure is: Write down the associated wff and determine if the last column of its truth table is all T's.

In Table 7.7a we have constructed the truth tables for (1) and (2). Since the last column is all T's, the argument related to wff1 is valid. Note that the premises  $(P \implies Q)$  and  $(Q \implies R)$  are both true only in rows 1, 5, 7, and 8 so that in the other rows wff1 is true without a need to consider the conclusion.

In Table 7.7b, the premises  $(P \implies Q)$  and  $Q$  are both true in lines 1 and 3 but wff2 is false in line 3 and so our argument about astrology was not valid. (Bet you knew that anyhow!) The interpretation of the F in the last column is that one of the hypotheses could be true (i.e., the economy is improving) even if the conclusion is false (i.e., astrology is not a true science) (whew!).

A wff which has the property, as in expression (1), that it is true for all possible values of its atomic propositions, is called a **tautology** and is said to be **logically true** or **necessarily true**. In the previous section we noted that when a wff of the form  $A \implies B$  is true for all values of its atoms and is, thus, a tautology, we say that  $A$  **logically implies**  $B$ . It's important that you understand the distinction between  $A$  *implies*  $B$ , that is,  $A \implies B$ , which is a wff whose truth value may be true or false depending upon the truth values of its atoms, and  $A$  *logically implies*

TABLE 7.7

Truth Tables for the Arguments of Example 1, Section 7.1:

(a)  $wff1 = ((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R)$ ;(b)  $wff2 = ((P \Rightarrow Q) \wedge Q) \Rightarrow P$ 

| (a) | $P$ | $Q$ | $R$ | $P \Rightarrow Q$ | $Q \Rightarrow R$ | $(P \Rightarrow Q) \wedge (Q \Rightarrow R)$ | $P \Rightarrow R$ | $wff1$ |
|-----|-----|-----|-----|-------------------|-------------------|----------------------------------------------|-------------------|--------|
|     | T   | T   | T   | T                 | T                 | T                                            | T                 | T      |
|     | T   | T   | F   | T                 | F                 | F                                            | F                 | T      |
|     | T   | F   | T   | F                 | T                 | F                                            | T                 | T      |
|     | T   | F   | F   | F                 | T                 | F                                            | F                 | T      |
|     | F   | T   | T   | T                 | T                 | T                                            | T                 | T      |
|     | F   | T   | F   | T                 | F                 | F                                            | T                 | T      |
|     | F   | F   | T   | T                 | T                 | T                                            | T                 | T      |
|     | F   | F   | F   | T                 | T                 | T                                            | T                 | T      |

| (b) | $P$ | $Q$ | $P \Rightarrow Q$ | $(P \Rightarrow Q) \wedge Q$ | $wff2$ |
|-----|-----|-----|-------------------|------------------------------|--------|
|     | T   | T   | T                 | T                            | T      |
|     | T   | F   | F                 | F                            | T      |
|     | F   | T   | T                 | T                            | F      |
|     | F   | F   | T                 | F                            | T      |

$B$ , in which the truth value of  $A \Rightarrow B$  is always true whatever the truth values of the atoms.

This distinction is clearly important. For this reason some texts use different symbols for the two concepts, typically  $A \rightarrow B$  for implies and  $A \Rightarrow B$  for logically implies.

*Note:* In ordinary English usage a “tautology” is a statement devoid of information. What’s the use of making a statement that is true independent of the facts? If I ask you, “Were you there?” and you answer, “Either I was or I wasn’t”, you haven’t been helpful. However, in mathematical logic, tautologies are often highly nonobvious and no stigma is attached to this word.

We can summarize what we have done so far in this section with a formal definition of a valid argument and a theorem that relates validity and tautology.

**Definition 1.** An **argument** is a claim that a conclusion  $B$  follows from zero or more premises  $A_1, A_2, \dots, A_n$ . If  $A_1, \dots, A_n$  and  $B$  are wffs in the propositional calculus, then the argument is **valid** if  $B$  is true for all truth values of the atoms that make all the premises true. If there are no premises, the argument is said to be valid if  $B$  is true for all truth values of its atoms.

The notion of a zero-premise argument may seem strange. But it will become less so when you see some examples of such arguments later in this section.

---

**Theorem 1.** Let  $B$  and  $A_1, A_2, \dots, A_n$  be wffs. The argument that a conclusion  $B$  follows from premises  $A_1, A_2, \dots, A_n$  is valid if and only if

$$(A_1 \wedge A_2 \wedge \cdots \wedge A_n) \implies B \tag{3}$$

is a tautology. If  $n = 0$  (i.e., there are no premises), then the argument is valid if and only if  $B$  itself is a tautology.

---

**PROOF** If expression (3) is a tautology, then  $B$  must be true for all truth values of the atomic propositions which make

$$A_1 \wedge A_2 \wedge \cdots \wedge A_n \tag{4}$$

true, that is, for all values of the atoms which make each of  $A_1, \dots, A_n$  true. So by Definition 1 the argument is valid.

Conversely, if the argument is valid, then  $B$  is true when all the premises are true, and, therefore, wff (3) is true whenever (4) is true. But (3), being an implication, is automatically true whenever (4) is false. Therefore the wff (3) is true for any values of its atoms and so is a tautology. This completes the proof when  $n \neq 0$ .

When  $n = 0$  there is nothing to prove, because both the definition and the theorem say that  $B$  must be true for all values of the atoms in this case. ■

(Do you see why it was necessary to define implication as we did when the hypothesis is false in order for this theorem to be true?)

So it seems we have said everything that needs to be said about proving theorems in the propositional calculus. We've outlined a nice mechanical procedure that will always work (calculate the truth table of the associated wff). But it isn't enough. Why not?

- One answer might be that, when the number of atomic propositions gets fairly large, the truth table becomes long and unwieldy. The number of lines in a truth table, when there are  $n$  atomic propositions, is  $2^n$  so the complexity of an algorithm to generate a truth table is  $\text{Ord}(2^n)$ . Actually things are a bit worse than this since, as the number of atoms increases, so, too, generally does the effort to evaluate each line of a truth table. Still, when the order of an algorithm is already exponential, things are so bad that making things a bit worse hardly matters.

If you haven't done so already, try generating some truth tables with your CAS. You'll find that this is practical only when the number of premises  $n$  is quite small. Still, with computers around, the answer above won't wash unless the number of variables is quite large (say,  $n > 20$ ) in which case the complexity of a truth table algorithm may still defeat you.

- A better answer is that proof by truth table does *not* lead to insight about an argument or to possible generalizations of it. Another method might do so. In particular, a more insightful approach might suggest what is provable and what is interesting to prove.
- But perhaps the best answer is that very few portions of mathematics allow a completely mechanical proof procedure like the truth table technique of the propositional calculus. (If you studied Euclidean geometry in high school, imagine how much easier it would have been if there were a crank you could have turned to generate proofs!<sup>†</sup>) Because one reason for studying the propositional calculus is to provide a simple subsystem of mathematics from which to get a better feel for the logical methods of mathematics in general, developing a deductive method of proving theorems (i.e., validating arguments) in the propositional calculus, as well, would be wise.

Thus pursuing another method of proving things in the propositional calculus is important in itself. We'll introduce such a method in Section 7.5 and give a brief introduction to a third method in Section 7.8.

**Further Uses of Tautologies.** So far all our tautologies have been implications. But from our definition of tautology, any wff can be a tautology if only it is true for all values of its atoms. For instance,

$$P \vee \neg P$$

is a tautology. It is known as the Law of the Excluded Middle (why?).

In particular, the notion of a tautology is very useful when we want to determine whether two wffs are logically equivalent. (Remember that this means that the two wffs have the same truth value for all possible values of their atoms.) Suppose, as we shall discuss further in Section 7.5, that we are designing an integrated circuit chip for a computer or some other electronic device. The output function of such a circuit is often expressed as a wff of its inputs. The form of this wff will determine just how much physical circuitry must be used to realize the desired output. If the wff can be simplified, it may be possible to save on circuitry and, therefore, on cost. The situation is:

- You are given wff1 describing the output in terms of the inputs.
- Someone presents you with a simpler wff2, purporting to give the same output as wff1 for all values of the input.

Question: Are they really logically equivalent? That is, is

$$\text{wff1} \iff \text{wff2} \tag{5}$$

a tautology?

How would you answer the question above? The obvious way at this point is just to calculate the truth table, as in the following examples (see also [14] and the discussion about simplifying Boolean expressions in Section 7.5).

---

<sup>†</sup>Actually, there *is* such a crank for Euclidean geometry in the form of a decision procedure due to Tarski, but it is neither easy to explain nor insightful.

**EXAMPLE 1**

Are the two wffs

$$(P \wedge Q) \vee (P \wedge R) \vee (\neg Q \wedge R) \quad \text{and} \quad (P \wedge Q) \vee (\neg Q \wedge R)$$

logically equivalent? That is, can we just omit the middle term in the first wff?

**Solution** The relevant truth table, shown in Table 7.8, indicates that the two wffs are equivalent because the last column is all T's; that is, the second term in wff1,  $P \wedge R$ , is superfluous because the truth value of the wff is the same whether it is present or absent. ■

**TABLE 7.8****Truth Table for the Equivalence of Two wffs:**

$$\text{wff1} = (P \wedge Q) \vee (P \wedge R) \vee (\neg Q \wedge R);$$

$$\text{wff2} = (P \wedge Q) \vee (\neg Q \wedge R)$$

| P | Q | R | $P \wedge Q$ | $P \wedge R$ | $\neg Q \wedge R$ | wff1 | wff2 | $\text{wff1} \iff \text{wff2}$ |
|---|---|---|--------------|--------------|-------------------|------|------|--------------------------------|
| T | T | T | T            | T            | F                 | T    | T    | T                              |
| T | T | F | T            | F            | F                 | T    | T    | T                              |
| T | F | T | F            | T            | T                 | T    | T    | T                              |
| T | F | F | F            | F            | F                 | F    | F    | T                              |
| F | T | T | F            | F            | F                 | F    | F    | T                              |
| F | T | F | F            | F            | F                 | F    | F    | T                              |
| F | F | T | F            | F            | T                 | T    | T    | T                              |
| F | F | F | F            | F            | F                 | F    | F    | T                              |

**EXAMPLE 2**

Prove that the following expressions are tautologies.

a)  $\underbrace{\neg\neg\dots\neg}_{2n} P \iff P.$

b)  $\underbrace{\neg\neg\dots\neg}_{2n-1} P \iff \neg P.$

**Solution**a) When  $n = 1$  the wff is  $\neg\neg P \iff P$  for which the truth table is

| P | $\neg P$ | $\neg\neg P$ |
|---|----------|--------------|
| T | F        | T            |
| F | T        | F            |

Thus the wff is a tautology. Now how would you prove this for any  $n$ ? By induction, of course! We leave the proof to a problem [12a].b) The proof for b) is essentially the same as that for a). Alternatively, define  $Q$  to be the proposition  $\neg P$  and use a) immediately [12b]. ■

Did you notice that the examples above actually are cases of the paradigm (3) with  $n = 0$ ? Just write (5) in the form

$$\implies (\text{wff1} \iff \text{wff2}) \quad (6)$$

and calculate the truth table as we have.

Still another way to prove that a wff like (5) is a tautology is to show that

$$(\text{wff1} \implies \text{wff2}) \wedge (\text{wff2} \implies \text{wff1})$$

is a tautology. This works because

$$(\text{wff1} \iff \text{wff2}) \iff ((\text{wff1} \implies \text{wff2}) \wedge (\text{wff2} \implies \text{wff1}))$$

is a tautology. That is,  $\text{wff1} \iff \text{wff2}$  is logically equivalent to  $(\text{wff1} \implies \text{wff2}) \wedge (\text{wff2} \implies \text{wff1})$  which is easy to show with a truth table [2]. So, instead of showing the equivalence of two wffs,  $\text{wff1}$  and  $\text{wff2}$ , we can show that  $\text{wff1}$  implies  $\text{wff2}$  and that  $\text{wff2}$  implies  $\text{wff1}$ .

## Problems: Section 7.3

---

1.
  - (1) Create truth tables for the following wffs. It is best if your table contains columns for intermediate results. For instance, your table for a) would contain columns for  $\neg R$  and  $Q \vee \neg R$ .
    - a)  $P \wedge (Q \vee \neg R)$
    - b)  $(P \vee W) \iff R$
    - c)  $(P \implies Q) \implies (Q \implies P)$
  2.
    - (1) Show by a truth table that, for any wffs  $A$  and  $B$ ,
$$(A \iff B) \iff ((A \implies B) \wedge (B \implies A)).$$
  3.
    - (2) Which of the following wffs are tautologies? (Use Fig. 7.3 for the definitions of the operators.)
      - a)  $(P \wedge Q) \implies P$
      - b)  $(P \vee Q) \implies P$
      - c)  $P \implies (P \vee Q)$
      - d)  $(A \implies B) \vee (A \implies \neg B)$
      - e)  $(P \mid Q) \vee (P \downarrow Q)$
      - f)  $(P \implies Q) \vee (Q \implies P)$
    4.
      - (2) Which of the following pairs of wffs are logically equivalent?
        - a)  $\neg P \wedge \neg Q$  and  $\neg(P \vee Q)$
        - b)  $(P \iff Q) \wedge (Q \iff R)$  and  $P \iff R$
        - c)  $P \implies Q$  and  $\neg P \vee Q$
        - d)  $(P \vee Q) \implies R$  and  $(P \implies R) \vee (Q \implies R)$
      5.
        - (2) For which of the following pairs of wffs does the first logically imply the second? (Use Fig. 7.3 for the definitions of the operators.)
          - a)  $P \mid Q$  and  $P \downarrow Q$ .
          - b)  $P$  and  $P \vee \neg Q$
          - c)  $(P \vee Q) \implies R$  and  $(P \implies R) \vee (Q \implies R)$
          - d)  $(P \iff Q) \iff R$  and  $P \iff R$
        6.
          - (2) Prove:  $(P \wedge Q) \implies R$  is equivalent to  $(P \wedge \neg R) \implies \neg Q$ . You might call this the *equivalence of conditional contrapositives*, because it says that if we assume  $P$ , then  $Q \implies R$  and  $\neg R \implies \neg Q$  continue to be equivalent.
        7.
          - (2) Show that  $(P \wedge \neg P) \implies Q$  is a tautology. Thus if you could ever find a proposition  $P$  for which you could prove within mathematics both  $P$  and  $\neg P$ , then any  $Q$  whatsoever would follow. This is the reason we hope that mathematics is “consistent”, that is, free of contradictions. If it isn’t, then we can prove everything with mathematics and so mathematics would have no ability to differentiate truth from falsehood.
        8.
          - (2) Suppose that a wff is a tautology. Explain in words why it is still a tautology if one or more of its atoms are replaced by other wffs. If more than one instance of a given atom is replaced, it must be replaced by the *same* wff each time. Why?

9.  $\langle 2 \rangle$  Let  $A$  be any set of premises and let  $B$  be a desired conclusion. We want to determine whether it is a valid argument that  $B$  follows from  $A$ . Using [8] and a tautology, explain why it is just as good to determine if it is a valid argument that  $\neg A$  follows from  $\neg B$ .

10.  $\langle 2 \rangle$  For each of the following arguments in words, first explain informally in words why you think it is valid or invalid. Then translate it into symbols and produce a truth table to determine its validity.

a) Either a Republican is President or a Democrat is President.

If a Democrat is President, then the Democrats control the Senate.

The Democrats control the Senate.

Therefore a Democrat is President.

b) If Bob lives in Chicago, then he lives in Illinois.  
If Bob lives in Chicago, then he lives in the USA.

Bob lives in Illinois.

Therefore Bob lives in the USA.

c) If you have a headache, then you feel nervous.  
You do not feel nervous.

Therefore you do not have a headache.

11.  $\langle 2 \rangle$  Determine whether each of the following symbolic arguments is valid.

a)  $A \implies B$       b)  $(A \vee C) \implies (B \vee C)$

$\neg A \implies C$

-----

$\neg B$                      $A \implies B$ .

-----

$C$ .

c)  $(A \implies B) \vee C$

$\neg B$

-----

$A \implies C$ .

12.  $\langle 2 \rangle$  Complete the proof of Example 2 by:

a) Induction for both parts of Example 2.

b) Using the result of [8] for the second part of Example 2.

13.  $\langle 2 \rangle$  Which of the following sentences are tautologies?

a) Without food people die.

b) If John said that he would do it and he did it, then he did it.

c) If 2 is an integer and 2 is not an integer, then the moon is made of blue cheese.

d) Canadians are North Americans.

14.  $\langle 2 \rangle$  Suppose that you need to find the truth table for a logical expression that involves just three atoms,  $P$ ,  $Q$ , and  $R$ . Write an algorithm using nested loops to list all triples of truth values for these three atoms. They should be listed in the same order as in the text. (For a more general algorithm, see [15].)

15.  $\langle 3 \rangle$

a) Sketch an algorithm to generate a truth table for a wff whose atoms are  $P_1, P_2, \dots, P_n$ . The only part you should write in detail is the part that generates the next line of values for  $P_1, P_2, \dots, P_n$  from the previous line.

b) Display the first  $n$  columns that would result from the algorithm sketched in a) when (i)  $n = 4$ ; (ii)  $n = 5$ .

c) Give a proof that your algorithm generates a truth table for any  $n$ .

## 7.4 Algorithm Verification

We touched on proving algorithms correct at various times in Chapters 1–6. The proofs of algorithm correctness given in those chapters were all informal in that they consisted of arguments in discursive English. In this section we shall discuss a much more formal approach to proofs of algorithm correctness. We believe that our previous proofs of algorithm correctness were convincing, so why is a more formal approach useful? Here are three answers:

1. Algorithms themselves are rather formal structures which are closely related to those still more formal structures, computer programs. Not only would it be nice if we could prove the correctness of programs but, if we had a formal technique for doing so, then we *might* be able to mechanize these proofs (i.e., have a computer do them). Thus being able to prove algorithms correct formally is closely related to the goal of being able to prove programs correct mechanically. We must admit, however, that research in this direction has proceeded quite slowly because achieving such mechanical proofs is inherently very difficult.
2. Nevertheless, the techniques developed to *try* to prove programs correct have proved very useful in the writing of algorithms and programs. One aspect of this approach, mentioned several times already in this book, is the importance of thinking about loop invariants when writing algorithms and proving them correct. Loop invariants are an important part of the technique for proving algorithms correct that we shall discuss in this section.
3. Finally, the formal discipline to be introduced in this section breaks the proof of an algorithm up into a number of subproofs. This approach is useful in thinking about proofs of long, complicated algorithms even when each subproof will be done informally.

*Note:* The examples in this section and in Section 7.7 are both of iterative algorithms. The formalism discussed in this section and extended in Section 7.7 is not applicable to recursive algorithms without the introduction of far more paraphernalia than we can justify here. However, as noted at the beginning of Section 2.4, 1) the standard approach to validating recursive algorithms is to do induction on the statement that the recursive procedure does what it is supposed to do, and 2) such validations tend to be simpler and more straightforward than ordinary deductive proofs of the correctness of iterative algorithms. Thus the arguments above for formalizing the proofs of algorithms are less compelling for recursive algorithms than for iterative ones.

We'll describe the ideas involved in proving algorithms correct by carrying through an example. The example we will use is Example 3, Section 7.1, where we posed the question of how to prove that the multiplication algorithm, which we've reproduced in Fig. 7.4, is correct.

The crucial step in proving an algorithm correct is to add to the algorithm **assertions** about the variables in the algorithm. These assertions take the form of one or more predicates (i.e., expressions that are true or false depending upon the values of the variables in them; we first introduced this term in Section 0.4; predicates will be the subject of Section 7.6). Each predicate contains variables of the algorithm; multiple predicates in an assertion are joined by the logical operators  $\wedge$ ,  $\vee$ , and  $\neg$ . Two of the assertions in Fig. 7.4 — at the beginning and end of the algorithm — are called the **input specification (IS)** and **output specification (OS)**, respectively. What we really want to prove is that, if the input specification is true, the execution of the algorithm allows us to assert the truth of the output specification. That is, we want to prove the theorem

|                                     |                              |
|-------------------------------------|------------------------------|
| <b>Input</b> $x$                    | $[ \text{Integer } \geq 0 ]$ |
| $y$                                 | $[ \text{Integer} ]$         |
| <b>Output</b> $prod$                | $[ = xy ]$                   |
| <b>Algorithm</b> MULT               |                              |
| $\{x \geq 0\}$                      | [Input specification]        |
| $prod \leftarrow 0; u \leftarrow 0$ |                              |
| $\rightarrow \{prod = uy\}$         | [Loop invariant]             |
| <b>repeat until</b> $u = x$         |                              |
| $prod \leftarrow prod + y$          |                              |
| $u \leftarrow u + 1$                |                              |
| <b>endrepeat</b>                    |                              |
| $\{prod = uy \wedge u = x\}$        | [Loop termination condition] |
| $\{prod = xy\}$                     | [Output specification]       |

**FIGURE 7.4**

Algorithm MULT  
with assertions.

$$\text{IS} \implies \text{OS}, \quad (1)$$

which, because of the temporal aspect of executing an algorithm, should be read:

If IS is true at the beginning of execution of the algorithm,  
then OS is true when the algorithm terminates.

The use of “assertion” is standard in the language of algorithm verification. Except for the output specification, however, assertions are just hypotheses or premises, as we have used these terms earlier in this chapter.

Now let’s use Fig. 7.4 to explain what we mean by the terms in the preceding paragraph. We distinguish assertions *about* an algorithm from the statements of the algorithm itself by placing the assertions in braces ( $\{$  and  $\}$ ). The input specification

$$\{x \geq 0\}$$

provides a constraint on the input which must be observed if the algorithm is to produce the desired output. Thus, the IS should be the *least restrictive* assertion that can be made about the input variables in the sense that any assertion that restricts the variables less than the IS would be such that the implication (1) would not be a theorem [4a]. This IS is a predicate, since  $x$  in this example can be any nonnegative integer. The truth values of this assertion and the ones that follow will depend on the current values of the variables in the algorithm. Thus we view the input specification as asserting that the particular value of  $x$  input to the program will be nonnegative.

To be completely precise and comprehensive, we should have stated the input specification as

$$\{x, y \text{ integers}, x \geq 0, -\infty < y < \infty\}. \quad (2)$$

But we want to avoid being quite so pedantic because

- we'll take the *type* of the variables (i.e., integer) to be understood; whenever this is not obvious, the type should be given as in expression (2);
- when no constraint on a variable, such as  $y$  in Fig. 7.4, is asserted, it is implicit that any value of the type is allowable; thus, the  $-\infty < y < \infty$  in expression (2) is really redundant because, once  $y$  is specified to be an integer, saying nothing else implies the range in (2).

Note, by the way, that the use of the word “specification” is natural because the IS *specifies* the character of the input as the OS does similarly for the output.

The assertions within the algorithm itself are intended to express truths about the values of the variables contained in them whenever control reaches that particular place in the algorithm. But how do we know where to put assertions in the algorithm (other than the input and output specifications, which must always be at the beginning and end, respectively)? A general rule is that the assertions should be inserted so as to break up the proof of the correctness of the entire algorithm into manageable (i.e., relatively simple) subproofs. A specific rule which follows from this is

Each loop should be preceded and followed by assertions. The former is the loop invariant; the latter, the **loop termination condition**, combines the loop invariant with the expression that is tested each time through the loop.

This rule is reasonable because, since loops are crucial portions of all significant algorithms, we should want to prove that each loop does what it is supposed to do. In the example in this section — and in the example that we shall present in Section 7.7 — each assertion except for the input and output specifications will be at the beginning or end of a loop. However, in longer algorithms, or in those which contain complicated if-structures, it is often useful to add further assertions that break up the proof of the correctness of the algorithm into more subproofs.

Now let's get to the assertions in Algorithm MULT. The assertion preceding a loop will always be the loop invariant for the loop, and it is intended to apply the first time the loop is entered and each time it is reentered. In this instance the assertion is

$$\{prod = uy\}.$$

This loop invariant asserts that each time the computation comes to the top of the repeat-endrepeat loop, the current value of *prod* must equal *uy*, using the current values of these variables. We emphasize this requirement by drawing the arrow from **endrepeat** to the assertion, not to **repeat**.

The loop termination condition

$$\{prod=uy \wedge u=x\} \tag{3}$$

is an assertion about the variables which should be true when the loop is exited. We could have written this assertion as

$$\{prod=xy\} \tag{4}$$

because the truth of (3) implies the truth of (4) (why?). We prefer (3) because the

proof that the algorithm is correct is more straightforward if the loop termination condition consists of the conjunction of the loop invariant and the expression that is tested each time through the loop. We do, however, use (4) as the output specification.

The output specification (4) is, of course, just the assertion of the desired output of the algorithm. Getting the output specification right is relatively easy; you usually know pretty well what you want your algorithm to do. On the other hand, getting the input specification right is often much harder. Specifying the conditions on the input variables that must be satisfied in order for the output to be correct may be very difficult. (Think, for example, of an algorithm to compute a payroll whose input must include all the constraints imposed by the tax laws.)

To prove the correctness of MULT, let's first name the assertions:

$$\begin{array}{ll} P_1 : x \geq 0. & [\text{Input specification}] \\ P_2 : prod = uy. & \\ P_3 : prod = uy \wedge u = x. & \\ P_4 : prod = xy. & [\text{Output specification}] \end{array}$$

What we would like to prove is that

$$P_1 \implies P_4.$$

To prove this it would be sufficient (why?) to show that

$$\begin{aligned} P_1 &\implies P_2, \\ P_2(\text{before}) &\implies P_2(\text{after}), \\ P_2 &\implies P_3, \\ P_3 &\implies P_4, \end{aligned}$$

by showing that each of these is a theorem of logic. The second theorem is necessary because we must prove that, if the loop invariant is true at any entry into the loop, then it remains true when we enter the loop the next time.

These theorems have quite a different flavor from those considered heretofore in this chapter because the assertions (e.g.,  $prod = uy$ ) contain variables that can take on any value in their underlying domain (typically some infinite subset of the integers). They cannot, therefore, be proved using truth tables (why?).

Our approach to each proof will be to give an (informal) argument which attempts to show that each assertion in the algorithm follows from its predecessor:

1. The first assertion after the input specification is the loop invariant assertion. When we first come to this assertion ( $prod = uy$ ), it is correct because both  $prod$  and  $u$  have just been set to 0. Thus  $P_1 \implies P_2$ .
2. Is  $P_2$  still true when we return to it from the end of the loop? That is, assuming that the loop invariant is true when we enter the loop, is it still true when we reenter the loop the next time? Each “next time” involves a value of  $u$  that is 1 greater than the previous one. So, if we define  $P_2(k)$  to be the claim that

$P_2$  holds for  $u=k$ ,

the theorem  $P_2(\text{before}) \implies P_2(\text{after})$  becomes the theorem

$$P_2(k) \implies P_2(k+1) \quad (5)$$

for values of  $k$  from 0 to  $x - 1$ . Does expression (5) look familiar? What we have done is recast this theorem in terms of a *finite induction* with basis case  $k = 0$ . (The induction is finite rather than infinite because  $P_2(k)$  need only be true for  $k = 0, 1, \dots, x$ ; finite inductions are usually what are needed to prove algorithms and programs correct.) If the IS is true, the basis case  $P_2(0)$  is true because  $P_1 \implies P_2$  is really  $P_1 \implies P_2(0)$ . So all we need to do to prove the theorem (5) is the inductive step. Let  $\text{prod}_k$  denote the value of  $\text{prod}$  before execution of the loop for a particular value  $u = k$ . Then the induction hypothesis is

$$\text{prod}_k = ky.$$

Because of the statements in the loop itself, the value of  $\text{prod}$  after this execution (and before the next execution) is

$$\begin{aligned} \text{prod}_{k+1} &= \text{prod}_k + y \\ &= ky + y \quad [\text{By the induction hypothesis}] \\ &= (k+1)y, \end{aligned} \quad (6)$$

which is just  $P_2(k+1)$ . This completes the proof of the theorem (5).

3. The loop termination condition must be true when we reach it because we have just shown that  $\text{prod} = uy$  each time we reach the end of the loop and  $u = x$  is just the condition which causes us to exit from the loop. Therefore  $P_2 \implies P_3$ .
4. Finally,  $P_3 \implies P_4$  is true because, as we argued above, the truth of the loop termination condition implies the truth of the output specification.

Does this complete our proof of the correctness of Algorithm MULT? Not quite. A proof of algorithm correctness requires not only that we prove  $\text{IS} \implies \text{OS}$ , but also that we prove the algorithm actually does terminate if the input specification is true. In particular, to prove the correctness of any loop, we must not only prove that the loop invariant is, indeed, invariant but we must also prove that the loop terminates. We prove that Algorithm MULT terminates as follows:

1. The only thing that could prevent its termination would be that the repeat-loop is executed endlessly.
2. Because  $u$  is initialized to 0 and  $x$  is nonnegative if the IS is true,  $u \leq x$  when the loop is first entered.
3. Each time the loop is executed,  $u$  is increased by 1. Therefore, sooner or later,  $u$  will have the same value as  $x$  and the loop will be exited, which completes the proof.

Do you see why this proof is invalid if the IS is not true [4b]? Did you notice that this is just another induction proof [4c]?

Algorithm MULT is a very simple algorithm, so you shouldn't be misled into thinking that proofs of the correctness of algorithms are always as easy and straightforward as this one. They're not. But even when they are much more difficult, the discipline of trying to prove an algorithm correct by inserting assertions in it *as you develop it* can be a valuable aid in constructing correct algorithms. In Section 7.7 we'll give a somewhat more difficult example of proving an algorithm correct.

## Problems: Section 7.4

1. ⟨2⟩ In this section, “assertion” and “hypothesis” mean the same thing. As used in most writing, how do they differ?
2. ⟨2⟩ Write an algorithm which inputs a real number  $x$  and a positive integer  $n$  and computes  $x + n$  by repeated addition of 1. Insert appropriate assertions and use them to prove that the algorithm is correct and terminates.
3. ⟨2⟩ Algorithm RECIPROCAL below is a trivial algorithm to compute the reciprocal of a nonzero number. Obviously the correctness of Algorithm RECIPROCAL depends on the input meeting the input specification. Use assertions to prove the algorithm correct. (The assertions and the proof are quite simple. The point of the exercise is to show how the input specifications are used in the verification process.)
4. ⟨2⟩
  - a) Suppose that the input specification for MULT is changed to assert only that  $x$  is an integer. Where does the proof of correctness break down?
  - b) Why is the proof of the termination of MULT invalid if the input specification is false?
  - c) Restate the proof of the termination of MULT as an induction.
5. ⟨2⟩ Write input specifications for the following algorithms.
  - a) An algorithm to determine whether three positive integers satisfy the Pythagorean theorem
  - b) Algorithm BINSEARCH (Section 1.5)
  - c) Algorithm PERMUTE-1 (Section 4.9)
6. ⟨2⟩ Write output specifications for each of the algorithms in [5]. You'll have to use ellipsis for c). In Section 7.6 we'll develop an easier way to express this OS.
7. ⟨3⟩ For each of the following write the algorithm requested, insert appropriate assertions in it, and then prove that your algorithm is correct.
  - a) Determine whether a nonnegative integer  $n$  is even by successive subtraction of 2 from  $n$ .
  - b) Sum  $n$  given numbers.
  - c) Find the greatest power of 2 less than or equal to a given positive integer  $n$ .
  - d) Given an array of numbers  $a_1, a_2, \dots, a_n$ , find the subscript of the largest number in the array.

## Algorithm

|                             |                      |
|-----------------------------|----------------------|
| <b>Input</b> $x$            | $[x \neq 0]$         |
| <b>Output</b> $y$           | [Reciprocal of $x$ ] |
| <b>Algorithm</b> RECIPROCAL |                      |
| $y \leftarrow 1/x$          |                      |

4. ⟨2⟩
  - a) Suppose that the input specification for MULT is changed to assert only that  $x$  is an integer.

## 7.5 Boolean Algebra

It is remarkable that the subject of this chapter — originally developed a century and a half ago by George Boole to provide a formal way to describe and manipulate logical propositions in an algebraic framework — has so many applications today. Probably the most important application is in the design of digital circuits. At the

end of this section we'll solve the problem introduced in Example 2, Section 7.1, concerning the design of a complements.

Essentially, Boolean algebra is a reformulation of the propositional calculus that should remind you of the algebra you learned in high school. As a result, applications are much easier to do using Boolean algebra than using the propositional calculus directly. Moreover, the notation and perspective on truth values and variables in Boolean algebra provide an illuminating contrast to those used in the propositional calculus.

First in this section we'll introduce the language of Boolean algebra and then we'll show how to manipulate expressions in Boolean algebra and prove theorems. Finally we'll show how to apply the paraphernalia we'll have developed to the design and simplification of electronic circuits.

## Boolean Algebra: Constants, Variables and Expressions

**Algebra**, as used in the title of this section, has a more general meaning than what you know as high school algebra. In mathematics, an algebra consists of

- a set  $S$ , the *constants* of the algebra, and
- the definition of operators, which map tuples (see Section 0.1)

$$(s_1, s_2, \dots, s_p), \quad s_i \in S$$

to elements of  $S$ .

Thus, for example, if  $p = 2$ , we would have a binary operator, which maps  $(s_1, s_2)$  into  $s$ , which is an element of  $S$ . In general, an operator on a  $p$ -tuple is called a  **$p$ -ary operator**.

In Section 7.2 on the propositional calculus,  $S$  was the set of all propositions. That is, variables such as  $P, Q, R, \dots$  ranged over the set of all propositions, and the operators were binary (as in the case of **and** and **or**) or unary (in the case of **not**). This is analogous to high school algebra, where the set  $S$  is usually the (infinite) set of all the real numbers and, although most operators are binary, the operator “ $-$ ” can be unary, as in  $-a + b$ .

However, since each proposition has a truth value T or F, it is more convenient in the context of this section to take  $S$  to be the set {T, F}. Thus **and** and **or** are binary operators ( $p = 2$ ) that map two elements from {T, F} into one element of this set, while **not** is a unary operator ( $p = 1$ ) that maps one element of this set into the other element. To go one step further, in Boolean algebra it is convenient to use the constants 1 and 0 instead of, respectively, T and F, since one of the main applications of Boolean algebra is to the design of computer circuits and computers normally operate on binary digits (bits), 0 and 1. Thus in Boolean algebra  $S = \{0, 1\}$ .

Instead of using five operators corresponding to the five logical operators ( $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\implies$ ,  $\iff$ ) that we previously emphasized, we shall only use three operators corresponding to  $\wedge$ ,  $\vee$ , and  $\neg$  with equivalence handled using the familiar equal sign ( $=$ ) as described below. Implication is not as directly useful in Boolean algebra as

it is in the propositional calculus. In addition, it can, if need be, be represented using  $\wedge$ ,  $\vee$  and  $\neg$  [9].

In this section we'll represent the  $\wedge$ ,  $\vee$  and  $\neg$  operators by, respectively,  $\cdot$ ,  $+$  and  $\bar{\phantom{x}}$  (overbar) because this notation is traditional in Boolean algebra. Nonetheless, we shall continue to refer to these three operators as the **and**, **or**, and **not** operators, respectively.

The operators  $\cdot$ ,  $+$  and  $\bar{\phantom{x}}$  are defined precisely analogously to the corresponding operators in the propositional calculus. Table 7.9 gives the truth tables for the three Boolean operators. Note that the format of these tables is different from that in Table 7.2, Section 7.2 and is more like operation tables in abstract algebra.

In particular, you should note that  $1 + 1$  in Boolean algebra is 1. It couldn't be 2 because there is no value 2 in Boolean algebra. It is defined to be 1 because  $1 + 1 = 1$  in Boolean algebra corresponds to  $T \vee T = T$  in the propositional calculus (i.e., the conjunction of two propositions, each with truth value T, is T).

---

**TABLE 7.9**  
**Truth Tables for  $+$ ,  $\cdot$ , and  $\bar{\phantom{x}}$**

---

|                       |                       | <u><math>p + q</math></u> |   | <u><math>p \cdot q</math></u> |                       |   |   |
|-----------------------|-----------------------|---------------------------|---|-------------------------------|-----------------------|---|---|
| <u><math>p</math></u> | <u><math>q</math></u> | 0                         | 1 | <u><math>p</math></u>         | <u><math>q</math></u> | 0 | 1 |
| 0                     | 0                     | 0                         | 1 | 0                             | 0                     | 0 | 0 |
| 1                     | 1                     | 1                         | 1 | 1                             | 0                     | 0 | 1 |

---

Corresponding to the atoms,  $P, Q, R, \dots$ , which were the variables of the propositional calculus and which could vary over the set of all propositions, each in turn having truth values, we have Boolean variables, which we will denote by  $p, q, r, \dots$  and which are the atoms of Boolean algebra. These vary over the truth values themselves, namely 0 or 1. By analogy with our use of  $A, B, C, \dots$  to express either atomic or compound propositions in the propositional calculus, we shall use  $a, b, c, \dots$  in Boolean algebra for either atomic expressions (i.e., individual Boolean variables without operators) or compound expressions (i.e., containing both variables and operators).

The first four rules for forming valid expressions in Boolean algebra are precisely the same as those for forming wffs in the propositional calculus:

1. An atom is a Boolean expression.
2. If  $a$  is a Boolean expression, then so is  $\bar{a}$ .
3. If  $a$  and  $b$  are Boolean expressions, then so are  $a \cdot b$  and  $a + b$ .
4. If  $a$  is a Boolean expression, then so is  $(a)$ .

Rule 4 says that parentheses *can* be added whenever you wish, as in the corresponding rule in Section 7.2. And, as in Section 7.2, parentheses *should* be inserted if they will improve readability. But if we left it at this, there would be a plethora

of parentheses in the expressions in this chapter since these expressions tend to be more complex than those in Section 7.2. Instead we shall avoid all possible ambiguities by using the convention of always writing  $pq$  in place of  $p \cdot q$ . (Note, in particular, that, since our Boolean variables will always be single letters,  $pp$  should be interpreted as  $p \cdot p$ .)

How does this convention avoid the problem of ambiguity? Because, whereas it is not immediately obvious whether  $p + q \cdot r$  should be interpreted as  $(p + q) \cdot r$  or  $p + (q \cdot r)$ , there can be no doubt about  $p + qr$  since this form clearly implies that the “and” is to be applied before the “or”. Thus, in effect, we have another rule

5. The Boolean “and”,  $\cdot$ , has precedence over the Boolean “or”,  $+$ , in the sense that all “and”s inside a given set of parentheses are evaluated before any “or”s.

This rule is, of course, the same as the convention normally used with ordinary multiplication of numbers. Indeed, the  $\cdot$  operator in Boolean algebra is sufficiently akin to multiplication in ordinary algebra and the  $+$  is sufficiently akin to addition, that  $\cdot$  is sometimes called the “logical multiplication operator” and  $+$  the “logical addition operator”. We shall, in fact, sometimes refer to  $a + b$  as a sum and to  $ab$  as a product.

How about the “not” operator,  $\neg$ ? How come it is not covered in Rule 5? Actually, it could be but we take it as obvious that, for example,  $\bar{p}$  requires that  $p$  be negated before handling any other operators associated with  $p$ . This corresponds to our assumed precedence rule in Section 7.2 that  $\neg$  has precedence over every binary operator.

Do these five rules suffice? How about ambiguities like that in (4) in Section 7.2 where, without parentheses, the order in which the operators are applied makes a difference? In fact, no such ambiguity is possible in Boolean algebra, as we have presented it, since the evaluation of repetitions of both binary operators,  $+$  and  $\cdot$ , is independent of the order in which they are evaluated (see (v) and (vi) of Theorem 1 below).

Thus, these five rules produce unambiguous wffs. As with expressions in the propositional calculus, you are encouraged to insert more parentheses than may be necessary if doing so will enable easier readability of your expressions. Boolean expressions may be regarded as **Boolean functions** which map  $k$ -tuples of 0’s and 1’s, where  $k$  is the number of variables in the expression, to a 1-tuple, which is either 0 or 1. Thus when  $p = 1$ ,  $q = 0$ , and  $r = 1$ , the expression

$$\bar{p}q + p\bar{q}r + p\bar{r}$$

maps  $\{1, 0, 1\}$  to  $\{1\}$ , since 1 is the value of the expression when  $p = 1$ ,  $q = 0$ , and  $r = 1$  (why?).

You may have noted that Boolean algebra is closely related to set algebra in Section 0.1 (although we didn’t call it an algebra there). In set algebra

- the objects, that is the domain of  $S$ , are any sets whatsoever in contrast to the domain of Boolean algebra,  $\{0, 1\}$ ;

- the operations of intersection, union and complement correspond to  $\cdot$  (**and**),  $+$  (**or**), and  $\neg$  (**not**) in Boolean algebra. Note, however, the unfortunate confusion of notation: For set intersection and union, we use a notation,  $\cap$  and  $\cup$ , akin to that in the propositional calculus, whereas for set complement we use the same notation as in Boolean algebra. We do this not to confuse but because it is conventional.

In fact, if we interpret 1 in Boolean algebra as the universal set of set algebra and 0 as the empty set, the theorems of Boolean algebra are exactly the same as the theorems of set algebra. For example,  $P \cap \overline{P} = \emptyset$  in set algebra becomes  $p\bar{p} = 0$  in Boolean algebra (discussed in the next subsection; also see [2–3] for more translations). This is no coincidence. There are, indeed, many *Boolean algebras*, all of which obey the same rules. Our Boolean algebra (the original) and set algebra are two examples.

## Boolean Theorems

Recall that the valid arguments of the propositional calculus are just those arguments for which the associated implications are tautologies. A tautology of the propositional calculus corresponds in Boolean algebra to an expression which evaluates to 1 for any values of the atoms in it, since 1 corresponds to True. But using only the three operators  $\cdot$ ,  $+$  and  $\neg$ , almost all expressions in Boolean algebra that always evaluate to 1 are pretty uninteresting (e.g.,  $p + \bar{p}$  or  $pq + p\bar{q} + \bar{p}q + \bar{p}\bar{q}$ ). Therefore, we'll move on to *Boolean equations*. An example of a Boolean equation is

$$pq + pr + \overline{q}r = pq + \overline{q}r, \quad (1)$$

where as always in mathematics, “=” means the two sides are asserted to have the same value (here either 0 or 1). Actually, in our Boolean algebra “=” has the same meaning as  $\iff$  (why?); we use  $=$  instead of  $\iff$  as part of the general Boolean approach of using more familiar algebraic notation. That “=” stands for  $\iff$  has one perhaps surprising consequence. Not only do both sides of a Boolean equation have a value (either 0 or 1) but so does the whole equation (1 if both sides are equal, 0 otherwise). Thus we can talk about an equation that is true (equals 1) for all values of the atoms. Such a Boolean tautology is called a **Boolean theorem** or **identity**.

### EXAMPLE 1

Show that (1) is a theorem.

**Solution** Table 7.10 is a truth table in which both sides of (1) are evaluated. Since the last two columns have the same value in each row, both sides of the equality (1) are the same for all values of the atoms and, therefore, (1) is a theorem. ■

Example 1 gives essentially the same proof by truth table that we did in Example 1 in Section 7.3.

We have just demonstrated, in effect, that any Boolean theorem can be proved (or disproved) using truth tables. But Boolean theorems can also be proved deductively by algebraic manipulation. In order to do this, it is useful to have available

**TABLE 7.10**  
**Truth Table for  $pq + pr + \bar{q}r = pq + \bar{q}r$**

| $p$ | $q$ | $r$ | $pq$ | $pr$ | $\bar{q}r$ | $pq + pr + \bar{q}r$ | $pq + \bar{q}r$ |
|-----|-----|-----|------|------|------------|----------------------|-----------------|
| 0   | 0   | 0   | 0    | 0    | 0          | 0                    | 0               |
| 0   | 0   | 1   | 0    | 0    | 1          | 1                    | 1               |
| 0   | 1   | 0   | 0    | 0    | 0          | 0                    | 0               |
| 0   | 1   | 1   | 0    | 0    | 0          | 0                    | 0               |
| 1   | 0   | 0   | 0    | 0    | 0          | 0                    | 0               |
| 1   | 0   | 1   | 0    | 1    | 1          | 1                    | 1               |
| 1   | 1   | 0   | 1    | 0    | 0          | 1                    | 1               |
| 1   | 1   | 1   | 1    | 1    | 0          | 1                    | 1               |

Note that we list the values of the atoms in their natural binary order, whereas in the truth tables of Section 7.2 we always started with a row of T's (= 1's). This is merely conventional and has no other significance.

a set of basic *identities* and then use these to prove other theorems. In proceeding this way we'll also discuss some unusual aspects of these identities.

The following theorem states the most important identities in Boolean algebra:

**Theorem 1.** If  $a$ ,  $b$ , and  $c$  are Boolean expressions, the following equations are Boolean theorems:

- (i)  $a + 0 = a.$  } [Additive identity]
- (ii)  $a1 = a.$  } [Multiplicative identity]
- (iii)  $a + b = b + a.$  } [Commutative laws]
- (iv)  $ab = ba.$  } [Commutative laws]
- (v)  $(a + b) + c = a + (b + c).$  } [Associative laws]
- (vi)  $(ab)c = a(bc).$  } [Associative laws]
- (vii)  $a(b + c) = ab + ac.$  }
- (viii)  $a + bc = (a + b)(a + c).$  } [Distributive laws]
- (ix)  $a + \bar{a} = 1.$  }
- (x)  $a\bar{a} = 0.$  } [De Morgan's laws]
- (xi)  $\overline{a + b} = \bar{a}\bar{b}.$  }
- (xii)  $\overline{ab} = \bar{a} + \bar{b}.$  } [De Morgan's laws]

In the remainder of this section we'll refer to these equations by using the lowercase Roman numerals.

We can easily prove each of these identities using truth tables [1]. Eqs. (ix) and (x) have no analogs in ordinary algebra but are true in Boolean algebra because one of  $a$  and  $\bar{a}$  is 1 and the other is 0.

Of the first eight identities in Theorem 1, the only one which would not have been familiar to you from high school algebra is the second distributive law (viii), which has no analog in ordinary algebra. Whereas the first distributive law (vii), like its counterpart in ordinary algebra, says that multiplication distributes over addition, (viii) says that addition distributes over multiplication. Does it disturb you that in Boolean algebra

$$(a + b)(a + c) = a + bc,$$

whereas in ordinary algebra [5]

$$(a + b)(a + c) = a^2 + ab + ac + bc ? \quad (2)$$

It shouldn't because, in any mathematical system, of which ordinary algebra and Boolean algebra are two examples, we are free to define the rules of the game as we see fit. In Boolean algebra the definitions of the **and** and **or** operators determine that (viii) is a theorem, while in ordinary algebra the definitions of addition and multiplication determine that Eq. (2) is a theorem. Of course, if the "games" we play in mathematics are to be useful practically, then the rules we define must correspond to some sphere of application.

The distributive law (viii) is not the only "weird" fact in Boolean algebra. For instance,

$$a + a = a \quad \text{and} \quad aa = a \quad (3)$$

also look weird compared to high school algebra, but in fact you can easily justify them in terms of what  $+$  and  $\cdot$  mean in Boolean algebra, and after reading further you will also be able to prove them deductively from the identities in Theorem 1 [4].

The six pairs of identities in Theorem 1 exemplify another fact about Boolean algebra. In any Boolean theorem, if logical addition (i.e.,  $+$ ) is replaced everywhere by logical multiplication (i.e.,  $\cdot$ ) and  $\cdot$  by  $+$ , and if 0 is replaced by 1 and 1 by 0, the result is still a theorem. We consider this **principle of duality** further in a problem [11].

Why did we single out the twelve identities in Theorem 1? It is easy to justify including the first ten because it turns out that these ten are a *complete* set; every Boolean theorem can be proved using them. Thus they could be construed to be the **axioms** of Boolean algebra since, starting from them, all theorems of Boolean algebra can be proved. Even if you wish to prove that a Boolean expression, such as  $p + \bar{p}$ , always evaluates to 1 (so that it is in effect a *Boolean tautology*), you can use Theorem 1 by setting the expression equal to 1, thus obtaining an equation.

The last two identities in Theorem 1, De Morgan's laws, are given a place equal in prominence to the others because they are so often useful in Boolean algebra.

We now show by some examples what we mean by a **deduction** in Boolean algebra. Basically, one does algebraic manipulations, including substitutions of

equals, but only using the rules in Theorem 1 or other results you have already proved by this method. For our first example, there are no previous results to use. So here is a deduction that  $1 + 1 = 1$ . On the right in brackets we indicate which portions of Theorem 1 are used in each line.

$$\begin{aligned}
 1 &= 1 + 0 && [(i) \text{ with } a = 1] \\
 &= 1 + 0 \cdot 1 && [\text{Applying (ii) to second term}] \\
 &= (1 + 0)(1 + 1) && [(viii) \text{ with } a = c = 1, b = 0] \\
 &= 1(1 + 1) && [\text{Applying (i) to first term}] \\
 &= (1 + 1)1 && [(iv) \text{ with } a = 1 + 1, b = 1] \\
 &= 1 + 1. && [(ii) \text{ with } a = 1 + 1]
 \end{aligned}$$

In this first example we limited ourselves to one step per line. To speed things up below, we will allow two or three steps per line and just state which rules are used.

This deduction was pretty tricky, especially when the result is obvious from the meaning (True or True is True) or from a truth table. Why is it so tricky and why bother to do it?

It is tricky for several reasons. First, some of the rules are unfamiliar (“weird”). Second, many familiar rules are unavailable. For instance, the most common step in showing that  $x$  equals something in ordinary algebra is cancellation, e.g., from  $a + x = a$  you cancel (subtract the  $a$ 's) and get  $x = 0$ . Not only is there no such rule in Theorem 1 but cancellation is simply not valid in Boolean algebra [6]. Third, the first deductions in any system are often the hardest because you haven't yet worked up a large set of facts to work with. The problems will provide many more useful Boolean facts. Fortunately, the deductions one most often has to do (the expansion to normal forms discussed in the next subsection on Karnaugh maps) are relatively easy, so we are not going to belabor general deductions.

Why bother with deductions? We've noted that there are in fact many Boolean algebras. Whereas the set  $S$  for the original Boolean algebra is small ( $S = \{0, 1\}$ ), for most Boolean algebras  $S$  is large or infinite. For example, for set algebra  $S$  is infinite. Therefore, for most Boolean algebras a finite decision procedure like truth tables is either unavailable or too long. Therefore, it is important to have another method, and deduction is it.

Here is a second, somewhat subtler example of how Theorem 1 can be used to prove a Boolean theorem.

**EXAMPLE 2** Prove that  $a = \bar{\bar{a}}$ .

**Solution** How should we begin? Well, since we know we want bars, we start by multiplying  $a$  by 1 in a form that uses bars.

$$\begin{aligned}
 a &= a1 && [(ii)] \\
 &= a(\bar{a} + \bar{a}) && [(ix) \text{ with } a \text{ replaced by } \bar{a} \text{ and then (iii)}] \\
 &= a\bar{a} + 0 && [(vii) \text{ and (x)}]
 \end{aligned}$$

$$\begin{aligned}
 &= \bar{\bar{a}}a + 0 && [(iv)] \\
 &= \bar{\bar{a}}a + \bar{\bar{a}}\bar{a} && [(\text{x}), \text{ with } a \text{ replaced by } \bar{a}, \text{ and (iv)}] \\
 &= \bar{\bar{a}}(a + \bar{a}) && [(vii)] \\
 &= \bar{\bar{a}}1 && [(ix)] \\
 &= \bar{\bar{a}}. \blacksquare && [(ii)]
 \end{aligned}$$

The problems provide some practice both with the method of Example 2 and with using truth tables to prove theorems [10, 12].

## Boolean Simplification and Karnaugh Maps

We noted in Section 7.3 that one purpose of deductions in the propositional calculus is to take a given wff and simplify it (see Example 1 in that section). Actually, such simplification is more common in Boolean algebra because the Boolean algebra formulation is normally used in designing electronic circuits. But what do we mean by *simplification*, a term we have thus far left to your intuition? For our purposes here we define the *complexity of an expression* as the number of Boolean operators in it. (Don't forget the implied "and"s in Boolean expressions like  $ab$ .) Thus to simplify an expression is to reduce the number of operators in it.

For example, consider the Boolean theorem

$$\bar{p} + q + p\bar{q} = 1.$$

To prove this Boolean theorem deductively as in Example 2, we would write:

| Derivation                    | Justification                                                                                                          |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------|
| $\bar{p} + q + p\bar{q}$      |                                                                                                                        |
| $= \bar{p}\bar{q} + p\bar{q}$ | $\bar{p} + q = \bar{p}\bar{q}$ by (xii) with $b$ replaced by $\bar{b}$ and<br>using $\bar{\bar{b}} = b$ from Example 2 |
| $= 1$                         | By (ix) with $p\bar{q}$ for $a$                                                                                        |

This derivation takes an expression with five operators ( $\neg$ ,  $+$ ,  $+$ ,  $\cdot$ , and  $\neg$ ) and simplifies it to an expression with none.

The approach in the simplification above has one important drawback. It is not systematic in the sense that the steps that reduce  $\bar{p} + q + p\bar{q}$  to 1 are not clear at all when you start. It takes considerable ingenuity to decide just which identity to apply at each stage.

Another considerably more systematic method of simplifying Boolean expressions is through the use of **Karnaugh maps**. The idea of a Karnaugh map is connected to the idea of writing a Boolean expression in **disjunctive normal form (DNF)**. An expression in DNF is a sum of products in which each variable in the entire expression appears *exactly once* in each product, complemented or uncomplemented. For example, here is a three-variable expression in disjunctive normal form:

$$\bar{p}qr + \bar{p}q\bar{r} + p\bar{q}r.$$

Each product in a DNF expression corresponds to one row of a truth table for the entire expression — the row in which that product is true and which is, therefore, one of the rows in which the entire expression is true. For example,  $\bar{p}q\bar{r}$  corresponds to the row in which  $p = 0$ ,  $q = 1$ , and  $r = 0$ .

Note that when we apply (vii) of Theorem 1 to the preceding expression, it becomes

$$\bar{p}q(r + \bar{r}) + p\bar{q}r = \bar{p}q + p\bar{q}r. \quad [\text{Since } r + \bar{r} = 1]$$

Thus the DNF version of a Boolean expression is usually not its simplest form but, as you will see, we can use it as an intermediate stage when simplifying Boolean expressions.

### EXAMPLE 3

Convert the Boolean expression

$$pq + pr + \bar{q}r$$

to DNF.

**Solution** We will, in effect, present an algorithm for converting any Boolean expression to DNF although, for the sake of brevity, we won't state this algorithm in our usual notation.

1. In the first step we insert, for each variable missing from any term, the sum of that variable and its complement. For this example we get

$$pq(r + \bar{r}) + p(q + \bar{q})r + (p + \bar{p})\bar{q}r.$$

This expression must be equivalent to the original one because the sum of any variable and its complement is 1 and the product of 1 and any expression is the expression.

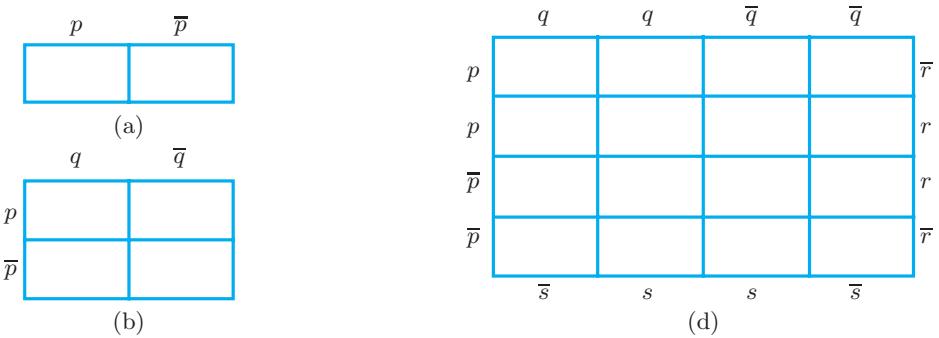
2. Then we apply the distributive law (vii) and delete any terms which appear more than once (because  $p + p = p$ ). In our example we obtain

$$\begin{aligned} pqr + pq\bar{r} + pqr + p\bar{q}r + p\bar{q}r + \bar{p}\bar{q}r \\ = pqr + pq\bar{r} + p\bar{q}r + \bar{p}\bar{q}r. \end{aligned} \tag{4}$$

The resulting expression is still equivalent to the original one and is now in DNF. ■

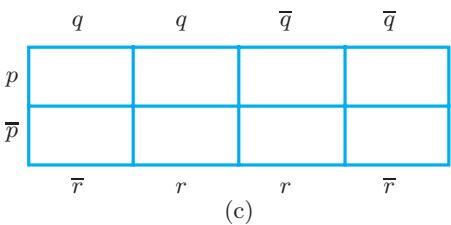
An alternative algorithm for converting an expression to DNF would be to write the truth table for the expression and then, for each row where the value of the expression is 1, create the product that is 1 just for that row (as we discussed two paragraphs before Example 3) and then sum these products to get the DNF [15, 19]. For expressions with relatively few terms but many variables, this approach would be quite tedious.

If you thought we were trying to simplify the expression in Example 3 (and we are), then with (4) we seem to have made things worse. But bear with us. The next step is to convert the expression in DNF to a Karnaugh map. One-, two-,



**FIGURE 7.5**

Karnaugh map templates: (a) 1-variable template; (b) 2-variable template; (c) 3-variable template; (d) 4-variable template.



three-, and four-variable Karnaugh map templates are shown in Fig. 7.5. The key to the construction of these maps is that each box should correspond to a different possible term in a DNF expression. If there are  $n$  variables in a Boolean expression, there are  $2^n$  possible terms in the DNF (why?), which explains the number of boxes in each part of Fig. 7.5. Note that the labels of each box in each part of the figure are chosen so that their product is one of the possible terms in a DNF expression. For example, in the three-variable template, the upper right-hand box corresponds to the term  $p\bar{q}\bar{r}$ .

#### EXAMPLE 4

Take the DNF output of Example 3 and simplify it using a Karnaugh map.

**Solution** A Karnaugh map template is converted into a Karnaugh map for a particular Boolean expression by inserting 1's in each box corresponding to a term of the DNF for this expression. This step is shown in Fig. 7.6 for expression (4).

The pattern of *regions* in the Karnaugh map can be used to simplify Boolean expressions. For example, Fig. 7.6 shows two  $2 \times 1$  regions outlined in black. The terms corresponding to the upper left-hand  $2 \times 1$  region are  $pqr$  and  $pq\bar{r}$ . As two of the three parts of each term are the same ( $pq$ ) and the other appears uncomplemented in one term and complemented in the other ( $r$  and  $\bar{r}$ ), the sum of these two terms is

$$pqr + pq\bar{r} = pq(r + \bar{r}) = pq. \quad (5)$$

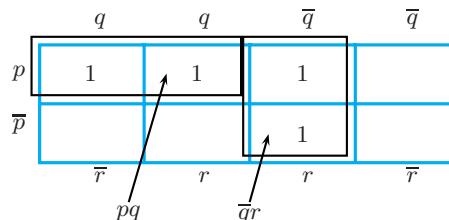
Similarly for the  $2 \times 1$  region at the right, the sum of the two terms is

$$p\bar{q}r + \bar{p}\bar{q}r = (p + \bar{p})\bar{q}r = \bar{q}r. \quad (6)$$

When we put Eqs. (5) and (6) together, the four-term DNF expression in Eq. (4) simplifies as follows:

**FIGURE 7.6**

Karnaugh map  
for expression (4),  
Example 3.



$$\begin{aligned}
 & pqr + p\bar{q}r + p\bar{q}\bar{r} + \bar{p}\bar{q}\bar{r} \\
 &= pq(r + \bar{r}) + (p + \bar{p})\bar{q}r \\
 &= pq + \bar{q}r. \blacksquare
 \end{aligned}$$

Putting Examples 3–4 together, we conclude that

$$pq + pr + \bar{q}r = pq + \bar{q}r,$$

a reduction from 6 operators to 4. Incidentally, this simplification is the one we obtained in Example 1, Section 7.3, using a truth table.

The Karnaugh map approach provides an approach to simplification of Boolean expressions which, at first glance, may seem to rely on human visual pattern recognition of map regions. However, it isn't hard to imagine designing an algorithm which, given a Boolean expression as input, forms its DNF equivalent, then searches for patterns in the corresponding Karnaugh map, and, finally, simplifies the original expression, although perhaps not to the greatest possible extent.

Here is a second example of the use of Karnaugh maps, which illustrates some other aspects of these maps.

### EXAMPLE 5

Simplify the DNF expression whose Karnaugh map is shown in Fig. 7.7a.

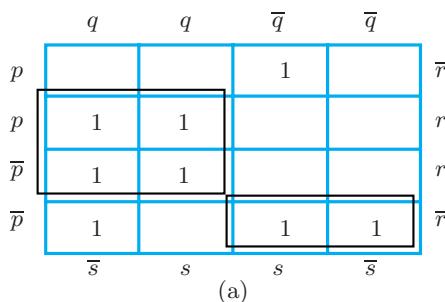
**Solution** The  $2 \times 2$  region corresponds to four terms, all of which contain  $q$  and  $r$  and which otherwise contain the four possible combinations of  $p$  and  $s$ , complemented and uncomplemented:

$$pqrs, \quad pqr\bar{s}, \quad \bar{p}qrs, \quad \text{and} \quad \bar{p}qr\bar{s}.$$

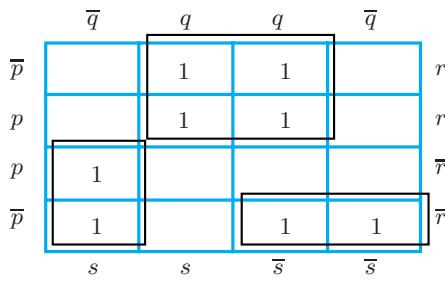
Therefore, the sum of these four terms can be simplified to  $qr$ . If you don't see this, write the sum of the four terms and then simplify it by factoring out  $(p+\bar{p})(s+\bar{s})$ . The general rule is that, in any  $2 \times 2$  region, the variables which are not constant throughout the region drop out [21]. In the  $2 \times 1$  region in Fig. 7.7a, the  $s$  part of the two terms drops out, leaving  $\bar{p}\bar{q}\bar{r}$ . For the remaining two boxes, no simplification seems possible, so we get the expression

$$qr + \bar{p}\bar{q}\bar{r} + pqr\bar{s} + \bar{p}qr\bar{s}.$$

Is this the simplest possible expression that can be obtained from the Karnaugh map in Fig. 7.7a? No, it's not, and the reason is that there are two other  $2 \times 1$  regions that we could have used instead of the one shown in Fig. 7.7a. The way the regions appear in the Karnaugh map depends on which of the many possible ways to label the boxes we choose. Figure 7.7b shows another labeling, in which once again



(a)



(b)

**FIGURE 7.7**

Karnaugh maps for Example 5.

the 16 boxes are labeled by the 16 possible combinations of four variables. The 1's in Fig. 7.7b correspond to precisely the same terms as the 1's in Fig. 7.7a. The  $2 \times 2$  square corresponds, as before, to  $qr$ . But now there are two  $2 \times 1$  rectangles, which correspond to  $\bar{q}\bar{r}s$  and  $\bar{p}\bar{r}\bar{s}$ , both different from the  $2 \times 1$  term we obtained using the map in Fig. 7.7a. Thus we may write the complete expression as

$$qr + \bar{q}\bar{r}s + \bar{p}\bar{r}\bar{s},$$

which is certainly simpler than the preceding one. ■

It's a bit unfortunate if the expression we get depends on how we label the boxes in the Karnaugh map. Could we have generated the simpler expression using Fig. 7.7a? Yes, we could have, simply by recognizing that a  $2 \times 1$  rectangle which enables two four-variable terms to be combined into one three-variable term need not necessarily be formed from two adjacent squares. The pattern can also be formed by a *wraparound* from the bottom to the top or the right to the left or vice versa. Another way of saying the same thing is that we can consider opposite borders of the map to be contiguous. Then, referring to Fig. 7.7a and ignoring the  $2 \times 1$  rectangle of actually contiguous squares, we have two other  $2 \times 1$  rectangles: The first contains the  $p\bar{q}\bar{r}s$  and  $\bar{p}\bar{q}\bar{r}s$  squares; the second contains the  $\bar{p}q\bar{r}\bar{s}$  and  $\bar{p}\bar{q}\bar{r}\bar{s}$  squares. Using these rectangles leads to the expression obtained directly from Fig. 7.7b.

How do we know which rectangles and squares to choose to get the simplest possible expressions? This isn't an easy question, and we won't attempt to answer

it here. Dealing with Karnaugh maps and simplifying Boolean expressions involves a great deal more than we can possibly present here. However, some further topics are covered in [23, 24].

To conclude this section we return now to Example 2 in Section 7.1 which allows us to illustrate several additional issues related to Karnaugh maps.

## EXAMPLE 6

Find Boolean expressions for the complements outputs in terms of the inputs in Example 2, Section 7.1.

**Solution** In Table 7.11 and Fig. 7.8, we have repeated Table 7.1 and Fig. 7.1, except that we have added some labels and displayed the standard symbols for the **logic circuits** which implement the “and”, “or” and “not” operations. Our object is to determine what needs to go in the rectangle in Fig. 7.8, so that the outputs will be correct for any inputs which represent the digits 0–9. The question we need to answer is: What Boolean function of the inputs  $b_8$ ,  $b_4$ ,  $b_2$ , and  $b_1$  is each of the outputs  $c_8$ ,  $c_4$ ,  $c_2$ ,  $c_1$ ? If we knew the answer to this question, we could — as you’ll soon see — quite easily design the desired logic circuit.

---

**TABLE 7.11**  
**Complementing a Decimal Digit**

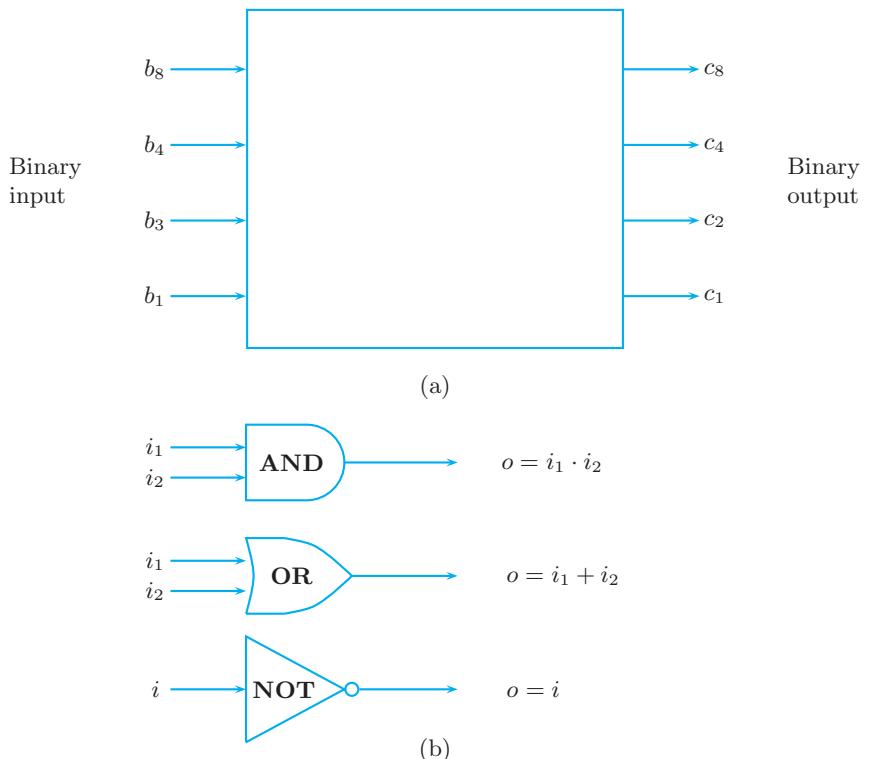
---

| Decimal | Input Digit |       |       |       | Output Digit |       |       |       | Decimal |
|---------|-------------|-------|-------|-------|--------------|-------|-------|-------|---------|
|         | $b_8$       | $b_4$ | $b_2$ | $b_1$ | $c_8$        | $c_4$ | $c_2$ | $c_1$ |         |
| 0       | 0           | 0     | 0     | 0     | 1            | 0     | 0     | 1     | 9       |
| 1       | 0           | 0     | 0     | 1     | 1            | 0     | 0     | 0     | 8       |
| 2       | 0           | 0     | 1     | 0     | 0            | 1     | 1     | 1     | 7       |
| 3       | 0           | 0     | 1     | 1     | 0            | 1     | 1     | 0     | 6       |
| 4       | 0           | 1     | 0     | 0     | 0            | 1     | 0     | 1     | 5       |
| 5       | 0           | 1     | 0     | 1     | 0            | 1     | 0     | 0     | 4       |
| 6       | 0           | 1     | 1     | 0     | 0            | 0     | 1     | 1     | 3       |
| 7       | 0           | 1     | 1     | 1     | 0            | 0     | 1     | 0     | 2       |
| 8       | 1           | 0     | 0     | 0     | 0            | 0     | 0     | 1     | 1       |
| 9       | 1           | 0     | 0     | 1     | 0            | 0     | 0     | 0     | 0       |

---

Note that the subscript associated with each binary digit is the value of the power of 2 associated with that place in positional notation.

We'll solve this problem two ways. First we'll use Karnaugh maps and, as an example, we'll focus on  $c_4$ . Referring to Table 7.11, we observe that  $c_4$  equals 1 in only four positions. For example, the 1 in the  $c_4$  column in the third row (decimal input 2, output 7) corresponds to the input values



**FIGURE 7.8**

(a) Designing a completer;  
 (b) available logic circuits.

$$b_8 = 0, \quad b_4 = 0, \quad b_2 = 1, \quad \text{and} \quad b_1 = 0,$$

for which the product

$$\bar{b}_8 \bar{b}_4 b_2 \bar{b}_1 = 1.$$

Corresponding to each of the other three 1's in the  $c_4$  column is a similar product of the  $b_i$ 's that equals 1. Now suppose that we logically add these four products together to obtain

$$\bar{b}_8 \bar{b}_4 b_2 \bar{b}_1 + \bar{b}_8 \bar{b}_4 b_2 b_1 + \bar{b}_8 b_4 \bar{b}_2 \bar{b}_1 + \bar{b}_8 b_4 \bar{b}_2 b_1. \quad (7)$$

The crucial point is that the value of this sum will be 1 if and only if the  $b_i$ 's have just the values of one of the rows in Table 7.11 in which  $c_4 = 1$ . The  $b_i$ 's in each of these rows are such that one and only one of the terms in expression (7) is 1 and all the rest are 0. And for all other rows, the  $b_i$ 's are such that all four terms are 0. Therefore for any row in Table 7.11 the sum in expression (7) gives the value of  $c_4$ .

Now the sum in expression (7) is in DNF. Therefore we may try to simplify it using the Karnaugh map shown in Fig. 7.9. This Karnaugh map has two  $2 \times 1$  rectangles, the one on the left resulting from a top-to-bottom wraparound. The two terms corresponding to these two rectangles are  $b_2 \bar{b}_4 \bar{b}_8$  and  $\bar{b}_2 b_4 \bar{b}_8$ , so the equation for  $c_4$  is

|             | $b_2$       | $b_2$ | $\bar{b}_2$ | $\bar{b}_2$ |             |
|-------------|-------------|-------|-------------|-------------|-------------|
| $b_1$       | 1           |       |             |             | $\bar{b}_4$ |
| $b_1$       |             |       |             | 1           | $b_4$       |
| $\bar{b}_1$ |             |       |             | 1           | $b_4$       |
| $\bar{b}_1$ | 1           |       |             |             | $\bar{b}_4$ |
|             | $\bar{b}_8$ | $b_8$ | $b_8$       | $\bar{b}_8$ |             |

**FIGURE 7.9**

Karnaugh map for expression (7).

$$c_4 = b_2 \bar{b}_4 \bar{b}_8 + \bar{b}_2 b_4 \bar{b}_8.$$

Similarly, we could generate equations for  $c_1$ ,  $c_2$ , and  $c_8$ , but we'll leave that to a problem [33]. Before showing you how the equation for  $c_4$  might be converted into a logic circuit, we'll present our other approach to this problem.

In this approach, we simply look at Table 7.11 and — using visual pattern recognition but rather differently from that used with Karnaugh maps — try to infer from it the equation for each  $c_i$  in terms of the  $b_i$ 's. We reason as follows:

- As the  $b_2$  and  $c_2$  columns are identical,

$$c_2 = b_2. \quad (8)$$

- Each entry in the  $c_1$  column is the complement of the corresponding entry in the  $b_1$  column; therefore

$$c_1 = \bar{b}_1. \quad (9)$$

- There is a 1 in the  $c_4$  column if there is a 1 in the  $b_2$  column or the  $b_4$  column but not both (which logical operator discussed in Section 7.2 does this correspond to?); therefore  $c_4$  is 1 if and only if both  $b_2 + b_4$  and  $\bar{b}_2 + \bar{b}_4$  are 1. From this it follows that

$$c_4 = (b_2 + b_4)(\bar{b}_2 + \bar{b}_4). \quad (10)$$

- There is a 1 in the  $c_8$  column only when there are 0's in the  $b_2$ ,  $b_4$ , and  $b_8$  columns; therefore

$$c_8 = \overline{b_2 + b_4 + b_8}. \quad (11)$$

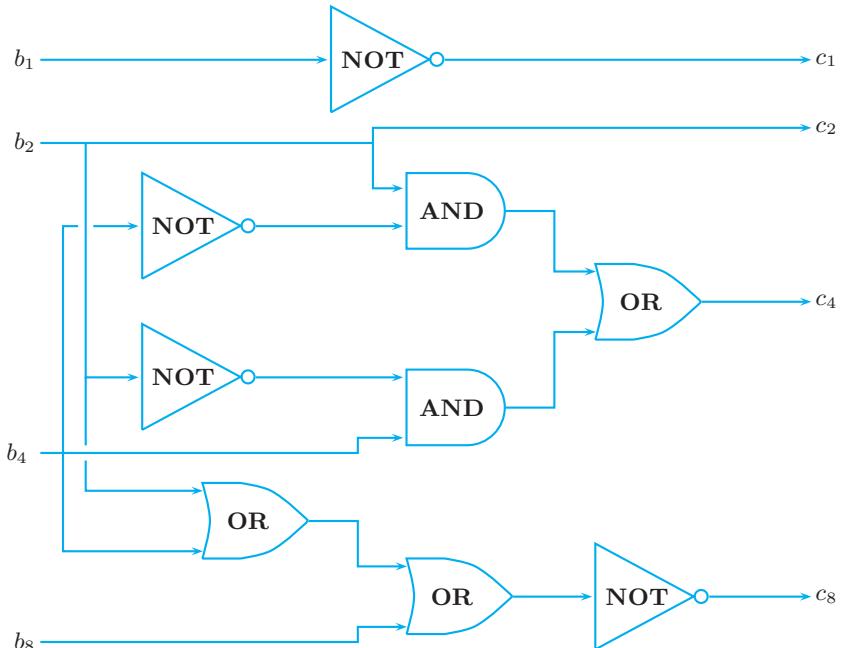
But wait a minute. Eq. (10) for  $c_4$  is different from the one we obtained earlier. How can this be? Could it be that they are really the same but just expressed in a different form? If we use the distributive law (vii) on Eq. (10), we obtain

$$\begin{aligned} c_4 &= (b_2 + b_4)(\bar{b}_2 + \bar{b}_4) \\ &= b_2 \bar{b}_2 + b_2 \bar{b}_4 + \bar{b}_2 b_4 + b_4 \bar{b}_4 \\ &= b_2 \bar{b}_4 + \bar{b}_2 b_4. \end{aligned} \quad [Since p\bar{p} = 0] \quad (12)$$

This result looks a little better, but there is still no  $\bar{b}_8$ , as in the earlier equation. Actually, both equations are correct because  $\bar{b}_8 = 1$  in all rows in which  $c_4 = 1$ .

That is to say, if we multiply  $b_2\bar{b}_4 + \bar{b}_2b_4$  by  $\bar{b}_8$ , its value remains the same for all rows of Table 7.11. The Karnaugh map approach, as we have described it, is not capable of determining this result and thereby enabling deletion of the  $\bar{b}_8$ . The cause of this failure is that only 10 of the 16 possible combinations of the  $b_i$ 's are relevant in this example. By recognizing that we don't care what the other six combinations give, we can in fact use the Karnaugh map approach to derive Eq. (12) [23]. Thus both of our approaches will give the same result if properly used. ■

Which solution method in Example 6 is to be preferred? We like the Karnaugh map approach better because of its inherently algorithmic, constructive flavor. The second approach, while quicker here, smacks too much of ad hoc cleverness. It would likely fail for more complex circuit designs than the simple one we've considered. To be fair, we should admit that the Karnaugh map approach is not of much use when there are more than 5 or 6 variables because the maps and their labeling become unwieldy. But there are systematic approaches to the design of logic circuits which take an approach similar to that of the Karnaugh map idea and are applicable to circuits of any complexity. This situation is analogous to the situation in graph theory, where a visual method for, say, finding a shortest path only works for relatively small graphs, but the idea can be systematized into an algorithm whose input is a representation of a graph of any size.



**FIGURE 7.10**

Logical circuit for a completer.

Finally, what does the logic circuit which implements Eqs. (8), (9), (11), and (12) look like? It is shown in Fig. 7.10 and is actually nothing more than a pictorial representation of these equations. For example,  $c_4$  as given by Eq. (12) is the

logical sum of two logical products or, stated another way, the “or” of two “and”s. Therefore we “and”  $b_2$  and  $\bar{b}_4$ , as well as  $\bar{b}_2$  and  $b_4$ , and then “or” the two results together. The rest of Fig. 7.10 may be interpreted similarly.

In the actual design of logic circuits, it turns out that circuits which implement the “nand” and “nor” connectives (see Fig. 7.3, Section 7.2) are more common than those that use “and”, “or” and “not”. We’ll explore some of the reasons for this in [4–8, Supplement].

## Problems: Section 7.5

---

1.  $\langle 1 \rangle$  Verify all 12 parts of Theorem 1 using truth tables:

- a) parts (i)–(iv)
- b) parts (v)–(viii)
- c) parts (ix)–(xii).

2.  $\langle 2 \rangle$  Translate the following set statements into Boolean algebra notation. For each one, state whether it is always true (i.e., a theorem).

- a)  $\overline{A \cap B} = \overline{A} \cup \overline{B}$
- b)  $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
- c)  $A \cap (B \cup C) = (A \cap B) \cup C$

3.  $\langle 2 \rangle$  Translate the following Boolean equations into set statements. For each one, state whether it is a theorem.

- a)  $pq + p\bar{q} = p$
- b)  $1 + p = p$
- c)  $(p + q)(\bar{p} + \bar{q}) = 0$
- d)  $\overline{\overline{a}} = a$

4.  $\langle 3 \rangle$  For each dual pair of Boolean identities below, i) explain intuitively why they are true, ii) check them with truth tables, and iii) give a deductive proof of both parts.

- a)  $\bar{0} = 1$  and  $\bar{1} = 0$ . Hint: Substitute  $\bar{0}$  into the identity  $a + 0 = a$ .
- b)  $1 + 1 = 1$  and  $0 \cdot 0 = 0$
- c)  $a + a = a$  and  $aa = a$
- d)  $a + 1 = 1$  and  $a0 = 0$ . Hint: Start with  $a + 1 = (a + \bar{a})(a + 1)$ .

5.  $\langle 2 \rangle$  Actually, Eq. (2) is true in Boolean algebra.

- a) Prove this by first denoting  $a + b$  as  $d$  and using the familiar distributive law twice.
- b) It thus follows from Eqs. (1) and (2) that we have the Boolean identity

$$a^2 + ab + ac + bc = a + bc.$$

Prove this by truth tables as well.

6.  $\langle 3 \rangle$  In ordinary algebra, whenever you see  $a + x = a$ , you can immediately “cancel” and conclude  $x = 0$ .

- a) In the 0-1 Boolean algebra find values for  $a, x$  so that  $a + x = a$  is true but  $x \neq 0$ . Thus the usual cancellation law is false.
- b) Fortunately, the following modified cancellation law is true in (any) Boolean algebra: If  $x$  satisfies  $a + x = a$  for every  $a$  in a Boolean algebra, then  $x = 0$ . Prove this.

7.  $\langle 2 \rangle$  What are the dual results to the two parts of [6]?

8.  $\langle 3 \rangle$  Fact: If  $a + b = 1$  and  $ab = 0$ , then  $b = \bar{a}$ . Prove this two ways:

- a) In the 0-1 Boolean algebra by considering all possibilities for  $a$  and  $b$ .
- b) In any Boolean algebra by deduction. Hint: Expand  $b = b(a + \bar{a})$  and  $\bar{a} = \bar{a}(a + b)$  and use the givens to reach the expression.

9.  $\langle 2 \rangle$

- a) Show that the Boolean expression  $\bar{p} + q$  corresponds to the implication operator in the propositional calculus.
- b) Consider the expression  $\bar{p} + pq$ . Compare its truth values to those of the expression in a). How do you explain what you find?

10.  $\langle 1 \rangle$  Prove the following Boolean theorems using truth tables.

- a)  $p\bar{q} + \bar{p}q = (p + q)(\bar{p} + \bar{q})$ .
- b)  $\overline{pr + qr} = \bar{p}r + \bar{q}r$ .
- c)  $\bar{p}\bar{q} + \bar{q}\bar{r} + \bar{q}r + \bar{p}q = p\bar{q} + q\bar{r} + \bar{p}r$ .

11. (2) We noted that parts (i)–(x) of Theorem 1 could be taken to be the axioms of Boolean algebra in that all possible theorems of Boolean algebra can be deduced from these 10 identities.

a) Verify that in each pair of identities in Theorem 1 one part can be obtained from the other by exchanging  $\cdot$  for  $+$  and 0 for 1. For this reason each member of a pair is said to be the **dual** of the other.

b) Why therefore is it true that, from every theorem in Boolean algebra, you can derive a **dual theorem** by exchanging  $\cdot$  for  $+$  and 0 for 1?

c) Write the dual of each of the Boolean theorems in [10].

12. (3) Prove the following Boolean theorems deductively.

a)  $p + pq = p.$  [Hint: Use (ii) of Theorem 1]

b)  $p(p + q) = p.$

c)  $p + (\bar{p} + q) = 1.$

d)  $p(\bar{p}q) = 0.$

13. (3) Both of De Morgan's laws [(xi) and (xii) of Theorem 1] can be proved deductively using (i)–(x). Do it. Hint: Use (vii) and (viii), c) and d) of [12], and [8].

14. (3)

a) Prove the generalized distributive law for Boolean algebra:

$$p(q_1 + q_2 + \cdots + q_n) = pq_1 + pq_2 + \cdots + pq_n.$$

b) Prove deductively that

$$(p + q)(p + r)(p + s) = p + qrs.$$

15. (2)

a) Describe how the truth table of a Boolean expression can be used to derive the DNF for that expression.

b) Apply the result of part a) to  $pq + pr + \bar{q}r$  to get Eq. (4).

16. (2) Using DNF, show that  $(p + q)(p + r) = p + qr.$  That is, expand both sides into DNF (either algebraically or by checking which rows of the truth table are true for each side) and check that they are the same. Which side represents the simpler circuit to build?

17. (3) Use Karnaugh maps to simplify the following expressions.

a)  $pr + (pq + \bar{q})\bar{r}$

b)  $(p + q)r + \bar{p}r + p\bar{r}$

c)  $\bar{p}\bar{q} + \bar{q}\bar{r} + \bar{q}r + \bar{p}q;$  Find two distinct simplifications both with the same number of terms (compare to [10c]).

18. (1) Display circuit diagrams for the Boolean expressions in [17] using AND, OR, and NOT gates as the basic components.

19. (2) Write DNF expressions for the Boolean expressions represented by the last columns of the following truth tables.

a) 

| <u><math>p</math></u> | <u><math>q</math></u> | <u><math>r</math></u> |
|-----------------------|-----------------------|-----------------------|
| 0                     | 0                     | 0                     |
| 0                     | 1                     | 1                     |
| 1                     | 0                     | 0                     |
| 1                     | 1                     | 1                     |

b) 

| <u><math>p</math></u> | <u><math>q</math></u> | <u><math>r</math></u> |
|-----------------------|-----------------------|-----------------------|
| 0                     | 0                     | 0                     |
| 0                     | 1                     | 1                     |
| 1                     | 0                     | 1                     |
| 1                     | 1                     | 0                     |

c) 

| <u><math>p</math></u> | <u><math>q</math></u> | <u><math>r</math></u> |
|-----------------------|-----------------------|-----------------------|
| 0                     | 0                     | 0                     |
| 0                     | 0                     | 1                     |
| 0                     | 1                     | 0                     |
| 0                     | 1                     | 1                     |
| 1                     | 0                     | 0                     |
| 1                     | 0                     | 1                     |
| 1                     | 1                     | 0                     |
| 1                     | 1                     | 1                     |

d) 

| <u><math>p</math></u> | <u><math>q</math></u> | <u><math>r</math></u> |
|-----------------------|-----------------------|-----------------------|
| 0                     | 0                     | 0                     |
| 0                     | 0                     | 1                     |
| 0                     | 1                     | 0                     |
| 0                     | 1                     | 1                     |
| 1                     | 0                     | 0                     |
| 1                     | 0                     | 1                     |
| 1                     | 1                     | 0                     |
| 1                     | 1                     | 1                     |

20. (2) Apply the Karnaugh map method to rewrite  $p(q + r)$  in another form. What do you end up with? Does the result represent a simpler circuit than the original?

21. (2) Show by truth tables that the DNF for the  $2 \times 2$  region discussed in the second paragraph of Example 5 can be expressed as  $qr.$

22. (3)

a) Design an array for Karnaugh maps with five variables. Hint: Since there are only four sides to an array but five variables, some side must have more than one set of labels.

b) If we could draw them in three dimensions, what would be the best way to arrange boxes for Karnaugh maps of three variables? What would be the advantage of this over the two-dimensional arrangement we have shown?

- 23.** ⟨2⟩ If we don't care what the output is for a certain set of inputs, we can put a "d" in the Karnaugh map box for those inputs and include that box in a group or not include it, whichever will help the simplification most. Use this observation to obtain Eq. (12).
- 24.** ⟨3⟩ In this problem we refer to the rows of a 4-variable truth table by their decimal equivalents. Thus 9 refers to the row whose first four columns are 1 0 0 1 since 1001 in binary is equivalent to 9 in decimal. Simplify the following Boolean expressions where "don't care" refers to rows for which the value of the expression is irrelevant. That is, the original expression may evaluate to 1 in a "don't care" row, but the simplified expression can evaluate to 0 for such a row, or vice versa. This allows you to put 1's in any "don't care" box in a Karnaugh map if doing so will make a useful pattern of 1's which can be simplified.
- a)  $\overline{pqr}\overline{s}$  [Don't care: 10, 11, 12, 13, 14, 15]  
 b)  $\overline{p}\overline{r} + pq\overline{s} + \overline{p}\overline{q}rs + pqr s$  [Don't care: 2, 6, 7]  
 c)  $\overline{p}\overline{q}\overline{r} + \overline{p}\overline{r}s + \overline{q}\overline{s}$  [Don't care: 0, 2, 3, 7, 9, 10, 11, 13, 14, 15]
- 25.** ⟨2⟩ Show that the Boolean expression defined by any possible truth table (i.e., a table with one more column than the number of atoms), regardless of how many atoms are involved, can be expressed using just **and**, **or**, and **not**. *Hint:* Consider DNF.
- 26.** ⟨3⟩ Show that the expression defined by any possible truth table can be expressed using just **and** and **not**.
- 27.** ⟨3⟩ Show that the expression defined by any possible truth table can be expressed using just **or** and **not**.
- 28.** ⟨3⟩ Use De Morgan's laws to give an informal proof of the following rule: To find the complement of a Boolean expression (i.e., an expression whose value is the complement of the original expression, whatever the values of the atoms), change all  $\cdot$  to  $+$  and all 0's to 1's, and vice versa, and replace each variable by its complement. *Hint:* Use the fact that any Boolean expression which contains  $+ \text{ or } \cdot$  can be written as  $a + b$  or  $a \cdot b$  for suitable Boolean expressions  $a$  and  $b$ .
- 29.** ⟨2⟩ **Conjunctive Normal Form (CNF)** is any expression of the form:
- $$(p + q + r)(p + q + r) \cdots (p + q + r),$$
- where each variable under consideration appears once in each factor, either plain or complemented, and no factor appears more than once. Otherwise, the number of factors is not fixed. Do you see that CNF is just like DNF with the roles of  $\cdot$  and  $+$  interchanged? Therefore CNF is the dual of DNF. Just as every wff can be written in DNF, so can every wff be written in CNF.
- a)** If you were given a truth table and asked to derive the CNF representation of the wff it represents, which rows of the truth table would you have to pay attention to? Why?
- b)** Figure out the CNF for the truth tables in [19].
- 30.** ⟨2⟩ Karnaugh maps can also be used to simplify CNF, with each box now representing a sum rather than a product. Use the following CNF Karnaugh map to answer a) and b).
- |                |     |                |                |                |
|----------------|-----|----------------|----------------|----------------|
|                | $q$ | $\overline{q}$ | $\overline{q}$ | $q$            |
| $p$            |     | 1              | 1              |                |
| $\overline{p}$ | 1   |                |                |                |
|                | $r$ | $r$            | $\overline{r}$ | $\overline{r}$ |
- a)** To what three terms in a CNF expression do the three 1's correspond?
- b)** Show that the CNF corresponding to the  $2 \times 1$  rectangle can be simplified to  $p + \overline{q}$ .
- 31.** ⟨2⟩ Rewrite each of the wffs in [17] in CNF.
- 32.** ⟨3⟩ Simplify each result in [31] using a Karnaugh map. Show that each result is equivalent to the result of [17].
- 33.** ⟨2⟩ Use Karnaugh maps to find expressions for  $c_1$ ,  $c_2$ , and  $c_8$  in the completer example.
- 34.** ⟨2⟩ Use the same technique for designing a completer to design a circuit to add 1 to a decimal digit, where  $9 + 1 = 0$ .

## 7.6 The Predicate Calculus

Recall from Section 7.2 that propositions with variables in them, such as

$$\lfloor j/3 \rfloor * 3 = j, \quad (1)$$

are called predicates. Each assignment of specific values to the variables in a predicate results in a proposition whose truth value may then be calculated. Thus the predicate above will be true whenever  $j$  is a multiple of 3 and false otherwise.

In this section we explain the need for extending logic to include predicates, introduce the new notation and concepts, and provide practice in reading, writing, and manipulating logical expressions containing predicates. Among its many virtues, the use of predicates lets us define a language which makes it much clearer what is going on in complicated reasoning. Formal proofs with predicates are discussed briefly in Section 7.8.

The **predicate calculus** is an extension of the propositional calculus, which, as its name implies, lets us deal with predicates without requiring that each variable, like  $j$  above, first be given a value in order to convert the predicate into a proposition. Thus the predicate calculus is a much more powerful language than the propositional calculus for expressing complicated mathematical thoughts.

As an instance of a situation that cannot be handled by the propositional calculus, consider the following common type of argument:

All prime numbers are integers.

7 is a prime number.

Therefore 7 is an integer.

(2)

Superficially this looks like an argument in the propositional calculus with the first two lines the premises and the third line the conclusion. But how would you prove it? Certainly not by a truth table, because the first line is not a single proposition but an infinite number. Note also that the conclusion is just one of this infinite number of propositions. To handle arguments like this, we shall somehow have to find a way to express the relationship between “all primes are integers” and “7 is a prime” that will allow us to justify the conclusion. The predicate calculus will allow us to do just this.

Sometimes even when the the propositional calculus could be used, it is inconvenient and inefficient. This is illustrated in the following algorithm verification example. Suppose that you are given a sequence of numbers,  $n_1, n_2, \dots, n_m$ , and you want to *sort* them; that is, you want to transform the sequence  $\{n_i\}$  into a new sequence  $\{N_i\}$  such that the  $N_i$ 's will be in numerical order. There are many algorithms for doing this. Suppose that you had designed or had been given such an algorithm and you wanted to insert assertions into it, as we discussed in Section 7.4. The assertion for the input specification might be

$$\{(1 \leq m) \wedge n[1:m]\},$$

where the bracket notation  $([1:m])$  means that  $n$  is a list of  $m$  numbers.

Now what would the assertion for the output specification look like? You want to say that the algorithm results in a list  $N[1:m]$  with the property that  $N_1 \leq N_2$ ,  $N_2 \leq N_3$ ,  $N_3 \leq N_4$ , etc. So you would have to write something like

$$\{N_1 \leq N_2 \wedge N_2 \leq N_3 \wedge \cdots \wedge N_{m-1} \leq N_m\}$$

or

$$\{N_1 \leq N_2 \leq N_3 \leq \cdots \leq N_m\}.$$

Even the second form is not very pretty. It would be nice to have a notation that is more compact and avoids ellipsis. Well, you might argue, the second form above is really not that ugly. But suppose there were a constraint on the input which stated that all the numbers in the list  $n[1:m]$  must be distinct. How would you express that in an assertion [3]?

Situations like those in the examples above, which involve statements of infinite or arbitrary length, are common in mathematical thinking but are cumbersome to express using the propositional calculus. The predicate calculus, however, provides an effective and elegant way to handle such situations.

The display (1) illustrates that a predicate may be thought of as a **propositional function** of its variables. A common — and natural — notation for a predicate with variable  $j$ , such as that in (1), is  $P(j)$ , with  $j$  the argument of the predicate  $P$ . If the predicate has two variables, say,  $x$  and  $y$ , we would write  $P(x, y)$ . And so on for more variables, though we won't give any examples with more than two variables.

Another example of a predicate, this time mixing mathematical notation and ordinary English, is

$$P(x) : x \text{ is a Republican},$$

where  $x$  can be any living animal (i.e., this is the underlying domain). This predicate is true if  $x$  is George Bush, false if  $x$  is John Kerry, and (presumably) false if  $x$  is your pet cat Rusty.

This example indicates that the underlying domain or the **universe** that the arguments range over can be anything (so long as we specify it), but most of the time in this book it will be  $N$  or  $N^+$  or some subset of these. Occasionally it will be  $R$ . Sometimes it will read better to write  $P_j$  instead of  $P(j)$ , or to write  $P_{ij}$  instead of  $P(i, j)$ , especially when the universe consists only of integers.

Here are two further examples of predicates when the universe is  $N^+$ :

$$P_i: 1 + 2 + \cdots + i = i(i+1)/2;$$

$$P_j: \lfloor j/2 \rfloor * 2 = j.$$

Note that the first predicate is true for all values of  $i$  in  $N^+$ , but the second is true only for certain values of  $j$ . (Which ones? [1])

A wff in the propositional calculus becomes a wff in the predicate calculus if any of its atomic propositions is replaced by a propositional function. For example, corresponding to the propositional calculus wff

$$P \wedge Q \implies R$$

there would be a predicate calculus wff

$$((x > 7) \wedge Q) \implies (y < 5),$$

if  $P$  and  $R$  are replaced as above. Note that a predicate may contain some atomic propositions, such as  $Q$ , in addition to propositions with variables. Also note that predicates can be combined using logical operators just like atoms of the propositional calculus.

## EXAMPLE 1

We have encountered predicates in decision statements in AL (see Section 1.2). And you may well have seen them in programming languages, where they are often called Boolean expressions. Here are some examples:

$$\begin{array}{ll} i < j; & [\text{Two-variable predicate } P_{ij}] \\ [(\sqrt{i})]^2 = i; & [\text{Is } i \text{ a perfect square?}] \\ (i < 10) \vee (j > 20); & \\ ((x > 10) \wedge (x < 20)) \implies (x = 15). & \blacksquare \end{array}$$

For each value (or tuple of values) of the variable(s) in a predicate, the predicate will have the value T or F [2].

## Quantifiers

When you deal with predicates, you may want to consider more, perhaps many more, than one value of the variable(s) in the predicate. For example, if you have the predicate

$$P_i: i^2 > 8i + 7,$$

you might want to consider all  $i \in N$  or some range of values of  $i$ . The mechanism in the predicate calculus for doing this kind of thing is called **quantification**. A **quantifier** (first introduced in Section 0.6) enables us to gather many propositions under a single rubric. The first of the two most important quantifiers used in mathematical logic is defined as follows:

---

**Definition 1.** The expression

$$(\forall x)P(x) \tag{3}$$

represents the assertion that  $P(x)$  is true for all  $x$  in the universe  $U$  under discussion. For instance, if  $U = N^+$ , then  $(\forall i)P_i$  is the infinite conjunction

$$P_1 \wedge P_2 \wedge P_3 \wedge \dots$$

The symbol  $\forall$  (an upside down A, where “A” is short for “all”) is called the **universal quantifier**.

---

The notation  $(\forall i)P_i$  should be read, “For all  $i$ ,  $P(i)$  is true”, which is the same as saying that  $P_1$  **and**  $P_2$  **and**  $P_3$ , ..., are true. So a quantified predicate is just a shorthand for writing an infinite number of propositions, instead of just listing some of them and using an ellipsis to indicate the rest. Sometimes, by the way, you wish to specify not an infinite number of propositions but a finite number greater than 1. You can do this with quantifier notation as we’ll show below.

## EXAMPLE 2

Write the proposition:

All odd positive integers are prime

using quantifier notation.

**Solution** Of course, our proposition is false but that doesn’t prevent us from expressing it in the notation of logic. Indeed, when a proposition is expressed in English and not obviously true or false, expressing it in logic notation will often be a necessary first step in determining its truth value. So, with

$$P_i : i \text{ is an odd positive integer} \quad \text{and} \quad Q_i : i \text{ is prime}$$

and  $N^+$  as the universe, the given proposition becomes

$$(\forall i)(P_i \implies Q_i). \tag{4}$$

Actually our convention that  $P \implies Q$  is true when  $P$  is false means that we don’t have to worry too much about the universe. In (4) the universe could be any superset of  $N^+$  since  $P_i$  is surely false if  $i$  is not a positive integer. Note also how in this example an English sentence of the form “all A’s are B’s” turns into a universally quantified implication. This is very common. ■

The display (3) is a *proposition*; either  $P(x)$  is true for all  $x$  in the universe or it isn’t. We have noted that a predicate  $P(x)$  is not a proposition because of the variable. Heretofore the only way we could make it into a proposition was to substitute a specific value for  $x$ . But now we know another way: quantify the predicate.

Although the idea of a quantifier sometimes seems strange or difficult when first encountered, you actually have met the essential idea before. Summation and product notation, which we first discussed in Section 0.4, are both notations which *summarize* something unwieldy or impossible to write out completely, as in

$$\sum_{i=1}^n a_i = a_1 + a_2 + \cdots + a_n, \tag{5}$$

and

$$\prod_{i=1}^{\infty} b_i = b_1 \cdot b_2 \cdot b_3 \cdots. \tag{6}$$

Quantifiers enable us to summarize conjunctions, as in Definition 1 (and also disjunctions, as in Definition 2 below), of many propositions in much the same way that summation and product notation allow us to summarize the addition and multiplication of mathematical expressions. We illustrate this below.

Although logicians and mathematicians almost always apply the universal quantifier to infinite sets of propositions, computer scientists often apply it to finite sets of propositions, as in the sorting example at the beginning of this section. The output specification there that the numbers be in numerical order could be stated

$$(\forall i : 1 \leq i < m) N_i \leq N_{i+1}. \quad (7)$$

In (7), the portion after the colon indicates the range of values of  $i$  to which the quantifier applies. Here,

$$N_i \leq N_{i+1}$$

is a predicate whose underlying universe is the finite number of values of  $i$  from 1 to  $m - 1$ . (Make sure, by the way, that you understand why " $< m$ " rather than " $\leq m$ " appears in expression (7).)

Now suppose you want the universe for a predicate  $P(x)$  to be the real numbers  $R$ . You could write

$$(\forall x)(R(x) \Rightarrow P(x)),$$

where  $R(x)$  is the predicate " $x$  belongs to  $R$ ". Or, analogously to expression (7), you might write

$$(\forall x: R(x)) P(x),$$

or

$$(\forall x \in R) P(x).$$

The analogy of quantifier notation with summation or product notation becomes even more evident when you realize that something like Eq. (5) might be written in a "linear" notation as

$$\sum i : 1 \leq i \leq n : a_i.$$

However, this is rarely done. More common is to go in the other direction and write expression (7) as

$$(\forall i)_{1 \leq i < m} N_i \leq N_{i+1},$$

which is like the notation we used in Section 0.4.

In informal mathematical writing the universal quantifier is generally used implicitly rather than explicitly. For example, the fact that 0 is the additive identity is usually stated as

$$x + 0 = x,$$

but this really means

$$(\forall x) x + 0 = x.$$

Similarly, the commutative law for the sum of two numbers,

$$x + y = y + x,$$

is really the statement

$$(\forall x)(\forall y) \quad x + y = y + x, \tag{8}$$

where  $x + y = y + x$  is a two-variable predicate. Here, double quantification is directly analogous to double summation. You should read expression (8) as “For all  $x$  and for all  $y$ ,  $x + y = y + x$ . ”

Indeed, in all mathematical identities the universal quantifier is implicit because such identities apply to all values of their variables in the underlying universe. This is really the distinction between an *identity* and an *equation*. The former is true for all values in the appropriate domain, whereas the latter is true for only one or a few values.

### EXAMPLE 3

Restate argument (2) using quantifier notation and use this notation to explain why the argument is valid.

**Solution** Using the predicates

$$P_i : i \text{ is a prime}$$

$$Q_i : i \text{ is an integer}$$

this argument may be captured in symbols fully as

$$(\forall i) (P_i \Rightarrow Q_i)$$

$$\begin{array}{c} P_7 \\ \hline Q_7. \end{array}$$

As for validity, from the first line we get  $P_7 \Rightarrow Q_7$  (or  $P_i \Rightarrow Q_i$  for any other single number  $i$ , but 7 is the one we want). Next,  $Q_7$  follows from  $P_7 \Rightarrow Q_7$  and  $P_7$  because

$$[(P_7 \Rightarrow Q_7) \wedge P_7] \Rightarrow Q_7$$

is a particular case of the tautology

$$[(P \Rightarrow Q) \wedge P] \Rightarrow Q,$$

provable by truth tables. ■

We often use quantifiers in ordinary speech as we did with the English equivalent of  $P_i \Rightarrow Q_i$ : “All prime numbers are integers”. Sometimes we make them explicit, using words like “all” or “every”, but sometimes they are implicit, as in “Refrigerators are heavy”, which would usually be taken to mean “All refrigerators are heavy”.

**Quantification and Induction.** Universal quantifier notation allows us to express the principles of mathematical induction compactly and with less chance for misunderstanding. Using this notation the First (or weak) Principle becomes: In order to prove

$$(\forall n) P(n), \tag{9}$$

$$n \geq n_0$$

it suffices to prove

$$P(n_0) \wedge (\forall n: n > n_0)[P(n-1) \implies P(n)]. \quad (10)$$

Using this notation, we can give a better answer than before to the common question: Why isn't it enough to do just the inductive step? That is, isn't the proof finished as soon as  $P(n)$  is proved from  $P(n-1)$ ? The answer is: when we say we want to prove  $P(n)$ , we are using implicit quantification; we are really trying to prove  $(\forall n: n \geq n_0)P(n)$ . This is not what we get in the inductive step, because the  $(\forall n)$  in (10) applies to the whole implication  $P(n-1) \implies P(n)$ , not just to  $P(n)$ .

Quantifiers also allow us to rebut the claim sometimes made that induction is valid if we prove  $P(n-1) \implies P(n)$  but invalid if we prove  $P(n) \implies P(n+1)$ . In the latter case (so the argument goes), we are assuming what we want to prove rather than concluding with it. The fallacy here is revealed if we again make all the quantifiers explicit. What we want to prove is  $(\forall n: n \geq n_0)P(n)$ , and this is neither the conclusion of  $(\forall n: n > n_0)[P(n-1) \implies P(n)]$  nor the assumption of  $(\forall n: n \geq n_0)[P(n) \implies P(n+1)]$ .

Strong induction is also precisely stated with predicate notation. The principle needed to prove (9) by strong induction is

$$P(n_0) \wedge (\forall n: n > n_0)[((\forall j: n_0 \leq j < n)P(j)) \implies P(n)]. \quad (11)$$

Note the use of the second universal quantifier to express that the truth of all the propositions  $P(j)$  for  $n_0 \leq j < n$  may be used to prove that  $P(n)$  is true.

The other essential quantifier used in mathematical logic is defined as follows:

**Definition 2.** The expression

$$(\exists x)P(x) \quad (12)$$

represents the assertion that there exists at least one  $x$  in the universe  $U$  under discussion for which  $P(x)$  is true. For instance, if  $U = N^+$ , then  $(\exists i)P_i$  is the infinite disjunction

$$P_1 \vee P_2 \vee P_3 \vee \dots$$

The symbol  $\exists$  (a backwards E, where “E” is short for “exists”) is called the **existential quantifier**.

The notation  $(\exists i)P_i$  should be read, “There exists an  $i$  for which the predicate  $P_i$  is true”, which is the same as saying that  $P_1$  or  $P_2$  or  $P_3$ , …, is true. Corresponding to expression (7) there is a similar form for the existential quantifier of which an example is:

$$(\exists i: m \leq i \leq n)P_i, \quad (13)$$

which is true if  $P_m \vee P_{m+1} \vee \dots \vee P_n$  is true and false otherwise. As with the universal quantifier, expression (12) defines a proposition which is true or false. Either there exists an  $i$  for which  $P_i$  is true or there doesn't.

When we make statements in ordinary English which correspond conceptually to the existential quantifier, we usually use words like “one” or “some”, as in

One of these days I'll ...

Some people really like mathematics.

In these cases the claim is that at least one day or person satisfies whatever condition follows. That is, in either case there *may* be more than one day or person for which the statements are true. In general, there *may* be many  $P_i$  which are true, perhaps even all of them, but there *must* be at least one if expression (12) is to be true; see Section 0.6.

**Multiple Quantifiers.** As with the universal quantifier, we may use multiple existential quantifiers, as in

$$(\exists x)(\exists y)[x^2 + y^2 = 1].$$

You should be able to see from this example and from (8) that it doesn't matter how you order the variables when a number of quantifiers of the same kind are concatenated. That is, just as you can change the order of summation when all the limits of summation are constants, so you can change the order of repeated universal or repeated existential quantifiers. We leave the proof to a problem [21], and we also leave to a problem [22] the situation in which the universe of values of an inner quantification depends on a value of the variable in an outer quantification.

Just as we can have summation and product notation in the same expression, we can have both existential and universal quantifiers in the same expression. For example,

$$(\forall i)(\exists j) [2^j \leq i < 2^{j+1}]$$

says that any integer  $i$  lies between two successive powers of 2. But what about the order in which you write the quantifiers when there is a mixture of universal and existential quantifiers? For example, is

$$(\forall i)(\exists j)P_{ij} \iff (\exists j)(\forall i)P_{ij} \quad (14)$$

**logically valid?** That is, is the truth value of the two sides of expression (14) always the same, independent of the predicate  $P_{ij}$  and the universe for  $i, j$ ? The answer is No, as the following example illustrates.

## EXAMPLE 4

Consider the predicate

$$P_{ij}: i = j. \quad (15)$$

Then

$$(\forall i)(\exists j)P_{ij}$$

is true because for any  $i$  there exists a  $j$  such that  $i = j$ . On the other hand,

$$(\exists j)(\forall i)P_{ij}$$

is false because there is no  $j$  such that  $i = j$  for all  $i$ . Thus the two sides of expression (14) can have different truth values. ■

However,

$$(\exists j)(\forall i)P_{ij} \implies (\forall i)(\exists j)P_{ij} \quad (16)$$

is logically valid, because whenever there exists a  $j$  (say,  $j_0$ ) such that  $P_{ij}$  is true for all  $i$ , then for all  $i$  and this  $j_0$  the right-hand side is also true. Note that for the predicate in (15) the left-hand side of (16) is false, but (16) is true since implication is defined so that ( $F \implies$  anything) is true.

Is expression (16) a tautology as we defined that term in Section 7.3? Yes, but “tautology” is normally reserved for use only in the propositional calculus. Logically valid is the preferred terminology in the predicate calculus.

## EXAMPLE 5

The following are logically valid formulas in the predicate calculus:

$$(\forall i)P_i \implies P_j.$$

[If the left-hand side is true,  
each  $P_i$  is true]

$$(\exists i)P_i \vee (\forall i)\neg P_i. \blacksquare$$

[Since, if no  $P_i$  is true,  
all  $P_i$  must be false]

**Negation and Quantification.** Consider the following statement in the universe of real numbers:

$$(\forall x)(\exists y)(xy = 1).$$

This is false (why?), so we would prefer to assert its negation, which must be true. One can always negate a statement by sticking “not” in front of it:

$$\neg(\forall x)(\exists y)(xy = 1), \quad (17)$$

but it is often helpful if one can rework such a statement into a more positive form.

We now show that negation passes through quantifiers in a way that can be applied mechanically to (17) and to more complicated wffs.

Consider  $(\forall x)P(x)$ . It’s false that  $P(x)$  is true for all  $x$  if and only if  $P(x)$  is false for some  $x$ . Thus

$$\neg(\forall x)P(x) \iff (\exists x)\neg P(x) \quad (18)$$

is logically valid, so that either side can always be substituted for the other.

Similarly,  $(\exists y)Q(y)$  is false if and only if  $Q(y)$  is false for all  $y$ , that is,

$$\neg(\exists y)Q(y) \iff (\forall y)\neg Q(y) \quad (19)$$

is logically valid.

Let us apply these observations to (17). Apply (18) to (17), with  $(\exists y)(xy = 1)$  playing the role of  $P(x)$ . We get

$$\neg(\forall x)(\exists y)(xy = 1) \iff (\exists x)\neg(\exists y)(xy = 1).$$

Now apply (19) to  $\neg(\exists y)(xy = 1)$ , with  $Q(y) = (xy = 1)$  (with  $x$  just a parameter with respect to  $Q(y)$ , called in logic a *free variable* – see the subsection Bound and Free Variables below), to obtain

$$(\exists x)\neg(\exists y) (xy = 1) \iff (\exists x)(\forall y) (xy \neq 1).$$

The right-hand side follows because  $\neg(xy = 1)$  is equivalent to  $xy \neq 1$ . Then by transitivity,

$$\neg(\forall x)(\exists y) (xy = 1) \iff (\exists x)(\forall y) (xy \neq 1).$$

(What is the  $x$  that is claimed to exist? Is there more than one such  $x$ ?)

In short, you can pass a negation through a string of quantifiers, but you must change each quantifier to the other kind of quantifier while negating the remaining proposition.

Here's another example. Do you agree that [23]

$$(\forall c)(\forall x)(\exists y) x + y = c ?$$

If not, you should assert the negation, which is

$$(\exists c)(\exists x)(\forall y) x + y \neq c.$$

**Bound and Free Variables.** Is

$$(\forall i)(i^2 > i) \tag{20}$$

a predicate with variable  $i$ ? No, because the variable  $i$  has been “quantified out”. It makes no sense to talk about giving  $i$  a value in expression (20) because the quantifier encompasses *all* values of  $i$ . What we have in expression (20) is not a propositional function but, as noted earlier, just a proposition itself. Another way to think about this is that  $i$  in (20) is a dummy variable, a concept first explained in Section 0.4.

What about

$$(\exists i)(i^2 = j)? \tag{21}$$

In expression (21) we are concerned with whether  $j$  is a perfect square. For each  $j$  the proposition is true or false but expression (21) itself is still a predicate whose value depends on the value assigned to  $j$ . In an expression such as (21) the variable  $j$  is said to be **free** because its value is unconstrained (except by the underlying universe), and  $i$  is said to be **bound** because the quantifier does not allow us to give specific values to  $i$ . By contrast, in

$$(\forall j)(\exists i)(i^2 = j), \tag{22}$$

both variables are bound —  $j$  by the universal and  $i$  by the existential quantifier. Thus expression (22) is a proposition with truth value T or F (which is it?). Only when each variable in a predicate is quantified, as in expression (22), do we get a proposition rather than a predicate.

## Problems: Section 7.6

1. **<1>** For what values of  $j$  is the predicate  $P_j$  :  $\lfloor j/2 \rfloor * 2 = j$  true?
2. **<2>** The parts of this problem refer to the four predicates of Example 1.

- a) For  $j = 17$ , for what values of  $i$  is  $P(i, j)$ :  $i < j$  true?
- b) For what values of  $i > 0$  is the predicate “ $i$  is a perfect square” true?
- c) Describe or draw a diagram to illustrate the pairs  $(i, j)$  such that  $(i < 10) \vee (j > 20)$  is true.
- d) For what real numbers  $x$  is the predicate  $[(x > 10) \wedge (x < 20)] \implies (x = 15)$  true?

3. ⟨2⟩

- a) Without using quantifiers, write an input specification for an algorithm to sort a list  $n[1, m]$  of  $m$  distinct numbers.
- b) Rewrite this IS using quantifiers.

4. ⟨2⟩

- a) Use quantifiers to write an output specification for an algorithm to sort a list  $n[1, m]$  of  $m$  distinct numbers.
- b) Use quantifiers to write an output specification for PERMUTE-1 (Section 4.9).

5. ⟨2⟩ The upshot of the verification of Algorithm MULT in Section 7.4 is effectively the predicate calculus statement

$$(x \geq 0) \implies (\text{prod} = xy).$$

Is there implicit quantification here? If so, what is it?

6. ⟨2⟩ For the following statements, let the domain be the set  $R$  of real numbers. For each statement, try to say it in words. If it is false, give an example to show why. Example:

$$(\forall x)(\exists y)(x < y)$$

says “for every number there is a bigger number”. This is true.

- a)  $(\forall x)(\exists y)[x \neq 0 \implies xy = 1]$ .
- b)  $(\exists y)(\forall x)[x \neq 0 \implies xy = 1]$ .
- c)  $(\exists x)(\forall y)[y \leq x]$ .
- d)  $(\forall x)(\exists y)(x + y = x)$ .
- e)  $(\exists y)(\forall x)(x + y = x)$ .
- f)  $(\forall x)(\forall y)(\exists z)[x < z < y]$ .
- g)  $(\forall x)(\forall y)[(x \neq y) \implies (\exists z)(x < z < y \vee x > z > y)]$ .
- h)  $(\forall x)(\forall y)(\forall z)[(x > y \wedge y > z) \implies x > z]$ .

7. ⟨2⟩ Let the domain be  $R$  again. Translate each of the following sentences into a formula of the predicate calculus.

- a) Every number has an additive inverse.
- b) There is a smallest number.
- c) Every positive number has a square root.
- d) Every positive number has a positive square root.
- e) There is a multiplicative identity (1, but don’t say so!).
- f) Any two numbers have a sum (that is, the sum is another real number).
- g) No number equals twice itself.

8. ⟨2⟩ Use predicate logic notation to express what it means for a relation  $R$  to be

- a) symmetric
- b) reflexive
- c) transitive.

See Section 0.1 for the definitions. Assume that  $R$  is defined on  $S \times S$  and that  $S$  is the universe of quantification. Let  $R(x, y)$  stand for the assertion that  $x$  and  $y$  satisfy the relation.

9. ⟨3⟩ Now let the domain be the integers  $Z$ . Try to say each of the following statements in words. If it is false, give an example to show why.

- a)  $(\forall i)(\exists j)(j = i + 1)$ .
- b)  $(\forall i)(\exists j)[Z(i) \implies (Z(j) \wedge j = i + 1)]$ .  
 $[Z(i)$  is the predicate “ $i$  is an integer”]
- c)  $(\forall i)(\forall j)[(i^2 = j) \implies ((-i)^2 = j)]$ .
- d)  $(\forall i)(\exists j)(j = i^2)$ .
- e)  $(\forall i)(\forall j)(\forall k)[(i \mid j) \wedge (j \mid k)] \implies (i \mid k)$ .  
 $[\mid]$  means “divides without remainder”]
- f)  $(\exists j)(\forall i)(ij = i)$ .
- g)  $(\forall i)(\forall j)(\forall k)[(i \mid j \wedge j \mid k) \implies \lfloor \lfloor k/j \rfloor / i \rfloor = \lfloor k / (j/i) \rfloor]$ .

10. ⟨2⟩ With the domain the nonnegative numbers  $N$ , translate the following sentences into formulas of the predicate calculus.

- a) There is a smallest nonnegative integer.
- b) For each number not a prime, there is another integer which divides it without remainder.
- c) All numbers less than 100 are less than  $2^7$ .
- d) All numbers less than 100 lie between two successive powers of 2.
- e) Every number is even or odd.
- f) The product of two odd numbers is odd.

- g)** The product of an odd and an even number is even.
- h)** If one number divides a second and the second divides a third, then there is an integer which, when multiplied by the first number, gives the third.
- 11.**  $\langle 2 \rangle$  State the following using predicate logic notation. Make all quantifiers explicit.
- $2x + 3 = 5$  iff  $x = 1$ .
  - The solutions to  $x^2 = 4$  are  $x = \pm 2$ .
  - All squares are rectangles. (Use  $S(x)$  for  $x$  is a square, and  $R(x)$  for rectangles.)
- 12.**  $\langle 3 \rangle$  In Section 0.1 various function properties were defined. These properties are fairly easily turned into predicate calculus statements. (All mathematical statements can be put into logic language if enough notation is introduced, but the ideas in Section 0.1 are relatively easy.) For instance, a function  $F: S \rightarrow T$  is onto if
- $$(\forall y \in T)(\exists x \in S)(f(x) = y).$$
- As discussed in the current section, there are various other ways to express the restrictions of the variables to  $S$  and  $T$ ; for instance,  $f$  is also onto if
- $$(\forall y)(\exists x)[T(y) \implies (S(x) \wedge f(x) = y)],$$
- where now  $S(x)$  means  $x$  has the property of belonging to set  $S$ , and similarly for  $T$ .
- Find at least two ways to express each of the following:
- $f$  is one-to-one
  - $f$  is a constant function
  - $f$  is an invertible function.
- 13.**  $\langle 3 \rangle$
- To say that a set  $S$  has a minimal element  $y$  means that  $y \in S$  and every element of  $S$  is equal to or bigger than  $y$ . Restate this definition using the predicate calculus. Regard  $S$  as a fixed set, that is,  $S$  will be a constant in your statement.
  - Fact: Every set  $S \subset N$  has a minimal element. State this fact using the predicate calculus. (The universe for your quantification will now have at least two types of objects, numbers and sets of numbers. There are more sophisticated types of predicate calculus in which objects and sets of objects are handled separately,
- but not here. Look up the Theory of Types in any book on foundations of mathematics.)
- 14.**  $\langle 3 \rangle$  Translate the following arguments into predicate calculus, defining whatever predicates you need. Make your translation just detailed enough to capture the argument. For instance, in a) having 4 right angles can be a predicate of figures; you don't need names for the angles or a two-argument predicate that says that  $x$  is an angle of figure  $y$ .
- All squares are rectangles. All rectangles have 4 right angles. Therefore all squares have 4 right angles.
  - Every planar graph is 4-colorable. If a graph is  $k$ -colorable, then it contains a set of at least  $v/k$  independent vertices (no adjacencies), where  $v$  is the number of vertices in the graph. Therefore, every planar graph contains an independent of at least  $v/4$  vertices.
- 15.**  $\langle 2 \rangle$
- Rewrite expression (10), replacing  $P(n-1) \implies P(n)$  with  $P(n) \implies P(n+1)$ .
  - Rewrite expression (11), replacing  $P(n)$  with  $P(n+1)$ .
- 16.**  $\langle 2 \rangle$  Recall that there are many variant induction schemes. One is split induction, of which the simplest case is: if you prove  $P(0)$  and  $P(1)$  and then prove that  $P(n)$  implies  $P(n+2)$ , you have proved  $P(n)$  for  $n \in N$ . State this variant in predicate calculus.
- 17.**  $\langle 3 \rangle$  Below are two statements in predicate logic. For each one, decide whether it is logically true, and give an informal explanation.
- $(\forall x)[P(x) \implies (Q(x) \vee R(x))] \implies (\forall x)[((P(x) \implies Q(x)) \vee (P(x) \implies R(x))]$
  - $(\forall x)[P(x) \implies (Q(x) \vee R(x))] \implies (\forall x)[((P(x) \implies Q(x)) \vee (\forall x)[P(x) \implies R(x)])]$
- 18.**  $\langle 3 \rangle$  (Passing quantifiers through half an implication) Explain in words why the following two statements are logically valid:
- $[(\forall x)(P \implies Q(x))] \iff [P \implies (\forall x)Q(x)]$
  - $[(\exists x)(P \implies Q(x))] \iff [P \implies (\exists x)Q(x)]$
- The principle here is easy to remember: a quantifier passes through unchanged to the *conclusion* of an implication, if the quantified variable does not appear in the hypothesis.

19. ⟨2⟩ It is often possible to drop the parentheses around quantifiers and keep the scope clear. Combined with writing  $P_x$  instead of  $P(x)$ , this can sometimes make long expressions more readable. For instance, a) in [18] may be rewritten as

$$[\forall x(P \Rightarrow Q_x)] \iff [P \Rightarrow (\forall x Q_x)].$$

Such rewrites work especially well when the quantifier applies to a short expression, such as  $Q(x)$  or  $P \Rightarrow Q(x)$ .

- a) Rewrite b) in [18] similarly.
- b) Rewrite the wffs in [17] similarly.

20. ⟨4⟩ Passing quantifiers through to the *hypothesis* of an implication is much trickier than passing them through to the conclusion, as in [18]. Determine which of the following statements are valid and then draw a general conclusion. Throughout, we use the easier-to-read notation of [19].

- a)  $[\forall x(P_x \Rightarrow Q)] \Rightarrow [\forall x P_x \Rightarrow Q]$
- b)  $[(\forall x P_x) \Rightarrow Q] \Rightarrow [\forall x(P_x \Rightarrow Q)]$
- c)  $[\forall x(P_x \Rightarrow Q)] \Rightarrow [(\exists x P_x) \Rightarrow Q]$
- d)  $[(\exists x P_x) \Rightarrow Q] \Rightarrow [\forall x(P_x \Rightarrow Q)]$
- e)  $[\exists x(P_x \Rightarrow Q)] \Rightarrow [(\exists x P_x) \Rightarrow Q]$
- f)  $[(\exists x P_x) \Rightarrow Q] \Rightarrow [\exists x(P_x \Rightarrow Q)]$
- g)  $[\exists x(P_x \Rightarrow Q)] \Rightarrow [\forall x P_x \Rightarrow Q]$
- h)  $[(\forall x P_x) \Rightarrow Q] \Rightarrow [\exists x(P_x \Rightarrow Q)]$

21. ⟨2⟩ Explain in words why the order of two universal quantifiers or two existential quantifiers can be switched, but  $(\forall i)(\exists j)$  cannot be switched.

22. ⟨3⟩ Rewrite the following assertions so that the order of the quantifiers is switched. “Rewrite” means that the result must have exactly the same meaning. In each case the domain is  $N$ .

- a)  $(\forall n: 1 < n < 10)(\forall m: n < m < 10)P(m, n).$
- b)  $(\forall n: 0 \leq n \leq 5)(\forall m: 0 \leq m \leq n)P(m, n).$
- c)  $(\forall n: 1 \leq n \leq 5)(\forall m: 0 \leq m \leq n)P(m, n).$
- d)  $(\forall n: 0 \leq n < 10)(\forall m: n \leq m \leq n^2)P(m, n).$

23. ⟨2⟩ Either

$$(\forall c)(\forall x)(\exists y) x + y = c$$

or its negation

$$(\exists c)(\exists x)(\forall y) x + y \neq c$$

must be true. Which one and why?

24. ⟨3⟩ Negate your statements in both parts of [13], both in symbols and in words.

25. ⟨3⟩ In the text we showed that, to negate an expression with a single quantifier, we can replace it with the other quantifier and negate the predicate inside. This generalizes to arbitrary strings of quantifiers. For instance,

$$\neg(\forall x)(\exists y)(\exists z)(\forall w)P(x, y, z, w) \iff \\ (\exists x)(\forall y)(\forall z)(\exists w)\neg P(x, y, z, w).$$

State this generalization and prove it by induction.

26. ⟨3⟩ Apply the result of [25] to:

- a) The formulas of [6].
- b) The formulas of [9].

In each case state the resulting formula in words and, if it is false, give an example to show why.

27. ⟨2⟩ In the following expressions, identify free variables and bound variables. Assume there are no implicit quantifications.

- a)  $x + y = y + x.$
- b)  $(\forall x)(x + y = y + x).$
- c)  $(\forall x)(\forall y)(x + y = y + x).$
- d)  $(\forall x)[(x > 0 \Rightarrow (\exists y)(x = 2y)).$
- e)  $(\forall x)P(x) \Rightarrow (x > 0 \wedge (\exists y)Q(y)).$
- f)  $g(n) = \sum_{i=1}^n (2i + 1).$

28. ⟨2⟩ Describe the set of real numbers for which the expression

$$(\exists x)(\exists y)[x^2 + y^2 = z]$$

is true. And is this set of numbers a set of  $x$ -values, a set of  $y$ -values, or a set of  $z$ -values? Explain.

29. ⟨3⟩ Consider the statement: Let  $f$  be a parabola of the form  $f(x) = (x - a)^2$ . What does this statement have to do with free and bound variables? There is no quantification here, so there is no clear right answer based on the text, but we hope you see an analogy.

## 7.7 Algorithm Verification Using the Predicate Calculus

At the beginning of Section 7.6 we considered the problem of taking a list  $n[1:m]$  of numbers and *sorting* it into a list  $N[1:m]$  of numbers in numerical order. We also considered this problem in Section 5.8, where we analyzed the merge sort method. (See also Section 3.7.) Here we'll consider another algorithm for sorting, which illustrates nicely how quantification plays a role in verifying algorithms.

The method we'll consider here is called **insertion sort**. The idea is illustrated in Fig. 7.11. We begin with the list  $n[1:m]$  and apply the recursive paradigm as follows. Suppose we had a list whose first  $i - 1$  elements were in numerical order (Fig. 7.11b), where the primes indicate that the elements in positions 1, ...,  $i - 1$ , will normally not be the elements that were there originally. In order to insert the  $i$ th element in its proper place among the already sorted  $i - 1$  elements, we

- first take  $n_i$  and put it in a temporary location (*temp* in Fig. 7.11c and d);
- then compare  $n_i$  with  $n'_{i-1}$ ; if  $n_i \geq n'_{i-1}$  then stop (because  $n_i$  is already in its proper place — not necessarily its *final* place but its correct place among the first  $i$  items); otherwise move  $n'_{i-1}$  into the slot previously occupied by  $n_i$ ; the result if  $n_i < n'_{i-1}$  is shown in Fig. 7.11c;
- continue comparing  $n_i$  with elements  $n_j$  of successively lower index; move each  $n_j$  to the right one position until a  $j$  is found such that

$$n_i \geq n'_j,$$

as shown in Fig. 7.11d;

- insert  $n_i$  into slot  $j + 1$ , as shown in Fig. 7.11e, where " indicates that the items in these slots are not the ones that were there at the start of  $i$ th step.

Of course, if  $n_i < n'_1$ , no  $j$  will be found for which  $n_i \geq n'_j$ ; this situation is easily recognized when  $j$  gets to 0, in which case  $n_i$  gets inserted into slot 1. Summarizing, insertion sort works by inserting the  $i$ th element in its proper place in the already sorted list of  $i - 1$  elements. As we start with a sorted list of one element (namely,  $n_1$ ), the result of applying the steps above for  $i = 2, 3, \dots, m$  should result in a completely sorted list.

An example of the entire process is illustrated in Fig. 7.12 and is translated into our algorithmic language in Algorithm 7.2, **INSERTION**. Algorithm **INSERTION** also includes assertions, as discussed in Section 7.4. Look first at the algorithm, ignoring the assertions, to make sure that you understand how it implements sorting by insertion. Although we have described sorting by insertion using the recursive paradigm, as with binary search, implementing the algorithm using iteration rather than recursion is easier and more straightforward.

Now let's see whether we can use the assertions to *prove* the algorithm correct. Remember: Informal reasoning that sorting by insertion works is one thing; proving that an algorithm to implement the insertion idea is correct is something quite different. We reason as follows:

|       |       |  |  |  |  |  |         |       |
|-------|-------|--|--|--|--|--|---------|-------|
| $n_1$ | $n_2$ |  |  |  |  |  | $\dots$ | $n_m$ |
|-------|-------|--|--|--|--|--|---------|-------|

(a)

|        |        |         |  |  |            |       |         |  |       |
|--------|--------|---------|--|--|------------|-------|---------|--|-------|
| $n'_1$ | $n'_2$ | $\dots$ |  |  | $n'_{i-1}$ | $n_i$ | $\dots$ |  | $n_m$ |
|--------|--------|---------|--|--|------------|-------|---------|--|-------|

$$n'_1 \leq n'_2 \leq \dots \leq n'_{i-1}$$

(b)

|        |        |         |  |  |            |       |            |         |       |
|--------|--------|---------|--|--|------------|-------|------------|---------|-------|
| 1      | 2      | $\dots$ |  |  | $i-2$      | $i-1$ | $i$        | $m$     |       |
| $n'_1$ | $n'_2$ |         |  |  | $n'_{i-2}$ |       | $n'_{i-1}$ | $\dots$ | $n_m$ |

(c)

temp    $n_i$

|        |         |  |  |        |       |            |         |            |         |       |
|--------|---------|--|--|--------|-------|------------|---------|------------|---------|-------|
| 1      | $\dots$ |  |  | $j$    | $j+1$ | $j+2$      | $m$     |            |         |       |
| $n'_1$ |         |  |  | $n'_j$ |       | $n'_{j+1}$ | $\dots$ | $n'_{i-1}$ | $\dots$ | $n_m$ |

(d)

temp    $n_i$

|        |         |  |  |        |                   |         |           |            |         |       |
|--------|---------|--|--|--------|-------------------|---------|-----------|------------|---------|-------|
| 1      | $\dots$ |  |  | $j$    | $j+1$             | $i$     |           |            | $m$     |       |
| $n'_1$ |         |  |  | $n'_j$ | $n''_{j+1} = n_i$ | $\dots$ | $n''_i =$ | $n'_{i-1}$ | $\dots$ | $n_m$ |

(e)

Insertion sort. (a) A list to be sorted; (b) the list after the first  $i - 1$  elements are in numerical order; (c) the list after the first comparison of  $temp$  with  $n'_{i-1}$ , assuming that  $n_i < n'_{i-1}$ ; (d) the list with  $n_i$  ready to be inserted in its proper place; (e) the list with the first  $i$  elements in numerical order.

**FIGURE 7.11**

|                           |             |     |            |               |               |               |               |               |               |
|---------------------------|-------------|-----|------------|---------------|---------------|---------------|---------------|---------------|---------------|
| $n[1: 9]:$                | [15]        | 12  | 9          | 17            | 6             | 12            | 2             | 8             | 16            |
| End of loop with $i = 2:$ | [ <u>12</u> | 15] | 9          | 17            | 6             | 12            | 2             | 8             | 16            |
| End of loop with $i = 3:$ | [ <u>9</u>  | 12  | 15]        | 17            | 6             | 12            | 2             | 8             | 16            |
| End of loop with $i = 4:$ | [ <u>9</u>  | 12  | 15         | [ <u>17</u> ] | 6             | 12            | 2             | 8             | 16            |
| End of loop with $i = 5:$ | [ <u>6</u>  | 9   | 12         | 15            | [ <u>17</u> ] | 12            | 2             | 8             | 16            |
| End of loop with $i = 6:$ | [ <u>6</u>  | 9   | 12         | [ <u>12</u>   | 15            | [ <u>17</u> ] | 2             | 8             | 16            |
| End of loop with $i = 7:$ | [ <u>2</u>  | 6   | 9          | [ <u>12</u>   | 12            | [ <u>15</u>   | [ <u>17</u> ] | 8             | 16            |
| End of loop with $i = 8:$ | [ <u>2</u>  | 6   | [ <u>8</u> | 9             | 12            | 12            | 15            | [ <u>17</u> ] | 16            |
| End of loop with $i = 9:$ | [ <u>2</u>  | 6   | 8          | [ <u>9</u>    | 12            | 12            | 15            | [ <u>16</u>   | [ <u>17</u> ] |

An example of insertion sort (item inserted is underlined; brackets enclose sorted portion of list).

**FIGURE 7.12**

## Algorithm 7.2

### Insertion

|                                                                                                                                                                 |                                    |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------|
| <b>Input</b> $m$                                                                                                                                                | [Integer $> 0$ ]                   |
| $n[1: m]$                                                                                                                                                       | [List to be sorted]                |
| <b>Output</b> $n[1: m]$                                                                                                                                         | [In numerical order]               |
| <b>Algorithm</b> INSERTION                                                                                                                                      |                                    |
| { $m > 0\}$                                                                                                                                                     | [Input specification]              |
| { $(\forall k: 1 \leq k \leq i-1) n_k \leq n_{k+1}$ }                                                                                                           | [Outer loop invariant]             |
| <b>for</b> $i = 2$ <b>to</b> $m$                                                                                                                                |                                    |
| $temp \leftarrow n_i$                                                                                                                                           |                                    |
| $j \leftarrow i - 1$                                                                                                                                            |                                    |
| { $(\forall k: 1 \leq k \leq j-1) n_k \leq n_{k+1} \wedge (\forall k: j+2 \leq k \leq i-1) n_k \leq n_{k+1} \wedge (\forall k: j+2 \leq k \leq i) temp < n_k\}$ | [Inner loop invariant]             |
| <b>repeat until</b> $j = 0$ <b>or</b> $temp \geq n_j$                                                                                                           |                                    |
| $n_{j+1} \leftarrow n_j$                                                                                                                                        |                                    |
| $j \leftarrow j - 1$                                                                                                                                            |                                    |
| <b>endrepeat</b>                                                                                                                                                |                                    |
| { $(\forall k: 1 \leq k \leq j-1) n_k \leq n_{k+1} \wedge (\forall k: j+2 \leq k \leq i-1) n_k \leq n_{k+1} \wedge (j=0 \vee temp \geq n_j)\}$                  | [Inner loop termination condition] |
| $n_{j+1} \leftarrow temp$                                                                                                                                       |                                    |
| <b>endfor</b>                                                                                                                                                   |                                    |
| { $(\forall k: 1 \leq k \leq i-1) n_k \leq n_{k+1} \wedge i=m\}$                                                                                                | [Outer loop termination condition] |
| { $(\forall k: 1 \leq k \leq m-1) n_k \leq n_{k+1}\}$                                                                                                           | [Output specification]             |

1. The input specification,  $m > 0$ , states only the assumption that the list is not empty.
2. The loop invariant for the outer loop states that, before entering this loop with a new value of  $i$ , the first  $i$  items in the list are sorted. When the loop is entered for the first time with (implicitly)  $i = 1$ , this assertion is vacuously true because there are no values of  $k$  between 1 and  $i - 1$  ( $= 0$ ).
3. The purpose of the inner loop is to insert the  $i$ th element in its proper position with respect to the prior  $i - 1$  elements.
  - a) The first part of the inner loop invariant says that the first  $j$  elements are properly ordered. When the loop is entered the first time with  $j = i - 1$ , this says that the first  $j = i - 1$  elements are properly ordered. But this is precisely what the outer loop invariant says on entry into the outer loop. (Remember: As the outer loop is entered,  $i$  is increased by 1; thus in terms of this new  $i$ , the first  $i - 1$  elements were in order when the loop was entered.) Nothing between the beginning of the outer loop and entry into the inner loop changes any element of the list, so this condition must still be true at the first entry into the inner loop.
  - b) The second part of the inner loop invariant says that the elements from  $j + 2$  to  $i$  are in order. At the first entry into the inner loop, this is vacuously true because there are no values of  $k$  between  $j + 2$  ( $= i + 1$ ) and  $i - 1$ .
  - c) The third part of the inner loop invariant says that  $\text{temp}$  ( $= n_i$ ) is less than all values it has been compared with thus far. This condition is also vacuously true the first time the loop is entered.
4. Does the truth of the inner loop invariant on one entry into this loop imply its truth on the next entry? What happens in the loop (assuming that the condition for exit is not satisfied)? The value that was in the  $j$  position is moved to the  $j + 1$  position and  $j$  is reduced by 1. The first part of the inner loop invariant is still true because it now refers to one less element, as the new  $j$  ( $j_{\text{new}}$  below) is one less than the old  $j$  ( $j_{\text{old}}$  below), and these elements are unchanged.

Is the second portion of the loop invariant now true for

$$j_{\text{new}} = j_{\text{old}} - 1?$$

The only new value of  $k$  is

$$j_{\text{new}} + 2 = j_{\text{old}} + 1. \quad (1)$$

So, is

$$n_k \leq n_{k+1}$$

for this value of  $k$ ? Yes, because these two values are both from the  $i - 1$  elements of the list which were sorted on entry into the outer loop.

For the third part of the loop invariant again the only value of  $k$  that we must consider (why?) is  $j_{\text{new}} + 2 = j_{\text{old}} + 1$ . On entry to the loop  $\text{temp} < n_{j_{\text{old}}}$  or the exit would have been taken. Since in the loop  $n_{j_{\text{old}}+1}$  becomes  $n_{j_{\text{old}}}$ , on exit  $\text{temp} < n_{j_{\text{old}}+1} = n_{j_{\text{new}}+2}$  which is what we want.

This is the only hard part of the proof of correctness. The rest is all downhill.

5. When the inner loop is exited, either  $\text{temp} \geq n_j$  or  $j = 0$ . The loop termination condition is the conjunction of the exit condition with the loop invariant.
6. Now, how about the outer loop? If its loop invariant is true on one entry, is it true on the next entry? When the inner loop is exited, elements 1 through  $j$  and  $j + 2$  through  $i$  are in order. Suppose that the inner loop was exited with  $j \neq 0$ . We need then only show that the elements  $j$ ,  $j + 1$ , and  $j + 2$  are in order. What the outer loop does, after the inner loop is exited, is to put  $\text{temp}$  (the old  $n_i$ ) in the  $j + 1$  position. Now  $\text{temp}$  is greater than or equal to the element in the  $j$  position because that's what caused the exit from the inner loop. And it's less than the element in the  $j + 2$  position because that's part of what the third clause of the inner loop termination condition says. (The clauses are the assertions between the  $\wedge$ 's.)

Suppose that  $j = 0$  on exit from the inner loop. We'll let you provide the appropriate argument [1].

Therefore the outer loop invariant really is invariant from one pass through the loop to the next.

7. The outer loop termination condition is just the conjunction of the loop invariant and the exit condition from the for-loop ( $i = m$ ). Is the output specification true? Yes, because it is simply the result of replacing  $i$  by  $m$  in the outer loop termination condition.

This completes our verification of Algorithm `INSERTION` except that we haven't shown that it terminates. Does it always? We leave this to a problem [2].

How important was the use of the universal quantifier in the assertions for Algorithm `INSERTION`? Well, we could have done without it if we had been willing to use a lot of ellipsis, which would have made the assertions considerably less readable. For more complicated algorithms with more complex loop invariants, quantifier notation would be even more desirable.

The verification of `INSERTION` was not simple. But, if you understood it and — even better — understand how to construct proofs for your own algorithms, you will have developed an important ability. As with all purported mathematical proofs, you should convince yourself that our proof really is a proof.

We didn't use induction explicitly in the verification of `INSERTION`. But you may have seen or, at least, had a sneaking suspicion that verification of an algorithm with one loop nested inside another, where both loops are controlled by a counter, was likely to have involved some kind of induction. Indeed, the verification that each of the loop invariants really was invariant was an induction proof in each case [4].

# Problems: Section 7.7

1. ⟨1⟩ Argue that the outer loop invariant in INSERTION is satisfied on reentry to the outer loop when the condition which causes exit from the inner loop is  $j = 0$ .
2. ⟨2⟩ Give a proof that INSERTION always terminates if the input specification is satisfied. What would happen if the input specification is not satisfied?
3. ⟨2⟩ When the list to be sorted by INSERTION has a length of 1, so that sorting it means doing nothing, the input specification is satisfied. But does the algorithm work properly? Why?
4. ⟨3⟩ Restate as proofs by induction the proofs that the inner and outer loop invariants hold on initial entry to the loop and at each reentry.
5. ⟨2⟩ Bubble sort. Here the idea is to compare successively adjacent items on a list and interchange them if they aren't in numerical order.
  - a) Suppose that you start at the beginning of a list and proceed to the end, first comparing the elements in the first two positions, interchanging them if they are out of order, then comparing the elements in positions 2 and 3 and interchanging, if necessary, and so on. Show that, after comparing and, if necessary, interchanging the elements in the last two positions, the largest element will be at the end of the list.
  - b) Then, going back to the beginning again and proceeding as before, show that the next-to-largest element ends up in the next-to-last position. You don't need to compare all successive pairs of elements this time. Where can you stop?
  - c) Argue that by continuing this procedure, you will eventually get the list in sorted order.
6. ⟨2⟩
  - a) Write Algorithm BUBBLE to implement the notation in [5].
  - b) Show how your algorithm works on the data in Fig. 7.12.
7. ⟨2⟩
  - a) How many passes are required (i.e., the number of times you went back to the start of the list) in Algorithm BUBBLE?
  - b) If the list were initially ordered, how could you modify your algorithm to prevent unnecessary passes? Do it.
  - c) Apply your modified algorithm to the data in Fig. 7.12.
8. ⟨2⟩ In bubble sort, as described in [5], larger items are “bubbled” (if you visualize the list as going from top to bottom, really they sink) to the end of the list. Modify BUBBLE so that small items bubble to the beginning.
9. ⟨3⟩ Your Algorithm BUBBLE should have two nested loops.
  - a) Determine a loop invariant for each. *Hint:* The outer loop invariant must state not only which items are already sorted but also the relation of these items to the unsorted elements. The inner loop invariant only has to state the relationship between the last item on a sublist and the ones which precede it.
  - b) For both loop invariants argue that they are true on entry to their respective loops and on each subsequent reentry.
  - c) State appropriate loop termination conditions and use the outer loop termination condition to show that the appropriate output specification is satisfied.
10. ⟨3⟩ We can do still better than the improvement suggested in [7]. Sometimes a pass results in putting more than one number in its correct final place. How could you discover this and thereby improve your algorithm? *Hint:* Each time you interchange two values, record the position of the interchange. Rewrite the modified algorithm from [7] to exploit your answer to the preceding question.
11. ⟨3⟩ [Computer project] Convert INSERTION and your BUBBLE from [5], [7], or [10] to computer programs. Compare their run times for a variety of data to try to determine which executes most rapidly.

## 7.8 Theorems and Proofs

Mathematical logic lies at the very foundations of mathematics. Since the proof of theorems by deductive methods — start with some premises and reason until you arrive at a desired conclusion — is a crucial part of all mathematics, you may find it surprising then that thus far in this chapter, with the exception of a brief excursion into deductive proofs in Boolean algebra (Section 7.5), the only method of proof we have featured is the use of truth tables to prove theorems of the propositional calculus (i.e., to prove that certain wffs are tautologies). But truth tables are clearly a very special kind of tool to use in proofs, since they depend upon being able to list all the possible instances of the values of the variables in a tautology.

In the predicate calculus the truth table method of proof is generally not even available. Even when the domain of a predicate variable is finite, as in Eq. (7), Section 7.6, there is normally a variable [ $m$  in Eq. (7), Section 7.6] which can assume values from an infinite set.

In any event, it makes sense in both the propositional calculus and the predicate calculus to be able to prove theorems by traditional deductive means. In Section 7.3 we noted several good reasons. Here we just mention that logic must include such deductions, defined in a precise way, if it is to model carefully and formally the normal deductive process of mathematics and thus help put it on a firmer basis. In this section we shall give an introduction to how to put deduction into logic. But be forewarned: This is a very large topic, whose surface we'll only barely scratch.

Just as has been the case with theorems in the foregoing chapters of this book, a theorem within a logical system is a justified claim that a stated conclusion follows from stated premises. Using the language earlier in this chapter (Section 7.3), a theorem is thus an *argument* that is “certified” by some accepted rules (i.e., proved). One certainly hopes that these rules ensure that the conclusion is true whenever the premises are true, that is, that every theorem is a *valid* argument.

Recall further that, in the propositional calculus, every valid argument is associated with an implication that is logically true, i.e., a tautology. (A similar statement is true for predicate calculus.) For example, consider the tautology

$$((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R). \quad (1)$$

which was (1) in Section 7.3. This is associated with the valid argument that, given premises  $P \Rightarrow Q$  and  $Q \Rightarrow R$ , then the conclusion  $P \Rightarrow R$  follows; that is,

$$\begin{array}{c} P \Rightarrow Q \\ Q \Rightarrow R \\ \hline P \Rightarrow R. \end{array} \quad (2)$$

This is indeed a theorem of propositional calculus as well, as we show in Example 1 below. The proof will consist of a sequence of steps that take us from the premises to the conclusion in a believable fashion.

But in logic “believable” means something considerably more formal than heretofore in this book. In a **formal deductive system** each step of such a

proof will have to be justified using a precisely stated **rule of inference**. In the remainder of this section we shall give two examples, one from the propositional calculus and one from the predicate calculus, of how to prove theorems in logic deductively.

To summarize, we are in a position to define precisely the concept of theorem within a logical system because we have not previously used “theorem” to describe a result within logic (except during a brief introductory foray into deduction in the special case of Boolean algebra, where the usage was consistent with what we do now). Otherwise we have only talked about wffs being tautologies and arguments being valid. To be sure, we had a theorem in this chapter (Theorem 1, Section 7.3), but it was a theorem *about* logic, not a theorem *within* logic. So here is our definition: a **theorem** within a logic system is any derivation of a conclusion from 0 or more premises, where the derivation consists solely of a sequence of steps allowed by the stated rules of inference.

## Proof in the Propositional Calculus

The theorem we shall prove deductively is (2). Before we can begin, however, we need to state the rules of inference we shall use to prove this theorem. The two we shall need are:

---

### Rule 1: Modus Ponens<sup>†</sup>

If  $A$  and  $B$  are wffs, then given  $A \Rightarrow B$  and  $A$ , we may infer  $B$ .

---

So this rule of inference says that any time you can assert the truth of the two premises ( $A \Rightarrow B$ ) and  $A$ , you may also assert the truth of the conclusion  $B$ . (Does this make intuitive sense to you? We hope it does because rules of inference in a deductive system should be completely acceptable, for otherwise how is one to believe the less obvious results one gets from many applications of these rules? In any event, compare this rule of inference with our argument about astrology in Section 7.1. What do you conclude?)

An argument in English that takes the form of modus ponens is

|                                                  |                   |
|--------------------------------------------------|-------------------|
| If you are reading this book, you must be crazy. | $A \Rightarrow B$ |
| You are reading this book. (Aren't you?)         | $A$               |
| Therefore you must be crazy.                     | $B$               |

The first two lines are the premises and the third is the conclusion that can be inferred from the premises. Here “you are reading this book” plays the role of  $A$  and “you must be crazy” plays the role of  $B$ . Please remember that this argument

---

<sup>†</sup>Latin for, roughly, a method of placing.

has the form: *If* the implication is true *and A* is true, *then B* is true. We don't, of course, believe that  $A \Rightarrow B$  is true!

Put another way, Modus Ponens asserts that

$$((A \Rightarrow B) \wedge A) \Rightarrow B. \quad (3)$$

is a tautology, but this is not the form in which it is used in our formal deductive system (see below).

---

### Rule 2: Conditional Proof

If, by temporarily *assuming* that  $A$  is true, it is possible to derive that  $B$  is true (in any number of steps using any available inference rules), then we may assert  $A \Rightarrow B$ .

---

Thus, this rule says that, if there is a valid argument beginning with the assumption that  $A$  is true and ending with the conclusion  $B$ ,

$A$

⋮

---

$B,$

then we may assert that  $A \Rightarrow B$ . When we use Rule 2 in a deductive proof, we indent the **conditional assumption**  $A$  and all the subsequent steps up through the conclusion  $B$ . Then we write  $A \Rightarrow B$  unindented because Rule 2 says that  $A \Rightarrow B$  is true independently of the truth of  $A$ . (Do you see why, after assuming  $A$  and deriving  $B$  from it, it is then sensible to conclude  $A \Rightarrow B$  whether  $A$  is true or not?)

#### EXAMPLE 1

Prove theorem (2) using rules of inference.

**Solution** In two-column format, we proceed as follows:

| Derivation           | Justification                                |
|----------------------|----------------------------------------------|
| 1. $P \Rightarrow Q$ | Premise                                      |
| 2. $Q \Rightarrow R$ | Premise                                      |
| 3. $P$               | Assumption                                   |
| 4. $Q$               | Apply Rule 1 (modus ponens) to lines 1 and 3 |
| 5. $R$               | Apply modus ponens to lines 2 and 4          |
| 6. $P \Rightarrow R$ | Apply Rule 2 to lines 3 and 5                |

The little subroutine 3–4–5 enables us to show that there is an argument beginning with the assumption  $P$  and concluding with the assertion of  $R$ . This argument makes use of lines 1 and 2, which, in this sense, act like “global” data to the subroutine. ■

It’s important that you understand why the proof in Example 1 is a *formal* proof. Each step is

- a restatement of a premise,
- a conditional assumption, or
- a new wff which is the result of a purely formal manipulation of the symbols involved using one of the rules of inference.

We used the term “subroutine” above because, in fact, the notion of assuming that something  $A$  is true and then using it to conclude that  $B$  is true corresponds very closely to the idea of a subroutine in a computer program in which some input parameters (the assumption  $A$ ) are provided and some output (the conclusion  $B$ ) is computed.

Sometimes in formal proof systems you start with some *axioms* and use these together with the premises and any theorems previously proved to prove new theorems. If you studied Euclidean geometry in high school, this will be how you proceeded. In the example above, we didn’t need any axioms to get started but only the rules of inference. There are a number of other possible rules of inference in addition to the two we have introduced. Some are given in [4]. Using them, we could prove more extensive theorems (most of which, by the way, would make very tedious reading). An obvious question is: Is there a finite set of axioms and rules of inference such that *any* valid argument in the propositional calculus can be proved with this set of rules? The answer is Yes, though we shall not state such a set. (In fact, no axioms are needed; rules of inference are enough!) Such a set of axioms and rules is called a **complete** set and the propositional calculus is, therefore, called a **complete system**. In addition, the propositional calculus is **consistent**: There are complete sets of rules of inference that do not allow you to prove anything that is invalid. To summarize: Every valid argument is a theorem (completeness) and every theorem is a valid argument (consistency).

This concludes our very brief introduction to deductive methods in the propositional calculus. Now on to a similar discussion for the predicate calculus.

## Proof in the Predicate Calculus

Although in the predicate calculus, as in the propositional calculus, there are rules for forming wffs, we didn’t discuss them in Section 7.6 because they are rather complicated. Instead, we’ll rely on your intuitive feel for appropriate formulas whenever necessary below.

A theorem in the predicate calculus, as in the propositional calculus, begins with some wffs that are the premises of the theorem. Then, much as for the propositional calculus, we use rules of inference to derive a conclusion, which must

be true if the premises are true. To prove theorems in the predicate calculus, we shall need some rules of inference involving quantifiers in addition to those in the previous subsection for the propositional calculus. Then we use deduction as in the propositional calculus.

As with the propositional calculus, we will content ourselves with a single example of a proof in the predicate calculus.

## EXAMPLE 2

Prove using rules of inference: If all primes are integers and all integers are rational, then all primes are rational. (Are the premises true?)

**Solution** First we state the wff associated with this theorem:

$$[(\forall x)(PM(x) \Rightarrow I(x)) \wedge (\forall x)(I(x) \Rightarrow R(x))] \Rightarrow [(\forall x)(PM(x) \Rightarrow R(x))], \quad (4)$$

where  $PM(x)$  is the predicate that  $x$  is a prime,  $I(x)$  is the predicate that  $x$  is an integer, and  $R(x)$  is the predicate that  $x$  is rational.

Our proof will use two rules of inference for the predicate calculus (which we number 3 and 4 following Rules 1 and 2 above):

- **Rule 3, Universal Instantiation.** If  $(\forall x)P(x)$  is true, we may assert  $P(x)$  for any  $x$  in the domain of the quantification.
- **Rule 4, Universal Generalization.** If  $P(x)$  is true for an  $x$  chosen *arbitrarily* in a domain, we may assert  $(\forall x)P(x)$  where the quantification is over the same domain. (To prove  $P(x)$  for an arbitrary  $x$  means that in the proof we cannot use any special properties of the  $x$  chosen. Thus to prove a theorem about quadrilaterals, we couldn't consider a square and then use some property of “squareness” in the proof.)

Now here's the proof that the argument associated with (4) is a theorem:

|    | Derivation                             | Justification |
|----|----------------------------------------|---------------|
| 1. | $(\forall x)(PM(x) \Rightarrow I(x)).$ |               |
| 2. | $(\forall x)(I(x) \Rightarrow R(x)).$  |               |
| 3. | $PM(x) \Rightarrow I(x).$              |               |
| 4. | $I(x) \Rightarrow R(x).$               |               |
| 5. | $PM(x).$                               |               |
| 6. | $I(x).$                                |               |
| 7. | $R(x).$                                |               |
| 8. | $PM(x) \Rightarrow R(x).$              |               |
| 9. | $(\forall x)(PM(x) \Rightarrow R(x)).$ |               |

We've omitted the justifications above and left it to a problem [12] for you to fill them in. If you can do this, you will have understood each step of this proof and you will probably also have understood the basic ideas of quantification. One caution, though: It is *not* true that everything in the predicate calculus can be proved by stripping the quantifiers, doing some propositional calculus, and then putting the quantifiers back. You always can strip them, but you can only put them back if the

variables are still arbitrary. We explained “arbitrary” in an intuitive way above, but in a formal deductive system such a concept must be reduced to rules, and this takes some doing. We won’t pursue this topic further here. ■

As with the propositional calculus, there is a complete and consistent set of rules of inference for the predicate calculus. This may not surprise you. Indeed, for most of history mathematicians have pretty much assumed (if they thought about it) that any mathematical truth could be proved. However, the completeness and consistency of the predicate calculus is a hard theorem, and worse than that, the predicate calculus is about as far as you can go along these lines. Roughly speaking, every deductive system that adds enough more expressive power to include arithmetic is either inconsistent or incomplete. This is Gödel’s famous *Incompleteness Theorem*.

One final thought. Completeness and incompleteness results are theorems about theorems (that is, **metatheorems**). Just as we needed a careful formulation of algorithmic language in order to prove theorems about algorithms, one needs a careful formulation of logic to prove theorems about theorems. While we have merely stated some theorems about theorems in the previous paragraph, we hope we have given you a taste of how logic can be used to study the foundations of mathematics.

## Problems: Section 7.8

1. ⟨1⟩ Another common rule of inference is **modus tollens** (Latin for, roughly, a method of removing) which states that: If  $A$  and  $B$  are two wffs such that  $A \implies B$  and  $\neg B$  are true, then we may assert the truth of  $\neg A$ .

- a) State the wff associated with modus tollens.  
b) Use a truth table to show that this wff is a tautology.  
c) Similarly show that the wff associated with modus ponens in (3) is a tautology.

2. ⟨2⟩ Prove that the argument

$$P \implies Q$$

$$Q \implies R$$

$$\neg R$$

$$\hline \hline$$

$$\neg P.$$

is a theorem,

- a) By reasoning informally using discursive English.

- b) By computing the truth table for the associated wff.  
c) By a formal deduction using only modus tollens.  
3. ⟨2⟩ Explain informally why the rule of conditional proof is true. That is, if

$$A$$

$$\hline$$

$$B$$

is a valid argument, why must

$$\hline \hline$$

$$A \implies B$$

be a valid argument.

4. ⟨1⟩ Consider the following additional rules of inference:

$$A \wedge B$$

$$\hline$$

$$B \wedge A.$$

$$A \wedge B$$

$$\hline$$

$$A.$$

$$A$$

$$\hline$$

$$A \vee B.$$

- a) Explain each of these rules in words.

- b) Prove that each is valid using a truth table for the associated wff.

The middle rule is called the **Rule of Detachment**, because you get to detach the first of two propositions from a conjunction.

5. ⟨2⟩ Prove that the wff

$$(P \implies Q) \implies (\neg Q \implies \neg P)$$

is a tautology

- a) By reasoning informally using discursive English.

- b) By using a truth table

- c) By a formal deduction of the associated theorem using modus tollens and conditional proof.

6. ⟨3⟩ There is a problem with [5]. Whereas every argument has a unique wff associated with it, each wff can have several arguments associated with it. You probably took the “associated theorem” to be

$$\begin{array}{c} P \implies Q \\ \hline \neg Q \implies \neg P \end{array}$$

but you could have taken it to be

$$\begin{array}{c} \hline (P \implies Q) \implies (\neg Q \implies \neg P) \end{array}$$

that is, an argument with no premise. In this case, now prove the latter argument by rules of inference. In doing so, you will get an idea about how Conditional Proof makes it possible for propositional logic to be complete using only rules of inference, not any axioms. You will also appreciate some of the additional rules of inference in [4].

7. ⟨3⟩ Use the rules of inference in the text and those of [1] and [4] to come up with formal deductions for the following arguments, where  $A$  and  $B$  are arbitrary wffs. That is, provide the necessary intermediate steps and the justification for each step.

- a) Given  $A \wedge B$ , one may infer  $B$ . Note: This is *not* a case of the 2nd rule of [4]. You must take two steps and combine two rules to get it. In a formal system, you may not make without comment the obvious jumps which are standard in informal proofs.

- b) From premises  $A$ ,  $A \implies (B \wedge C)$  and  $C \implies D$ , one concludes  $D$ .

- c) If  $(A \implies (B \wedge C)) \wedge (B \implies D)$ , then  $A \implies D$ . Hint: Use conditional proof.

- d) Assuming  $(A \implies B) \wedge ((B \vee C) \implies D)$ , then  $A \implies D$ .

8. ⟨2⟩ Suppose that the assumption of  $P$  allows you to prove both  $Q$  and  $\neg Q$  where  $Q$  is any wff. State in words why you can then assert  $\neg P$ . This fact is formalized as yet another rule of inference, the **Law of Contradiction**.

9. ⟨2⟩ Suppose you are in a subproof of a formal deduction as in lines 3–5 of Example 1. Explain why any line before the subproof (e.g., line 2 in Example 1) could be, if desired, inserted as a line of the subproof.

10. ⟨2⟩ Use the ideas in [8–9] to prove:

a)  $\neg\neg P \implies P$ .      b)  $P \implies \neg\neg P$ .

(Hint:  $Q$  in [8] can itself be  $P$ .)

11. ⟨3⟩ Prove by formal deduction:  $(P \wedge Q) \implies R$  is equivalent to  $(P \wedge \neg R) \implies \neg Q$ . To prove that two statements are equivalent in a formal system, you prove that each can be deduced from the other.

12. ⟨2⟩ Fill in the Justification column in Example 2.

13. ⟨2⟩ (set inclusion and predicate logic) Recall that  $S \subset T$  was defined to mean every element of  $S$  is an element of  $T$ . That is,  $S \subset T$  means

$$(\forall x)[x \in S \implies x \in T].$$

(We could go a step further and write the predicate  $x \in S$  as  $S(x)$ , where  $S(x)$  is the property of belonging to set  $S$ , but there is no need to do this.) With this translation into logic notation, prove: If  $S \subset T$  and  $T \subset U$  then  $S \subset U$ .

14. ⟨2⟩ Recall that the definition of  $\overline{S}$ , the complement of set  $S$ , is the set of all elements (in the universe under discussion) not in  $S$ . In other words,  $x \in \overline{S}$  means  $x \notin S$ . With this understanding, use predicate logic to prove: If  $S \subset T$  then  $\overline{T} \subset \overline{S}$ .

15. ⟨2⟩ In both algorithm verifications in the text, of Algorithm MULT in Section 7.4 and Algorithm INSERTION in Section 7.7, there is an implicit Universal Instantiation at the beginning and a Universal Generalization at the end. Explain. (See [5, Section 7.6].)

16. ⟨3⟩ Here are two more rules of inference in predicate calculus. 1) **Existential Generalization**: if  $P(x_0)$  has been asserted for any constant  $x_0$ , then we may assert  $(\exists x)P(x)$ . 2) **Existential Instantiation**: If  $(\exists x)P(x)$  has been asserted, then we

may assert  $P(x_0)$ , where  $x_0$  is a constant that has not appeared previously in the deduction.

Using one or both of these rules as well as earlier ones, deduce:

- a) If  $(\forall x)[P(x) \implies Q]$ , then  $[(\exists x)P(x)] \implies Q$ .

b) The converse.

Compare with the intuitive explanation in [20, Section 7.6].

17.  $\langle 2 \rangle$  Do a deduction to prove the graph argument in [14b, Section 7.6].

## Supplementary Problems: Chapter 7

1.  $\langle 2 \rangle$  Suppose you want to show that a whole series of statements  $S_1, S_2, \dots, S_n$  are equivalent, that is, that  $S_i \iff S_j$  is a tautology for each  $i$  and  $j$ . Typically, such a mutual equivalence is proved by proving a number of implications; e.g.,  $S_1 \implies S_2$ ,  $S_2 \implies S_3$ , and so on. For instance, once these two implications are proved, it isn't necessary to prove  $S_1 \implies S_3$  directly, so half of  $S_1 \iff S_3$  has been done. What is the least number of implications necessary to prove the mutual equivalence of all the  $S_i$ 's? Prove that you are right. Hint: Use graph theory.

2.  $\langle 3 \rangle$  You are kidnapped by a gang of logic-fiends and placed in a room with two doors. The leader tells you that the red door leads to freedom and the green door to death but you can't see the colors. He also tells you that you may ask his assistant one Yes-No question and that his assistant understands logic perfectly, but either is always truthful or is a complete liar. Use the methods of this chapter to figure out a question to ask. Hint: There are only two variables — which door leads to freedom and whether the assistant lies or tells the truth. Let's say that you want an answer of Yes to your question to mean take the red door. For all four pairs of values for the variables, figure out what the truthful answer to your question would have to be. From this result you can construct the question.

3.  $\langle 3 \rangle$  The five major logical operators,  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\implies$ , and  $\iff$ , are not independent. For instance, if somebody doesn't understand  $\iff$ , you can explain that  $P \iff Q$  means  $[(P \implies Q) \wedge (Q \implies P)]$ .

- a) Using only  $\wedge$  and  $\neg$ , can you explain any of the other three operators? Which ones? How?
- b) Same as a), but start with  $\vee$  and  $\neg$ .
- c) Same as a), but start with  $\implies$  and  $\neg$ .

d) Show that, using  $\iff$  only, none of the other four operators can be explained. Hint: First show that truth tables for any wffs using only  $P, Q$  and  $\iff$  must have 0, 2, or 4 T values.

4.  $\langle 3 \rangle$  Suppose you found a tribe whose language included a word for  $|$  (the Sheffer stroke — see Fig. 7.3), but which didn't have words for any of the other logical operators.

- a) How could you explain "not" to these people?
- b) How could you explain "and"?
- c) How could you explain "or"?

It follows that you can express every wff to these people; see [25], Section 7.5.

5.  $\langle 3 \rangle$  Suppose that you found another tribe whose language included a word for  $\downarrow$  (Peirce's arrow — see Fig. 7.3), but which didn't have equivalents for any of the other logical operators.

- a) How could you explain "not" to these people?
- b) How could you explain "and"?
- c) How could you explain "or"?

It follows that you can express every wff to these people; see [25], Section 7.5.

6.  $\langle 2 \rangle$  Show how to construct circuits for all five standard logical operators ( $\neg, \wedge, \vee, \implies$ , and  $\iff$ ) using NAND ( $=|$ ) gates. Hint: See [3] and [4].

7.  $\langle 2 \rangle$  Show how to construct circuits for all five standard logical operators using NOR ( $=\downarrow$ ) gates. Hint: See [6].

8.  $\langle 2 \rangle$  Why are most actual logic circuits constructed with just NAND and NOR gates rather than AND, OR, and NOT gates?

9.  $\langle 3 \rangle$  Section 7.6, doesn't give a complete description of how to negate a formula in the predicate calculus. Why? Because it tells us to replace a quantifier with the other quantifier and negate the

predicate inside the quantifier, but “inside” may consist of several pieces joined by operators like  $\wedge$ , and the pieces themselves may have quantifiers. For instance, the formula might be

$$(\forall i) [((\exists j)P_j \wedge (\forall k)Q_{ik}) \implies \neg(R_i \wedge \neg S)].$$

Come up with a recursive description of how to negate all predicate calculus formulas by changing the quantifiers. *Suggestion:* All valid formulas in the predicate calculus may be viewed as having a binary tree structure, with operators and quantifiers on the internal nodes and predicates and atoms on the leaves. For instance, the formula above can be drawn as

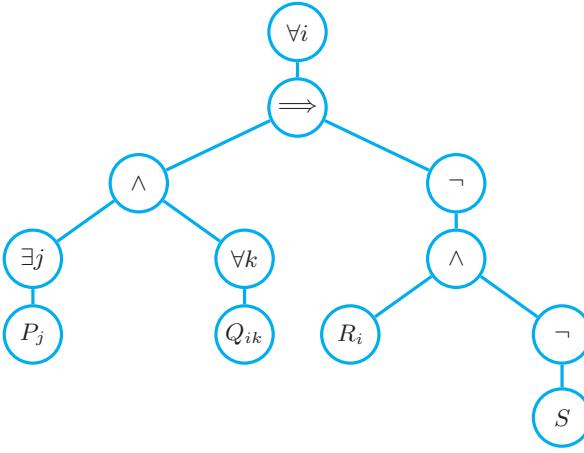


Figure out a rule for negation which explains what to do to the root and to the subtree hanging from the root. For the given formula, give both the negation tree and the negation formula. Describe in general terms how to negate any valid formula using its tree.

10. {3} A formal deduction system may be viewed as a game in which you start with some set of symbols and rules for forming these symbols into *legal* strings such as the rules for forming wffs in Section 7.2. Then you are given additional rules, which we have called axioms and rules of inference, that let you elevate some of these legal strings to the status of *winning* strings. The Game of Stars and Slashes is played with four symbols,  $A$ ,  $B$ ,  $*$ , and  $/$ . Legal strings are formed using these four symbols, where each may be repeated as often as desired. The rules for forming winning strings are:

- $A$  or  $B$  by itself is a winning string.
  - If  $S_1$  and  $S_2$  are winning strings, then  $/S_1 * S_2/$  is a winning string.
- a) Construct all winning strings with nine or fewer symbols.
- b) From the result of a) can you infer a formula for the length of possible winning strings? Can you prove that your formula is correct?
- c) Using the rules for winning strings, determine whether each of the following is a winning string.
- $//A * A/ * //B * A/ * A//$
  - $/A * /A * B/$
  - $////B * B/ * A/ * B/ * B * B//$

11. {2} Consider an  $n$ -variable truth table. Let’s call the value of the expression in the  $i$ th row and last column  $f_i$  and denote the DNF product corresponding to row  $i$  by  $m_i$ . (Recall that each variable appears once in this product, plain or complemented, depending on the row.) Show that the DNF for the expression defined by the truth table is given by

$$\sum_{i=0}^{2^n-1} f_i m_i. \quad (1)$$

12. {2} Argue that the complement of the expression in (1) is given by

$$\sum_{i=0}^{2^n-1} \bar{f}_i m_i. \quad (2)$$

13. {3} As in [11] let  $m_i$  represent the DNF product of the variables in the  $i$ th row of a truth table. Similarly, let  $M_i$  represent the sum of the same variables. Prove the correctness of the following identities.
- a)  $\bar{m}_i = M_{2^n-1-i}$ .
- b)  $\bar{M}_i = m_{2^n-1-i}$ .
14. {3} Use De Morgan’s laws repeatedly and the result of [13] applied to expression (2) in [12] to show that the CNF form of any expression defined by a truth table is

$$\prod_{i=0}^{2^n-1} (f_i + M_{2^n-1-i}).$$

# Coming Full Circle with Biology and Minimax Theorems

## E.1 DNA Matching

---

Molecular biology is a great source of problems for discrete algorithmic mathematics. DNA can be represented as long “words” using 4 letters (representing the 4 organic bases whose sequence completely determines the DNA). Proteins can be presented as long words using 20 letters (representing the 20 amino acids). To study how close two species are, you want to study how close their DNA is. To study a mutation (which amounts to a small change in the DNA) you want to find two sequences of DNA that are almost but not quite the same. These strings tend to be very long, so you need algorithms to analyze closeness. You need mathematics to prove that the algorithms are correct and to analyze their efficiency.

Thus molecular biology is a wonderful topic with which to review some of the key themes of this book, and to show their relevance to current work in yet another discipline.

There is so much need for good algorithms in biology today that this approach is now called *computational biology*. There is a lot more to it than the study of closeness of DNA symbol strings. For instance, the geometry of the molecules of living organisms turns out to be very important for determining interactions (e.g., whether a cell can fend off a virus). How DNA entangles itself is studied with knot theory. How molecules fold up involves a lot of continuous mathematics to model the forces involved and to find stable equilibria.

For our foray into biology we will limit ourselves to the issue of determining the closeness of strings of symbols, a subject also known as **pattern matching**, a name that suggests correctly that the subject has lots of applications outside biology as well. Pattern matching is a big subfield of computational biology — probably the most studied subfield to date. Discrete mathematics has good answers for many of the problems and there are working algorithms that biologists consult regularly. After you read this section, check out the BLAST algorithms (Basic

Local Alignment Search Tool) at [www.ncbi.nlm.nih.gov/BLAST](http://www.ncbi.nlm.nih.gov/BLAST). Of course, these are industrial strength algorithms that involve many more ideas than discussed here, but the ideas discussed here are the starting point.

We title this Epilogue *Coming Full Circle* because, as we will see shortly, the mathematics will bring us all the way back to the Prologue. Thus, after treating the biology problem in the current section, in Section E.2 we will conclude by extending some ideas from the Prologue a little further, independently of the biology discussion.

## Defining the Problem

First, a quick biology lesson. DNA (deoxyribonucleic acid) is the fundamental encoding agent of all living organisms. It is a double helix, built around two long sequences of just four small organic base molecules: Adenine, Cytosine, Guanine, and Thymine, henceforth A, C, G and T. These bases always come in specific pairs, A with T, C with G, one base from each pair in corresponding positions on each strand of the helix. Although many other molecular pieces join around these pairs to make the whole huge DNA molecule, a DNA molecule is completely determined by the sequence of bases in one strand. So that's what we'll deal with, sequences of A, C, G and T.

We will denote sequences by script capitals, typically  $\mathcal{S}$  and  $\mathcal{T}$ , sometimes with primes ( $\mathcal{S}'$ ) and sometimes with hats ( $\hat{\mathcal{S}}$ ), for particular purposes to be explained below. The individual letters in  $\mathcal{S}$  will be denoted by the variables  $s_1, s_2, \dots, s_k$ . For instance, if  $\mathcal{S} = \text{ACCT}$ , then  $s_1 = \text{A}$  and  $k = 4$ .

Second, we need to distinguish between subsequences and substrings. Each word is a sequence. A **subsequence** is what remains after zero or more letters are removed but the order is maintained in the remainder. Thus ACT is a subsequence of ACGTCCT (in more than one way) but ATG is not. A **substring** is a subsequence which is consecutive in the original word. Thus GTCC is a substring of ACGTCCT, but GCCT is not, although the latter is a subsequence.

The measure of closeness we will develop is related to how DNA mutates. The basic mutation processes are *deletion, insertion, and substitution*. Take the word ACGTG. The word AGTG is obtained by deleting the C. The word ACGTGG is obtained from ACGTG by inserting G between T and G (or after the final G). Finally, the word ACGAG is obtained from ACGTG by substituting an A for the T. These processes don't need to affect just one letter. For instance, ATG is obtained from ACGTG by deleting the adjacent letters CG.

Given two strings that we think are related, we often wish to look for the minimum way that both strings could have been created by mutation from some common ancestor string. To get an idea about how to measure closeness, let's first suppose we know the ancestor. Let's use the example in the previous paragraph, with ACGTG the ancestor, with one descendant being  $\mathcal{S} = \text{AGTG}$ , gotten by deleting C, and with another descendant being  $\mathcal{T} = \text{ACGAG}$ , gotten by substituting A for T. So  $\mathcal{S}$  and  $\mathcal{T}$  differ by only 2 mutations.

But in actuality, we don't usually know the ancestor DNA. We only have  $S$  and  $T$ . Or, it might not be a question of finding an ancestor;  $S$  might come from the normal strain of some organism,  $T$  might be from a variant, and we might be looking for the source of the variation. In any event, how do we *align*  $S, T$  (line them up with every symbol accounted for — we give a formal definition later) to discover that they differ by only two mutations? Well, first let's align them naively:

$$\begin{array}{cccccc} A & G & T & G & - \\ & & & & \\ A & C & G & A & G \end{array} \quad (1)$$

Look at this column by column. The first column is a **match**. The next column is a **mismatch**, as are columns 3 and 4. Notice the underscore in the 5th column of the top line. Since the two strings are different lengths, we have to allow letters in one string to line up with blanks in the other, and we indicate a blank with an underscore. Let us call a column containing a blank and a letter a **pass**. So this alignment contains one match (good), 3 mismatches (bad) and one pass (also bad). Mismatches and passes are bad because they correspond to mutations, which are rare events, so the more mutations we need to explain an alignment, the less likely the two strings are really related, at least in the way required by that alignment.

But there is no reason to put all the blanks at the end. Since insertions and deletions can occur anywhere, to account for them we have to allow blanks anywhere. And once we allow this, we can get a much better alignment:

$$\begin{array}{ccccc} A & - & G & T & G \\ & & & & \\ A & C & G & A & G \end{array} \quad (2)$$

Now we have 3 matches, 1 mismatch, and 1 pass, for a total of only 2 demerits. In fact, for this small example it is easy to see that this is the best we can do, but for larger strings we will need a systematic method. Anyway, the point here is that aligning the strings in the way that creates the fewest mismatches and passes probably shows how they are actually related.

But there is another issue: Should mismatches and passes count equally as demerits? We address this issue through an example.

## EXAMPLE 1

Align AACC to ACCC with and without blanks.

**Solution** The no-blank alignment is

$$\begin{array}{cccc} A & A & C & C \\ & A & C & C & C \end{array} \quad (3)$$

which has 3 matches and one mismatch. Do blanks allow an improvement? Well, one can always trade a mismatch for two passes as follows:

$$\begin{array}{ccccc} A & A & - & C & C \\ & & & & \\ A & - & C & C & C \end{array} \quad (4)$$

The number of matches is unchanged from (3). ■

So which alignment in Example 1 is better, that is, more likely to indicate the biological history: one mismatch (representing a substitution) or two passes (representing two insertions or deletions)? It turns out that insertions and deletions

are usually rarer than substitutions, but in certain instances there may be specific information to the contrary. Fortunately, we as mathematicians don't have to decide the issue in advance. Instead of merely counting demerits, we can weight them. If matches are much better than mismatches and passes, then we will weight each match much higher than either a mismatch or a pass. In fact, for the rest of this section we will assume that matches have positive weight and mismatches and passes have negative weight (although this assumption is not strictly necessary [9]). We will leave it to the biologists to decide the relative weights of passes and mismatches. In any event, the best alignment will be the one with the highest total weight, where the weight of an alignment is the sum of the weights in its columns.

This leads to the following definition.

**Definition 1.** An **alignment**  $\alpha$  of two strings  $\mathcal{S}, \mathcal{T}$  of lengths  $k$  and  $l$  is any way of inserting blanks into one or both of them so that they are equally long, say length  $\ell$ , and so that no position has a blank in both words.

The **value** of alignment  $\alpha$  of  $\mathcal{S}$  and  $\mathcal{T}$  is defined to be

$$V(\mathcal{S}, \mathcal{T}, \alpha) = c_m N_m + c_u N_u + c_p N_p, \quad (5)$$

where the  $c$ 's are constants and the  $N$ 's are respectively the number of letter matches, the number of letter mismatches (un-matches), and the number of passes (blanks versus a letter) in the alignment.

Equivalently, define  $\hat{\mathcal{S}}, \hat{\mathcal{T}}$  to be the **enlargements** of  $\mathcal{S}, \mathcal{T}$  with blanks according to the alignment  $\alpha$ , so that the terms in  $\hat{\mathcal{S}}$  are  $\hat{s}_1, \hat{s}_2, \dots, \hat{s}_\ell$  and the terms in  $\hat{\mathcal{T}}$  are  $\hat{t}_1, \hat{t}_2, \dots, \hat{t}_\ell$ . Then

$$V(\mathcal{S}, \mathcal{T}, \alpha) = \sum_{i=1}^{\ell} v(\hat{s}_i, \hat{t}_i), \quad (6)$$

where

$$v(s, t) = \begin{cases} c_m & \text{if } s = t, \\ c_u & \text{if both } s \text{ and } t \text{ are letters and } s \neq t, \\ c_p & \text{if one of } s \text{ and } t \text{ is a blank.} \end{cases} \quad (7)$$

By the ***i*th column** of an alignment  $\hat{\mathcal{S}}, \hat{\mathcal{T}}$ , we mean the pair  $(\hat{s}_i, \hat{t}_i)$ , usually arranged vertically as  $\begin{bmatrix} \hat{s}_i \\ \hat{t}_i \end{bmatrix}$ .

A **best alignment**  $\alpha$  is one for which  $V(\mathcal{S}, \mathcal{T}, \alpha)$  is maximized. The value of a best alignment is called the **measure of similarity** of  $\mathcal{S}$  and  $\mathcal{T}$  and is denoted  $V(\mathcal{S}, \mathcal{T})$ . That is

$$V(\mathcal{S}, \mathcal{T}) = \max_{\text{all } \alpha} \{V(\mathcal{S}, \mathcal{T}, \alpha)\}. \quad (8)$$

Note the use of  $\hat{\mathcal{S}}$  and  $\hat{\mathcal{T}}$ . We will always use hats on string names to represent enlargements with blanks so that the resulting strings are equally long and can be lined up column by column.

The reason for insisting that blanks not line up in the same position is because it doesn't make sense biologically: it would mean that we are supposing additional rare deletion events that aren't necessary to explain the similarity of two strings.

Since double blanks are not allowed, it is easy to see that  $\ell \leq k + l$ ; why?

## EXAMPLE 2

Find the values of the two alignments in Example 1. Use  $c_m = 1$ ,  $c_u = -1.5$  and  $c_p = -1$ .

**Solution** We use this example to show how all the notation in Definition 1 works. Let  $\mathcal{S} = \text{AACC}$ ,  $\mathcal{T} = \text{ACCC}$ . Let  $\alpha$  be the first alignment,  $\beta$  the second. In  $\alpha$ ,  $\hat{\mathcal{S}} = \mathcal{S}$  and  $\hat{\mathcal{T}} = \mathcal{T}$ , so the columns of  $\alpha$  are

$$\begin{bmatrix} A \\ A \end{bmatrix}, \begin{bmatrix} A \\ C \end{bmatrix}, \begin{bmatrix} C \\ C \end{bmatrix}, \begin{bmatrix} C \\ C \end{bmatrix}$$

with  $v$ -values 1,  $-1.5$ , 1, 1. Therefore  $V(\mathcal{S}, \mathcal{T}, \alpha) = 1.5$ .

In  $\beta$ ,  $\hat{\mathcal{S}} = \text{AA\_CC}$  and  $\hat{\mathcal{T}} = \text{A\_CCC}$ . Thus the columns of  $\beta$  are

$$\begin{bmatrix} A \\ A \end{bmatrix}, \begin{bmatrix} A \\ \_ \end{bmatrix}, \begin{bmatrix} \_ \\ C \end{bmatrix}, \begin{bmatrix} C \\ C \end{bmatrix}, \begin{bmatrix} C \\ C \end{bmatrix}$$

with  $v$ -values 1,  $-1$ ,  $-1$ , 1, 1, so  $V(\mathcal{S}, \mathcal{T}, \beta) = 1$ .

Thus with these  $c$ 's,  $\alpha$  is the better alignment. However, for some different values of the  $c$ 's, the other alignment has the higher value. We leave it to you (see [2]) to show that so long as  $c_u < 2c_p \leq 0$ , then  $\beta$  is the better alignment, because in this case the best alignment of any two strings never contains a mismatch but only passes and matches. (Recall, however, that replacing mismatches by passes is rarely a reasonable biological outcome.) ■

## Matching Algorithms

All right, how shall we devise an algorithm to find a best alignment? Why, the Recursive Paradigm of course! Suppose we already knew how to best align shorter sequences. How would we align  $\mathcal{S}$  and  $\mathcal{T}$ ?

As always with recursion, we also need to generalize. Since we will need to know about shorter sequences than  $\mathcal{S}$  and  $\mathcal{T}$  anyway, we ask: How could we best align *any* initial substrings of  $\mathcal{S}$  and  $\mathcal{T}$ , assuming we knew how to align even shorter substrings.

So, let  $\mathcal{S}_i$  be the substring of  $\mathcal{S}$  consisting of its first  $i$  symbols and similarly for  $\mathcal{T}_j$ . Our goal is to optimally align  $\mathcal{S}_i, \mathcal{T}_j$  and we suppose we already know how to optimally align substrings of them. So consider any alignment of  $\mathcal{S}_i$  and  $\mathcal{T}_j$ . That is, we enlarge them to  $\hat{\mathcal{S}}$  and  $\hat{\mathcal{T}}$  with zero or more blanks in each so that both have the same length,  $\ell$ . Let's look at the last ( $\ell$ th) position in the hopes that we can somehow peel it off and reduce to a previous case. Well, in this enlargement

either 1) the items in that last position are letters, in which case they are the last letters of  $\mathcal{S}_i$  and  $\mathcal{T}_j$  (that is,  $\hat{s}_\ell = s_i$  and  $\hat{t}_\ell = t_j$ ) or else 2) one of the items in that position is a blank (that is, either  $\hat{s}_\ell$  or  $\hat{t}_\ell$  is a blank). In case 1), the last position contributes a summand of  $c_m$  or  $c_u$  to  $V$  (depending on whether or not the letters are equal). In case 2) it contributes  $c_p$  (since one is a blank).

In case 1), all the previous positions are an alignment of  $\mathcal{S}_{i-1}$  with  $\mathcal{T}_{j-1}$ . Now here's the crucial observation. By Eq. (6) we can align  $\mathcal{S}_{i-1}, \mathcal{T}_{j-1}$  any way we please (i.e., insert blanks anywhere we wish in  $\mathcal{S}_{i-1}$  and  $\mathcal{T}_{j-1}$  so long as we don't violate the no-two-blanks-in-a-column rule), and it won't affect the contribution of the  $\ell$ th position to the value of the final alignment of  $\mathcal{S}_i$  and  $\mathcal{T}_j$ . Therefore, since we wish to maximize  $V(\mathcal{S}_i, \mathcal{T}_j, \alpha)$  we should use the best alignment of  $\mathcal{S}_{i-1}, \mathcal{T}_{j-1}$ . By the recursive paradigm, we assume we already know that best alignment.

Similarly, for case 2) if the  $\ell$ th position in  $\hat{\mathcal{S}}$  is a blank, the recursive problem to is to optimize the alignment of  $\mathcal{S}_i$  and  $\mathcal{T}_{j-1}$  because before the  $\ell$ th position you have to match all of  $\mathcal{S}_i$  with  $\mathcal{T}_{j-1}$ . If instead the  $\ell$ th position of  $\hat{\mathcal{T}}$  is a blank, the recursive problem is to optimize the alignment of  $\mathcal{S}_{i-1}$  and  $\mathcal{T}_j$ .

Which of these three reductions is best? Whichever has the highest value when the  $\ell$ th position is put back in. Setting

$$V(i, j) = V(\mathcal{S}_i, \mathcal{T}_j)$$

to simplify the notation slightly, we get the recurrence

$$V(i, j) = \max \left\{ \begin{array}{l} v(s_i, t_j) + V(i-1, j-1), \\ c_p + V(i-1, j), \\ c_p + V(i, j-1) \end{array} \right\}. \quad (9)$$

For this recurrence to determine  $V$  properly, there must be a basis case, or more likely basis cases, as this is a recurrence in two variables. Indeed, just as in Fig. 5.5, Section 5.3, the basis cases are all those cases along the left and bottom borders. In the current problem that means values  $V(i, j)$  for which  $i$  or  $j$  is 0, that is, for which  $\mathcal{S}$  or  $\mathcal{T}$  is empty. Specifically,

$$V(i, 0) = V(0, i) = c_p i, \quad (10)$$

because the only way to align a sequence of length 0 with a sequence of length  $i$  is to enlarge the empty sequence with  $i$  blanks.

In conclusion, from basis (10) and inductive step (9) one can easily write for any fixed values of  $c_m, c_u, c_p$  a recursive algorithm to find  $V$ , the measure of similarity of any two sequences. See Algorithm E.1. With a little more work of a sort we have seen many times, the algorithm can be enhanced to list an optimal alignment [28].

One might worry a little if this algorithm is guaranteed to terminate, since it involves two variables, and unlike in our various Euclid algorithms, we can't argue termination using just one of the variables, because neither variable is guaranteed to decrease every time. However, at least one of them is decreased each time, so eventually one of them reaches 0 and a basis case kicks in. Put another way, proof of termination is by finite descent on the sum  $i + j$ .

**Algorithm E.1****StringAlign-  
Rec**

|                                                                                                                                                    |                                         |
|----------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|
| <b>Input</b> $\mathcal{S}, \mathcal{T}$                                                                                                            | [Strings of length $k, l$ ]             |
| $c_m, c_u, c_p$                                                                                                                                    | [Typically $c_m > 0, c_u, c_p \leq 0$ ] |
| <b>Output</b> $V(k, l)$                                                                                                                            | [Measure of similarity]                 |
| <b>Algorithm</b> STRINGALIGN-REC                                                                                                                   |                                         |
| <b>function</b> $V(i, j)$                                                                                                                          |                                         |
| <b>if</b> $i = 0$ <b>then</b> $V(i, j) \leftarrow c_p j$                                                                                           | [Case if]                               |
| <b>j</b> = 0 <b>then</b> $V(i, j) \leftarrow c_p i$                                                                                                |                                         |
| <b>ij</b> $\neq 0$ <b>then</b> $V(i, j) \leftarrow \max \begin{cases} v(s_i, t_j) + V(i-1, j-1) \\ c_p + V(i-1, j) \\ c_p + V(i, j-1) \end{cases}$ |                                         |
| <b>endif</b>                                                                                                                                       |                                         |
| <b>endfunc</b>                                                                                                                                     |                                         |
| $V(k, l)$                                                                                                                                          | [Main algorithm]                        |

So this algorithm works. But would we want to use it? We have seen before (e.g., Algorithms HANOI and FIB-B in Section 5.8) that whenever a recursive procedure calls itself even twice within its definition, the order of the algorithm is exponential. Algorithm E.1 calls itself 3 times. Therefore it seems plausible that this recursive approach is exponential and probably worse than HANOI, which is  $\text{Ord}(2^n)$ . We can't check this belief directly by the methods of Chapter 5 because those methods assumed a single index variable in all recurrences and made essential use of this assumption. (Review the characteristic equation method in Section 5.5. How does it break down for two indices?) Still, by a simple “collapsing variables” argument one can show that this algorithm is indeed at least  $\text{Ord}(3^{\min\{k,l\}})$  [29]. With more work one can show that the algorithm runs even much more slowly than that [30].

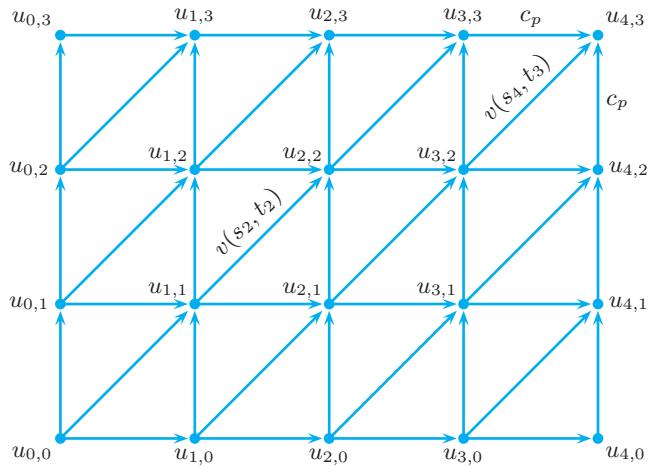
So what to do? Well, has this algorithm reminded you of anything you have seen before? Was there some other problem where we generalized from the specific evaluation we wanted to many subevaluations, each subevaluation was something like that in Eq. (9), and we were able to compute the answer efficiently by iteration instead of recursion?

Here's one more hint. Let's represent the problem of aligning  $\mathcal{S}_i$  with  $\mathcal{T}_j$  by a vertex  $u_{ij}$  in a graph. Eq. (9) says that this vertex has information fed into it from vertices  $u_{i-1,j-1}$ ,  $u_{i-1,j}$ , and  $u_{i,j-1}$ . Represent this relationship by directed edges. The additional cost of extending the best alignment from, say,  $u_{i-1,j}$  to  $u_{ij}$ , is a known term from Eq. (9) and can thus be represented as a weight on the directed edge between the vertices. For instance, with  $k = 4$  and  $l = 3$ , we get Fig. E.1. All the vertices and edges are shown, but to avoid crowding only some of the edge weights are shown. Note, however, that the weight on every horizontal or vertical edge is  $c_p$  (because these edges represent extending a partial alignment with a pass in the next column). The weight on the diagonal edge from  $u_{i-1,j-1}$  to  $u_{ij}$  will always be  $v(s_i, t_j)$  (because these edges represent expanding an alignment of  $\mathcal{S}_{i-1}$

and  $T_{j-1}$  by adding a column containing  $s_i, t_j$ ).

## FIGURE E.1

The computation graph for finding the measure of similarity between two sequences of length  $k = 4$  and  $l = 3$ . Each vertex  $u_{i,j}$  is positioned by regarding  $i$  as an  $x$ -coordinate and  $j$  as a  $y$ -coordinate. All edges have weights but only a few of the weights are shown.



Anyway, with this graph Eq. (9) has a new interpretation:  $V(i, j)$  is the length of the *longest* directed path from  $u_{0,0}$  to  $u_{ij}$ , where the length of a path is the sum of the weights on its edges. Since we want  $V(k, l)$  in particular, we are seeking the longest path from the lower left corner to the upper right. Why is this longest path interpretation correct? Because each edge of any directed path from lower left to upper right represents another column in an alignment, and the weight on the edge is the value contributed by that column to the alignment. Since both the value of the alignment and the length of the path are defined to be the sum of these weights, the value and the length are the same. We want the highest weight alignment, and therefore we want the longest path.

The graph representation is very much like the graph in the Prologue. The real-world problem there sounded quite different — find the minimum time to complete a project with many subtasks — but the mathematical model was much the same. In both cases we have a directed graph where we want to get from start to finish with the best value, and to compute the value at each vertex  $v$  we have to maximize over expressions involving the value at vertices adjacent into  $v$  and certain weights. The exact recurrence there was Eq. (1), p. 4. The situation there wasn't quite the same as here, because the weights were on the vertices, not the edges. (The weights were the durations of the tasks, and the tasks were the vertices.) However, an edge-weighted version of the minimum completion time problem was considered in [5, Prologue]. The recurrence for that version was essentially the same as Eq. (9), and the iterative algorithm for solving that version was little different from the version for the vertex-weighted problem.

The trick to solving that Prologue problem efficiently was to work up from the start. That is, once we *understood* what was going on through the Recursive Paradigm, we used the Inductive Paradigm to do the computation iteratively from the bottom up. Can we do the same here?

There the vertices came numbered so that all arrows went from lower to higher indices. Thus, because we iterated from 0 to  $n$ , when we wanted to use the recurrence to compute the  $k$ th value we had already computed all the values we needed. Here there are two indices. Can we order the vertices so that once again, whenever we want to evaluate the LHS of Eq. (9), we have already computed all the values on the RHS?

Yes we can, in several different ways. Look at Fig. E.1. Perhaps the simplest way to order them is to proceed by rows, each row left to right, starting with the bottom row and working up. Algorithm STRINGALIGN uses this order. (We could just as well proceed by columns, each column bottom to top, starting with the left-most column and working right; [3] shows yet another way. In any event, for a rectangular grid with all edges directed right or up, it's easy to eyeball orders that work. But in fact, any directed graph that has no directed cycles has such an order; see [6, Prologue] and [16, 17, Supplement, Chapter 3].)

## Algorithm E.2

### StringAlign

|                                                                                                                                                                           |                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|
| <b>Input</b> $\mathcal{S}, \mathcal{T}$                                                                                                                                   | [Strings of length $k, l$ ]             |
| $c_m, c_u, c_p$                                                                                                                                                           | [Typically $c_m > 0, c_u, c_p \leq 0$ ] |
| <b>Output</b> $V(k, l)$                                                                                                                                                   | [Measure of similarity]                 |
| <b>Algorithm</b> STRINGALIGN                                                                                                                                              |                                         |
| <b>for</b> $j = 0$ <b>to</b> $l$                                                                                                                                          |                                         |
| <b>for</b> $i = 0$ <b>to</b> $k$                                                                                                                                          |                                         |
| <b>if</b> $i = 0$ <b>then</b> $V(i, j) \leftarrow c_p j$                                                                                                                  | [Case if]                               |
| <b>if</b> $j = 0$ <b>then</b> $V(i, j) \leftarrow c_p i$                                                                                                                  |                                         |
| <b>if</b> $ij \neq 0$ <b>then</b> $V(i, j) \leftarrow \max \left\{ \begin{array}{l} v(s_i, t_j) + V(i-1, j-1) \\ c_p + V(i-1, j) \\ c_p + V(i, j-1) \end{array} \right\}$ |                                         |
| <b>endif</b>                                                                                                                                                              |                                         |
| <b>endfor</b>                                                                                                                                                             |                                         |
| <b>endfor</b>                                                                                                                                                             |                                         |

As stated, Algorithm STRINGALIGN finds the measure of similarity, that is, the  $V$ -value of a best alignment, but it does not make public what this alignment is. To retrieve the alignment, you can make the same sort of change made to retrieve the shortest path in Algorithm DIJKSTRA in Section 3.4: For each vertex  $u_{ij}$ , set up a pointer  $\text{prev}(i, j)$  that names whichever vertex was previous on the longest path to  $u_{ij}$ . Then starting at  $(k, l)$ , work backwards through the pointers. We leave the algorithm modifications to a problem [12]. Once you have this longest path, you still have to convert it into an alignment [13]. Or, figure out how to generate the alignment directly without generating the longest path as an intermediate step [14].

## EXAMPLE 3

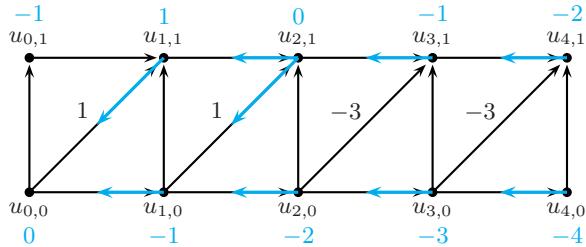
Apply Algorithm STRINGALIGN to strings  $\mathcal{S} = \text{AACC}$ ,  $\mathcal{T} = \text{ACCC}$  of Example 1 with the new weights  $c_m = 1$ ,  $c_u = -3$  and  $c_p = -1$ . (Recall that these  $c$ -values are not realistic for biology, because  $2c_p > c_u$  and thus the optimum alignment will

never use mismatches, replacing any potential mismatch by two passes. However, we use this example because it provides a simple illustration of several things that can happen while running the algorithm.)

**Solution** We work up row by row, as the algorithm says, and display the results graphically. Note that  $k = l = 4$ , so there are 5 rows and 5 columns. In the 0th row  $V(i, 0) = c_p i$  and it is set by assignment:



In the second row things get interesting; see the next drawing. To avoid clutter, we don't show the weight on any horizontal or vertical edge; it is always  $c_p = -1$ . We do show the weights on each diagonal edge. Each diagonal edge into row 1 represents matching  $t_1 = A$  in turn against each letter in  $\mathcal{S} = \text{AACC}$ , so these weights are 1, 1, -3, -3.



The colored reverse arrows are not from the algorithm itself but rather from the enhancement of it in [12] that allows the determination of the best alignment. They point to the previous vertex that provided the best value for  $V$ . Notice there is a tie for best direction back from  $u_{2,1}$ . Following a vertical edge down means pairing the last letter currently under consideration in  $\mathcal{T}$  with a blank because only the  $\mathcal{T}$ -coordinate changes. Likewise, following a horizontal edge left means pairing the last letter currently under consideration in  $\mathcal{S}$  with a blank. Thus following the colored path  $u_{2,1}, u_{1,1}, u_{0,0}$  gives on the right an (A, blank) pairing and then to the left of that an (A,A) pairing, and so corresponds to the alignment

$$\begin{array}{c} \text{A A} \\ \text{A } \end{array}$$

of  $\mathcal{S}_2$  and  $\mathcal{T}_1$ , whereas following the colored path  $u_{2,1}, u_{1,0}, u_{0,0}$  corresponds to

$$\begin{array}{c} \text{A A} \\ - \text{ A} \end{array}$$

Sure enough, both alignments have value 0 (the colored number over vertex  $u_{2,1}$ ) and are optimal for matching  $\mathcal{S}_2$  and  $\mathcal{T}_1$ . Other paths back from  $u_{2,1}$  correspond to less good alignments, for instance,  $u_{2,1}, u_{2,0}, u_{1,0}, u_{0,0}$ , which corresponds to the alignment

$$\begin{array}{c} \text{A A } - \\ - - \text{ A} \end{array}$$

which has value -3.

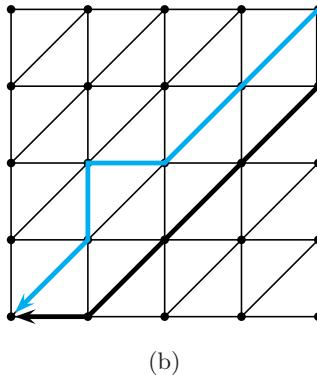
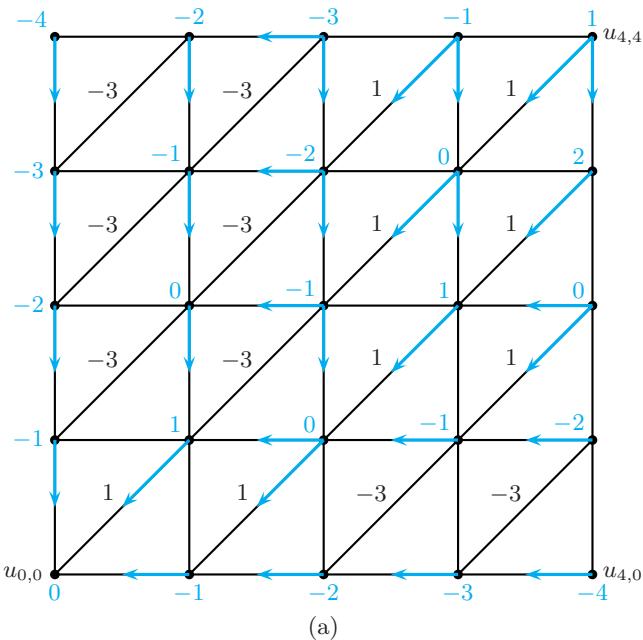
Figure E.2a shows the complete results of the algorithm. We are interested in the longest path back from  $u_{4,4}$ . There are many, because of the frequent ties for best previous vertex. The path highlighted in color in Fig. E.2b corresponds to

$$\begin{array}{cccc} A & \underline{\quad} & A & C \\ A & C & \underline{\quad} & C \\ & & & C \end{array}$$

which is the alignment we found before in display (4). The path highlighted in black in Fig. E.2b corresponds to

$$\begin{array}{ccccc} A & A & C & C & \underline{\quad} \\ & \underline{\quad} & A & C & C \\ & & C & C & \end{array}$$

one of the many other equally optimal alignments for our choice of the  $c$  values. ■



**FIGURE E.2**

Part (a) shows the complete results of running STRINGALIGN on  $\mathcal{S}, \mathcal{T}$  from Example 1. Part (b) highlights two longest paths from  $u_{4,4}$  back to  $u_{0,0}$ . Each such path corresponds to a best alignment.

*Efficiency and correctness of STRINGALIGN.* The efficiency analysis of this algorithm is very simple. Since it consists of one double loop, there are  $(k+1)(l+1)$  passes through the interior. Since the amount of work in any one pass is bounded, we conclude that STRINGALIGN is an  $\text{Ord}(kl)$  algorithm. Quite an improvement from exponential!

Why is STRINGALIGN correct? Since it is a loop algorithm, the standard method is to use a loop invariant. It's not hard to see what the invariant needs to be: Upon entrance to the inner loop, all previous assignments to  $V$  are correct. Then the next assignment of  $V$  will be correct, because it uses Eq. (9). Thus the invariant continues to be correct at the next iteration. We don't present this proof formally, because as we saw in the proof of Insertion Sort in Section 7.7, formal proofs by loop invariant for nested loops can get quite involved.

## Other Matching Problems

Algorithm STRINGALIGN is more versatile than it might seem. The option to choose the  $c$ 's arbitrarily allows it, or variants involving modest changes, to solve several problems that initially seem quite different from the alignment problem we have considered so far. We give two examples of this and leave others to the problems.

### EXAMPLE 4

Show how to use Algorithm STRINGALIGN to find the longest common subsequence of  $\mathcal{S}$  and  $\mathcal{T}$ .

**Solution** We are asked to ignore passes and mismatches rather than subtract off for them. Then the right values for the  $c$ 's are:  $c_m = 1$ ,  $c_u = c_p = 0$  (why?). The measure of similarity will be the length of a longest common subsequence. To actually find the subsequence, after you find the alignment, delete all columns where the entries aren't identical. ■

The maximization problem we have set ourselves so far — find the similarity between two whole strings — is called the **global alignment** problem. It is not always the right problem biologically. Suppose you have a DNA sequence from some unusual organism. You probably wouldn't expect it to be very close to known DNA sequences *over its entirety*. But you could expect that certain substrings would match well to substrings of known DNA, because you would expect that some functions performed by the organism (and thus coded in the DNA) would be functions in common with other organisms. Knowing which DNA substrings from this organism are familiar and which are not could thus be very valuable information. The problem of finding well matched substrings of two given strings, where neither the substrings nor even their size is specified in advance, is called the **local alignment** problem.

### EXAMPLE 5

Modify Algorithm STRINGALIGN to find the most similar substrings of  $\mathcal{S}$  and  $\mathcal{T}$ , that is, the substrings  $\mathcal{S}'$ ,  $\mathcal{T}'$  for which  $V(\mathcal{S}', \mathcal{T}')$  is highest.

**Solution** It appears at first that we have to consider every possible starting and ending point for  $\mathcal{S}'$  and  $\mathcal{T}'$ . We already considered every possible ending point by computing every  $V(i, j)$  with  $i \leq k, j \leq l$ . Recall that  $V(i, j)$  is the measure of similarity of the initial substrings  $\mathcal{S}_i$  and  $\mathcal{T}_j$ . We could, therefore, solve the problem as follows: for each  $i', j', i' = 0, 1, \dots, k, j' = 0, 1, \dots, l$ , lop off  $\mathcal{S}_{i'}$  and  $\mathcal{T}_{j'}$  (leaving final substrings) and run STRINGALIGN again; then find the max of all the  $V(i, j)$  computed in all the runs. This is a lot of work!

Fortunately, there is a better way. Define  $V^*(i, j)$  to be the best value of any alignment of *any* substring of  $\mathcal{S}$  ending at  $s_i$  against any substring of  $\mathcal{T}$  ending at  $t_j$ . It seems like a tall order to compute this, but again think recursively. As before, if the best alignment of any such substrings ends with  $s_i$  and  $t_j$  matched in the last column, then everything to the left of that should be the best alignment of any substrings that end with  $s_{i-1}$  and  $t_{j-1}$ , because then  $V^*(i, j)$  is the sum of  $v(s_i, t_j)$  with the value of whatever comes before. This is just the first case of Eq. (9) with  $V^*$  substituted for  $V$ . Similarly, the two other cases in Eq. (9) also hold with  $V^*$  instead of  $V$ . However, there is now one further case. What if all 3 values in this modified RHS of Eq. (9) are negative? Then the highest-valued alignment of substrings ending at  $s_i$  and  $t_j$  is the empty alignment (null substrings), with value 0. In other words, the correct recurrence for  $V^*$  is

$$V^*(i, j) = \max \left\{ \begin{array}{l} v(s_i, t_j) + V^*(i-1, j-1), \\ c_p + V^*(i-1, j), \\ c_p + V^*(i, j-1), \\ 0 \end{array} \right\}. \quad (11)$$

Also, the basis cases are now different than in Eq. (10). The right basis values are

$$V^*(i, 0) = V^*(0, i) = 0. \quad (12)$$

Why? Anyway, it is easy to modify STRINGALIGN to incorporate Eqs. (11–12). Then there is one more step. The maximum of  $V^*(i, j)$  must be found. In fact, this can be done simultaneously with computing the values of  $V^*$  [16]. The resulting Algorithm SUBSTRINGALIGN is, once again, an  $\text{Ord}(kl)$  algorithm. If we want to know which substrings are aligned, and how they are aligned, the same sort of changes that accomplish this for STRINGALIGN can be effected. ■

**Another Discrete Math DNA Problem.** We close this section with brief mention of a related problem. Before biologists can compare long sequences of DNA, they have to know what those long sequences are. One of the standard methods for determining sequences has been to use various chemical agents to cleave multiple copies of the same DNA into short overlapping segments, determine the sequence of bases for these small segments (for small segments there are good sequencing methods), and then somehow put all the small overlapping sequences back in order. That last step uses discrete mathematics too, in fact, Euler graph theory. See [10–11, Section 3.3] for a sketch of how this problem is solved.

# Problems: Section E.1

1. ⟨2⟩ Using  $c_m = 1$  and  $c_u = c_p = -1$ , find the values of the two alignments in displays (1) and (2). You'll find, not surprisingly, that one alignment is much worse than the other. Then find values for  $c_m, c_u, c_p$  so that what was the worse alignment is now the better one. (These  $c$  values won't be useful for biology because you'll have to choose  $c_u > c_m$ ; why?)
2. ⟨2⟩ Finish Example 2 by verifying the claims in the final paragraph of its solution.
3. ⟨2⟩ Algorithm STRINGALIGN2 below processes the vertices in a different order than STRINGALIGN in the text. What order? (Illustrate with a figure.) Is this a legitimate order — will values of  $V$  be known when they are needed? What advantages (perhaps only psychological) does STRINGALIGN2 have over STRINGALIGN?

## Algorithm E.3 STRINGALIGN2

|                                                                                                                                         |                              |
|-----------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| <b>Input</b> $\mathcal{S}, \mathcal{T}$                                                                                                 | [Strings of length $k, l$ ]  |
| $c_m, c_u, c_p$                                                                                                                         | $[c_m > 0, c_u, c_p \leq 0]$ |
| <b>Output</b> $V(k, l)$                                                                                                                 | [Measure of similarity]      |
| <b>Algorithm</b> STRINGALIGN2                                                                                                           |                              |
| <b>for</b> $i = 0$ <b>to</b> $k$                                                                                                        | [Initialize row 0]           |
| $V(i, 0) \leftarrow c_p$                                                                                                                |                              |
| <b>endfor</b>                                                                                                                           |                              |
| <b>for</b> $j = 1$ <b>to</b> $l$                                                                                                        | [Initialize column 0]        |
| $V(0, j) \leftarrow c_p$                                                                                                                |                              |
| <b>endfor</b>                                                                                                                           |                              |
| <b>for</b> $j = 1$ <b>to</b> $l$                                                                                                        |                              |
| <b>for</b> $i = 1$ <b>to</b> $k$                                                                                                        |                              |
| $V(i, j) \leftarrow \max \left\{ \begin{array}{l} v(s_i, t_j) + V(i-1, j-1) \\ c_p + V(i-1, j) \\ c_p + V(i, j-1) \end{array} \right\}$ |                              |
| <b>endfor</b>                                                                                                                           |                              |
| <b>endfor</b>                                                                                                                           |                              |

4. ⟨1⟩ Suppose we believe that strings  $\mathcal{S}, \mathcal{T}$  are descendants of a common ancestor, and suppose we are right that the best alignment according to our weighting corresponds to the actual mutations that took place. For instance, suppose alignment (2) is correct in suggesting that the  $\mathcal{S}$  in the first row there and the  $\mathcal{T}$  on the second evolved from some common ancestor by two mutation events. Is

this enough to determine the common ancestor? Explain.

5. ⟨2⟩ Do Example 3 again, but this time let  $c_u = -1$  ( $c_m, c_p$  unchanged).
6. ⟨2⟩ Find the best alignment of  $\mathcal{S} = \text{AGTG}$  and  $\mathcal{T} = \text{ACGAG}$  using  $c_m = 1, c_u = -1$  and  $c_p = -2$ . Is the best alignment unique? Is it the alignment in display (2)?
7. ⟨2⟩ Show that multiplying all of  $c_m, c_u, c_p$  by the same constant  $d$  merely multiplies  $V(\mathcal{S}, \mathcal{T}, \alpha)$  by  $d$ , for every  $\mathcal{S}, \mathcal{T}$ , and  $\alpha$ . Explain why, therefore, if  $d > 0$ , then the best alignment for each pair  $\mathcal{S}, \mathcal{T}$  is unchanged. We say therefore that the global alignment problem is **invariant** under positive scaling of the weights.
8. ⟨2⟩ Let  $\mathcal{S}$  have  $k$  letters, and  $\mathcal{T}$  have  $l$  letters. As in Definition 1, let  $N_m, N_u, N_p$  count the number of matches, mismatches and passes in an alignment. Prove:

$$2(N_m + N_u) + N_p = k + l.$$

This means: Although the numbers  $N_m, N_u, N_p$  vary as you use different alignments, yet so long as you use the same  $\mathcal{S}$  and  $\mathcal{T}$ , the quantity  $2(N_m + N_u) + N_p$  does *not* vary. It is an invariant over all alignments of these two strings (indeed, over all strings of length  $k$  and  $l$ .) Hint: Count all the letters in  $\mathcal{S} \cup \mathcal{T}$  by how they are matched.

9. ⟨3⟩ Use [8] to prove: If the weights  $c_m, c_u, c_p$  are changed to  $c_m+b, c_u+b$  and  $c_p+\frac{1}{2}b$ , then for each pair of strings  $\mathcal{S}, \mathcal{T}$  the optimal alignment of them does not change. We say that the global alignment problem is invariant under modified translation of the weights. (Translation means adding a fixed amount; we say "modified" because one weight,  $c_p$ , has a different amount added.) Hint: Show that  $V(\mathcal{S}, \mathcal{T}, \alpha)$  is increased by  $(k+l)b/2$ , independently of  $\alpha$ .
10. ⟨3⟩ Use [7, 9] to show that, whatever weights  $c_m, c_u, c_p$  you had in mind to use (so long as  $c_m > c_u$ ), there is a value  $c'_p$  so that instead you can use  $c'_m = 1, c'_u = 0$  and  $c'_p$  and you will get the same optimal alignments. Therefore, the global alignment problem really has only one free parameter, the weight of a pass.

11. ⟨3⟩ Now assume only that  $c_m > 2c_p$  (very reasonable biologically — why?). Similarly to [10], show that you can set  $c'_m = 1$ ,  $c'_p = 0$ , letting the mismatch parameter be the free one, and still get the same optimal alignments. The advantage of this is that Algorithm STRINGALIGN becomes more efficient (but not by a change in order) because the two latter plus-signs in the max line can be eliminated.
12. ⟨3⟩ Modify Algorithm STRINGALIGN so that it either outputs a longest directed path from  $u_{0,0}$  to  $u_{k,l}$  or outputs information from which it is easy to determine this longest path.
13. ⟨3⟩ Show how to convert the longest path output by STRINGALIGN into a specific alignment of  $\mathcal{S}$  and  $\mathcal{T}$ . Note: An alignment is completely specified by listing where blanks are inserted. For instance, for AAC and AGCT, the alignment
- $$\begin{array}{ccccc} \text{A} & \text{A} & \text{C} & \text{—} & \\ & \text{—} & \text{A} & \text{G} & \text{C} \end{array} \quad \begin{array}{c} \text{T} \\ \text{—} \end{array}$$
- is completely described by saying  $\mathcal{S}: 3, 5; \mathcal{T}: 1$ . Of course, direct descriptions are possible: just list  $\hat{s}_1, \hat{s}_2, \dots, \hat{s}_\ell$  and  $\hat{t}_1, \hat{t}_2, \dots, \hat{t}_\ell$ .
14. ⟨3⟩ Show how to modify STRINGALIGN so that it outputs the best alignment directly, without going through the intermediate step of listing the longest path, as in [12–13].
15. ⟨2⟩ For  $\mathcal{S}, \mathcal{T}$  in Example 1 find the most similar substrings, using  $c_m = 1$ ,  $c_u = -2$  and  $c_p = -1$ . The algorithm to do this, SUBSTRINGALIGN, is described (without algorithmic language) in Example 5. Write up your computations in the format of Example 3 or get a machine to do it.
16. ⟨2⟩ Write Algorithm SUBSTRINGALIGN of Example 5 in algorithmic language. Write it so that it computes the maximum of all the  $V^*(i, j)$  simultaneously with computing the  $V^*(i, j)$ .
17. ⟨2⟩ Algorithm STRINGALIGN can be interpreted as a path optimization algorithm. Algorithm SUBSTRINGALIGN from [16] also has a path interpretation. What is it? Explain.
18. ⟨3⟩ Show that the local alignment problem solved by SUBSTRINGALIGN is invariant under positive scaling of the weights, just as the global alignment problem is. However, also show that the local alignment problem is *not* invariant under modified translation of the weights. Therefore, the local alignment problem has two free parameters, not one. Compare with [7–10].
19. ⟨4⟩ Modify SUBSTRINGALIGN from [16] to output not only the value of the best substring alignment, but also which substrings are aligned, and how they are aligned. That is, give the enlargements, including blanks, that are lined up column by column.
20. ⟨3⟩ The methods in [13–14] result in finding the optimal alignment backwards. That is, you find the last column first, then the next to last, and so forth. This is a little disconcerting to humans. Devise an alternative version of STRINGALIGN which results in the alignment being determined from left to right instead of right to left.
21. ⟨2⟩ Use STRINGALIGN to find the longest common subsequences of ACGATC and CAGT. This example is easier to do by sight, but practice choosing the weights and running the algorithm with these strings. You may wish to program a machine to do it.
22. ⟨3⟩ Show how to use STRINGALIGN to determine, for any two sequences  $\mathcal{S}, \mathcal{T}$ , if  $\mathcal{S}$  is a subsequence of  $\mathcal{T}$ . Note: When a problem is solved by a special case of a general algorithm, it is often the case that there is also a much simpler algorithm. That's true here. However, it suffices here to show why this problem can be solved by STRINGALIGN.
23. ⟨3⟩ Use one of the algorithms of this section, or a variant, to show how to find the longest common substring of two sequences.
24. ⟨3⟩ A specialization of the question in [23] is to determine if  $\mathcal{S}$  is a substring of string  $\mathcal{T}$ .
- a) This is a problem someone using a computer confronts all the time — forget about DNA. Explain.
  - b) Specialize your algorithm in [23] to solve this problem.
  - c) As is often true of special cases, there is also a much simpler algorithm. Describe it.
25. ⟨3⟩ Given string  $\mathcal{S}, \mathcal{T}$ , the **semiglobal alignment** problem is to find the highest valued alignment of *any* substring of  $\mathcal{S}$  with all of  $\mathcal{T}$ . Thus the semiglobal problem is halfway between the global alignment problem solved by STRINGALIGN (find the best alignment of all of  $\mathcal{S}$  with all of  $\mathcal{T}$ ) and the local alignment problem of Example 5 (find

the best alignment of any substring of  $\mathcal{S}$  with any substring of  $\mathcal{T}$ ).

- a) Solve the semiglobal alignment problem where  $\mathcal{S} = \text{ACGAG}$   $\mathcal{T} = \text{AGTG}$ ,  $c_m = 1$ ,  $c_u = 0$ , and  $c_p = -2$ . Since we have not asked you to develop a algorithm for this problem yet, the intent here is for you to give a special argument that works for these two words.

- b) Modify STRINGALIGN to solve the semiglobal alignment problem. *Hint:* It suffices to modify the weights (or border rows) of STRINGALIGN to solve it. Explain.

26. (4)  $\mathcal{T}$  is a **supersequence** of  $\mathcal{S}$  if  $\mathcal{S}$  is a subsequence of  $\mathcal{T}$ . Devise an algorithm to find the shortest common supersequence of two sequences.

27. (3) The **edit distance** of  $\mathcal{S}, \mathcal{T}$  is the minimum number of blanks or mismatches in any alignment of the two strings.

- a) Find the edit distance for  $\mathcal{S} = \text{ACGAG}$  and  $\mathcal{T} = \text{AGTG}$ . It is easy to do this by a special argument; you don't need a general method yet.

- b) Explain how to use or modify STRINGALIGN to find the edit distance between any two sequences.

28. (3) Consider again Algorithm STRINGALIGN-REC, the recursive global alignment algorithm. Modify the algorithm to output the best alignment as well. You will need a function  $\text{prev}(j)$  which finds and stores the best previous vertex to each vertex  $j$ .

29. (4) Let's bound the run time of the recursive algorithm STRINGALIGN-REC. A good way to do this is to bound the number of *calls* to the recursive function in STRINGALIGN-REC. This is good because it simplifies the computation without changing the order of the result. For instance, calls of basis cases have just one assignment. Calls that invoke Eq. (9) have lots of operations, but never more than some bound.

- a) Why do the last two sentences imply that, when STRINGALIGN-REC is invoked, the total number of operations and the total number of calls have the same order?

- b) So let  $C(k, l)$  be the number of calls, including the main call, when your procedure or function is invoked for  $k, l$ . Explain why the following recurrence is correct:

$$C(k, 0) = C(0, l) = 1,$$

$$C(k, l) = 1 + C(k, l-1) +$$

$$C(k-1, l) + C(k-1, l-1).$$

- c) Now let  $D_l = C(l, l)$ . Explain why  $D_l \geq 3D_{l-1} + 1$ .

- d) Prove that  $D_l$  is at least  $\text{Ord}(3^l)$ .

- e) Explain therefore why  $C(k, l)$  is at least  $\text{Ord}(3^{\min\{k, l\}})$ .

30. (5) Let's try to get more exact information about  $C(k, l)$  as defined in [29]. Use the inductive paradigm: compute some terms.

- a) Find and prove a formula for  $C(1, l)$ . Note that by symmetry you also have a formula for  $C(k, 1)$ .

- b) Find and prove a formula for  $C(2, l)$ .

- c) Keep going as far as you can. You can probably go a lot further if you just ask for order, not an exact formula.

31. (3) **Multiple (simultaneous) alignment** takes 3 or more sequences and looks for similarities, as before, by finding the best way to enlarge them to equal length with blanks to get a high scoring match. But now there are many more choices than match, mismatch and pass. For instance, in the first column two sequences might match and the third might be blank; or two might mismatch and the third be a blank; or all three might mismatch; or the first enlarged sequence might have a letter and the latter two have blanks, causing two of the enlarged sequences to have blanks in the same column, something not allowed when only two sequences are aligned.

- a) How many distinct cases are there for each column, where the actual letters don't matter but just how many and whether they match?

- b) The usual measure is this. Decide how you want to weight matches, mismatches and passes in the 2-sequence case, and compare each pair of sequences by this weighting. Then add the results for the three pairs of 2 sequences. *For the rest of this problem* make things even simpler by weighting each match as 1 and each pass or mismatch as 0 (so that in the 2-sequence case STRINGALIGN finds the longest common subsequence and so that when two blanks line up it doesn't matter because it doesn't change the score).

- c) What is the value of the alignment below?

|   |   |   |
|---|---|---|
| A | — | C |
| A | C | G |
| — | C | T |

- d) Explain why, for any triple alignment  $\alpha$

$$V(\mathcal{S}, \mathcal{T}, \mathcal{U}, \alpha) = V(\mathcal{S}, \mathcal{T}, \alpha) + V(\mathcal{S}, \mathcal{U}, \alpha) + \\ V(\mathcal{T}, \mathcal{U}, \alpha),$$

where  $V(\mathcal{S}, \mathcal{T}, \mathcal{U}, \alpha)$  means the value as defined in part b).

- e) Nonetheless, the following can be *false*:

$$V(\mathcal{S}, \mathcal{T}, \mathcal{U}) = V(\mathcal{S}, \mathcal{T}) + V(\mathcal{S}, \mathcal{U}) + \\ V(\mathcal{T}, \mathcal{U}).$$

Show it is false for the three words

ABC, ACD, ADB.

- f) Find an even smaller example (shorter sequences) to show that the equality in part e) can be false.

## E.2 Algorithmic Mathematics and Minimax Theorems

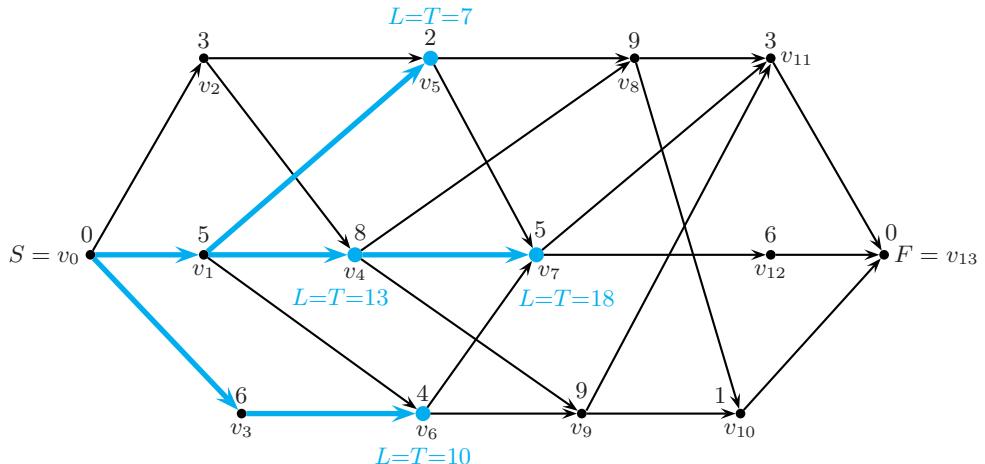
---

In this final section, we dwell on the middle word in our book's title, *algorithmic*. In the previous section, biology connected us back to the Prologue. Looking at that connection more closely will lead us to the middle word. It will also lead us to a brief discussion of a wonderful topic for your next discrete math course, *minimax theorems*.

The connection between the previous section and the Prologue was this: In each of them the problem studied was modeled with a directed graph where we found a value for the final vertex by iterating on nearly identical recurrences. But the DNA problem was to find a *maximum* weight alignment, while the Prologue problem was to find a *minimum* completion time. Could one applied max problem be essentially the same as another applied min problem?

The answer is Yes, but we'll state this as a theorem about something intrinsic in the underlying graph rather than as a result about two applications. The DNA problem reduced to the intrinsic problem of finding longest paths. So our theorem will relate shortest schedules to longest paths.

To state the theorem, we need to review and extend notation from the Prologue. Let  $D$  be a **vertex activity graph**: The vertices represent the *tasks* that form the pieces of a *project*; the vertices are numbered  $v_0, v_1, \dots, v_n$ , with  $v_0$  the start and  $v_n$  the finish; edges represent precedence and are directed from lower to higher numbered vertices; and each vertex has a weight  $d(j)$  representing the time (duration) that task  $j$  takes by itself, with  $d(0) = d(n) = 0$ . Let  $T(j)$  be the shortest time possible to complete task  $j$  from the start of the whole project. Let  $L(j)$  be the length of the longest directed path from  $v_0$  to  $v_j$ , where length is the sum of the weights of the vertices on the path (not on the edges, as in Section E.1). It is assumed that there are enough resources to carry on any number of the tasks simultaneously. These ideas are illustrated in Fig. E.3, which repeats Fig. P.2 from the Prologue but with some values of  $T(j)$  and  $L(j)$  added.



**Theorem 1.** For a project represented by a vertex activity graph, the minimum completion time equals the maximum path length from  $v_0$  to  $v_n$ . More specifically,

- 1) The minimum completion time is accomplished by scheduling each task  $j$  to begin at time  $L(j) - d(j)$ , which is the earliest time task  $j$  could start under any schedule.
- 2) For any task  $j$  as well as for the project as a whole,  $T(j) = L(j)$ ; thus, minimum completion time equals longest path length from  $v_0$ .

This is an example of a **minimax theorem**: the minimum of one thing equals the maximum of something else. It is a type of structure theorem: It says that some type of problem (in this case, scheduling projects with precedences) always has a certain sort of structure to it (here, that in the associated vertex activity graph the longest paths answer the scheduling questions). The theorem says that the structure exists, but it doesn't hint at how to actually compute what you want to know (here, to find the optimal schedule, or equivalently, the longest paths).

Contrast Theorem 1 with the following theorem, which was implicit in our solution to this same problem in the Prologue:

**Theorem 2.** The minimum completion time for a project represented by a vertex activity graph is correctly computed by Algorithm E.4, PROJECTSCHED. The minimum completion time is the output  $T(n)$ , and a schedule that obtains this minimum time is obtained by finishing each task  $j$  at the time  $T(j)$  computed by the algorithm.

## Algorithm E.4

### ProjectSched

|                                                          |                           |
|----------------------------------------------------------|---------------------------|
| <b>Input</b> $D(V, E)$ ,                                 | [Vertex activity digraph] |
| $d(0)=0, d(1), \dots, d(n-1), d(n)=0$                    | [Vertex weights]          |
| <b>Output</b> $T(n)$                                     | [Minimum completion time] |
| <b>Algorithm</b> PROJECTSCHED                            |                           |
| $T(0) \leftarrow d(0)$                                   |                           |
| <b>for</b> $j = 1$ <b>to</b> $n$                         |                           |
| $T(j) \leftarrow d(j) + \max_{i \rightarrow j} \{T(i)\}$ | [Eq. (1), Prologue]       |
| <b>endfor</b>                                            |                           |

This theorem statement clearly announces a method to compute the desired quantities, but it gives no hint about the nifty minimax structure. In this way, Theorem 2 is an example of *algorithmic* mathematics, whereas Theorem 1 is an example of *existential* mathematics, also called *traditional* mathematics or *classical* mathematics, because it was the predominant style from the late 1800's to the late 1900's. Now both styles are flourishing.

But perhaps a traditional proof of Theorem 1 might provide the constructive approach that the statement of Theorem 1 lacks. See for yourself:

**TRADITIONAL PROOF OF THEOREM 1** With  $L(j)$  and  $T(j)$  as defined before the theorem statement, we will show two things for  $j = 0, \dots, n$ :  $T(j) \geq L(j)$  and  $T(j) \leq L(j)$ . That means we'll be done.

The first inequality is easy. The tasks along *any* directed path from  $v_0$  to  $v_j$  must be done one after the other, because edges represent precedence. At the very best, each task on the path might begin immediately after the previous one ends. Given how the path length is defined, it follows that it takes time at least equal to the length of this path to complete the project through task  $j$ . Thus  $T(j) \geq$  the length of any path to  $v_j$ . In particular,  $T(j) \geq$  the length of the longest such path, whose length is  $L(j)$  by definition.

To show  $T(j) \leq L(j)$ , we construct a specific schedule for the tasks in which, for each  $j$ , task  $j$  finishes exactly  $L(j)$  units after the start. Since we do not claim from the construction itself that this is the optimum schedule, the optimum,  $T(j)$ , might be less. In other words,  $T(j) \leq L(j)$ . (Of course, once we demonstrate such a schedule, from our previous knowledge of the other inequality we will know that this schedule *is* optimum, for every task  $j$ .)

The schedule is the one in part 1) of the Theorem statement: start each task  $j$  at time  $L(j) - d(j)$ . Since task  $j$  takes  $d(j)$  units, this schedule certainly ensures that task  $j$  finishes  $L(j)$  units after the start. The only hitch is: is it a *feasible* schedule? That is, at the time we start task  $j$ , have all precedent tasks been completed? (Since we have enough resources to do any number of tasks simultaneously, precedence is the only constraint on starting a task.) In other words, if task  $i$  immediately precedes task  $j$ , is  $L(i) \leq L(j) - d(j)$ ? This is equivalent to asking

Is  $L(i) + d(j) \leq L(j)$  whenever  $v_i$  immediately precedes  $v_j$ ?

(For a specific example, consider the graph in Fig. E.3 and let  $i = 4, 5, 6$ , and  $j = 7$ .) Well,  $L(i)$  is the length of the longest path from  $v_0$  to  $v_i$  (as shown in thick lines for  $i = 4, 5, 6$  in the figure). And  $d(j)$  is the additional length if we extend that path along one more edge to  $v_j$ . This gives us *some* path from  $v_0$  to  $v_j$ . Thus it is at most as long as the longest path from  $v_0$  to  $v_j$ . (In the figure, this extension is not the longest path to  $v_7$  when  $i = 5, 6$ , but it is longest when  $i = 4$ , and this longest path to  $v_7$  is also shown in thick lines.) In other words,  $L(i) + d(j) \leq L(j)$ , and we are done. (You can visually check that scheduling task 7 to finish at time  $T = L = 18$  is feasible if scheduling the predecessor tasks 4, 5 and 6 to finish at times 13, 7, and 10, respectively, is feasible.) ■

So, does this proof construct the schedule? Well, sort of. It talks about the schedule, and even deduces a formula for when each task starts, but the formula is in terms of longest paths and it *still* gives you no idea of how these path lengths are determined.

So traditional mathematics can really be quite different from algorithmic mathematics. As we saw earlier, the two types of mathematics can differ in what is considered important enough about a solution method to state as a theorem. As we just saw, they can also differ in the type of proof they favor.

Please understand: It is not our intent to bash traditional mathematics. We find Theorem 1 very pleasing and interesting. Rather our point is that both styles have strengths and there is no need to take sides so long as both styles are appreciated. Indeed, look for ways to combine both ways of thinking.

As a case in point, we now give a second proof for Theorem 1.

**ALGORITHMIC PROOF OF THEOREM 1** Jump into the middle! Suppose we knew  $L(i)$  for all vertices  $i < j$ . Is there a recurrence for  $L(j)$ ? Sure. The longest path from  $v_0$  to  $v_j$  must pass through one of the vertices that immediately precedes  $v_j$ . Furthermore, the subpath up through that preceding vertex must be a longest path to that vertex (why?). Consequently,

$$L(j) = d(j) + \max_{i \rightarrow j} \{L(i)\}. \quad (1)$$

Note two things. First, Eq. (1) is the same as (1) in the Prologue, except that now we have  $L(j)$  instead of  $T(j)$ . Furthermore, the  $T$  and  $L$  sequences have the same initial value:  $T(0) = L(0) = d(0) = 0$ . Thus, since the two sequences start the same and satisfy the same recurrence, by induction they must be the same. Furthermore, Eq. (1) allows us to compute  $L$  just as Eq. (1) in the Prologue allowed us to compute  $T$ , by working up. ■

In short, the recursive method of the second proof shows us how to compute the answer at the same time it gives us the structural result. We kill two birds with one stone!

## Minimax Theorems

Although Theorem 1 is the only minimax theorem you have seen in this book, you'll see quite a few of them, mostly deeper and more surprising, if you take a second course in discrete mathematics, say a combinatorics course or an operations research course.

In addition to their structural elegance, minimax theorems have a nice algorithmic consequence. They allow for a quick check that a claimed optimal solution is indeed optimal. We develop this idea through an example.

### EXAMPLE 1

One of the early big uses of the scheduling method developed in the Prologue was to put a man on the moon. (The US was in a hurry, lest the Soviet Union beat it.) Suppose you were the person called upon to minimize the project time, given all the task information. You show your boss the optimal schedule, based on the algorithm we've just discussed. Your boss is grateful but he is under a lot of pressure to be fast and be right. He doesn't have time for you to give him a lecture on the theory, which he might not understand anyway. What check can you provide to convince him you're right?

**Solution** You do two things. First, you convince him that your schedule is feasible. For this you give him a listing showing that each task  $j$  is scheduled to finish at least  $d(j)$  time units after each preceding task ends. That is, you give him the data to verify that Eq. (1) in the Prologue is satisfied for each  $j$ .

Second, you list for him just one path of tasks from start to finish, along with the sum of the times on that path. (He can verify that this *is* a path by verifying that each task  $i$  is listed among the immediate precedents for the task after it on your proposed path. He can also verify your sum by adding all the times  $d(i)$  from the initial data.) The path you show him happens to be a longest path, so that its length equals  $T(n)$ , the completion time for the whole project, but you don't yet point out this fact or try to explain how you picked your path.

Now you show your boss that your check is done. How? Well, you convinced him that the schedule you proposed is feasible, so the only issue left is: is it optimal? You also showed him a path of tasks that would have to be done sequentially in any schedule, so any schedule will take at least as long as the sum on that path. But your schedule takes exactly as long as that sum. Therefore, none could be better. ■

Do you see how we have used the minimax theorem? Since any path length has to be  $\leq$  any schedule length, and we have found a path and a schedule where the values are equal, then we must have found a minimum time schedule (and a maximum length path, which itself turns out to be useful [2]).

So minimax results allow for a useful check. Is this check also quick to perform, at least relative to the time it takes to find the optimal solution in the first place? For this problem, yes and no. The run time of the checking procedure is quite efficient; it is  $\text{Ord}(|E|)$ , where  $|E|$  is the number of edges in the graph. And this is also the order of the Algorithm PROJECTSCHED that found the solution.

So in this case, yes, the check is quick, but no, it is not quicker than finding the solution. But in many other cases, the check *is* much faster than the original computation. For example, suppose you are asked to find a minimum proper coloring of a large graph  $G$ , and you color it with 5 colors (i.e., the chromatic number  $\chi(G)$  is  $\leq 5$ ) and you also find a complete subgraph of 5 vertices. Thus by the first inequality in Theorem 3, Section 3.6, your coloring is optimal (i.e.,  $\chi(G) = 5$ ). Thus to convince your boss that you have an optimal coloring, all you have to do is have him verify that your coloring is proper, and then verify that what you claim to be a clique really is. All this takes  $\text{Ord}(|E|)$  time. However, to find an optimal coloring in a general graph no known algorithm runs even as fast as some high power of  $|E|$ .

Because this minimax method of convincing your boss never requires that your boss understand how you found the solution, and often takes much less time than finding the solution, it is sometimes called the **lazy supervisor method**.

*Note:* Theorem 3 in Section 3.6 is not a minimax theorem; the clique number is often much smaller than the chromatic number. But if you are lucky — in your graph they are the same and you discover it — then the lazy supervisor method is available to you. In other words, this method is sometimes available even if there is only a minimax *inequality*.

However, when you have a minimax theorem, that is, when a minimax equality is guaranteed to exist, then a lazy supervisor check is always available in principle. We say in principle because you still have to find both things in order to use the method. (In the scheduling problem, you have to find not just the schedule to minimize but also the path whose length is maximized.) Therefore, the best proofs of minimax theorems, to our minds, are proofs by algorithm that construct both objects. For many minimax theorems, such proofs are known.

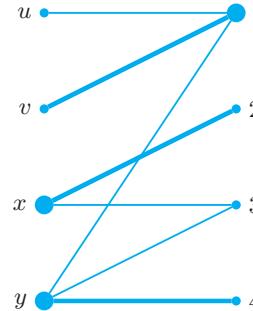
We conclude by stating one more minimax theorem, one for which such a proof is known (though we won't give it).

In the **assignment problem**,  $m$  people are available to fill  $n$  jobs. Each person is qualified for some subset of the jobs. The problem is to fill as many jobs as possible with qualified people, with the constraint that no person gets more than one job and no job is big enough for two people. This problem is a special case of the weighted assignment problem discussed in [18, Section 3.7]. The weighted assignment problem also has a minimax theorem, but it's a bit more complicated to state so we won't.

The assignment problem is represented by a bipartite graph: On one side the vertices are people, on the other jobs, and there is an edge  $\{p, j\}$  iff person  $p$  is qualified to do job  $j$ . Fig. E.4 is an illustration, where, for instance, person  $u$  is qualified to do job 1 only, and person  $y$  can do any of jobs 1, 3 and 4. An assignment corresponds to a **matching**, a set of edges with no vertices in common. (This matching concept makes sense for arbitrary graphs, but we will be interested in it only for bipartite graphs, where matchings correspond to assignments.) In the figure, one matching is shown in heavier lines: People  $v, x, y$  are assigned to jobs 1, 2, 4, respectively.

## FIGURE E.4

The bipartite graph of an assignment problem. The thick edges are a maximum assignment and the large vertices are a minimum cover.



In fact, this matching is optimal: It is not possible to assign more than 3 people to jobs, even though there are  $m = 4$  people and  $n = 4$  jobs. How do we know? Look at the vertices drawn large in the figure. They form a **cover**: every edge is incident to at least one of them. We claim that, for any bipartite graph, the number of edges in any matching must be  $\leq$  the number of vertices in any cover. Why? Because every edge in a matching, like any edge at all, must touch at least one cover vertex, but in a matching no two edges can touch the same cover vertex (or else it isn't a matching — either the same person is used twice or the same job is). So we have a minimax inequality, and since in the figure both the matching and the cover have size 3, the matching is as large as possible (and the cover is as small as possible, though this is not of direct interest for the real-world application).

Unlike in the coloring example, it was not just luck that we found an equality. The famous minimax theorem is

---

**Theorem 3.** In any bipartite graph, the maximum size of a matching equals the minimum size of a cover.

---

There are many proofs of this theorem, some of them not very long, but we leave them all for another course. Our favorite proof is by algorithm, and produces a minimum cover simultaneously with a maximum matching. These are the two things to show your lazy supervisor.

## Concluding Note

This completes our Epilogue. We hope you agree that considerable portions of the mathematics introduced in this book, as well as the methodology we have emphasized, have reappeared. Discrete mathematics — especially algorithms, mathematical induction, and the recursive and inductive paradigms — are generally a powerful combination with which to attack problems in modern applied mathematics.

## Problems: Section E.2

1. **(1)** Name two other places in DAM where we have discussed both algorithmic and existential approaches to the same problem.
2. **(3)** In the DNA alignment problem, it was clearly useful to find the longest path as well as the optimal values at the nodes, because it is the path that actually corresponds to the alignment. However, in the minimum scheduling problem, the only use shown so far for producing a longest path is to convince a lazy supervisor that the schedule is optimal.

Actually, knowing the longest path is much more valuable than that. Why? Because the task duration  $d(j)$  is rarely fixed, contrary to what we have so far assumed. Almost any task can be sped up by throwing more money at it! But for many tasks, even if you do speed them up, that won't allow the whole project to finish sooner.

- a) Why not? Explain with a small graph example.

A task  $j$  in a project is called **critical** if reducing  $d(j)$  will reduce  $T(n)$ , the minimum time for the whole project.

- b) Explain why a necessary condition for task  $j$  to be critical is that it be on a longest path.
- c) Explain why, if the longest path is unique, then every task on it is critical.
- d) If the longest path is not unique, how do you identify which tasks are critical?

Because knowing longest paths are actually quite important for project management, the scheduling method we have introduced is often called **CPM** (the critical path method). Note that we already identified in Section E.1 how to augment our algorithms to find the longest path, for instance, in [12].

3. **(2)** Your boss at Genomics Inc asks you to align two long DNA sequences. You do so with the standard algorithms and show her the result. She knows you have used proven algorithms, but a very important genetic engineering project rests on the closeness of this match, so she wants a check that you really have the best alignment. Provide a check. Explain to your boss why it *is* a check.

4. **(3)** The iterative method we have used to solve the best alignment and best schedule problems of this Epilogue are examples of what is often called **dynamic programming**, especially in operations research circles. This method does not have a precise definition, but essentially it amounts to solving a recurrence relation in a single pass. A single pass solution is possible whenever a recursive problem can be represented by a directed acyclic graph, for then the vertices can be numbered 0 to  $n$  so that all edges go from lower to higher numbers. You then solve the recurrence iteratively from 0 to  $n$ . Because the availability of dynamic programming on directed acyclic graphs gives those graphs prominence, they have come to have a short name: **DAGs**.

- a) Use dynamic programming to devise a method for counting the number of directed paths from  $v_0$  to  $v_n$  in a DAG.
- b) Apply this method to Fig. P.2 in the Prologue.
- c) Apply this method to the directed grid graph  $G(m, n)$ , where Fig. E.1 shows  $G(4, 3)$ . Obviously, you are not actually going to run this algorithm on infinitely many graphs. Can you find and prove a general solution?
5. **(2)** Use dynamic programming (see [4]) to devise a method to find the *shortest* path from start to finish in a DAG, where (just to be specific) path length is the sum of weights on the edges. Show that this algorithm is simpler than DIJKSTRA of Section 3.4 and works even with negative edge weights, which DIJKSTRA does not. So why didn't we introduce this method back then instead of DIJKSTRA?
6. **(2)** Show that the assignment problem of this section is indeed a special case of the weighted assignment problem of [18, Section 3.7]. In that problem, every person was adjacent to every job in the graph, but here that need not be. *Hint:* Use binary weights.
7. **(2)** In Fig. E.4 it is easy to see that the assignment shown cannot be extended to more people, because the only person left out is  $u$ , the only job left out is 3, and  $u$  can't do job 3. And in general, for any assignment it is easy to check whether it

can be extended. It is plausible to think that if an assignment cannot be extended, that proves it is a best assignment. But that's not true. Find an assignment of just two people in Fig. E.4 that cannot be extended. Why can't it be extended?

8. ⟨2⟩ In the Prologue we weren't yet ready to state a theorem about the scheduling method or prove it. In this section we finally stated it: Theorem 2. Now prove it correct. But then see [11].
9. ⟨3⟩ Redo the analysis of the project scheduling problem for the edge-weighted version. That is
  - a) State a minimax theorem for edge activity graphs.
  - b) Give an existential proof.
  - c) State an algorithmic mathematics theorem for the same problem.
  - d) Give an appropriate proof for this second theorem.
  - e) Give an algorithmic mathematics proof for the first theorem.
10. ⟨4⟩ The algorithmic proof of Theorem 1 (p. 710) basically just said: The  $L$  and  $T$  sequences have the same initial value and recurrence so they are equal. But we already proved a general sequence uniqueness theorem: Theorem 1, Section 5.4. So why didn't we just invoke that theorem?
 

Answer: Because despite our efforts there to be general, that theorem as stated is not general enough to apply here.

  - a) Why not?
  - b) Therefore, prove by induction that  $L(j) = T(j)$  for all  $j$ , as the proof on p. 710 claims.
  - c) But better would be to come up with a really general uniqueness theorem for difference equations. Do your best.
11. ⟨3⟩ To derive Eq. (1) in the Prologue we reasoned as follows. Let  $i^*$  be the task immediately preceding task  $j$  for which  $T(i)$  is greatest. Then task  $j$  can be started at time  $T(i^*)$ , and thus finish at time  $d(j) + T(i^*)$ , because task  $i^*$  can be finished at  $T(i^*)$  and all other tasks preceding  $j$  can be finished then or earlier.

Yes, all the others can be finished then or earlier, but can they all be finished then or earlier *using a single schedule*? To be concrete, suppose task 7 is immediately preceded by tasks 5 and 6. Suppose also that  $T(5) = 10$  and  $T(6) = 9$ . That means there is *some* feasible schedule of tasks 0 through 5 which finishes task 5 after 10 units, and some *possibly different* feasible schedule of tasks 0 through 6 which finishes task 6 at time 9. Maybe the first schedule requires task 2 to start at time 3, but maybe the second schedule requires task 2 to start at time 1. Thus, if we insisted that task 5 finish at time  $T(5)$ , because we think this will allow task 7 to be completed as soon as possible, we would discover that task 6 might not finish up at time  $9 = T(6)$  after all because intermediate task 2 is delayed by 2 times units from the optimal task 6 schedule. Thus our derivation of Eq. (1) in the Prologue is flawed.

The traditional Theorem 1 does not have this flaw, because each proof actually exhibits a single schedule. By definition,  $L(i)$  is uniquely defined, and the proof shows that scheduling each task  $i$  to finish at time  $L(i)$  is feasible.

Fortunately, our original argument using Eq. (1) in the Prologue can be fixed up. One should expect this, since it leads to the same correct computations as Eq. (1) in the current section, which was used in one of the proofs of the traditional theorem. The fix is this. In deriving Eq. (1) in the Prologue by jumping into the middle, we assumed merely that we knew the optimum finish times for all  $i < j$ . Instead, let us assume more — that we know the optimum times for all  $i < j$  and that these finish times can be achieved by a single schedule for tasks 0 through  $j - 1$ . Explain why, with this assumption, Eq. (1) in the Prologue really is correct. Further, explain why with this assumption we can prove that the optimal schedule can be extended so that task  $j$  is optimally scheduled too. Thus by induction our original Prologue argument is OK after all.

This is yet another example of *strengthening the hypothesis* to make an induction work.

# Final Problems

This final problem set contains some challenging problems that recapitulate and extend various topics from the text. The problems are generally long with many parts; they can serve as introductions, or even outlines, for papers and projects.

---

1. ⟨4⟩ There are two versions of Straightline Towers of Hanoi (STOH), first discussed in [5–6, Section 2.4]. In one version you must move the tower between endpoints. In the other you must move it only from one end to the middle, or vice versa. Let us call the former LSTOH (L for long trip) and the latter S1STOH (S for short trip and 1 representing a move from the middle to the end) and S2STOH (2 for a move from the end to the middle).
  - a) Prove by induction that LSTOH can be won, but use only knowledge of previous cases of LSTOH, not any knowledge of S1STOH or S2STOH. More specifically for  $n$  rings let  $P(n)$  be the statement that LSTOH can be won, let  $Q(n)$  be the statement that S1STOH can be won, and let  $R(n)$  be the statement that S2STOH can be won. Then the inductive step for  $P(n)$  should use only earlier cases of  $P$  and not any earlier cases of  $Q$  or  $R$ . (In principle, it's perfectly legitimate, say, to first prove  $Q(n)$  and  $R(n)$  for all  $n$  and then use them in the inductive step of  $P(n)$ , but keeping the two versions completely separate leads directly to later parts of this problem.)
  - b) Using the notation in part a), prove  $Q(n)$  and  $R(n)$  together by “co-induction”. That is, if we let  $S(n)$  be the proposition  $Q(n) \wedge R(n)$ , do induction on  $S(n)$ . However, do not use your knowledge of the truth of  $P(n)$  from part a). (As soon as you start the proof, you'll see why we need to allow co-induction if we are not allowing use of  $P(n)$ .)
  - c) Prove that the number of ring moves for S1STOH and S2STOH are the same.
  - d) Turn your proofs in a) and b) into recurrences for the number of steps your method takes to win these games. Solve the recurrences.
  - e) From now on let's refer to both short-trip versions as SSTOH. No doubt your proof in b) for  $n$ -ring SSTOH requires somewhere in it two calls of  $(n-1)$ -ring SSTOH to be performed one right after the other. Conceivably fewer ring moves are required if one call of LSTOH is made instead. This leads to a system of linked recurrences, one for the minimum number of steps to solve  $n$ -ring LSTOH and one for minimal  $n$ -ring SSTOH. Set up and solve these linked recurrences. (How can you solve linked recurrences? One way is to do some algebra first to unlink them. Another is to use generating functions. After you turn the recurrences into statements about two generating functions, the algebra to unlink is standard. In any event, this pair of linked difference equations is only “partially” linked and you can solve them without a general method.)
  - f) Why does your answer in e) tell you that the minimum solution to LSTOH has the property that at one point the entire  $n$ -tower sits on the middle pole?
  - g) Prove the statement in f) directly by induction (without using any knowledge of the number of moves in minimal solutions).
2. ⟨4⟩ In Clockwise Towers of Hanoi (CTOH), the poles are on a circle and all the pieces are required to move clockwise but, as in STOH, always to an adjacent pole. There are two versions. In short-trip CTOH, you must move the tower to the next

pole in clockwise order. In long-trip CTOH, you must move the tower to the next pole in *counter-clockwise* order (but all the pieces must still move clockwise).

a) Prove that both versions can be won. As in [1], do each proof by an induction that depends only on its own previous cases.

b) Let  $L_n$  be the number of steps to win long-trip CTOH via the algorithm implicit in your proof in a). Similarly, let  $S_n$  be the number of steps in your short-trip algorithm. Write the recurrences implicit in your proofs and solve them.

c) The explicit formulas you got in b) do *not* give the minimum number of steps to win these games. For  $n = 2$ , find this minimum by playing the game yourself. Then explain conceptually what's "wrong" with the recurrences.

d) Set up linked recurrences for the minimum solutions for long- and short-trip CTOH. Justify why they are correct and solve them. (See the comments in [1d].)

e) Write a recursive algorithm for solving both long-trip and short-trip CTOH in the minimum number of steps. Your algorithm will have two procedures that call both themselves and each other. This is sometimes called **corecursion**.

3. ⟨5⟩ Again the poles of TOH are on a circle. Now, in addition to the basic rules, the smallest piece must always move clockwise to the adjacent pole (others can move either way). Determine, with proof, for which pairs of poles, if any, this  $n$ -ring game can be won. For each pair of start and finish poles for which the game can be won, show that there is a unique minimum set of moves that wins this version of the game. Find a formula for the minimum number of moves.

4. ⟨4⟩ Same as [3] except that the first move of the smallest ring must be clockwise, and thereafter each of its moves must be in the *opposite* direction from its previous move. (As in [3], there are no restrictions on the direction of movement of larger rings.)

5. ⟨5⟩ A society has  $n$  men and  $n$  women who want to get married. Each man has a preference ranking of all the women. That is, he knows who is his first choice, his second choice, ..., his  $n$ th choice. Similarly, each woman has ranked all the men. Here is an algorithm for matching them up:

i) Simultaneously, each man proposes to his first choice woman.

ii) If each woman has exactly one proposal, all proposals are accepted and the matchmaking is over; else

iii) Simultaneously, each woman keeps the best of her current suitors "on a string" and tells any others to get lost forever.

iv) Simultaneously, each man told to get lost forever proposes to the next woman on his list; return to ii).

a) Show that this algorithm always terminates with a set of  $n$  marriages. Among other things, you must show that you never reach a state where some man has been told to get lost by all  $n$  women. For that case, the algorithm doesn't say what to do, so it would terminate abnormally.

b) A set of marriages is **stable** if there is no man-woman pair, not married to each other, who prefer each other to their spouses. Prove that the set of marriages constructed by the algorithm is stable.

c) This algorithm is inspired by the traditional convention that "man proposes, woman disposes." Let's turn things around and have the women propose and the men dispose. We must get stable marriages, but do we always get the same set of marriages as when the men propose?

d) Suppose that there are more men than women. Find and justify a modification of the algorithm that still obtains a stable set of marriages. Part of the job is to redefine stability as well; remember, there will be men who are unhappy because they are not married at all.

e) Can you modify the algorithm so that it works if there are more women than men?

f) The initial assumption was that every man was willing to marry any of the women (perhaps reluctantly) and vice versa. Can the algorithm be made to work if each person is allowed to exclude people from his or her preference list, that is, if people are allowed to regard remaining unmarried as preferable to marriage to some people?

- g)** Can you modify the algorithm to make it work for college admissions? College admissions are like marriage where bigamy is allowed: Both colleges and applicants have preference lists, but colleges are allowed to “marry” more than one student.
- 6.** **(4)** (Multiplying  $n$ -digit numbers) The traditional (Indo-)Arabic algorithm for doing this is brilliant: It breaks down a single multiplication of long numbers into many multiplications and additions of two single-digit numbers. Calculation (i) below shows an example with two 3-digit numbers. Each line between the horizontal rules is a multiplication of 526 by one of the digits of 738. However, calculation (ii) shows that each such line is in fact the result of several single-digit multiplications added to each other with the aid of carrying.

$$\begin{array}{r}
 526 \\
 \underline{738} \\
 4208 \\
 1578 \\
 \underline{3682} \\
 388188
 \end{array}
 \quad
 \begin{array}{r}
 526 \\
 \underline{8} \\
 48 \\
 16 \\
 \underline{40} \\
 4208
 \end{array}$$

(i)                   (ii)

- a)** In the worst case, how many multiplications of two single-digit numbers are involved in multiplying two  $n$ -digit numbers by the Arabic method? How many additions of two single-digit numbers, where summing a column of  $k$  single-digit numbers counts as  $k - 1$  additions and each carry counts as one more addition.)

Let's try to do better by using the divide and conquer method. Suppose that  $n = 2m$  and that we already know how to multiply two  $m$ -digit numbers. If  $A$  has  $n$ -digits, we can think of the leftmost  $m$  digits as a single “superdigit”  $P$  and the rightmost  $m$  digits as another superdigit  $Q$ . In standard notation,  $A = P10^m + Q$ . If  $B$  is another  $n$ -digit number, we can likewise think of it as written with two superdigits  $R$  and  $S$ , where  $B = R10^m + S$ . Then we can calculate the product  $AB$  by four superdigit multiplications analogously to the Arabic multiplication of two 2-digit numbers.

- b)** Justify the last claim. *Hint:* Use algebra to express  $AB$  in terms of two-way products of  $P, Q, R$ , and  $S$  and then interpret this algebra using a calculation like calculation (i).

- c)** Write down a recurrence for the number of single-digit multiplications involved in multiplying two  $n$ -digit numbers by the divide and conquer method. Solve the recurrence. (See Section 5.8 for how to handle divide and conquer recurrences.) Has anything been saved in comparison with the Arabic multiplication method?
- d)** Similarly write down and solve a recurrence for the maximum number of single-digit additions using the divide and conquer method. There will be single-digit additions to compute the needed products of  $P, Q, R, S$ , and there will also be single-digit additions to add these various products. Assume (reasonably) that the number of single-digit additions to add these products is  $cn$  for some  $c > 2$ .
- e)** With  $A$  and  $B$  as given, verify that

$$\begin{aligned}
 AB = & PR10^n + [(P+Q)(R+S) \\
 & - (PR+QS)]10^m + QS.
 \end{aligned} \tag{1}$$

Note that this formula shows that the  $n$ -digit numbers  $A$  and  $B$  can be calculated in terms of only *three* multiplications of  $m$ -digit superdigits. (Actually, in one case you might get  $m+1$  digits, but you may ignore this. Why?) Devise a recursive procedure for multiplying  $A$  and  $B$  using this observation. Determine the number of single-digit multiplications involved.

- f)** Equation (1) appears to have more additions and subtractions than the standard method so maybe this undermines the reduction in multiplications. Devise and solve a recurrence for the number of additions and subtractions. Is the savings in multiplications undermined? (Use the same assumption as in d.)

- 7.** **(5)** (Fast matrix multiplication) If  $A$  is a  $2m \times 2m$  matrix, we may represent it as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

where  $A_{11}$ ,  $A_{12}$ ,  $A_{21}$ , and  $A_{22}$  are  $m \times m$  matrices. Similarly, if  $B$  is  $2m \times 2m$ , we may write

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}.$$

- a)** Verify that  $2m \times 2m$  matrices satisfy the same formula for multiplication as  $2 \times 2$  matrices;

that is, with the preceding notation,

$$AB =$$

$$\begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}. \quad (2)$$

(Note, however, that the order of the factors in terms like  $A_{11}B_{11}$  is now important, as matrices don't commute in general.)

- b) Use this observation to describe a recursive procedure for multiplying  $2^n \times 2^n$  matrices. Come up with a recurrence for the total number of real-number multiplications and additions needed by your recursive method. How does the answer compare with the complexity of the standard method?
- c) Equation (2) involved eight multiplications and four additions of  $m \times m$  matrices. Suppose that someone came up with a different method to multiply  $2m \times 2m$  matrices, call it procedure  $X$ , that involved seven  $m \times m$  multiplications and 18  $m \times m$  additions. This doesn't sound promising, but determine the complexity of the recursive algorithm required to multiply  $2^n \times 2^n$  matrices using procedure  $X$ . How does the answer compare with the standard method?
- d) There is a procedure  $X$ ! Show this by verifying the following. Using the preceding notation, if

$$P = (A_{12} - A_{22})(B_{21} + B_{22}),$$

$$Q = (A_{11} + A_{22})(B_{11} + B_{22}),$$

$$R = (A_{11} - A_{21})(B_{11} + B_{12}),$$

$$S = (A_{11} + A_{12})B_{22},$$

$$T = A_{11}(B_{21} - B_{22}),$$

$$U = A_{22}(B_{21} - B_{11}),$$

$$V = (A_{21} + A_{22})B_{11},$$

then

$$AB = \begin{bmatrix} P + Q - S + U & S + T \\ U + V & Q - R + T - V \end{bmatrix}.$$

How does this fact result in a procedure  $X$ ?

- e) How could you apply the method in d) to square matrices whose order is not a power of 2? Is the complexity the same for general  $n$ ? What about using this method for multiplying an  $m \times n$  matrix by a  $n \times p$  matrix?

8. (4) In [14, Section 1.5], you gathered data about the worst-case efficiency of Algorithm 1.6, EUCLID, which finds gcd's. (What we say below applies to EUCLID and to its recursive formulations, EUCLID-RECPRO and EUCLID-RECFUNC from Section 1.3. For the worst-case efficiency of some other variants, see [9].) It turns out that determining the exact worst case isn't too difficult if you look at it the right way, and the answer is a pleasant surprise. The key idea is to run the problem backwards, or more precisely, run EUCLID as always, but think about how to describe the quantities computed in the reverse order. The first line in the following display is the labeling we have used previously:  $m$  and  $n$  are the inputs, the  $r$ 's are the remainders, and  $r_{k-1}$  is the gcd, obtained on the  $k$ th iteration (since  $r_k = 0$ ). The second line is a new labeling for running backwards: What used to be  $m$  is now  $s_{k+1}$ , and what used to be  $r_k$  is now  $s_0$ .

$$\begin{array}{cccccc} m, & n, & r_1, & r_2, \dots, & r_{k-1}, & r_k = 0; \\ s_{k+1}, & s_k, & s_{k-1}, & s_{k-2}, \dots, & s_1, & s_0 = 0. \end{array}$$

- a) Assume that we always start with a pair  $(m, n)$ , where  $m \geq n$ . Show that with the relabeling  $\{s_0, s_1, \dots, s_{k+1}\}$ , there is a sequence of positive integers  $\{Q_1, \dots, Q_k\}$  so that

$$s_0 = 0,$$

$s_1$  = a positive integer, and

$$s_{j+1} = Q_j s_j + s_{j-1}, \quad j \geq 1.$$

- b) Explain why, for  $k > 1$ ,  $Q_1$  must be at least 2, but otherwise  $Q_j$  can be any positive integer.
- c) Argue from a) and b) that if  $(m, n)$  is any pair with  $m \geq n$  that requires at least  $k > 1$  iterations of Algorithm EUCLID to find the gcd, then

$$m \geq f_{k+1}, \quad n \geq f_k,$$

where  $f_k$  is the  $k$ th Fibonacci number. Argue further that  $(m, n) = (f_{k+1}, f_k)$  takes exactly  $k$  iterations and that this is the only pair  $(m, n)$  with  $m = f_{k+1}$  which takes exactly  $k$  iterations. This means that consecutive Fibonacci numbers are the worst cases (and the only worst cases with  $m \geq n$ ) for the Euclidean algorithm!

- d) What changes if we don't require that  $m \geq n$ ? You may be interested to know that, for large  $n$ , the *average* number of iterations to compute

$\gcd(m, n)$  is  $(12 \ln 2 \ln n)/\pi^2$ . The proof of this claim is quite deep.

9. ⟨5⟩ In [4, Supplement Ch. 1] we introduced EUCLID2, which sometimes finds a smaller remainder than EUCLID by rounding  $num/denom$  to get the quotient. (A recursive version of the same algorithm was introduced in [14, Section 2.4].)

- a) Run EUCLID and EUCLID2 on several pairs  $(f_{k+1}, f_k)$ , including  $(21, 13)$ . What happens? Can you prove the behavior you observe for EUCLID2?
- b) The Fibonacci pairs  $(f_{k+1}, f_k)$  are *not* the worst case for EUCLID2. Find a pair  $(m, n)$  with  $m < 21$  which takes EUCLID2 as many iterations as  $(21, 13)$ . (As you probably discovered in part a),  $(21, 13)$  is the smallest Fibonacci pair for which EUCLID2 takes 4 iterations.)
- c) Do as much as you can to bound or precisely determine the worst-case efficiency of EUCLID2. (This question is deliberately left vague. The treatment of EUCLID, starting with our initial bound on the worst case in Section 1.5 and continuing through various problems until [8], might provide some ideas.)

10. ⟨3⟩ Suppose that  $\{g_n\}$  satisfies the inequality

$$0 < g_n \leq cn^2 + g_{n-1} + g_{n-2}, \quad n > 1,$$

where  $c \geq 0$  is a constant. Prove that  $g_n = O(f_n)$ , where  $\{f_n\}$  is the Fibonacci sequence. This result is useful in [11].

11. ⟨4⟩ An independent set of vertices of a graph  $G$  is a set in which no two vertices are joined by an edge. We want an algorithm to find the **independence number**  $\alpha$  for any graph  $G$ , that is, the maximum size of an independent set of vertices in  $G$ . When

there is only 1 vertex, clearly  $\alpha = 1$ . More generally, suppose that we already know how to find  $\alpha$  for graphs with  $< n$  vertices and that we are now given  $G$  with  $n$  vertices. (When we say  $G$  is given, we mean that we have its adjacency matrix.) Pick any vertex  $v$ . Either a maximum independent set  $S$  contains  $v$  or it doesn't. If it doesn't,  $S$  is an independent set of  $G - v$ ; if it does,  $S - v$  is an independent set of  $G - N(v)$ , where by the **neighborhood**  $N(v)$  of  $v$  we mean  $v$  and all adjacent vertices. (Of course, when you take vertices from a graph, you also remove all the edges incident to those vertices.)

- a) State a recursive algorithm for finding  $\alpha$  that corresponds to the preceding discussion.
- b) Let  $g_n$  be the worst case amount of work for your algorithm when  $G$  has  $n$  vertices. (You have to stipulate what counts as work.) The preceding discussion did not ask you to take special care in picking  $v$ . You might pick an isolated vertex. In light of this, what is the best inequality recurrence for  $g_n$ ? Solve it. In the worst case, then, is your algorithm any better than checking each and every subset of the vertices of  $G$  for independence?
- c) If  $G$  has no edges then, clearly,  $\alpha = n$ . (The only work in checking the adjacency matrix is to verify that there are no edges.) Otherwise, you may pick  $v$  with at least one incident edge. Modify your algorithm to incorporate these observations. Now devise and solve a new inequality recurrence for  $g_n$  and thus verify that your modification improves the worst-case behavior.
- d) Apply the algorithms in a) and c) when  $G$  is a 4-cycle. Show a tree of subcalls. Represent each node of the tree by a picture of the graph that the algorithm analyzes in that call.

# References

The following is a selection of books, arranged in categories and with brief annotations, to use during a course where DAM is the text or after such a course.

## 1. Discrete Mathematics Books Accessible to most First- and Second-Year Students

There are now a great many discrete math books on the market. Those below suggest some of the different flavors.

Dossey, John A., Otto, Albert D., Pence, Lawrence E. and Vanden Eynden, Charles. 2002. *Discrete Mathematics* 4/e, Addison-Wesley.

Fairly elementary approach, with a very good range of applications, intended to interest a broad audience, including future teachers.

Epp, Susanna S. 2004. *Discrete Mathematics with Applications*, 3/e, Brooks Cole.

Well-written with especially extensive and effective material for improving reasoning skills.

Johnsonbaugh, Richard. 2005. *Discrete Mathematics*, 6/e, Prentice Hall.

Perhaps the text closest to DAM in the emphasis on algorithmics.

Rosen, Kenneth H. 2003. *Discrete Mathematics and its Applications*, 5/e, McGraw-Hill.

Comprehensive and readily available.

## 2. More Advanced Discrete Math Books

The next books are either for a second discrete math course, or for a first course taught at the junior-senior level.

Biggs, Norman L. 2003. *Discrete Mathematics*, 2/e, Oxford University Press.

Terse but clear, covering many more topics than DAM, including much more number theory and algebra.

Grimaldi, Ralph P. 1998. *Discrete and Combinatorial Mathematics: An Applied Introduction*, 4/e, Addison-Wesley.

Starts at the beginning, but covers many more topics than DAM, especially algebraic topics (e.g., Polya theory, finite state machines, finite fields and codes).

Roberts, Fred. 1984 *Applied Combinatorics*, Prentice Hall.

Fairly general discrete math coverage, but especially broad in combinatorics (more than DAM). Many examples in biology and social sciences.

## Books on individual topics

### 3. Algorithms

Aho, Alfred V., Hopcroft, John E., and Ullman, Jeffrey D. 1974. *The Design and Analysis of Computer Algorithms*, Addison-Wesley.

A classic; terse, but covers a lot of ground, and everyone refers to it.

Cormen, Thomas H., Leiserson, Charles E., and Rivest, Ronald L. 2001. *Introduction to Algorithms*, 2/e, McGraw-Hill.

A popular, comprehensive general computer science text on algorithms, e.g., programming, data structures, important specific algorithms, as well as the mathematics of them.

Greene, Daniel H. and Knuth, Donald E. 1990. *Mathematics for the Analysis of Algorithms*, 3/e, Birkhauser.

Worked up from lecture notes on a course Knuth gave specifically to flesh out carefully the mathematics of algorithm analysis.

Horowitz, Ellis, Sahni, Sartaj, and Rajasekaran, Sanguthevar. 1997. *Computer Algorithms*, Computer Science Press.

Similar to the book by Cormen above, but perhaps a bit easier. Comes in both pseudocode and C++ versions.

Sedgewick, Robert, and Flajolet, Philippe. 1995. *An Introduction to the Analysis of Algorithms*, Addison-Wesley.

Special attention to average-case analysis and to algorithms of particular interest in computer science.

Wilf, Herbert S., 2002. *Algorithms and complexity*, 2/e, A K Peters.

Emphasizes problems of mathematical interest: algorithms in number theory (including public key encryption), Fast Fourier Transform, network flows.

### 4. Induction

There are few books exclusively on mathematical induction; instead there is usually a section or chapter on induction in more general math books. Here are the books we know.

Golovina, L. I., and Yaglom, I. M. 1963. *Induction in Geometry*, D. C. Heath.

While geometry is not really a discrete math topic, there are lovely and surprising uses of induction in this pamphlet.

Youse, Bevan K. 1964. *Mathematical induction* Prentice Hall.

This book (and the next) are mostly about traditional uses of induction (e.g., formulas for sums — no algorithms).

Sominskii, I. S. 1961. *The Method of Mathematical Induction*, Pergamon Press.

### 5. Graph Theory

Wilson, Robin J. 1996. *Introduction to Graph Theory*, 4/e, Longman.

A relatively short book for this subject, well written and excellent for self-study.

Bondy, J. Adrian, and Murty, U.S.R. 1976. *Graph Theory with Applications*, American Elsevier.

Considered a classic because of its elegant proofs and broad coverage for a thin volume.

Gross, Jonathan, and Yellen, Jay. 1998. *Graph Theory and its Applications*, CRC Press.

Comprehensive, with strong emphasis on computer science and operations research applications and algorithms.

West, Douglas B. 2001. *Introduction to Graph Theory* 2/e, Prentice Hall.

A more traditional approach (more structure theory, less algorithms) with broad coverage and many challenging problems.

## 6. Counting (Combinatorics)

Graham, Ronald L., Knuth, Donald E., and Patashnik, Oren. 1994. *Concrete Mathematics*, 2/e, Addison-Wesley.

Treats, in a unique style, advanced methods to solve problems of particular interest in computer science. While it combines CONtinuous and disCRETE mathematics, in fact it has very much the flavor of discrete mathematics.

Stanley, Richard P. 2000. *Enumerative Combinatorics*, Vol. 1, Cambridge University Press.

A standard advanced text by a leading researcher.

Stanton, Dennis, and White, Dennis. 1986. *Constructive Combinatorics*, Springer-Verlag, New York.

Combinatorics with a distinctly modern, algorithmic flavor. Like DAM, it uses algorithms to generate both theorems and their proofs.

Petkovsek, Marko, Wilf, Herbert S., and Zeilberger, Doron. 1996. *A = B*, A K Peters.

A very readable exposition of ground-breaking research on the automation of the discovery (or proof of non-existence) of combinatorial identities.

Wilf, Herbert S. 1994. *Generatingfunctionology*, 2/e, Academic Press.

A very thorough and lively treatment of the generating function approach to counting.

## 7. Difference Equations

Goldberg, Samuel I. 1986. *Introduction to Difference Equations*, Dover.

A classic, (originally published 1958) with a leisurely development of the theory. Contains many social science examples, and introduced many social scientists to the subject.

Mickens, Ronald E. 1991. *Difference Equations: Theory and Applications*, 2/e, CRC Press.

A more advanced work.

## 8. Probability

Devore, Jay L. 2003. *Probability and Statistics for Engineering and the Sciences*, 6/e, Duxbury Press.

Well written, assumes modest calculus, and covers substantial amounts of both probability and statistics.

Feller, William. 1968. *An Introduction to Probability Theory and Its Applications*, Vols. 1 and 2, 3/e, Wiley.

A classic. Wonderful examples. The first volume is mainly about discrete probability.

Grinstead, Charles M., and Snell, Laurie J. 1997. *Introduction to Probability*, 2/e, American Mathematical Society.

Introductory text that treats discrete and continuous in parallel and provides many computer programs for computation, simulation, and development of intuition.

## 9. Logic

Bergmann, Merrie, Moor, James, and Nelson, Jack. 2003. *The Logic Book*, 4/e, McGraw-Hill.

Popular general symbolic logic text, often used by philosophy departments.

Quine, Willard V. 1989. *Methods of Logic*, 4/e, Harvard University Press.

A classic by a leading logician.

Schagrin, Morton L., Rapaport, William J., and Dipert, Randall R. 1985. *Logic: A Computer Approach*, McGraw-Hill.

A fairly low-level book and one of the few books on logic aimed at computer scientists.

## 10. Discrete Math and Biology

Gusfield, Dan. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press.

A very thorough and current presentation by a computer scientist who got interested in computational biology.

## 11. Matching Algorithms

Almost any book on linear programming, linear optimization, or combinatorics will contain the basic matching results, and most will contain algorithmic proofs. But here is an advanced book devoted entirely to this subject.

Lovász, L., and Plummer, M.D. 1986. *Matching theory*, North-Holland.

## 12. Other Useful Books

Pemmaraju, Sriram, and Skiena, Steven S. 2003. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Cambridge University Press.

A guidebook to *Combinatorica*, probably the most powerful general discrete math software available currently — it's an add-on to the CAS *Mathematica*.

Polya, George. 1971. *How to Solve it*, Reissue edition, Princeton University Press.

Polya's books, especially this one, are *the* source on developing mathematical problem-solving skills.

Roberts, Eric S., 1986. *Thinking Recursively*, Wiley.

A charming introduction to the beauty and power of the Recursive Paradigm. Many unusual examples.

Rosen, Kenneth H., ed. 1999. *Handbook of Discrete and Combinatorial Mathematics*, CRC Press.

A very general quick reference — definitions, theorem statements, references, but no proofs — with brief presentations on just about every discrete topic.

Solow, Daniel. 2002. *How to Read and Do Proofs: an Introduction to Mathematical Thought Processes*, 3/e, Wiley, New York.

A very thorough and readable short book, made even more accessible because for examples it uses mostly high school mathematics.

# Limits

In Section 0.3 we introduced the concept of a limit informally and relied on your intuition to understand what we said about limits in our development of Order Notation. In analyzing the qualitative behavior of sequences (Section 5.6) we made informal use of facts about the limits of products and quotients to determine long-term behavior. In Chapter 6, Section 6.4, we used the idea of a limit once again in Example 7 to show how to develop the formula for the Poisson distribution. In this appendix we wish to formalize the definition of a limit and some of the theorems related to it. The definitions and proofs in this appendix are covered in some calculus courses and all real analysis courses.

In discrete mathematics the main interest in limits is in connection with sequences. Only occasionally do we mention limits of functions (e.g., in Section 0.3), but since the two kinds of limits are closely related and because many readers of this book will be familiar with limits of functions from a calculus course, we'll define both kinds of limits here. We begin with limits of sequences.

---

**Definition 1.** Let  $\{a_n\}$  be a sequence of real numbers. We say that the sequence **converges to a limit  $L$** , and write

$$\lim_{n \rightarrow \infty} a_n = L,$$

if, for each positive number  $\epsilon > 0$ , there exists a positive integer  $N$  such that

$$|a_n - L| < \epsilon \quad \text{for all } n > N.$$

---

In words,  $L$  is the limit of the sequence  $\{a_n\}$  if after some point (i.e., after some value  $N$  of the subscript  $n$ ) all values of  $a_n$  are arbitrarily close to  $L$ . What does “arbitrarily” mean here? Just that no matter how small you take  $\epsilon$  to be in Definition 1, you can still find an  $N$  such that all values of  $a_n$  after  $a_N$  will be within  $\epsilon$  of  $L$ .

It is important to understand that the notation  $\lim_{n \rightarrow \infty} a_n = L$  means first, that the sequence *has* a limit, and second that the limit is  $L$ . This distinction is important because in fact many sequences have no limit. For example, consider the sequence  $\{a_n\}$  where  $a_{2i} = 0$  and  $a_{2i+1} = 1$  for  $i = 0, 1, 2, \dots$ . That is, terms of the sequence with even subscripts are 0 and those with odd subscripts are 1.

There is no  $L$  to which, after any value at all of  $N$ , terms of the sequence will all be arbitrarily close. Specifically, take  $\epsilon = 1/3$ . No matter how large  $N$  is,  $L$  would have to be within  $1/3$  of 1 (for odd-indexed terms with index  $> N$ ) and also within  $1/3$  of 0 (for even-indexed terms); this can't be.

*Notes:*

1. We could, maybe should, have written  $N(\epsilon)$  instead of  $N$  above because generally the smaller the  $\epsilon$  you choose, the larger  $N$  you need. So  $N$  is really a *function* of  $\epsilon$ . But it is customary just to write  $N$  as we have.
  2. Using this definition you could formally show the correctness of all uses of limits in Section 0.3.
- 

**Definition 2.** Let  $f(x)$  be a function defined around a point  $x_0$ . Then we say that the function **converges to a limit  $L$**  as  $x \rightarrow x_0$ , and write

$$\lim_{x \rightarrow x_0} f(x) = L,$$

if, for each positive number  $\epsilon > 0$ , there exists another positive number  $\delta > 0$  such that

$$|f(x) - L| < \epsilon \quad \text{for all } x \neq x_0 \text{ such that } |x - x_0| < \delta.$$

---

*Notes:*

1. The reason  $x = x_0$  is excluded is because limits are about what happens as you *approach* something; for functions we are interested in what happens to the values as you approach  $x_0$  but not about the value at  $x_0$  itself. The issue of excluding the limit point does not arise in the sequence case, because the limit “point” is infinity, and a sequence has no value *at* infinity, that is, no term  $a_\infty$ . When a function converges to  $L$  at  $x_0$  and is also defined at  $x_0$  so that  $L = f(x_0)$ , we say that  $f$  is **continuous** at  $x_0$ .
2. Most people find Definition 2 harder to understand than Definition 1 because it uses two very small quantities,  $\epsilon$  and  $\delta$ , as opposed to only one such,  $\epsilon$ , in Definition 1. But, in fact, the restriction in Definition 2 that  $x$  be within  $\delta$  of  $x_0$  in order to assure that  $f(x)$  is within  $\epsilon$  of  $L$  is quite analogous to the requirement in Definition 1 that  $n$  be greater than  $N$  in order to assure that  $a_n$  is within  $\epsilon$  of  $L$ .

Now we shall prove a theorem, whose truth we assumed implicitly at a number of places in Section 0.3.

---

**Theorem 1.** Let  $\{a_n\}$  and  $\{b_n\}$  be two sequences with, respectively, limits  $L_1$  and  $L_2$ . Then

$$\lim_{n \rightarrow \infty} (a_n + b_n) = L_1 + L_2. \quad (3)$$

---

That is, if  $\{a_n\}$  and  $\{b_n\}$  both have limits, then the limit of the sum is the sum of the limits. This assertion may seem intuitively obvious, but, particularly with new concepts, it is always worth confirming your intuition with a proof.

**PROOF** Pick any  $\epsilon > 0$ . Then there is an  $N$  such that

$$|a_n - L_1| < \epsilon \quad \text{and} \quad |b_n - L_2| < \epsilon \quad \text{for all } n > N. \quad (4)$$

Why? Because by assumption the two sequences have limits,  $L_1$  and  $L_2$ , so that applying Definition 1 to our chosen  $\epsilon$ , there is an  $N_1$  such that  $|a_n - L_1| < \epsilon$  for all  $n > N_1$  and an  $N_2$  such that  $|b_n - L_2| < \epsilon$  for all  $n > N_2$ . Now simply let  $N = \max(N_1, N_2)$  and we have verified (4).

Next apply the Triangle Inequality. From (4) it follows that for all  $n > N$

$$\begin{aligned} |(a_n + b_n) - (L_1 + L_2)| &= |(a_n - L_1) + (b_n - L_2)| \\ &\leq |a_n - L_1| + |b_n - L_2| \quad [\text{Triangle Inequality}] \\ &< \epsilon + \epsilon = 2\epsilon = \epsilon'. \end{aligned} \quad (5)$$

Since  $\epsilon$  is an arbitrarily small number so is  $2\epsilon$ , justifying our replacement of  $2\epsilon$  by  $\epsilon'$ . Thus (5) shows that we can make  $a_n + b_n$  arbitrarily close to  $L_1 + L_2$  (namely,  $2\epsilon = \epsilon'$  close) by making  $n$  big enough. We conclude that the sequence  $\{a_n + b_n\}$  has limit  $L_1 + L_2$ . ■

*Note.* If you're not used to theorems about limits, you may not see immediately that our replacement of  $2\epsilon$  by  $\epsilon'$  is not sleight of hand. But think about it. All that need be understood is that, when dealing with an arbitrarily small number  $\epsilon$ , then  $c\epsilon$ , where  $c$  any positive constant, is also arbitrarily small. We could have given an equivalent proof in which we replaced  $\epsilon$  in (4) by  $\epsilon/2$  and then ended up with  $\epsilon$  in (5). Would you have liked such a proof better?

In Example 2, p. 32, we assumed that Theorem 1 is true for the sum of  $k$  sequences for any  $k$ . That is we assumed the following theorem:

---

**Theorem 2.** Let  $\{a_n^{(1)}\}, \{a_n^{(2)}\}, \dots, \{a_n^{(k)}\}$  be  $k$  sequences with, respectively, limits  $L_1, L_2, \dots, L_k$ . Then

$$\lim_{n \rightarrow \infty} (a_n^{(1)} + a_n^{(2)} + \dots + a_n^{(k)}) = L_1 + L_2 + \dots + L_k.$$

---

How would you prove this theorem? By induction, of course. We leave it to you. (See [14, Section 2.3].)

There are, as we hope you would expect, analogs of Theorem 1 for subtraction, multiplication and division of sequences. These are given by the following theorem:

---

**Theorem 3.** Let  $\{a_n\}$  and  $\{b_n\}$  be two sequences with, respectively, limits  $L_1$  and  $L_2$ . Then

- 1)  $\lim_{n \rightarrow \infty} (a_n - b_n) = L_1 - L_2.$
  - 2)  $\lim_{n \rightarrow \infty} (a_n b_n) = L_1 L_2.$
  - 3)  $\lim_{n \rightarrow \infty} (a_n / b_n) = L_1 / L_2,$  if  $L_2 \neq 0.$
- 

In other words, *if*  $\{a_n\}$  and  $\{b_n\}$  converge, then so do their difference and product, and to the limit you would guess. If also the limit of  $\{b_n\}$  is not 0, then the quotient converges to what you would expect.

We shall not prove this theorem, nor its function analog. We note only that the proof of the first part is precisely analogous to the proof of Theorem 1. The proof of the second part requires a rather more sophisticated application of the basic idea in the proof of Theorem 1. The proof of the last part is more difficult still.

# Hints and Answers

## Prologue

1. 50   2. 25   3.  $T(j) = j + T(j-1)$  so  $T(n) = n + (n-1) + \dots + 3 + 2 + 1 = n(n+1)/2$ .   4. a)  $S(j) = T(j) - d(j)$   
b)  $S(j) = \max_{(i,j) \in E} S(i) + d(i)$ .   5.  $T(j) = \max_{h(e)=j} \{d(e) + T(t(e))\}$  where  $d(e)$  is duration of task,  $h(e)$  is vertex at head of  $e$ ,  $t(e)$  is vertex at tail of  $e$ ,  $T(j)$  is minimum time to get to vertex  $j$  and  $h(e) = j$  means all tasks whose heads point to  $j$ .   6. For  $n$ -vertex graph assume solution for  $(n-1)$ -vertex graph, assign 1 to some vertex with no arrow into it (why must there be such a vertex?), and then assign 2 to  $n$  to other vertices using  $(n-1)$ -vertex solution.

## Chapter 0

### Section 0.1

1. a)  $\{\}, \{x\}, \{y\}, \{z\}, \{x,y\}, \{x,z\}, \{y,z\}, \{x,y,z\}$    2.  $P(\{a\})$  is  $\{\emptyset, \{a\}\}$ ;  $P(P(\{a\}))$  is  $\{\emptyset, \{\emptyset\}, \{\{a\}\}, \{\emptyset, \{a\}\}\}$   
3. c) all non-prime positive integers  $> 2$ .   4. a)  $\{x \mid 100 < x < 200 \text{ and } 2 \nmid x\}$    5. a)  $F \cap N^+$    c)  $Q - Z$    e)  $F \cap N^+ \cap E$   
g)  $Z - E - N$    i)  $Z - N$    6. c)  $S - T = \{1, 9\}$    9. No. (Why not?).   10.  $|A \cup B| = |A| + |B|$    12. Nothing; give an example to show why.   15. Least:  $m$ ; Most:  $m+n$    17. Build up a table with, for each value of  $p$  ( $1 \leq p \leq 10$ ), the number of fractions  $p/q$  which have not already been counted.   19. a) reflexive, transitive   c) symmetric  
20. a,b,d) reflexive, transitive   c) symmetric   21. (3,4), (6,7) are in; (3,5), (6,8) are not.   24. a) (1, 5), (2, 6) are; (1, 6), (2, 5) are not.   b) Transitive because if  $m \mid (p-q)$  and  $m \mid (q-r)$ , then  $m \mid [(p-q)+(q-r)] = (p-r)$ .   25. Use transitivity to prove both if and only if parts.   27. a)  $\{(x, y) \mid x < y\}$    d)  $\{(x, y) \mid x+1=y\}$    28. Yes for all three.  
29. a)  $\{(s, t) \mid s > t\}$    b) itself   30. a) child   b) ancestor   32. Need one-to-one so that no two people have same SS number; not onto so that more SS numbers may be assigned.   34. a) True   b) True   c) False; may not be onto   d) True   e) False; may not be onto   f) True   g) False; domain points with more than one image point could be balanced by codomain points not image points.   36. Image: Values of  $a_i$ ; if start with  $a_0$ , domain is  $N$ .   37. Same functions because all have same domain, same image of each point in domain.   39. Partial inverse is bijective because it associates each point in image uniquely with a point in the domain and vice versa.  
40. b)  $f(S) : 4 \leq x \leq 16$ ;  $f^{-1}(S) = \{x \mid 2 \leq x \leq 4\} \cup \{x \mid -4 \leq x \leq -2\}$ .   41. b) The pairs  $(x, f(x))$  form a Cartesian product on  $D \times C$ .

### Section 0.2

1. b) 2.531   4. a) 3   7. a)  $0 \leq x < 1$    9. Both sides positive with same value   11.  $[n/2] - [n/2]$    13.  $[\log_3(x-2)]$   
14. a) Any integer   c) All   e) All   g) Any integer or nonnegative real with fractional part  $< 1/2$  or negative real with fractional part  $> 1/2$    i) All multiples of  $n$    15. a)  $|x+y| \geq |x| + |y|$    c)  $|x-y| \geq |x| - |y|$    e)  $n|x/n| \leq |x|$ , equal for all  $m$ ,  $nm \leq x < nm+1$    16. b) Let  $w = x+y$  and use [15b].   18. a)  $R'(x) = \lceil x+.5 \rceil - 1$    20.  $\lfloor 10x+.5 \rfloor / 10$ ; rounds up if hundredths place is 5.   23. b) 4   c) Need number of multiples of each power of 5; therefore  $\lfloor n/5 \rfloor + \lfloor n/25 \rfloor + \lfloor n/125 \rfloor + \dots$    25. Some term in  $n_{(m)}$  is 0 if  $m > n$ .   27.  $(n-1/2)_n$    29.  $1^{(n)}$    31. Limit is 0.  
32. a) 15   b) 3; why are there no others?   34. b) 7   c) 127 mod 2 = 1.   35. a)  $\{x \mid x = 5n+2 \text{ and } n \in Z\}$   
36. a) 3   d)  $10/3$    37. a)  $kL$    c)  $-kL$    38. b)  $10^{1000}$    39. c) 4   43. For second equality, use result of [42].

45. Properties 1–3 hold but Property 4 does not;  $\lim_{x \rightarrow \infty} \log_b x = -\infty$ .

46. Product:  $x^6 - 4x^5$

47.  $x^4 - 1$

49. Just show that the  $x^2$  terms cancel.

### Section 0.3

1. a)  $n^3$ ; d)  $n^2$ ; g)  $n$ ; j)  $[(1.01)^{1/100}]^n$ ; 3.  $n^2 \log n$ ; by Theorem 2 the constants and other terms don't matter.  
5. b) Negative; use Theorem 1 to show that  $\text{Ord}(n^2) < \text{Ord}(n^3/\log n)$ . 6. Both  $a_n/1$  and  $a_n/c$  have a nonzero limit. A sequence if  $\text{Ord}(1)$  iff it is bounded. 8. No; if  $a < b$ , then  $a_n = (a/b)^n b^n = o(b^n)$ . 9. a) 45 since  $2^{45}$  just less than  $3.6 \times 10^{13}$ , number of operations in 10 hours; b) 514 10. a) About .7; b) About 37.  
12. No, because if  $b > a$ , then  $2^{\log_b n}/2^{\log_a n} = 2^{\log_b n(1-1/\log_b a)} \rightarrow 2^{-\infty} = 0$ . 14. c)  $(n^r \log n)/n^{r+1} = (\log n)/n \rightarrow 0$ .  
15. a) With  $x = a^n$ , then  $n = \log_a x$ , so  $n^r/a_n$  becomes  $(\log x)^m/x$ . b) Follows from (ii) and (iv) of Theorem 1.  
17. Greater than all others, because  $b_n/n! \rightarrow 0$  for any  $b$  as  $n \rightarrow \infty$ . 19. Let  $U'$  be the minimum of  $|a_n/b_n|$  for the exceptional  $n$ 's. If  $U' < U$ , replace  $U$  by  $U'$  in Inequality (3). Likewise define and use  $L'$ . 21. E.g., i) says: if  $a_n, a'_n$  are  $o(b_n)$ , then  $ca_n + c'a'_n = o(b_n)$ . 24.  $\frac{a_n}{c_n} = \frac{a_n}{b_n} \frac{b_n}{c_n}$  and  $\frac{a_n}{c_b}, \frac{b_n}{c_n}$  both  $\rightarrow 0$ . 26. a)  $\frac{3n-1}{3n} \rightarrow 1$  but  $\frac{3n-1}{n} \rightarrow 3$ .  
b)  $c = 1$  in Definition 1. e)  $\frac{a_n}{c_n} = \frac{a_n}{b_n} \frac{b_n}{c_n} \rightarrow 1 \cdot c = c$ . 28. a) If  $|x| \leq \frac{1}{2}$ , then  $\frac{1}{3} \leq \frac{x+1}{x+2} \leq \frac{3}{5}$ . But  $\frac{x}{x+1} \rightarrow 0$  as  $x \rightarrow 0$ , so no positive  $L$  exists.

### Section 0.4

1. d)  $\prod_{i=0}^{49} (2i+1)$  2. d)  $\sum_{i=0}^7 (i+1)x^{2i}$  3. a) 7451/110592 4. b) 2742 5. a) 1–1/5 for  $n = 4$ ; telescopes  
c) 1/6 for  $n = 5$ ; telescopes e) –8 for  $n = 6$  7.  $k = 0$  term is 0; index name is irrelevant. 9.  $i = 1$  to  $n+1$   
11.  $a = 0$ ,  $b = 4$ ,  $f(j) = (2j+1)^2$ . 13. a) 3 for  $n = 3$  14. 51; 200 17. a) (0, 10) d) [0, 1] h) Empty set; 0 is in no interval and no positive number is in every interval.

### Section 0.5

1.  $\begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$  3.  $\begin{bmatrix} 4 & 8 \\ 12 & 16 \end{bmatrix}$  5. a) [5 1 9] 7. a)  $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$  9.  $\begin{bmatrix} 285 \\ 335 \\ 560 \end{bmatrix}$  10.  $n \times n$  matrix 12.  $M^3 = \begin{bmatrix} 1 & 3 & 6 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix}$ .  $M^n$

is like  $M^3$  except that (1, 2) and (2, 3) elements are  $n$  and (1, 3) element is  $n(n+1)/2$ . 14.  $c = 22a_1 + 28a_2$ .

15. Columns of  $A$  = Rows of  $B$ , Columns of  $B$  = Rows of  $C$ ; double sum over  $r$  and  $s$  of  $a_{ir}b_{rs}c_{sj}$  to get  $(i, j)$  element of product.

### Section 0.6

1. b) If it's hot, then flowers wilt. 3. a) If  $n$  is even, then  $2n$  is even. c) If there is a 29 Feb. this year, then this is a leap year. 4. a) False c) True e) False 5. a) If  $ABCD$  is a parallelogram, then  $AB \parallel CD$ .  
6. a) If  $ABCD$  is not a parallelogram, then  $AB$  not parallel  $CD$ . b) If a computer program doesn't terminate, then the program is not correct. 7. b) If  $n$  is even, then  $2n$  is odd. 8. (i), (iii) and (iv) are equivalent;  
(ii) and (v) are equivalent. 10. a) True d) False g) True (but it would be false if  $\pi$  were replaced by  $x$ ).  
11. Pairs which are negations of each other are  $a-d, b-c, c-e, d-f$ . 12. a) Same meaning (but not very good English!) c, e) Not good English 14. Not valid because if  $A$  is not a single digit, then  $9|(A+B)$  is not true by hypothesis; but the germ of a correct proof is there. 15. Direct: If quadrilateral  $Q$  is a rectangle, all angles are  $90^\circ$  and, if a rhombus, all sides are equal; therefore,  $Q$  is a square. Indirect: Suppose  $Q$  not a square. Then either  $Q$  has unequal sides and so is not a rhombus or has angles not all  $90^\circ$  and so is not a rectangle; contradiction.  
17. a) A generic number in  $S$  is the sum of some odd number and some even number and thus is odd. b) Any generic odd number  $n$  is the sum of  $n-2$  (odd) and 2, hence is in  $S$ . 18. Subtract  $x$  from both sides to get  $0=1$ .  
21. If  $x$  is in  $S \cap T$ , then  $x \in S$  and  $x \in T$  and so  $x \in U$ . 23. a) Vacuously true; there is no (integer)  $k$ . b) True but not vacuously true because there is a  $k (= 2)$ . f) True (obviously so) but not vacuously true.

### Supplementary Problems: Chapter 0

1. a)  $315 = 5 \times 7 \times 9$ ;  $91 = 7 \times 13$ ; therefore  $\gcd = 7$ . b)  $440 = 2^3 \times 5 \times 11$ ;  $924 = 2^2 \times 3 \times 7 \times 11$ ; therefore  $\gcd = 2^2 \times 11 = 44$ . 3. b) The product of the two integers c) Show that if  $p^i$  is a factor of  $m$  and  $p^j$  is a factor of  $n$  (for  $p$  prime), then one of these is a factor of the gcd and the other is a factor of the lcm. 4. a) Domain:  $x > 1$ ; Range:  $x$  real 6. b) All  $x \geq 0$  c) None because no set is its own complement. 7.  $O_n = H_n - (1/2)H_{\lfloor n/2 \rfloor}$ ; therefore,  $\lim O_n/H_n = \lim(1 - (1/2)H_{\lfloor n/2 \rfloor}/H_n)$ . Since  $H_{\lfloor n/2 \rfloor} < H_n$  and (assuming this limit exists), it must

be between  $1/2$  and  $1$  which shows  $O_n = \text{Ord}(H_n)$ .    **8.**  $n2^{n-1}$ ; correct but less helpful:  $\sum_{k=1}^n k n! / [(n-k)!k!]$ .  
**10.c)** Add  $(-1)^k$  under summation.    **11.c)** Substitute  $b/d$  into  $f(x)$ , then multiply by  $d^{n-1}$  to get result for  $d$ , multiply by  $d^n/b$  to get result for  $b$ .    **d)**  $1/2, 1, 3/2, 2, 3, 6$  and their negations    **13.a)** Number of variations in sign excluding zeros of  $a_0, -a_1, a_2, \dots, (-1)^n a_n$     **14.b)** 0 positive zeros, up to 10 negative zeros

## Chapter 1

### Section 1.1

**1.** You can only repeat a loop a nonnegative integer number of times, but you can take nonnegative integer powers of any  $x$ .    **3.a)** Add  $x$  copies of  $y$ .    **b)** Strip the signs, use part a), then put the right sign back.  
**4.c)**  $-3 + -3 + \dots = -27$ .    **6.a)** Edges added:  $(1, 2), (2, 3), (2, 4)$ ; Total cost: 990    **7.a)** Same edges end up used, but picked in order of increasing cost: 150, 175, 250, 295.    **8.a)** Edges added:  $\{2, 4\}, \{2, 3\}, \{3, 6\}, \{4, 5\}, \{1, 5\}$ ; Total cost: 1050    **c)** Looking for maximum weight spanning tree is equivalent to negating all the weights and looking for a minimum weight tree; FIBERNET-MAX finds a maximum weight tree since it chooses the same edges as FIBERNET chooses when the weights are negated.    **11.** If  $m < n$ , the first iteration switches them. If  $n = 0$ , immediate termination ( $\text{gcd } m = m$ ). If  $m = 0$ , switch then terminate.    **12.a)** For  $(45, 26)$  the  $r_k$  sequence for EUCLID is 45, 26, 19, 7, 5, 2, 1, 0. For EUCLID1 it is 45, 26, 7, 2, 1, 0. So every other term skipped in the middle. Same for  $(21, 13)$ .    **b)** A number divides  $\text{denom}$  and  $\text{rem}$  iff it divides  $\text{denom}$  and  $\text{denom} - \text{rem}$ .

### Section 1.2

**2.** It prints “high” if  $n > 20$ , “low” if  $n < 10$ , and “middle’ otherwise (if  $10 \leq n \leq 20$ ).    **3.** With values of  $k$  bold, the trace is: **5 1 4 8 16 2 12 24 3 15 30**.    **5.**  $p = n!$     **6.b)** Both variables will have the old  $b$  value.

|                                          |                                |                                |                               |
|------------------------------------------|--------------------------------|--------------------------------|-------------------------------|
| <b>8.a)</b> $P = \sum_{i=0}^n a_i x^i$ . | <b>8.b)</b> $P \leftarrow a_n$ | <b>11.</b> $S \leftarrow 0$    | $S \leftarrow 0$              |
|                                          | $k \leftarrow 0$               | $k \leftarrow 1$               | $k \leftarrow 1$              |
|                                          | <b>repeat</b>                  | <b>repeat while</b> $k \leq n$ | <b>repeat until</b> $k = n+1$ |
|                                          | $k \leftarrow k+1$             | $S \leftarrow S+k$             | $S \leftarrow S+k$            |
|                                          | $P \leftarrow P x + a_{n-k}$   | $k \leftarrow k+1$             | $k \leftarrow k+1$            |
|                                          | <b>endrepeat when</b> $k = n$  | <b>endrepeat</b>               | <b>endrepeat</b>              |

|                                                                                            |                                                                       |
|--------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| <b>12. b)</b> <b>if</b> $a > b$ <b>then</b> <b>if</b> $a > c$ <b>then</b> $L \leftarrow a$ | <b>14.</b> Remember $i   N$ is true only if $i$ is a divisor of $N$ . |
| <b>else</b> $L \leftarrow c$                                                               | <b>16.</b> If a number is even, what condition must be true?          |
| <b>endif</b>                                                                               | <b>18. if</b> $n > 10$ <b>then if</b> $n \in Z$ <b>then print</b> $n$ |
| <b>else if</b> $b > c$ <b>then</b> $L \leftarrow b$                                        |                                                                       |
| <b>else</b> $L \leftarrow c$                                                               |                                                                       |
| <b>endif</b>                                                                               |                                                                       |
| <b>endif</b>                                                                               |                                                                       |

|                                                |                                            |                                               |
|------------------------------------------------|--------------------------------------------|-----------------------------------------------|
| <b>20. if</b> $n > 0$ <b>then print</b> $n$    | <b>22. for</b> $n = 1$ <b>to</b> 10        | <b>24. Use repeat while</b> $n^2 \leq 1000$ . |
| <b>else if</b> $n \in Z$ <b>then print</b> $n$ | $f \leftarrow f * n$                       |                                               |
| <b>endif</b>                                   | <b>if</b> $3   n$ <b>then print</b> $n, f$ |                                               |

|                                                               |                                                  |                                                  |
|---------------------------------------------------------------|--------------------------------------------------|--------------------------------------------------|
| <b>25. for</b> $i = a$ <b>to</b> $b$ becomes $i \leftarrow a$ | <b>28. rem</b> $\leftarrow m$ ; $q \leftarrow 0$ | <b>30. Output</b> $c$                            |
| Do Something                                                  | <b>repeat until</b> $rem < n$                    | $a \leftarrow 1$ ; $b \leftarrow 1$ $[f_0, f_1]$ |
| <b>repeat until</b> $i > b$                                   | $rem \leftarrow rem - n$                         | <b>repeat until</b> $c > 100$                    |
| Do Something                                                  | $q \leftarrow q + 1$                             | $c \leftarrow a+b$ [next $f$ ]                   |
| $i \leftarrow i+1$                                            | <b>endrepeat</b>                                 | $a \leftarrow b$                                 |
| <b>endrepeat</b>                                              |                                                  | $b \leftarrow c$                                 |
|                                                               |                                                  | <b>endrepeat</b>                                 |

**32.** If  $D_0 = 9$ , set  $D_0 = 0$  and add 1 to  $D_1$ . Repeat, possibly creating new digit  $D_{n+1}$  as last step. **34.** To test if odd  $i$  is prime, we need not check any numbers greater than  $\lfloor \sqrt{i} \rfloor$  to see if they divide  $N$  since, if  $d > \lfloor \sqrt{i} \rfloor$  is such that  $d | i$ , then  $i = dc$  where  $c$  is an integer less than  $\sqrt{i}$ ; in fact, we need only check to see if  $i$  is divisible by all primes less than  $\sqrt{i}$  since any composite  $d$  less than  $\sqrt{i}$  is the product of smaller primes.

### Section 1.3

1. If  $n > m$  the first argument increases at the first step and then decreases. **3. a,b)** In Algorithm 1.8, EUCLID-RECFUNC, and Algorithm 1.9, EUCLID-REC PRO, replace the **else** line with, respectively,

```
else rem ←  $i - j \lfloor 1/j \rfloor$  else rem ←  $i - j \lfloor 1/j \rfloor$ 
  if rem >  $j/2$  then rem ←  $j - rem$    if rem >  $j/2$  then rem ←  $j - rem$ 
  gcdf ← gcdf( $j, rem$ )   gcdp( $j, rem; result$ )
```

**4. b)** After “ $\text{gcd}(m, n)$ ” add “ $\text{lcm} \leftarrow mn/\text{answer}$ ”; you should also add a check that neither  $m$  nor  $n$  is zero.

**6. a)** Define both(**in**  $i, j$ ; **out**  $gcd, lcm$ ) as follows: **if**  $j = 0$  **then**  $gcd \leftarrow i$ ;  $lcm \leftarrow i$   
**else** both( $j, rem; gcd, lcm$ )  
 $lcm \leftarrow (i/rem)lcm$

**endif**

**b)** Functions as we have defined them can only return one value, so the answer is No unless we allow that value to be an ordered pair — and why not? **7. a)**  $1 \rightarrow 2$ ;  $1 \rightarrow 3$ ;  $2 \rightarrow 3$     **9. a)**  $1 \rightarrow 2$ ;  $1 \rightarrow 3$ ;  $1 \rightarrow 4$ ;  $3 \rightarrow 4$ ;  $2 \rightarrow 4$

**11.** Tail end: Move smallest disk to second pole, then move other  $n - 1$  disks. Front end: Do  $(n-1)$ st case, then move bottom ring to second pole.

### Section 1.4

**2. a)** **function** INT( $x$ )

```
  if  $x \geq 0$  then INT ←  $\lfloor x \rfloor$ 
  else INT ←  $- \lfloor -x \rfloor$ 
endfunc
```

**7.** Algorithm REVERSE

**procedure** Switch(**inout**  $x, y$ )

```
  temp ←  $x$ 
   $x \leftarrow y$ 
   $y \leftarrow temp$ 
```

**endpro**

```
for  $i = 1$  to  $\lfloor n/2 \rfloor$ 
  Switch( $a_i, a_{n+1-i}$ )
endfor
```

**11.** Where is  $prod$  initialized? **13.** This “cheating” procedure stacks rings on an intermediate pole in reverse order and then restacks them on a final pole in correct order. To make it iterative, use two for-loops. **14. a)** Each level calls the next level once and not twice. **15.** Biggest problem is that Max( $m$ ) is a procedure, so that statement “**if**  $a_m > \text{Max}(m-1)$ ” makes no sense; other problem is that many non-maximum values are printed. Better algorithm, with **in** and **out** variables: See Procedure Max in next set of algorithms below. **17.** SAMPLE1 prints four times. From within call Sum(4 :  $S$ ) it prints “4, 16, 10”. From the first line “print  $S, k, m, n$ ” it prints “10, 2, 3, 4” (not 10, 4, 3, 16). From within the call Sum(3 :  $S$ ) it prints “3, 9, 6”. From the second line “print  $S, k, m, n$ ” it prints “10, 2, 3, 4” again (not 6 something, because  $S$  in the second call has been associated with  $T$  at the main level). **19. b)**  $p_m \leftarrow 2$ ;  $q_m \leftarrow 3$ ;  $x_i \leftarrow p_m = 2$ ;  $y_i \leftarrow q_m = 3$ ;  $p_i$  becomes a synonym for  $p_m$ , same for  $q_i$  and  $q_m$ ;  $p_m = p_i \leftarrow 2^3 = 8$ ;  $q_m = q_i \leftarrow 3^2 = 9$ . ( $q_i \neq 3^8$  because  $x_i$  does not change as  $p_m = p_i$  changes within the call;  $x_i$  is an **in** variable.) So when the call to Powers ends, main algorithm is left with two variables,  $p = 8$ ,  $q = 9$ .

**20. b)**  $p_m \leftarrow 2$ ;  $q_m \leftarrow 3$ ;  $p_i$  made a synonym for  $p_m$ , ditto for  $q_i$  and  $q_m$ ;  $x_i$  made a synonym for  $p_m$ , ditto for  $y_i$  and  $q_m$ ;  $p_i (= p_m = x_i) \leftarrow 2^3 = 8$ ;  $q_i (= q_m = y_i) \leftarrow 3^8$ . So when the call to Powers2 ends, main algorithm is left with two variables,  $p = 8$ ,  $q = 3^8$ .

**4.**  $k = 3$ ,  $S = 14$ . In general,  $S$  is the first sum of squares  $> N$ , and  $k$  is the number of squares.

**6.** Instead of one line you have  
 $\text{fpro}(x; r_1)$   
 $\text{fpro}(3; r_2)$   
 $answer \leftarrow r_1 + r_2$

**9.** A big difference: the counting is backwards!

**10. b)** **function** MR( $p, q$ )

```
  if  $q = 0$  then MR ← 0
  else MR ←  $p + \text{MR}(p, q-1)$ 
endfunc
  prod ← MR( $x, y$ )
```

```

procedure Max(in m; out big)
    if m = 1 then big ← a1
        else Max(m-1; big)
            if am > big then big ← am
    endif
endpro
Max(n)
print big

```

22. a) Iterative:

```

F ← 1
for k = 2 to n
    F ← F × k
endfor

```

Recursive:

```

function Fact(k)
    if k = 1 then Fact ← 1
    else Fact ← k × Fact(k - 1)
F ← Fact(n)

```

c) Iterative:

```

S ← 1
for k = 2 to n
    S ← S + k
endfor
Recursive:
function T(k)
    if k = 1 then T ← 1
    else T ← k + T(k - 1)
endfunc
S ← T(n)

```

23. b) Iterative algorithm:

```

prod ← a1
for k = 2 to n
    prod ← prod × ak
endfor

```

Recursive algorithm:

```

function Prod(k)
    if k = 1 then Prod ← a1
    else Prod ← ak × Prod(k - 1)
endfunc
P ← Prod(n)

```

25. Assume vertices are labeled  $S = v_0, v_1, \dots, v_n = F$ . Then

a) for  $j = 1$  to  $n$

$$T(j) \leftarrow d(j) + \max_{i \rightarrow j} \{T(i)\}$$

endfor

b) function  $T(j)$

$$\text{if } k = 0 \text{ then } T \leftarrow 0$$

$$\text{else } T \leftarrow d(j) + \max_{i \rightarrow j} \{T(i)\}$$

endfunc  
 $T(n)$

26. b) Initialize  $words$  to 0 in main algorithm and then have single statement in processcharacter: if  $s_{i+1} = \emptyset$  then  $words \leftarrow words + 1$ .

## Section 1.5

1. a) A collection of sequences is, say,  $\text{Ord}(n)$ , if each of them is  $\text{Ord}(n)$ . Likewise, the collection is  $O(n)$  if each of them is  $O(n)$ . 2. a)  $n+1$  additions;  $n(n+1)/2$  multiplications 3. b) Best case is 1 when first coin is bad.

c) Worst case is 10, when last coin is bad. 5. a) Best case is 1, when  $N$  is odd. c) 1 division: odds; 2 divisions:  $n$  such that  $2|n$ ,  $4\nmid n$ ; etc. Av =  $\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \dots + \frac{1}{1024} \cdot 10 + \frac{1}{1024} \cdot 11 \approx 1.999$ .

7. b) for  $i = 1$  to  $m$

```

ci ← 0
for j = 1 to n
    ci ← ci + aij bj
endfor

```

d) For b) algorithm: Best, worst and average cases same:  $mn$ .

8. a) Line 1: Replace  $x_1$  by  $x_n$ ; Line 2:  $n-1$  **downto** 1 b) Initialize  $msub$  to 1 and update it whenever  $m$  is changed. 10. for  $n = 3$ : 5/6. 14. Here's part of the table: With  $m = 8$  and  $n = 1$  to 8: 1, 1, 3, 1, 4, 2, 2, 1 16. For each pass through the repeat-loop, either  $rem \leq denom/2$  from the line  $rem \leftarrow num - (quot \times denom)$ , or else  $rem < denom/2$  after  $rem \leftarrow denom - rem$ . We need to know how tight the worst case bounds for EUCLID and EUCLID1 are before we can conclude that the algorithm with the smaller bound is actually faster. 19. a) Don't do tests for  $w < w_i$  or  $w > w_i$ . c)  $n$  24. a) Correct 25. a) 2, 2, 3, 3, — not enough data to make a conjecture.

## Supplementary Problems: Chapter 1

1. A straightforward algorithm would just use the quadratic formula to calculate both roots; however, doing this  $x$  is undefined if  $a=0$ , yet equations of the form  $bx+c=0$  do have roots. Moreover, the quadratic formula often leads to the subtraction of two nearly equal quantities for one of the roots which is bad practice. A better algorithm is:

```
if  $a=0$  then if  $b=0$  then if  $c=0$  then print 'all  $x$  are roots' [case-if; linear case]
else print 'no roots'
```

```
endif
else  $x \leftarrow -c/b$ 
```

```
endif
```

```
 $b^2 < 4ac$  then  $x_1 \leftarrow (-b + i\sqrt{4ac - b^2})/2a$  [Complex roots]
 $x_2 \leftarrow (-b - i\sqrt{4ac - b^2})/2a$ 
```

```
else if  $b \leq 0$  then  $x_1 \leftarrow (-b + \sqrt{b^2 - 4ac})/2a$  [Real roots]
 $x_2 \leftarrow c/ax_1$ 
```

```
else  $x_1 \leftarrow (-b - \sqrt{b^2 - 4ac})/2a$ 
 $x_2 \leftarrow c/ax_1$ 
```

```
endif
```

2. d) function Dval( $k$ )

```
if  $k=0$  then Dval  $\leftarrow a_n$ 
else Dval  $\leftarrow Dval(k-1) \times b + a_{n-k}$ 
```

```
endfunc
```

```
 $D \leftarrow Dval(n)$ 
```

3. b) procedure Convert( $m, i$ )

```
if  $i < 0$  then stop
```

```
else  $v \leftarrow \lfloor m/b^i \rfloor$ 
 $m \leftarrow m - v \times b^i$ 
 $a_i \leftarrow v$ 
Convert( $m, i-1$ )
```

```
endif
```

```
return  $a_i, a_{i-1}, \dots, a_0$ 
```

```
endpro
```

```
 $I \leftarrow \lfloor \log_b N \rfloor$ 
```

```
Convert( $N, I$ )
```

3. Continued: For Convert to work,  $a_0, \dots$  must be set up as Input (global) variables or they must be listed as out variables within Convert.

4. a) Exactly the same sequence of rem values. 5. Best for both: all entries the same. Worst for both: all distinct. Both have same order, but StripDup1 generally takes twice as long.

6. a) K1: 20, 4; K2: 15, 14; K3: 9, 8, 7, 3; item of size 2 won't fit d) K1: 14, 8, 3; K2: 20, 7, 2; K3: 15, 9; item of size 4 won't fit

7. Pick the worker with the highest rating for any of the jobs and assign that worker to that job. Then of the remaining workers and jobs, find the worker who has the highest rating, and so on until everyone is assigned;

for data given, worker 1 gets task 1 and worker 2 gets task 2, a very poor result.

9. a)  $(\sqrt{5}-1)/2$  b) .38

10. b) 24 divisions, when  $N = 97$ . (24 is the number of primes  $< 97$ .)

12. Best  $b-1$ , worst  $2(b-1)$ , average  $(3/2)(b-1)$ .

## Chapter 2

Note: For induction proofs, we usually just sketch the key step, typically the breakdown of the next case (usually  $P(n+1)$ ) to obtain a previous case or cases (usually  $P(n)$ ). Proofs submitted for grading should be complete and should follow the format in the text.

### Section 2.2

1. Prove:  $1+3+\dots+(2n-1)=n^2$ . Inductive step:  $\sum_{k=1}^{n+1} (2k-1) = \sum_{k=1}^n (2k-1) + [2(n+1)-1] = n^2 + [2(n+1)-1] = (n+1)^2$ .

3.  $u_n = (2n-1)^2$ . Key step:  $(2n-1)^2 + 8n = [2(n+1)-1]^2$ .

6.  $\sum_{k=1}^n [a + (k-1)d] = \sum_{k=1}^n a + d \sum_{k=1}^n (k-1) = na + d \sum_{j=0}^{n-1} j$ .

8.  $(a_{n+2}-a_{n+1}) + (a_{n+1}-a_1) = a_{(n+1)+1} - a_1$ .

11.  $\prod_{n=2}^m = \frac{m+1}{2m}$ . Key step of  $P(m-1)$  to  $P(m)$ :  $\frac{m}{2(m-1)} \cdot \frac{m^2-1}{m^2} = \frac{m+1}{2m}$ .

15. The right triangle with base  $\sqrt{n}$  and height 1 has hypotenuse  $\sqrt{n+1}$ .

- 17.** Basis is  $n = 2$ . Also,  $(1+x)^{n+1} = (1+x)^n(1+x) > (1+nx)(1+x)$ .  
 be:  $x_n^2 > n$ . For the inductive step, expand  $x_{n+1}^2 = \left(x_n + \frac{1}{x_n}\right)^2$ .  
**23.** If  $x_n = \sqrt{1+x_{n-1}}$  were rational, then  $x_{n-1} = x_n^2 - 1$  would be rational.  
**25.** The difference between two consecutive sums is  $(n+3)^3 - n^3 = 9n^2 + 27n + 27$ .  
**27.**  $\frac{a^{n+1} - b^{n+1}}{a - b} = a^n + \frac{b(a^n - b^n)}{a - b}$ .  
**28.**  $2^n$ ; first show that the number of sets doubles when a new element  $e$  is allowed — for each old set, include  $e$  or don't.  
**30.** What if the man in back saw only white hats?  
**31. a)** 31254  
 (many others too)  
**b)** 25341  
**33.** Every game is a tie.  
**34.** Put an arrow from  $i$  to  $j \iff i$  wins the series with  $j$ . For each pair  $i, j$  there is still exactly one arrow between them, so the proof of Theorem 2 still holds.  
**36.** Given such a tournament with  $2n-1$  teams, arbitrarily let  $A$  be any  $n$  of the teams and  $B$  the remaining  $n-1$ . Add new teams  $c$  and  $d$ . Have  $c$  beat all in  $A$  and lose to all in  $B$ ; have  $d$  do the opposite. Between  $c$  and  $d$  who wins?  
**39.** In the inductive step of b), temporarily delete one player  $t$ ; the rest must be linear. Now show that no team  $t$  beats a team  $t$  loses to.  
**45.** In the  $N$  camper case, if you are the first to get a piece, you get exactly  $1/N$  as you see it. If not, there is  $\geq (N-1)/N$  left as you see it, and by induction you will get at least  $1/(N-1)$  of that.

### Section 2.3

- 1. a)**  $\left(\sum^{n+1} x_i\right)/m = \left(\sum^n x_i + x_{n+1}\right)/m \stackrel{P(2)}{=} \left(\sum^n x_i\right)/m + (x_{n+1}/m) \stackrel{P(n)}{=} \sum^n (x_i/m) + (x_{n+1}/m) = \sum^{n+1} (x_i/m)$ .
- c)** Write  $\sum^{n+1} a_i$  as  $\sum^n a_i + a_{n+1}$ , use  $P(2)$  then  $P(n)$ .  
**3.** If  $s_i$  is the length of side  $i$ , then  $s_i < \sum_{j \neq i} s_j$ . For  $n+1 > 3$ , one approach is to slice off two adjacent sides (not including side  $i$ ) and use  $P(n)$ , then  $P(3)$ .  
**4.** Write  $\prod^n a_i$  as  $(\prod^{n-1} a_i)a_n$ .  
**7.** To prove the left inequality, first write  $x = a + \epsilon$  where  $a = \lfloor x \rfloor$  and  $0 \leq \epsilon < 1$ . Likewise,  $y = b + \delta$ . For  $\lfloor x+y \rfloor \leq \lfloor x \rfloor + \lfloor y \rfloor + 1$ , note that  $\epsilon + \delta < 2$ . Generalization:  $\sum^n \lfloor x_i \rfloor \leq \lfloor \sum^n x_i \rfloor \leq \sum^n \lfloor x_i \rfloor + (n-1)$ . To prove it, write  $\sum^{n+1} \lfloor x_i \rfloor$  as  $\sum^n \lfloor x_i \rfloor + \lfloor x_{n+1} \rfloor$ .  
**9.** Write  $\bigcup^{n+1} B_i$  as  $(\bigcup^n B_i) \cup B_{n+1}$ , use  $P(2)$  then  $P(n)$ .  
**11.** Generalization:  $\bigcup A_i = \bigcap \overline{A_i}$ . Use  $\bigcup^{n+1} A_i = (\bigcup^n A_i) \cup A_{n+1}$ .  
**17.** Prove  $P(11)$  and  $P(12)$  ( $P(8)$  to  $P(10)$  are in the text), then prove  $P(n-5) \implies P(n)$ .  
**18.**  $P(1)$  and  $P(2)$   
**20. a)** If  $a_k, a_{k+1}$  are known to be integers, then the proof in Example 3 shows that  $a_n \in N$  for  $n \geq k$ . For  $n < k$  do downward induction:  $[P(n+1), P(n)] \implies P(n-1)$  because  $a_{n-1} = a_{n+1} - 2a_n$  is an integer if  $a_{n+1}$  and  $a_n$  are.  
**21.** Let  $\{b_n\}$  be the new sequence. Let  $P(n)$  be:  $b_n - a_n > b_{n-1} - a_{n-1} > 0$ .  
**23.** By strong induction you can assume that  $a_{\lfloor (n+1)/2 \rfloor} \geq a_{\lfloor n/2 \rfloor}$  and  $a_{\lceil (n+1)/2 \rceil} \geq a_{\lceil n/2 \rceil}$ .  
**24.**  $2^{n-2}$  for  $n \geq 2$ . For the inductive step, note that each strictly increasing sequence from 1 to  $n+1$  either includes  $n$  or it doesn't.  
**27.** Suppose  $t$  won (or tied for winning) the most games. If  $t'$  beat  $t$ , then  $t$  beat some  $t''$  that beat  $t'$ , for otherwise  $t'$  won more games than  $t$ . (Why and so what?)  
**29.** For  $n > 1$  do induction using formulas for  $\sin(\alpha+\beta)$  and  $\cos(\alpha+\beta)$  with  $\alpha = n\theta$  and  $\beta = \theta$ . What about  $n \leq 0$ ?

### Section 2.4

- 1.** For  $n+1$  rings there are  $2^n - 1$  moves from the call  $\text{TOH}(n, r, 6-r-s)$ , then 1 move, then  $2^n - 1$  moves from  $\text{TOH}(n, 6-r-s, s)$ .  
**2.** Any solution to the  $n$ -ring game must solve the  $(n-1)$ -ring game at least twice, once before moving the  $n$ th ring the first time and once after moving it the last time.  
**5. b)** For  $n > 1$  the call of  $\text{STOH}(n, r, s)$  is  $\text{STOH}(n-1, r, s); \text{robot}(r \rightarrow 2); \text{STOH}(n-1, s, r); \text{robot}(2 \rightarrow s); \text{STOH}(n-1, r, s)$ .  
**e)** Key idea: The  $(n-1)$ -tower must be on the goal pole when ring  $n$  first moves, and must be on the start pole when ring  $n$  moves to the goal pole.  
**7.** For an  $n$ -clutter, assemble the top  $n-1$  rings into a single pile by induction. If this pile is not already over ring  $n$ , move it over using [5] or [6b].  
**9.**  $n=0$  doesn't work because it is below the base case,  $n=1$ . HANOI works for  $n \in N^+$ .  
**11. a)** It puts the tower upsidedown on pole  $s$ . Let  $P(n)$  assert that TOH? moves the top  $n$  rings on pole  $r$  onto whatever is on pole  $s$  in reverse order.  
**13. b)** Induction on  $d$ , with basis  $d=0$ .  
**14.** In part a-i), one obtains 45, 26, 7, 2, 1, 0 for  $\gcd = 1$ . To prove the algorithm correct, the key is that  $\gcd(j, \text{roundrem}) = \gcd(i, j)$  because  $\text{roundrem} = \pm(i - kj)$ .  
**16. a)** For part i, the sequence is 45, 26, 7, 2, 1, 0.  
**17.** The key line in procedure  $\text{gcdp4}(\text{in } i, j; \text{out result})$  is  $\text{else gcdp4}(i, \lceil i/j \rceil j - i; \text{result})$ . For correctness, do induction on  $j$  that  $\text{result}$  returns  $\gcd(i, j)$ .  
**19.** By strong induction. If  $k \mid n$ , the factors (with multiplicities) of  $n$  are those of  $k$  and  $k/n$ . Also, if  $n = kj$  and  $k \leq j$ , then  $k \leq \sqrt{n}$ .  
**22.** In the inductive step, by  $P(q-1)$  the subcall returns  $\text{prod} = p(q-1)$ . Thus the line  $\text{prod} \leftarrow p + \text{prod}$  results in  $\text{prod} = pq$ .

## Section 2.5

2. Basis  $P(1)$ : add 1 once.  $P(n) \implies P(n+1)$ : get to  $n$  by  $P(n)$ , then add 1 once more. 4. The invariant is that, on entry to the loop,  $S$  is the sum of the integers from 0 to the loop-counter. 7. No. Yes, because no matter what word it checks, the remaining lists on either side will be shorter than than the previous list.
8. a) For the inductive step, if  $m, n$  will cause termination in  $k$  steps, then after one iteration the current values of  $num, denom$  (which have the same gcd) will cause termination in  $k - 1$  steps. b) Yes, termination 11. Let  $a_n = 2 + 2(\frac{1}{2})^n$ . Let  $P(n) = "a=a_n \text{ after the } n\text{th trip through the loop}"$ . Key step:  $\frac{1}{2}([2+2(\frac{1}{2})^n]+2)=2+2(\frac{1}{2})^{n+1}$ .
13.  $Q(n)$  asserts that the **repeat while** line is looked at  $n$  times and that  $a \neq 2$  the  $n$ th time. Thus, at the end of the  $n$ th pass, new  $a = \frac{a+2}{2} \neq 2$ , so the loop is also entered an  $(n+1)$ st time. 15. Termination iff  $a = b$  originally. 17. Mimic the proof in Example 5, replacing references to loops and additions by references to calls. 19.  $rem$  is cut at least in half with each iteration. By induction similar to that in Example 5, there are at most  $1 + \log_2 n$  iterations. 20. a) Clearly, any application of BINSEARCH that terminates finding the word would not end sooner if the word was not there. c)  $W_n = 1 + \max\{W_{\lfloor(n-1)/2\rfloor}, W_{\lceil(n-1)/2\rceil}\}$ . 23. A good loop invariant is: “The  $i$ th time the search loop is entered, either  $w$  is one of the words not yet considered or else  $w$  is not on the list at all.” A complete proof requires care. 25. a) After the first snap, break the two pieces into  $i$  and  $k-i$  pieces respectively. b) Each snap creates one new piece, keeping the number of pieces one ahead. Since at termination there must be  $k$  pieces, there were  $k-1$  snaps. 27. b)  $C_n = 1 + C_{\lceil n/2 \rceil} + C_{\lfloor n/2 \rfloor}$ . c)  $C_n = n-1$ . In the inductive step, the recurrence simplifies to  $C_n = 1 + (\lceil n/2 \rceil - 1) + (\lfloor n/2 \rfloor - 1) = \lceil n/2 \rceil + \lfloor n/2 \rfloor - 1 = n-1$ . 29.  $L(n) = \lceil \log_2 n \rceil$ . Prove with strong induction.

## Section 2.6

2. Easiest way would be to check the formula for  $n = 3$ , but if you tried to prove it first, you would find that  $(n^2-n+1)+(n+1) \neq (n+1)^2 - (n+1)+1$ . 4. a)  $(2n^3+3n^2+n)/6$ . Prove by induction. c)  $9n/(n+1)$  6. Sum is  $(n+1)! - 1$ . Prove by induction. 9. Move 1st ring to goal pole iff  $n$  is odd. 12.  $F(n, m) = nm$ . 14. 4 regions (3 lines parallel) or 6 regions (2 lines parallel or all concurrent). 16. a)  $r_0 = 1$  b) Yes c) Basis case slightly easier. 18. Two colors for any  $n \geq 1$ . For an inductive proof, ignore line  $n+1$ , 2-color the rest by assumption, then reverse the colors on one side of line  $n+1$ . 20.  $n^2$  segments; proof by induction or noting there are  $n$  segments per line. 23. Even ignoring tangency and concurrency, ovals can intersect in 2 or 4 points, creating different numbers of regions. 25. b) No three chords concurrent inside the circle. c)  $A_2 = 2, A_3 = 4, A_4 = 8, A_5 = 16$ . 26. a) Write  $a_{n+1}/a_n$  as  $b(\frac{1+c_{n+1}/(Ab^{n+1})}{1+c_n/(Ab^n)})$ . d) This sequence is essentially exponential with  $b = 3$  and  $A = 1$  (if  $1/2$  is called  $a_0$ ). 27. a)  $2^n + n$  c)  $\frac{1}{2}[3^n + (-1)^n]$  d)  $n^3 - 3n^2 + 2n = n(n-1)(n-2)$ . 29.  $L(n) = \lceil \log_2 n \rceil$ . 31. b) A triangulated  $n$ -gon always has  $n-3$  chords and  $n-2$  triangles. c) Strong induction; the first chord creates an  $r$ -gon and a  $(n+2-r)$ -gon with  $r \geq 3$ .

## Section 2.7

1.  $0! = 1; (n+1)! = (n+1)n!$  for  $n > 0$ . 3. Key line: **if**  $k = 0$  **then**  $sum \leftarrow 0$  **else**  $sum \leftarrow \text{Sum}(k-1) + a_k$ . 6. Initially  $prod \leftarrow 1$  (see solution for [7]). The loop body is  $prod \leftarrow prod * a_k$ . 7.  $\prod_{k=1}^0 a_k$  is the empty product that the algorithms in [5, refprob:algforprod] face initially. This product must be set to 1 for the algorithms to work, so generalize and define it to be 1 wherever. 9. Because it's an empty product. 13. The smallest set  $E$  such that i)  $0 \in E$ , and ii)  $n \in E \implies n+2 \in E$ . 15. Letting  $*$  be any roman letter and  $\beta$  be a single blank, the basis strings are  $*$ ,  $**$ , and  $*\beta*$ , and the production rule is from string  $A$  to produce  $*A*$  and  $*\beta A \beta*$ . 18.  $U = \{am+bm > 0 \mid a, b \in N\}$ . When  $m = 7, n = 11, U = \{7, 11, 14, 21, 22, \dots, 58, 60, 61, \dots\}$ . Can you explain why every integer  $\geq 60$  is in  $U$ ? 20. Substitute  $n = 0$  in ii), then use rules of addition and i) to obtain  $1 \cdot m = m$ . Now substitute  $n = 1$  into ii) and then  $n = 2$ . 22. Weak induction applies if  $P(0)$  is true and  $P(n) \implies P(n+1)$ . So by the definition of  $S$ ,  $0 \in S$  and  $n \in S \implies (n+1) \in S$ . We conclude  $N \subset S$ , so  $P(x)$  is true at least for  $x \in N$ . 23. a)  $S_k = \{k\}$  c) The alternative shows how to construct items in the set from earlier items, as with recursively defined sequences of numbers.

## Section 2.8

1. Assertion false. By assumption,  $r$  and  $s$  are either squares or primes. 3. Inductive step fine, but the basis

wasn't shown and is false. **5.** Assertion false. To prove  $P(n)$ , case  $P(n-2)$  was used, so two basis cases ( $x_3 < x_2$  and  $x_4 < x_3$ ) are needed; but  $x_4 \not< x_3$ . **6.** Assertion false (hardly obvious; smallest counterexample when  $n=6$ ), breakdown error in proof. When  $t^*$  is removed, the remaining tournament may have a total winner or loser, so  $P(n)$  need not apply. **9.** Assertion true but proof invalid: Side 1 may not be the longest in  $P'$ , so  $P(n)$  need not apply. Several easy fixes; e.g., strengthen  $P(n)$  by replacing "the longest" by "any". **11.** Assertion true, proof passable, but buildup approach from  $n$ -long sequences to  $(n+1)$ -long sequences of dubious merit. Either argue that all allowed  $(n+1)$ -long sequences are obtained by this buildup, or start with  $(n+1)$ st case and break down. **13.** Assertion false. In proof,  $P(n) \implies P(n+1)$  valid only for  $n > 0$  (where  $n \in M \implies n=n+1$ ) and the case  $n=0$  is needed. **15.** Assertion false, starting at  $n=4$ . At best the inductive step would show that there is only one triangulation using the initial chord you draw in, but there are multiple choices for that first chord. **17.** Assertion can be false for all odd  $n$ . Since the inductive step uses  $P(n-2) \implies P(n)$ , two basis cases are needed. **19.** Because we proved that the top-level "factorization" of any minimal TOH sequence (( $n-1$ )-game, single move, ( $n-1$ )-game) is unique. Then induction really does prove that the complete factorization is unique. **21.** All even-length palindromes missed. **23.** Assume  $Q(n+1)$  and break it down to obtain  $Q(n)$ . Now  $R(n)$  follows from assuming  $P(n)$ . Work back to  $R(n+1)$ . **24.** This  $P(2k+1)$  does not say that all  $(2k+1)$ -tournaments have a certain property, but only that *one*  $(2k+1)$ -tournament does. Buildup validly produces one item. **26.** Neither proof is correct, though the claim in a) is true. Both proofs require, and fail to give, two basis cases. The inductive step in both is fine.

## Supplementary Problems: Chapter 2

- When you can prove all the cases at once, go for it; but often you can't, especially when the cases grow in complexity (e.g., longer and longer sums). **3.** Can be won. For  $n+1$  rings, start by invoking case  $n$  twice. **5. a)**  $f(x)=cx$  for any constant  $c$ . **d)** Strong induction on the number of summands. **7.** Inductive step breaks down for  $n=3$ ; one of  $T$  and  $T'$  will not contain 2 known blondes. **9.** Strengthen the inductive hypothesis to  $P(n)$ : "Buffalo buffalo buffalo (that buffalo buffalo)" is a proper sentence *and* the last "buffalo" is a noun. Then adding "that ..." at the end is proper. **11.** Tough! Maximum total overhang when the  $i$ th brick (counting from the top) hangs distance  $1/2i$  over the brick below it. Just to prove this configuration (call it  $C$ ) stable takes induction. To prove optimality, strength the hypothesis to  $P(n)$ : With  $n$  bricks,  $C$  is optimal and, for any  $n$ -brick configuration  $C'$  with  $d$  less total overhang, the center of gravity of  $C'$  is  $< d$  farther to the right than in  $C$ . **13.** Don't try induction on "the  $n$ -game can be won" but rather on "Stage 1 of the  $n$ -game can be done", where Stage 1 gets the pieces to BXXO-XO (B = blank) or via mirror images to XO-XOXOB. Stage 1 can be accomplished recursively — play the inner  $(n-1)$ -game to its Stage 1 first. After Stage 1 of the  $n$ -game is done, a brief Stage 2 gets you to OXXO-OXB (or its mirror image). To finish and win, run the mirror image of Stage 1 backwards. **15. a)** One approach:  $\frac{1}{2n} \sum_{i=1}^{2n} a_i = \frac{1}{2}(b_1+b_2)$ , where  $b_1 = \frac{1}{n} \sum_{i=1}^n a_i$  and  $b_2 = \frac{1}{n} \sum_{i=n+1}^{2n} a_i$ . Now apply  $P(2)$  and then  $P(n)$  twice. **17.** Start with the Triangle Inequality, square, expand, subtract common terms, square again, and you get the Cauchy-Schwarz Inequality. Now verify that all steps reverse. **18.** Write  $\prod_{i=1}^{n+1} f_i$  as  $(\prod_{i=1}^n f_i) f_{n+1}$ . The proof is a standard generalization by strong induction, but the notation is a bit tricky. **20. a)**  $\sum_{i=0}^m \frac{1}{2i+1} = \sum_{k=1}^{2m+1} \frac{1}{k} - \sum_{k \text{ even}} \frac{1}{k}$ . **c)** By induction using  $H(2^{m+1}) = H(2^m) + (2^m \text{ terms all at least } 1/2^{m+1})$ . **21. a)** Key idea: Suppose  $u_m+d_n < C$ . Then  $u_i+d_n < C$  for all  $i$  since  $u_i \leq u_m$ ; so forget about  $d_n$ . Similarly, if  $u_m+d_n > C$ , forget  $u_m$ . **b)** Let  $P(k, l)$  be: If the repeat loop is ever entered with  $i=k$ ,  $j=l$ , then SUM-SEARCH correctly determines if  $C = u_i+d_j$  for any  $i \leq k$  and  $j \leq l$  and otherwise returns  $(0,0)$ . **23. a)** To show that  $\gcd(i, j) = ai + bj$ , in the inductive step use that  $\gcd(j, rem) = a'j + b'(rem)$ . **24.** Use that  $\gcd(m, n) = am + bn$  for some  $a, b \in \mathbb{Z}$ . **29.** Invalid. Let  $P(x)$  be the statement " $x \leq 1$ ". Then i) and ii) are true, but  $P(x)$  is false for all  $x > 1$ . Remember, the implication in (ii) is true whenever the premise is false.

## Chapter 3

### Section 3.1

- Can't even start; no trip with  $F$  gives legal group on near shore **4.** 12 legitimate states of  $H_1, H_2, W_1, W_2$ ,

**B**(oat) on near side of river; drawing graph and connecting states reachable from each other gives several solutions in 5 trips   **6.** Only three possible states after first move   **9.** Traverse edges 1, 11–14, 10, 2–9, 15–16   **10.** 00010111  
**12. a)** 6 trees; two have  $w_1$  as root with one left and one right branch   **b)** Two trees of height 1, four of height 2; average 5/3.   **13. b)**  $V = \{u, v, x, y\}$ ;  $E = \{\{u, v\}, \{u, x\}, \{u, y\}, \{v, x\}, \{v, y\}, \{v, v\}\}$ .   **14. b)**  $V = \{u, v, x, y\}$ ;  $E = \{(v, u), (v, x), (x, u)(x, y), (y, x)\}$ .   **15. a)** There is 1 with 0 edges; 1 with 1 edge, 2 with 2, 3 with 3, 2 with 4, 1 with 5, 1 with 6.   **b)** 1 with 0 edges, 1 with 1, 4 with 2, 10 with 3   **16. a)** Two graphs   **b)** Four graphs, two with edges in both directions between one pair of vertices   **18. a)** Reflexive and transitive, not symmetric  
**b)** Divisibility   **22.** Can't tell which of multiple edges is on path; definition must specify this   **23.** [n/2]  
**26. b)**  $\{v_1, v_2\}, \{v_1, v_4\}, \{v_3, v_2\}, \{v_3, v_6\}, \{v_5, v_2\}, \{v_5, v_4\}, \{v_5, v_6\}$    **27. a)** Two tournaments   **c)** Yes; draw it  
**28. b)** Only trees which consist of a single path from the root to a leaf   **29. b)** If connected and cyclic, each vertex on cycle would have indegree 1 but path from root to vertex on cycle would give that vertex indegree 2; if acyclic and not connected, a vertex in each component would have degree 1; otherwise could construct cycle.  
**30. b)** Adjacent to  $\rightarrow$  incident on;   **32. a)** 8 graphs   **b)** 4 graphs

### Section 3.2

**1. a)**  $\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}$    **b)**  $\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$    **c)**  $\begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 3 \\ 2 & 1 & 0 & 1 \\ 0 & 3 & 1 & 1 \end{bmatrix}$    **d)**  $\begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 \end{bmatrix}$    **3. a)**  $\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$    **c)** Sum of entries in

row  $i$  is degree of vertex  $i$ ; sum of columns is always two.   **d)**  $MM^T = A + D$  where  $A$  is adjacency matrix and  $D$  is a diagonal matrix all of whose entries are zero except on the diagonal where the  $i$ th element is the degree of vertex  $i$ .   **e)** Compute  $MM^T$  and consider diagonal, non-diagonal terms separately; note that  $m_{ik}m_{jk} = 1$  iff  $e_k$  is incident on both  $v_i$  and  $v_j$    **4. a)** Replace 1's by  $-1$ 's for vertices at the tail of an arrow (assuming no loops).

**6.** Build up adjacency matrix by processing edges one at a time   **7. a)**  $A^2 = \begin{bmatrix} 3 & 1 & 1 & 0 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$    **8. a)**  $uvwuw, uwvwu, uvwwu, uwuvw, uwuvw, uxuvw, uvww$    **12. a)**  $A^2$  is a  $4 \times 4$  matrix with 1's in (1,3), (1,4), (3,1) and (4,2) entries and 0's elsewhere.   **b)**  $uvw, vwx, wxu, xuv$    **13. a)**  $uwxu, vxuv, wxuvw, xuvwx$    **15.** For any path between two distinct vertices, remove all cycles; remaining path can at most go through all vertices in which case length is  $n - 1$ ; only from  $v$  to itself might the shortest path be an  $n$ -cycle.   **17. a)** When  $m = 0$ ,  $A^0 = I$ , the identity matrix which gives the one null path for each vertex   **b)** Change lower limit to 0.   **19.** With modification WALSHALL says there is always a path from a vertex to itself; therefore output differs only when such path is null path for any vertex.   **20.** Path matrix is all 1's.   **22.** Matrix is all 1's (even on the diagonal).   **24.** After "for  $i = 1$  to  $n$ ", skip inner loop if row  $i$  is all zeros; after "for  $j = 1$  to  $n$ ", if column  $j$  is all zeros, skip body of inner loop.   **25.** Innermost loop:  $j = i$  to  $n$  and after **then** also set  $p_{ji}$  to  $p_{ij}$ .   **26.** Suppose have shortest path from  $v_i$  to  $v_j$  using only first  $k - 1$  vertices; get next step from inner loop statement **if**  $l_{ik} \neq \infty$  and  $l_{kj} \neq \infty$  **then**  $l_{ij} \leftarrow \min(l_{ij}, l_{ik} + l_{kj})$

**27. Final matrix from modified WARSHALL is:**  $\begin{bmatrix} 4 & 6 & 8 & 2 & 3 & 12 \\ 6 & 4 & 2 & 4 & 3 & 6 \\ 8 & 2 & 4 & 6 & 5 & 4 \\ 2 & 4 & 6 & 2 & 1 & 10 \\ 3 & 3 & 5 & 1 & 2 & 9 \\ 12 & 6 & 4 & 10 & 9 & 8 \end{bmatrix}$  so answer is 12 (element in (1, 6) position);

diagonal elements are twice shortest edge from vertex; why?   **29. b)** Need only start from  $i$  and follow from one vertex to the next until there is edge from current vertex to  $j$    **32.** Negative weights allow possibility of reducing weight by going around cycle; just not true that statement in solution to [26] gives the minimum length for the next vertex.

### Section 3.3

- Two more bridges between distinct pairs of vertices; three choices; one is  $\{v_1, v_3\}, \{v_2, v_4\}$    **2. a)** Order of vertices: 1 2 5 6 2 3 6 7 8 3 4 1   **3.** Impossible; 2 vertices of odd degree; retrace two east-west blocks in center.
- 5. a)** Graph has 3 vertices, 9 edges.   **d)** One is: 0011022120.   **7.** Always a de Bruijn cycle because graph always has Eulerian cycle; why?   **9. a)** Yes; consider graph with 7 vertices 0, 1, ..., 6 and edges corresponding to dominoes (i.e., edge from 2 to 4 for (2, 4) domino; each vertex has degree 8 counting loop   **b)** No; now each vertex has degree 7.

**11.** Same graph as [10] except edges from one test are distinguished from others, say, by being colored red and black; need to find Eulerian cycle with alternating red and black edges and there is one: *AAAACCABCBCB*.  
**13.** Induction on the number  $n$  of vertices. Take any vertex in an  $(n+1)$ -vertex graph and replace each pair of edges at it by an edge between other ends of pair. Results in Euler cycle on  $n$ -vertex graph (or on each connected component). Now restore the  $(n+1)$ st vertex and its edges. **16. a)** All vertices on path found from initial call of Pathgrow have even degree on path and all vertices in original  $G$  have even degree. **b)** If component had no vertex on this cycle, original graph would not have been connected. **18.** Scan list of edges and pick out those that have  $v$  as a vertex. **20. a)** Graph with zero edges consists of a single vertex (recall that the graph is connected) and so Pathgrow trivially returns an empty (and trivially Eulerian) path. **b)** Either  $G$  consists of two loops at a single vertex (in which case Pathgrow finds the Eulerian cycle consisting of these two loops), or  $G$  has two vertices with two edges between them (in which case Pathgrow finds the Eulerian cycle consisting of the two edges).  
**22. a)** Need only consider 2 odd vertices case (why?); assume greater than 2 vertices; otherwise easy; Eulerian path must begin and end at odd-degree vertices; therefore choose  $v$  as in Proof 1 to be even-degree vertex and proceed as in Proof 1; both odd vertices must be in same component. **c)** Get Eulerian cycle on  $G'$ ; delete added edge; then have Eulerian path on  $G$ . **25.**  $G$  must have exactly 4 vertices of odd degree; add two edges to get all even vertices; get Eulerian cycle and then delete added edges. **26.** Traverse Eulerian cycle and direct the edges in the direction on cycle. **28.** In each component add enough edges to  $G$  to give all vertices even degree; traverse Eulerian cycle to give directions; then delete added edges; all outdegrees same as indegrees or differ by 1. **30. b)**  $\dots b_n \rightarrow b_2 \dots b_n c_1 \rightarrow b_3 \dots b_n c_1 c_2$  etc. **32.**  $(n-1)!$  — Start at any  $v$ ;  $n-1$  choices for next vertex;  $n-2$  for next etc.  
**33.** See Section 2.2, Theorem 2. **35. a,b)** Let each digit represent  $x, y, z$  coordinate; get 8 vertices of a cube; Hamiltonian path is 000, 001, 101, 100, 110, 111, 011, 010. **c)** Yes; solution in part b) has this property. **d)** Think recursively; let graph in  $n+1$  case,  $G_{n+1}$ , consist of 2 copies of  $G_n$  with cross edges between corresponding vertices of each; add 0 to all labels of one to get  $G0_n$ , 1 to all labels of the other to get  $G1_n$ ; path goes from edge of  $G0_n$ , to cross edge, to edge of  $G1_n$  that corresponds to next edge on  $G1_n$  path, to cross edge etc. **e)** Yes; prove by induction on  $n$ ;  $P(n)$  is: Graph in  $(n, k)$ -case has Hamiltonian cycle for any  $k$ ; take  $k$  copies of  $(n, k)$  graph, join corresponding labels, add 0 to all labels of one graph, 1 to all labels of next etc.; path consists of  $k$  edges joining successive  $(n, k)$  graphs followed by edge in an  $(n, k)$  graph etc. **36.** What is the sum of the first  $n-1$  integers?  
**38.** There are many groups of 5 integers (but none of 6 integers) between 1 and 25 such that the difference between any two in a group is greater than 4. **39. b)**  $mn$ .

### Section 3.4

**1. a)** Shortest path:  $v_0v_4v_3v_1v_2v_5$  **4.** Shortest path is (with length to each vertex in parens):  $v_5, v_2(4), v_1(6), v_3(9), v_4(10), v_0(12)$ . **7. b)** Shortest path has length 6 — along upper boundary of graph. **9.** Order in which vertices added:  $v_1v_6v_7v_3v_2v_4v_8$ ; path length to  $v_8$ : 11; path is  $v_1, v_6, v_7, v_3, v_2, v_4, v_8$ . **10.** Make bigraph into digraph by replacing each edge with two edges in opposite directions with appropriate weights; then apply DIJKSTRA to digraph.  
**11. a)** For each pair of vertices, make edge weight minimum of multiple values of weights of edges joining pair. **b)** If maximum number of edges joining pair of vertices  $< n$ , complexity still  $n^2$ ; if  $\geq n$  probably still  $n^2$  since finding minimum is easy. **14.** With weights nonnegative any path with cycle at least as long as path without cycle. **16.** DIJKSTRA can fail even to find the shortest simple paths, because once a vertex  $v$  joins  $U_k$ ,  $d(v)$  is never reconsidered. Can you construct an example? **19. a)** Just use  $\text{prev}(v)$  to work back from  $F$  to  $S$ . **b)** Easiest: Find in backwards order and then reverse. To find in forwards order within main algorithm must keep track of shortest path (in forwards order) found to each vertex thus far. **21.** Shortest distance to  $v \in U_k$  is a  $U_k$ -path and is the shortest distance to  $v$  in the whole graph. **23. a)** Apply DIJKSTRA with all weights 1. **c)** Unique; working back from 15, prior vertex always uniquely determined. **25.** For each edge delete it and find the length of the shortest path from the vertex at one end of the deleted edge to the vertex at the other end with all weights 1. **27.** Just use DIJKSTRA with  $+$  replaced by  $\times$  (or just replace weights by their logarithms). Why doesn't this work if weights can be  $< 1$ ? **29.** Need  $n(n-1)$  applications of DIJKSTRA to get all shortest paths so complexity  $n^4$ . However, change DIJKSTRA not to stop when  $F$  joins  $U_k$  and then need only  $n$  applications (for each possible  $S$ ); then complexity  $O(n^3)$ . Weighted WARSHALL is still  $\text{Ord}(n^3)$ . **32.** No; algorithm may fail to find longest path; consider path from vertex 1 to vertex 4 in graph with edges (weights after semi-colon)  $(1, 2; 5), (1, 3; 7), (2, 4; 3)$ ,

(2, 3; 4), (3, 4; 3).

### Section 3.5

3. 6 vertices — length 1; 8 vertices — length 2    4. By contradiction    7. a) Use DFS starting anywhere and test at end to see if all vertices visited.    8. a) 12435689    9. a) Edges from a vertex connect squares attacked by a queen at the vertex square.    10. c) 1213123132.

### Section 3.6

1. Ones with no edges    2. b) 3, 3, 5    3. 3    4. 2    6. 3; chemicals are vertices, edges connect vertices for chemicals that cannot be stored together.    8. 2; no cycles so BIPARTITE succeeds    9. 2 additional colors; use BIPARTITE ignoring red vertices; chromatic number is either 2 or 3    11. Vertices — courses; colors — exam periods; there is an edge  $\{u, v\}$  when at least one student takes both  $u$  and  $v$ .    13. An edge is needed between the Monday morning and the Physics 1 vertices and between the Tuesday evening and PoliSci 3 vertices; why?    15. Claim is correct; if graph 3-partite, a 3-way partition of kind described must exist.    16. b)  $O_3$  is an octahedron.    d)  $n$ ; prove by induction  
18. a) 3 and 5    b) 3; color vertex  $i$  as  $1 + (i - 1) \bmod 3$     21. Color high degree vertices first so don't end up with high degree vertex to color with all its adjacent vertices having different colors.    24. a) Simplest is triangle.    b) Add an edge at any vertex of triangle    c) Triangle with edges from each vertex meeting at common point inside triangle.    27. b) Both algorithms color  $v_{10}$  4 and otherwise color  $v_i$   $1 + (i - 1) \bmod 3$ .  
29. b) 3    d)  $n$  vertices:  $n - 1$  if  $n$  even,  $n$  if  $n$  odd in which case colors of edges incident on each vertex are a different set of  $n - 1$  colors; proof by induction; odd  $n \rightarrow$  even  $n + 1$ : just color each edge from new vertex different from all colors at each vertex of  $n$ -graph; even  $n \rightarrow$  odd  $n + 1$ : this is harder; change one edge at a time in  $n$ -vertex graph to color  $n$  or  $n + 1$     30. a) Planar; draw one diagonal outside square.    d) Planar; draw one diagonal outside each square.    33. Removal of any edge on cycle reduces number of regions by 1 but leaves  $G$  connected.  
34. b) Since  $n = 6, m = 9$ , from Euler  $r = 5$  if graph planar; show that structure of  $K_{33}$  requires that each region be surrounded by at least 4 edges; since each edge is boundary of two regions, would need 10 edges to get 5 regions.  
35. a)  $Q_{n+1}$  found from  $Q_n$  by making two copies of  $Q_n$  and putting an edge between each pair of corresponding vertices    b) 4    36. a) Remember each edge on boundary of 2 regions    b) use Euler's formula [33] and result in a)  
37. Suppose contrary so that  $5n \leq 2m$  and use [36b]    38. Same idea as [37]; at least one vertex of degree 3

### Section 3.7

2. b) (i) Edges:  $\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_5\}, \{v_2, v_6\}, \{v_3, v_4\}, \{v_5, v_7\}$     4. a) Each vertex is on shortest path from root; if cross edge spanned two or more levels, there would be a shorter path to lower vertex using the cross edge  
6. Algorithm will fail on line "Pick a ..." when tree spans initial component.    7. Weights of edges on tree in order added: 1, 8, 2, 3, 5, 6    9. By Theorem 2, MINSPANTREE finds a minimum spanning tree; suppose there were another; then use technique of Lemma 3 to get a contradiction.    12. Weights on edges in order added: 1 2 3 5 6 8.  
14. Just replace "minimum" by "maximum" and "shortest" by "longest" everywhere in algorithm and proofs.  
17. Nothing in proof assures that MST on  $n$  nodes plus minimum edge to node  $n + 1$  gives an MST on  $n + 1$  nodes.  
19. Start with 1 node, 0 edges; at each step add 1 node, 1 edge.    20. [(i), (ii)]  $\Rightarrow$  (iii): Use [19]. [(ii), (iii)]  $\Rightarrow$  (i): If  $G$  not connected (i.e., more than one tree), (iii) will not be satisfied. [(iii), (i)]  $\Rightarrow$  (ii): By [19] spanning tree has all the edges and tree is acyclic.    22.  $N_n H_{n+2}$ ; prove by induction on number of  $N$ 's; for  $n + 1$  case take  $N$  attached to only one other  $N$  (when will there not be one?) and replace it and  $H$ 's attached to it by an  $H$ .  
23. a) Same parent node    c) Both are descendants of node three levels higher.    e) Have common ancestor two levels higher for one, four levels for the other    24. b) There are 14.    26.  $B_n = B_{n-1}B_0 + B_{n-2}B_1 + \dots + B_0B_{n-1}$ ;  $B_3 = 5, B_4 = 14, B_5 = 42$ .    28.  $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1-i} T_i T_j T_{n-i-j-1}$     30. b) 1 2 4 5 8 9 3 6 7 10    31. Preorder: 1 2 5 6 3 4 7 10 11 8 12 9; Postorder: 5 6 2 3 10 11 7 8 12 9 4 1.    33. a) Yes; for every node in Preorder, Inorder tells you whether it is on left or right subtree of parent    34. b)  $W$     36. Tree has 7 nodes including root.  
37. Tree has 16 nodes including root.    39. a) Tree has 12 nodes; first player wins    c) After first move, game is reduced to 3- or 4-petal daisy.

### Supplementary Problems: Chapter 3

2. b) If leaf, just delete; otherwise replace deleted node by maximum value on left subtree or minimum value

on right subtree   **3. a)** Need at least 5 since complete graph of 4 vertices not allowed; but with 5 vertices sum of degrees would be odd   **5. b)** Leave out edge in  $\langle\{1, 2, 3\}\rangle$    **7. a)** By induction; remember  $B + B = B$

**9. a)** No   **b)** 7   **c)** 2   **11. a)** Need to find if allowing  $(k+1)$ st edge gives any new paths; no simple way to determine this from  $M^{(j)}$  matrices,  $j \leq k$    **b)** No simple way to go from  $N^{(k)}$  to  $N^{(k+1)}$    **13. c)** If degree of vertex currently at is one, removal of edge will not increase number of components; otherwise degree must be 2 or more (3 or more if not at original vertex); removal of *any* edge cannot result in new component because, before removal, all vertices in new component must have had even degree (why?) and removal of edge would result in vertex of odd degree at end of removed edge and in new component which is impossible; therefore, never need to increase number of components and so find Eulerian cycle   **15.** Argument in Section 2.2 assures that there is a Hamilton path; for  $k=3$ , use strong connectivity to get “back edge”; for inductive step use strong connectivity to insert one or two vertices into known cycle.   **17.** Columns of all zeros represent vertices with indegree 0; number one of these 1; then delete its row and column and repeat each time numbering a vertex with indegree 0 in reduced graph.

**20.** Doesn't work because shortest edge from start vertex may not go to nearest vertex since negative weight could give smaller weight to multi-edge path. See also [15, Section 3.4].   **21. a)** No shortest path can have cycle since cycle weights are positive; must be a shortest path since number of simple paths is finite.   **b)** DIJKSTRA doesn't work for same reason as in [20].   **23. c)** Same values of distance function   **d)** For Ford argue that, if distances to  $k$  nearest vertices already found, then algorithm next finds shortest distance to  $(k+1)$ st nearest vertex.

**25. a)** Find shortest path to each vertex and sum;   **c)** Find minimum spanning tree.   **27.** The complete graph of order 4 has 4 vertices of degree 3 and  $\chi(G) = 4$ ; for any graph with 3 or fewer vertices of degree 3 or more, color the vertices of degree 3 or more with distinct colors and apply K-COLORABLE to show  $\chi(G) < 4$ .   **31.** Need 31 dominos to cover 62 squares, 31 red, 31 black; but two opposite corner squares have same color; contradiction.

**32. a)** Those of odd length   **b)** Suppose vertex of degree  $n-2$  or less; delete it; if remaining graph is  $(n-1)$ -colorable, so was original graph   **35. a)** There are 3.   **b)** Same weights on each   **c)** Suppose not; then if edges ordered by increasing size, Tree2 has at some point larger edge than Tree1; put all vertices on Tree2 up to this edge in one set, remaining vertices in another set and apply Lemma 3 to get contradiction. See also [9, Section 3.7]

**36. a)** When multiple edges of weight  $w$ , consider graph with new edge weights  $w, w+\epsilon, w+2\epsilon$  etc. all less than next largest weight; apply Prim's algorithm; one such selection of weights must correspond to each minimum weight spanning tree   **b)** Assume weights are integers; using idea in a) no two spanning trees could have weights which differ by as much as 1 if  $\epsilon$  small enough; therefore, without  $\epsilon$  trees must have same weight   **38.** Stronger hypothesis: Free tree always has at least as many nodes of degree 1 as degree of vertex with maximum degree; for graph with  $n+1$  vertices delete vertex of degree one (why does the induction hypothesis assure that there is one?) not adjacent to vertex of maximum degree (if there isn't such a vertex then  $(n+1)$ -graph has a very simple form)

**40.** Proof not OK because suppose removed vertex disconnects the graph; fix by always removing a vertex which does not disconnect the graph; this is always possible (why?); then proof is OK.   **42.** Buildup error; haven't shown arbitrary  $(n+1)$ -tree can be formed this way; instead do inductive step by taking arbitrary  $(n+1)$ -tree and removing a leaf and its edge.   **48. a)**  $+ \times 42 - 3 / \times 45 + 37$    **b)** Associate each operator with two most recently written down operands to form new operand.   **50.** Player 1 wins if  $n$  is not a multiple of 3; Player 2 wins otherwise; if  $n$  is not multiple of 3, Player 1 can leave Player 2 with a multiple of 3; if  $n$  is a multiple of 3, Player 1 must leave Player 2 with a non-multiple of 3.

## Chapter 4

### Section 4.2

- 2. 21   4. 23   6. a)**  $26^3 = 17,576$    **c)**  $21^2 \cdot 5 = 2205$ .   **8. a)**  $9! = 362,880$    **b)**  $20 \cdot 19 \cdot 18 \cdots 12 \approx 6.09 \times 10^{10}$ .
- 10.** Any digit may be 0, so  $10^3 - 1 = 999$ . (Why subtract 1?)   **11.** At most 50, fewer if some students play on both teams.   **13.**  $26^3 \cdot 10^3 = 17,576,000$ . Counting commercial vehicles, this is probably not enough in CA or NY. Allowing digits to be first doubles the count and may be enough.   **15.** Each route after the first takes  $2n+2$  steps. Largest  $n$  is 13.   **17. 21   19. 136**   **20. a)**  $800 - 160 = 640$  exchanges and  $6.4 \times 10^6$  7-digit numbers.   **b)**  $136 \times 6.4 \times 10^6 \approx 870$  million phone numbers, which is enough.   **22. a)** The number of phone numbers should

grow exponentially with  $d$ ; this measure divides  $d$  out, leaving  $\log_{10}$  of the base, which will be between 0 and 1. **b)**  $(\log_{10} 10^7)/7 = 1$ . **d)**  $(\log_{10} 8^d)/d = (d \log_{10} 8)/d = \log_{10} 8$ . **24. c)**  $MN$  takes  $mn$  multiplications and then  $(MN)P$  takes at most  $(m+n)p$ .  $NP$  takes  $np$  multiplications and then  $M(NP)$  takes at most  $m(n+p)$ . The totals are the same:  $mn + mp + np$ . **25. b)**  $A(BC)$  takes  $2n^2$  multiplications,  $(AB)C$  takes  $n^3 + n^2$ , much more.

### Section 4.3

- 1.**  $3^n$  (if one or both can be empty)    **2.**  $3^n - 2 \cdot 2^n + 1$ ; subtract from  $3^n$  the situations where one or the other subcommittee is empty, then add back the one case where they are both empty.    **4.** Use Binary Search. Asker can win even with  $2^{20} > 10^6$  words.    **6.** Two positions per ring, starting with the largest, so  $2^n$     **8.**  $n$  for chairman,  $2^{n-1}$  for members; therefore  $n2^{n-1}$  ways.    **11.**  $9!/(2!4!3!)$     **13.** 5!;  $5!/5 = 4!$     **17.** Start by weighing coins 1,2,3,4,5 against 6,7,8,9,T, where T is the test coin.    **19.** Consider the case  $k+1 = 4$ . The inductive step  $P(k) \Rightarrow P(k+1)$  assumes too much, that the method works when  $k$  coins are a *subset* of the coins.    **21.** First weigh half the coins (all together). Lower bound:  $\lceil \log_2 13 \rceil = 4$ .    **23.**  $2^{\lceil n/2 \rceil}$  symmetric sequences; therefore no. of sequences =  $\frac{1}{2}(2^n - 2^{\lceil n/2 \rceil}) + 2^{\lceil n/2 \rceil} = \frac{1}{2}(2^n + 2^{\lceil n/2 \rceil})$ .    **25. a)** 52<sup>4</sup>    **b)**  $52^4/2^4 = 26^4$     **28.** 0, 1, 6, 8, 9 make sense when flipped. The number of usable signs (first digit nonzero) that flip to a different usable sign is  $4 \cdot 5 \cdot 5 \cdot 4 - 4 \cdot 5 = 380$ , where 4·5 is number of symmetric signs (same sign when flipped). So  $9000 - 190 = 8810$  different signs.    **30.**  $(n+1)(m+1) - (m+n+1) = mn$  eliminations if no coefficients zero.    **32.** 3 ways    **33.** 5 both; 35 field hockey only; 25 track only.    **34. a)**  $|F| = 40$ ,  $|T| = 30$ ,  $|\overline{F} \cap \overline{T}| = 1335$ .    **35. a)** 1291    **c)** 20  
**37.** 48

### Section 4.4

- 1. a)** 5, 20, 60    **b)** 5, 10, 10    **3.** Same    **5.**  $\binom{n}{1} + \binom{n}{2} = n(n+1)/2$ .    **7.**  $\binom{n-2}{k} + 2\binom{n-2}{k-1} = \binom{n}{k} - \binom{n-2}{k-2}$ . (Each side comes from a different approach.)    **9.** 8!,  $C(8, 5)P(8, 5)$     **10.**  $\binom{n}{3}$     **13. c)** LHS counts each intersection point on each line. So each intersection point is counted twice, once for each line that intersects there.    **16.** Inefficient, also overflow possible    **18.**  $\binom{n}{2}$     **19.** Each of the  $\binom{n}{2}$  possible edges is either in the graph or not; therefore  $2^{\binom{n}{2}}$ .    **22.**  $2^{\binom{n}{2}}, 3^{\binom{n}{2}}$ .

### Section 4.5

- 1.** Combinatorial: Choose a captain and  $k$  other team members various ways, one of which is to choose  $k$  members and then the captain.    **3.** Combinatorial: Choose  $k+j$  things from  $n$  various ways; e.g., for the LHS of a), choose  $k$  from  $n$  and then  $j$  from the rest.    **5.** Line up  $k$  out of  $n$  things all at once, or choose the first thing and then worry about the rest.    **8. a)** Adding each diagonal increases the number of polygons by 1, so there is always 1 more polygon.    **b)**  $3t = 2d + n$  (each diagonal gets 2 ticks and each side 1).    **10.** Induction on  $n$ : In the inductive step, rewrite  $\binom{k}{m}$  on the LHS using Theorem 2. Combinatorial: On the LHS, choose  $m+1$  numbers from  $\{1, 2, \dots, n+1\}$  by first deciding the largest to be chosen.    **11.** Second new row: 1, 6, 15, 20, 15, 6, 1.    **13. a)** Define a function  $C(n, k)$  with key line  $C \leftarrow C(n-1, k) + C(n-1, k-1)$ .    **c)**  $\binom{n'}{k'}^*$  (for  $n' \leq n$  and  $0 \neq k' \leq k \neq n'$ ) is computed  $\binom{n-n'}{k-k'}$  times.    **14.** For  $0 < k < p$ , no factor in the denominator of  $\frac{p!}{k!(p-k)!}$  divides the factor  $p$  in the numerator.    **16.** For most cases where  $k < 0$  or  $k > n$ , all terms in the theorems are 0. The cases  $k = 0$  and  $k = n$  in Theorem 2 need special attention because of the terms  $\binom{n-1}{k}$  and  $\binom{n-1}{k-1}$ .    **18.** In proving  $P(n) \Rightarrow P(n+1)$ , rewrite  $k\binom{n+1}{k}$  as  $k\binom{n}{k} + k\binom{n}{k-1} = k\binom{n}{k} + (k-1)\binom{n}{k-1} + \binom{n}{k-1}$ .    **21.** There is a 1-to-1 correspondence of terms by  $\binom{n}{k} \leftrightarrow \binom{n}{n-k}$ . Both sums =  $2^{n-1}$ .

### Section 4.6

- 1. b)**  $x^4 - 4x^3 + 6x^2 - 4x + 1$     **2. b)** .9412    **3.** LHS =  $(1+2)^n$ .    **4. b)**  $(-3)^n$     **e)**  $(-4)^n$     **5.** 12.68%    **7. a)** For any set, the number of odd-sized subsets equals the number of even-sized subsets.    **b)** It equals  $\frac{1}{2}2^n$ .    **9.** Expand the LHS and look at the signs of the “additional” terms.    **11.** Discovery:  $\sqrt{a^2 + \epsilon} = (a^2 + \epsilon)^{1/2} = (a^2)^{1/2} + \frac{1}{2}(a^2)^{-1/2}\epsilon + \dots$     **13.** When  $j = 0$  there is no  $y^{j-1}$  term in the first factor in the RHS of Eq. (5); when  $j = n+1$  there is no  $y^j$  term.    **15.** 30, 60    **17.**  $C(9; 2, 4, 3) = 9!/(2!4!3!)$ .    **20.**  $(-1)^k$     **23.** No. The only case that both extensions treat is  $n, k$

integers with  $0 \leq n < k$ . In that case, both extensions give  $\binom{n}{k} = 0$ . **24. b)**  $\sum_{k \geq 0} (k+1)x^k$  **26.** No, picking out one  $i$ -set leaves a second  $(n-i)$ -set. **28.** Theorem 2 becomes  $\binom{n}{i,j,k} = \binom{n-1}{i-1,j,k} + \binom{n-1}{i,j-1,k} + \binom{n-1}{i,j,k-1}$ . **30.** Substitute  $ax$  for  $x$ ,  $by$  for  $y$ , obtaining  $\sum C(n, k) a^{n-k} b^k x^{n-k} y^k$ . **32.** The  $j$ th element goes into set  $i \iff$  the letter in the  $j$ th position of the word is letter  $i$ . Counting permutations of words was discussed in Example 5, Section 4.3. **35. a)** Apply  $\frac{d^2}{dy^2}$  to the Binomial Theorem, then substitute  $x = y = 1$ . **b)** Apply  $\frac{d}{dy}$  to the Binomial Theorem, multiply by  $y$ , then differentiate again.

## Section 4.7

- 1.**  $\binom{10}{6} = 210$  configurations. **3.**  $26^4$  [dist. balls (positions) in dist. urns (letters)] **5.** Both dist.,  $3^4 = 81$ ; urns dist.,  $\binom{6}{2} = 15$ ; balls dist., 14; both indist., 4. **7.** one way **10. a)**  $\binom{13}{10} = 286$ . **c)**  $\binom{7}{6}\binom{5}{4} = 35$ . **12.** Dist. balls (permutation positions) in dist. urns (repeatable symbols) **14.** For  $n = 4$ : 4; 3,1; 2,2; 2,1,1; 1,1,1,1 **15. a)**  $6^9$  **17. a)** 12 **b)** 3 **19.** Each set of  $k$  elements from  $\{1, 2, \dots, n+k-1\}$  can be put into a strictly increasing sequence exactly one way. **22.** Consider all sequences of  $b$  numbered balls and  $u$  numbered walls, starting with a wall. E.g.,  $w_2, b_3, w_1, b_1, b_2$  means urn 2 is first, containing ball 3, and urn 1 is next, containing ball 1 then ball 2. Answer:  $u(b+u-1)!$ . **23. e)** Dist. balls in dist. stacks, but order of stacks fixed. So divide answer to [22] by  $u!$ . **24.**  $\binom{b-ku+u-1}{b-ku}$  **26. a)**  $p^*(b, u) = p(b-u, u)$  **d)**  $p(i, j) = \sum_{k=0}^{\min(i, j)} p(i-k, k)$ . **27.**  $p(6) = 11$ . In computing this from [26d], note that  $p(b, u) = p(b)$  whenever  $u > b$ . **29. b)**  $\lfloor b/2 \rfloor + 1$  **31.**  $P(b, u)u^{b-u}$  overcounts because it distinguishes the first ball into each urn. Try the case  $b = 2$ ,  $u = 1$ . **33.**  $S(0, 0) = T(0, 0) = 1$ .

## Section 4.8

- 1. b)**  $s^2/(1-s^2)$  **2. a)**  $1/(1-2s)$  **d)**  $1/(1+s^3)$  **3. a)**  $(1+x)^n$  **4. a)**  $a_k = (-1)^k$  **d)**  $a_k = k+1$ . **5. a)**  $a_k = 2+3 \cdot 2^k$ . **7.**  $1/(1-x) = (1+(-x))^{-1} = \sum C(-1, k)(-1)^k x^k = \sum (-1)^{2k} x^k$ . **9.**  $(1-s^{11})/(1-s)$ . **11.** Find  $a_{20}$  from  $(1+s+\dots+s^{20})(1+s^2+\dots+s^{18}+s^{20})(1+s^5+s^{10}+s^{15}+s^{20})$ . A CAS says  $a_{20} = 29$ . **13. a)**  $c_0 = 0$ ,  $c_k = 1$ ,  $k > 0$ . **c)**  $c_0 = c_2 = 1$ ;  $c_1 = 2$ ; all other  $c_k = 0$ . **14. a)**  $a_k = 1$ ,  $k = 0, 1, 2$ ; rest all 0;  $b_k = 1$  for all  $k$ ; thus  $c_0 = 1$ ,  $c_1 = 2$ ,  $c_k = 3$ ,  $k \geq 2$ . **15.**  $1+s+4s^2(1-s+s^2-\dots) = 1+s+4s^2-4s^3+\dots$ . **17. a)** 16, from  $s^3$  term in expansion of  $(1+s+s^2)^4$  or from  $s^3$  term in  $(1-s^3)/(1-s)^4 = (1-4s^3+\dots)(1+\dots+20s^3+\dots)$ . **19.** 25; by brute force consider cases of 1  $C$  through 9  $C$ 's and count combinations of  $A$  and  $B$  for each, or find coefficient of  $s^9$  in  $(1-s^{10})(1-s^5)^2/(1-s)^3$ . **21.**  $[1/(1-s)]^k = 1/(1-s)^k$ . **23. b)**  $\binom{n+k}{k} = \binom{n+k}{n} = \frac{k^n}{n!} + \text{lower order terms}$ . Thus if  $p(k) = \sum_{i=0}^n a_i k^i$ , then  $c_n = n! a_n$ . **25. b)** Also by induction, with Theorem 2 as the basis case, since  $\{c_{mk}\}$  is the convolution of  $\{a_k\}$  and  $\{c_{m-1,k}\}$ . **26.**  $(1+s)^n(1+s)^m = (1+s)^{n+m}$ .

## Section 4.9

1. Each input does not have a unique output. **2.** It never stops if, say, RAND[1,n] always outputs 1. The probability that some number is never output by RAND goes to 0 as time goes on. See Ch. 6. **3. b)** 2, 5, 3, 1, 4 **4.** The second loop should be **for**  $i = n$  **downto** 1 and should use  $r \leftarrow \text{RAND}[1, i]$ . **5. a)** RAND[n, n] has to be  $n$ . **b)** Terminate the loop at  $n-1$ . **c)** Still linear **7.** The probability that  $\text{Perm}(n) = n$  is too high ( $\frac{n-1}{n}$ ). **9.** First five: 12345, 12354, 12435, 12453, 12534. Last five: 54132, 54213, 54231, 54312, 54321. **11.** For PERMUTE-1 and PERMUTE-2, just end the for-loop at  $k$ . For PERMUTE-3, change  $n$  to  $k$  in the second for-loop and output only the first  $k$  entries of Perm. **13.** Any solution to [11] works. **15.** Let  $S \neq T$  be words. Let  $F(S)$  be the first symbol of  $S$  and let  $\overline{F}(S)$  be the remainder of the word. Assuming the order on individual symbols is known, then  $S < T \iff [F(S) < F(T) \text{ or } (F(S) = F(T) \text{ and } \overline{F}(S) < \overline{F}(T))]$ . If words can be different lengths, you also need to define the null word to come before all other words. **17. a)** 463 **b)** 43521 **18.** For any  $n$ -permutation  $P$ ,  $a_1, a_2, \dots, a_n$  are the values in Eq. (3) for computing  $\text{RAND}(P) \iff a_1+1, a_2+1, \dots, a_n+1$  are the values of  $r$  in PERMUTE-2 that cause PERMUTE-2 to output  $P$ . **20.**  $\text{Rank}(462135) = 258$ .  $\text{Unrank}(90) = 31$ . Let  $a_k$  be the number of digits to the right of the  $k$ th which are *larger*. **25.** Cycle 34125, length 1200 **27. (i, j)  $\leftarrow (\text{RAND}[1, n], \text{RAND}[1, n])$**  **29.** All first entries are equally likely and they shouldn't be. As a result, the probability of getting  $(1, n)$  is  $1/n^2$  (too low) and the probability of getting  $(n, n)$  is  $1/n$  (too high). **30.** Many ways; here's a wasteful one. **repeat until**  $y > x$ ;  $x \leftarrow \text{RAND}[1, n]$ ,  $y \leftarrow \text{RAND}[1, n]$ ; **endrepeat** **32. a)** For  $k$

from 1 to  $(n-1)!$ , let  $P$  be the  $k$ th  $(n-1)$ -permutation. If  $k$  is odd, replace  $P$  by  $n$  permutations, each with the symbol  $n$  inserted in  $P$ , first at the right end, then next to the right, ..., finally on the left. If  $k$  is even, make the same replacements but insert symbol  $n$  starting on the left. **c)** Every number from 0 to  $n! - 1$  has a unique representation in the form  $\sum_{k=1}^n a_k n!/k!$ ,  $0 \leq a_k \leq k-1$ .

## Section 4.10

- 1.**  $kn+1$    **3.** 26 socks   **5.** All 10 socks   **6.a)** There are 142 possible sums (6 to 147).   **c)**  $m=42$ .   **g)** Every sum from 3 to 47 must be obtained.  $1+2$  is the only way to get 3.  $1+24 = 2+23$ .   **7.a)** Compute and save the sums for all the 3-subsets, and set off an alarm if a sum is obtained twice.   **b)** The algorithm forces you to set up a correspondence between 3-subsets and sums, forcing you to think about how many subsets and how many sums there can be.   **9.** 5 trits   **10.** To determine  $u_k$  (or  $d_k$ ) you need to know  $u_j$  for each  $j < k$ . So procedure call  $k$  would have  $k-1$  subcalls. Even just two subcalls per call (as in Algorithm FIBB) usually leads to exponential run time. In fact, recursive MONOTONIC on an  $n$ -sequence would have exactly  $2^n$  calls.   **12.a)** Define  $U(k)$  to be the index of the term just before  $u_k$  in the longest up subsequence (found so far) ending at  $u_k$ . Include  $U(k)$  and a similar  $D(k)$  in the algorithm. For instance, in the line that begins “**if**  $a_j < a_k$ ”, when  $u_j + 1 > u_k$  update  $U(k)$ . If at the end  $d_7$  (say) is the largest  $d$  or  $u$  value (keep a running record of the biggest so far), then a longest sequence can be found by computing  $D(7), D(D(7))$  and so on.   **15.** If you still want a strictly monotonic subsequence, the conclusions are false. If any monotonic subsequence will do, the conclusions are true, but the inequalities in Algorithm MONOTONIC need to be changed.   **16.b)**  $L = n^3 + 1$ ,  $L' = n + 1$ .   **19.** From a set of  $P$  pigeons, let subset  $P_i$  be in hole  $i$ . If there is at most one pigeon per hole, then  $|P| \stackrel{\text{Sum Rule}}{=} \sum_{i=1}^n |P_i| \leq \sum_{i=1}^n 1 = n$ .

## Supplementary Problems: Chapter 4

- 1.** Let  $P_k$  be the 0–1 pattern in rows 0 through  $2^k - 1$ . Then  $P_{k+1}$  consists of three corner triangles and one center triangle. By induction show that all three corner triangles are  $P_k$  and the center triangle is all 0’s.   **3.**  $\binom{n}{3} + 4\binom{n}{4} + 5\binom{n}{5} + \binom{n}{6}$ . There are  $\binom{n}{3}$  triangles with all three vertices on the polygon. There are  $4\binom{4}{n}$  triangles with just two vertices on the polygon; the sides to the third triangle vertex extend to two other polygon vertices, and the chords from each set of 4 polygon vertices create four such triangles.  $5\binom{n}{5}$  counts triangles with one polygon vertex; there are  $\binom{n}{6}$  triangles with no polygon vertices.   **5.** If a chord has  $i$  polygon vertices on one side (hence  $n-2-i$  on the other), it contains  $i(n-2-i)$  intersection points. The answer is  $\frac{1}{4} \sum_{i=1}^{n-3} i(n-2-i)n$ . (The sum counts each chord twice and thus each intersection point 4 times.) Find a closed form.   **7.** Consider all  $n$ -permutations of  $1, 2, \dots, n$  with repetitions allowed. Count them by the location (2 through  $n+1$ ) of the first repetition ( $n+1$  means no repetition). Let this position be called  $k+1$ .   **9.** Starting with a simple graph with vertices labeled  $1, 2, \dots, n-1$ , create a new vertex labeled  $n$  and link it by an edge to each vertex previously of odd degree. In the unlabeled case, consider  $n=4$ .   **11.a)** 3   **c)** 10   **13.** Result after much algebraic manipulation is  $C(2n, n)/(n+1)$ .   **14.a)**  $1/(1-s)$    **b)**  $e^s$    **c)** Just calculate coefficient of  $s^n/n!$  in  $A(s)B(s)$ .   **d)** To choose from  $A \cup B$  can choose a  $k$ -list from  $A$  ( $a_k$ ) and a  $(n-k)$ -list from  $B$  ( $b_{n-k}$ );  $k$  elements from  $A$  can be placed in  $n$  positions in  $C(n, k)$  ways; sum over  $k$  to get  $c_n$ .   **e)** Interpretation of product of two terms is given in d); term in product of  $m$  exponential generating functions is given by  $(m-1)$ -fold summation involving  $m-1$  binomial coefficients and represents number of distinguished lists of elements taken from each of  $m$  non-intersecting sets.   **f)** Since  $a_k = b_k = 1$ ,  $c_n$  is  $2^n$ .   **16.b)**  $A(x)B(x)=1 \iff A(x)=B(x)=1 \iff x \in A \cap B \iff (A \cap B)(x)=1$ .   **d)** Direct from parts b) and c).   **18.**  $\binom{22}{2} - 3\binom{12}{2} + 6 = 36$ .   **19.a)** For  $n=2$ :  $a_{20}=0$ ,  $a_{21}=-1$ ,  $a_{22}=1$ .   **b)**  $a_{n+1,k} = a_{n,k-1} - na_{nk}$  when  $1 \leq k \leq n$ ;  $a_{00}=1$  and  $a_{n0}=0$ ,  $a_{nn}=1$  for  $n > 0$ .   **c)** Last row of table is: 0, 24, -50, 35, -10, 1.

## Chapter 5

### Section 5.2

- 1.**  $a_n = a_{n-1} + n^2$ ,  $a_1 = 1$ .   **3.**  $a_n = ra_{n-1} + d$ ,  $a_0$  (or  $a_1$ ) =  $a$ .   **5.a)**  $Y_n = cY_{n-1} + I + G$ .   **7.** (833.33, 1166.67), (805.56, 1194.44), (800.93, 1199.07), (800.15, 1199.85), (800.03, 1199.97)   **9.**  $N_n = \frac{3}{4}N_{n-1} + \frac{4}{5}L_{n-1}$ ;  $L_n = \frac{1}{4}N_{n-1} + \frac{1}{5}L_{n-1}$

$\frac{1}{5}L_{n-1}$ . **12.**  $r_n = r_{n-1} + 2r_{n-2}$ ;  $r_0 = 3$ ;  $3, 3, 9, 15, 33, \dots$ . **14.** The tempting analysis is wrong (why isn't  $r_{n-2}$  the number of rabbits born?), but  $r_n = r_{n-1} + r_{n-2} - r_{n-12}$  is correct anyway! Start with  $b_n = \sum_{i=2}^{11} b_{n-i}$ , where  $b_n$  are the number born in month  $n$ . **16.**  $c_n = c_{n-1}[\text{dot first}] + c_{n-2}[\text{dash first}]$ ;  $c_1 = 2$ ,  $c_2 = 3$ . **18.a)**  $S_{n+1} = S_n + r_n = S_n + \frac{1}{2}n(n+1) + 1$ . **b)**  $S_n = S_{n-1} + r_{n-1}$ ,  $r_n = r_{n-1} + n$ ,  $S_0 = r_0 = 1$ . **19.** Formulas for  $P_{k+1}, \hat{P}_{k+1}, V$  unchanged;  $\hat{P}_0 = P/(1-t)$ ;  $\hat{V} = \hat{P}_N - \hat{P}_{Nt}$ . **23.**  $c_{u,b} = c_{u-1,b}$  [don't use first urn] +  $c_{u-1,b-1}$  [do use it];  $c_{u,0} = 1$ ;  $c_{1,1} = 1$ ;  $c_{1,b} = 0$  for  $b > 1$ . **25.**  $P_k = 1$  for all  $k$ . **27.**  $P_n = (1+r)P_{n-1} - p$ ;  $P_0 = P$ ,  $P_N = 0$ . **29.**  $P(n+1, 0) = \sum_{i=10}^{50} c_i P(n, i)$ , where  $c_i$  is the yearly birth rate for females age  $i$ . For  $a > 0$ ,  $P(n+1, a) = (1-d_{a-1})P(n, a-1)$ , where  $d_a$  is the yearly death rate for females age  $a$ . **31.**  $N_n = \frac{7}{6}N_{n-1} - \frac{1}{6}N_{n-2}$ ;  $N_0 = 1000$ ,  $N_1 = 833\frac{1}{3}$ .

### Section 5.3

**1.** The pattern  $1, 1, 0, -1, -1, 0$  cycles **3.a)**  $1, 2, 4, 8, 16, 32$  **c)**  $1, 3, 9, 27, 81, 243$  **e)**  $0, 1, 5, 19, 65, 211$ . Note that a) and c) are geometric and e) is their difference. **4.**  $E_{2n} = (n/2) + E_n$ ;  $E_{2k} = (2^k + 1)/2$ . **7.** Using Eq. (5),  $\hat{V}/V$  is 1.004 in a), 1.28 in b) and 1.18 in c). **8.** Now  $\hat{V} = \hat{P}_N - (\hat{P}_N - P)(t+1)$ ; 22 years. **10.**  $V$  unchanged,  $\hat{V} = [P/(1-t)][(1+r)^N(1-t)] = P(1+r)^N$ , so  $\hat{V}/V = [(1+r)/(1+r(1-t))]^N > 1$ .  $\hat{V}/V$  for the 3 scenarios in [7] is 1.062, 1.640, 1.540. **13.** In the equations for [9, Section 5.2], substitute  $x$  for both  $N_n$  and  $N_{n-1}$  and  $2000-x$  for both  $L_n$  and  $L_{n-1}$ . The unique solution is  $2000(\frac{16}{21}) \approx 1524$ . **16.** Same as the statement in [15] except, in the display, replace  $P(m-1, n-1)$  by  $P(m, n-1)$ . **17.**  $f(m, n) = (m-n+1)^n$ . **19.a)**  $r_n = 1 + (1/r_{n-1})$ . **b)**  $r = 1 + (1/r) \implies r^2 - r - 1 = 0$ . The positive root is  $(1 + \sqrt{5})/2 \approx 1.618$ . **20.**  $\frac{L}{W} = \frac{L+W}{L}$ , which with  $r = \frac{L}{W}$  becomes  $r = 1 + \frac{1}{r}$ . Now see the solution to [19b]. **22.** The pattern is  $P_n = P(1+r)^n - p \frac{1 - (1+r)^n}{1 - (1+r)}$ . Since  $P_N = 0$ ,  $p = \frac{Pr(1+r)^N}{(1+r)^N - 1} = \frac{Pr}{1 - (\frac{1}{1+r})^N}$ . **24.** If  $a_0 > 0$ ,  $a_n \rightarrow 2$ ; if  $a_0 < 0$ ,  $a_n \rightarrow -2$ ; if  $a_0 = 0$ ,  $a_1$  is undefined.

### Section 5.4

**1.** first-order, nonlinear **3.** first-order, linear, nonhomogeneous, constant-coefficient **5.** second-order, linear, homogeneous, constant-coefficient **7.** second-order, linear, homogeneous **9.** linear, homogeneous, constant-coefficient (variable order) **11.a)** first-order, linear, homogeneous, constant-coefficient **b)** Same as in a) **14.** All sequences in which each term is either  $\pm 1$ . Theorem 1 says that  $a_1$  uniquely defines a solution to a first-order recursion if  $a_n$  is given as a function of  $a_{n-1}$ , that is, if  $a_n$  is uniquely defined in terms of  $a_{n-1}$ . That's not true here:  $a_n^2 = a_{n-1}^2$  leads to two values of  $a_n$ . **16.** Choose the  $k$ th term two different ways, and by Theorem 1 you already have two different sequences. **18.a)** If a piece ends at  $m$  units, there are  $d_m$  choices on the left and  $d_n$  on the right. If a domino goes from  $m-1$  to  $m+1$ , there are  $d_{m-1}$  choices on the left and  $d_{n-1}$  on the right. **19.** In the second proof, the induction has been pushed back into the proof of Theorem 1.

### Section 5.5

**1.**  $e_n = e_0 3^n$ . **4.**  $a_n = \frac{5}{6}2^n - \frac{1}{3}(-1)^n$ . **6.**  $a_n = 3^n - 2^{n+1}$ . **8.**  $a_n = 2^{n+2} - 3^n$  **10.a)**  $2^n$  **d)**  $2^n + 3^n$  **11.**  $a_n = Aw^n + \bar{A}\bar{w}^n$ , where  $A = \frac{1}{2} - \frac{i\sqrt{3}}{6}$ ,  $w = \frac{1+i\sqrt{3}}{2}$  and bars denote complex conjugate. **12.**  $A(-1)^n + Bn(-1)^n + Cn^2(-1)^n$  **15.** The degree of the characteristic polynomial is the difference between the indices of the highest and lowest indexed (nonzero) terms in the difference equation. By definition, this index difference is the order of the difference equation. **16.b)** It is the coefficient of the lowest order nonzero term in the difference equation. **18.** The trivial solution is obtained when  $A = B = 0$ . **20.a)**  $c_1 r_1^{n-1} + c_2 r_1^{n-2} = (r_1 + r_2)r_1^{n-1} + (-r_1 r_2)r_1^{n-2} = r_1^n$ . **21.** By Theorem 2, every solution is  $a_n = Ar_1^n + Br_2^n$  (in part a) or  $a_n = Ar_1^n + Bnr_1^n$  (part b). Now replace all exponents and subscripts  $n$  by  $k$  and then by  $k+1$ , and solve for  $A$  and  $B$ . Use the fact that neither root is 0. **24.a)** recursion **b)** closed form **c)** closed form **d)** probably recursion (perhaps even when fast computer algebra calculators are available) **25.** Show that the difference between the expression in the problem and Eq. (7) is  $< 1/2$ . **28.**  $a_n = 2F_n$ , where  $F_n$  is the  $n$ th Fibonacci number. **31.**  $a_n = -a_{n-1} + 12a_{n-2}$ . **33.**  $a_n = 2a_{n-1}$ ,  $a_0 = 1$ . **35.**  $c_n = 5c_{n-1} - 6c_{n-2}$ ;  $c_0 = 2$ ,  $c_1 = 5$ . **37.**  $e_n = e_{n-1} + 4e_{n-2} - 4e_{n-3}$ ;  $e_0 = 3$ ,  $e_1 = 7$ ,  $e_2 = 3$ . **39.**  $f_n = 5f_{n-1} - 3f_{n-2}$ ;  $f_0 = 2$ ,  $f_1 = r_1 + r_2 = 5$  since the sum of roots of a quadratic is negative the coefficient of  $x$ . **41.** The general solution

is  $a_n = Aw^n + B\bar{w}^n$ , where  $w, \bar{w}$ , defined in the solution to [11], satisfy  $w^6 = \bar{w}^6 = 1$ . Thus  $a_{n+6} = a_n$  for all n.

**43.** From [41],  $a_0 = a_6$  for any solution. **44.**  $a_n = (2 \cos \theta)a_{n-1} - a_{n-2}$ . **46.** Use formulas for  $\sin(\alpha+\theta)$  and  $\cos(\alpha+\theta)$  with  $\alpha = n\theta$ . **47. a)**  $r_{2k} = 0$  and  $r_{2k+1} = 2^k$  because  $G(s)$  satisfies  $G(s) = s + 2s^2G(s)$ . **c)**  $r_k = 1$  for all  $k$  because  $(1 - 2s + s^2)G(s) = 1 - s$ , so  $G(s) = 1/(1-s)$ . **48.** If the char. poly is  $r^2 + cr + d$ , then the polynomial in the denominator is  $1 + cs + ds^2$ . To associate  $a_{n-2}$  in  $a_n + ca_{n-1} + da_{n-2} = 0$  with  $s^n$ , one must look at  $s^2G(s)$ .

### Section 5.6

- 1. a)**  $\rightarrow \infty$  **b)**  $\rightarrow 3$  **d)** Oscillates to  $\infty$  like  $(-1.2)^n$  **f)** Oscillates to 0 **g)**  $\rightarrow 0$  **3.** Look at  $a_{k+1}/a_k$ ; the values for  $k = 8, 9, 10$  are 1.83951, 1.83893, 1.83942. Or find roots numerically of  $r^3 - r^2 - r - 1 = 0$ ; they are  $1.839, .420 \pm .606i$ . Either way, the long term growth is essentially geometric with ratio 1.839.

### Section 5.7

- 1. a)**  $A3^n - 2^{n+1}$  **b)**  $A3^n - 2^{n+1}(n+3)$  **c)**  $A3^n + n3^n$  **d)**  $A3^n - \frac{1}{2}$  **3. a)**  $A2^n + Bn2^n + \frac{1}{4}n^22^n$  **b)**  $A2^n + Bn2^n + n^32^n/12$  **c)**  $A2^n + Bn2^n + 3^{n+1}$  **d)**  $A2^n + Bn2^n + 1$  **5.**  $3^n - 1$  **8.** No, Yes **9.**  $\{3a_n\}$  satisfies  $s_n = s_{n-1} + s_{n-2} + 3n^2$ . **10.**  $f(n) = \sum_{i=1}^k P_i(n)r_i^n$ , where  $k$  is any positive integer, each  $P_i(n)$  is a polynomial, and each  $r_i$  is a constant. **12. a)** Because  $F(n) - F(n-1) = f(n)$ . **c)**  $n2^{n+1} - 2^{n+1} + 2$  **14.**  $b_n = A3^n + \sum_{k=2}^n 3^{n-k} \log k$ .

### Section 5.8

- 1. a)**  $n$  for both **b)**  $2n, 2n+1$  **3.** The savings are small for low-order polynomials. For people, HORNER requires more recording of partial results; e.g., many people can compute terms like  $2 \cdot 3^2$  (from STRAIGHTFORWARD) in their heads, but fewer can do  $(2 \cdot 3 + 4) \cdot 3$  mentally. Finally, STRAIGHTFORWARD emphasizes the standard algebraic form of polynomials, which is useful for other computations. **5.** Multiplications:  $1 + 2(n-1) = 2n - 1$ . Additions:  $1 + n - 1 = n$ . Between the other two in efficiency. **7.** Induction on  $n$  **9.** Each call of Procedure TOH involves two comparisons of  $n$  to 1; each call of TOH\* involves one. Both procedures make the same subcalls. **11.** FIBA uses  $2(n-1)$  subtractions and  $n-1$  additions ( $n-1$  more if you count updating  $k$ ). If FIBB uses  $a_n$  of these operations, then  $a_n = a_{n-1} + a_{n-2} + 3$ ; solution is  $a_n = 3F_n - 3$ . **12. c)**  $F(n)$  takes  $2n-1$  calls regardless of order: one call each for  $n, n-1$  and 0, two calls for 1 to  $n-2$ . A difference equation paralleling the recursion in the algorithm ( $c_n = c_{n-1} + c_{n-2}$ ) doesn't work; why? **14.** The initial nonhomogeneous solution is  $B_n = -1$ . This implies  $e_1 = e_2 = 2$ , so  $C$  and  $D$  are twice the values of  $A$  and  $B$  in Example 4, Section 5.5. Thus  $B_n = 2F_n - 1$ . **16. a)**  $a^{\log_b c} = (b^{\log_b a})^{\log_b c} = (b^{\log_b c})^{\log_b a} = c^{\log_b a}$ . **b)**  $b^k = b^{\log n} = n^{\log b}$  ( $\log = \log_2$ ). **19.**  $B_{2k} = 3^k$ , so  $B_n = n^{\log 3} (= 3^{\log n})$ . **22.**  $D_{2k} = 3^{k+1} - 2^{k+1}$ , so  $D_n = 3n^{\log 3} - 2n$ . **23.**  $F_n = (1 + \log n)n^2$ . **25.**  $H_n = \frac{1}{16}(21n^{\log 5} - 4\log n - 5)$ . **27.**  $J_n = 7n^{\log 3} - 2n(3 + \log n)$ . **29.**  $L_n = 7n^{\log 7} - 6n^2$ . **31.** Strong induction on the claim  $P(n)$  that  $D'_1, \dots, D'_n$  is increasing, with basis case  $n = 2$ . (Why not  $n = 1$ ?) **34.** Verify that  $ckb^k = bc(k-1)b^{k-1} + cb^k$ , obtained by plugging the guess  $a_k = ckb^k$  into (13). **36.** Strong induction using the previous integer case  $n/2$  or  $\lfloor n/2 \rfloor$ . **38.** The main line in one approach is: for  $k = m$  downto 0, if  $a_k = 1$  then  $p \leftarrow p^2x$  else  $p \leftarrow p^2$ . **40.** The recursion is  $A_n = 1 + A_{n/2}$ ,  $A_1 = 1$ . The solution is  $A_n = 1 + \log n$ . Do strong induction using Eq. (3) to show that  $A_n \leq 1 + \log n$  for all  $n \in N^+$ . **42.** Least =  $\min\{p, q\}$ . Let the  $p$ -list be 1,2 and the  $q$ -list be 3,4,5,6. **44.** 1 5 3 7 2 6 4 8 **46.** Solve  $\overline{C}_n = \overline{C}_{n/2} + 1$ ,  $\overline{C}_1 = 0$ , by Divide and Conquer to obtain  $\overline{C}_n = \log n$ . To prove  $\overline{C}_n \leq C_n$  for all  $n \in N$ , use strong induction and properties of logs in base 2. **50.** For  $n = 2^k$ ,  $B_n = 1 + B_{\frac{n}{2}+1}$ , not quite the right form. The  $B'$  recurrence is the right form, but it is likely to underestimate  $B$ . Workaround: Theorem 1 does apply to  $C_n = B_{n+1}$ ; why? **53.** One can find the median by first sorting (which can be done by Divide and Conquer). But to use Divide and Conquer to find only the medians  $m_1, m_2$  of two unsorted half lists tells you too little about the median of the whole list — only that it is  $m_1$ , or  $m_2$ , or between them. **56. a)** Example 6 shows how to get  $x^{2^k}$  in  $k$  multiplications. This is best possible since the largest power of  $x$  obtainable with  $\leq k-1$  multiplications is  $x^{2^{k-1}}$  (induction!). **b)** Lower bound: Since  $k$  multiplications yields at most  $x^{2^k}$ , if  $n \geq 2^k$  then  $M(n) \geq k = \log n$ . Upper bound:  $C_n = 2 \log n$  satisfies  $C_n \geq C_{\lfloor n/2 \rfloor} + 2$ ,  $C_1 = 0$ ; see Eq. (5).

### Section 5.9

- 2.**  $c_n = 3^n$ . **5.**  $g_n = n+1$ . **8.**  $m_n = 2^n(n+1)!$ . **9. a)**  $a_n = c^{\sum_{i=1}^k i} = c^{\binom{n+1}{2}}$ . **b)**  $a_n = \sum_{j=0}^n c^{\binom{n+1}{2}} - \binom{j+1}{2}$ . **10.**  $t_n =$

$(n+1)/n!$ . **11.**  $S_n = 1/n^2$ . **16. a)** As formulated in [21, Section 5.2],  $\hat{P}_k = (\hat{P}_{k-1} + P)(1+r) = \hat{P}_{k-1}(1+r) + P(1+r)$ , an arithmetic-geometric sequence. Also  $P_0 = 0$ . Thus  $\hat{P}_N = \frac{P}{r} [(1+r)^{N+1} - (1+r)]$ . For the non-IRA,  $P_N$  is the same with  $r^* = r(1-t)$  substituted for  $r$ . Finally,  $\hat{V} = \hat{P}_N - [\hat{P}_N - PN]t = \hat{P}_N(1-t) + PNt$ , and  $V = P_N$ . **b)**  $\hat{V} = \$286,415$ ,  $V = \$222,309$ . **17. a)**  $(s^{n+1} - r^{n+1})/(s-r)$  **b)**  $(n+1)r^n$  **20.**  $b_n = rb_{n-1} + a$  with  $b_0 = a$ . By the solution to Example 4,  $b_n = r^n(a + \frac{a}{r-1}) - \frac{a}{r-1} = \frac{ar^{n+1}-a}{r-1}$ . **21.**  $k = d/(c-1)$ ,  $b_n = b_0 c^n$ . **22.** The ratio of the  $(j+1)$ st product to the  $j$ th is  $d_{j+1}/(c_{j+1}d_j)$ , which is rational. **23.** For  $k \in Z$  but outside 0 to  $n$ ,  $\binom{n}{k}$  has been defined as 0; hence the sum is finite.  $S_n = 2S_{n-1}$ .

## Section 5.10

1. There's somewhat more evidence of convergence, but very slow. For instance,  $P_{50} = 1.01794$  and  $P_{100} = 1.01687$ ; also  $P_{49} = .98136$  and  $P_{99} = .982519$ . **5.** Period 8 for any positive  $P_0 < 1.3922$ , but you have to go out about 100 terms to see this. For  $P_0 > 1.3922$ ,  $P_n \rightarrow \infty$  very quickly. (You're not expected to discover the dividing point, which equals  $= (r+1)/r$ .) **8.** Proof breaks down in quadratic term of Eq. (1). For instance,  $rP_n^2 + rP_n'^2 \neq r(P_n + P_n')^2$ . **10. a)**  $P_n = P_{n+1} = P_{n+2} = \dots$  **b)** No **12.** When  $P_0 = 1$  the sequence is constant, and  $1 - rP_n = 1 - r = .93$ . When  $P_0 = 2$ , by induction  $\{P_n\}$  is positive decreasing, and thus  $1 - rP_n$  is increasing but  $< 1$ . **14.**  $r = 1$ ,  $P_0 > 2$  **16. a)**  $0 < P_1 \leq 1$ , and then by [15] or argument in text involving Fig. 5.13,  $P_n \rightarrow 1$ . **c)** 2 0 2 0 ... **18.** The lines  $y = x$  and  $y = f(x) = 2x$  intersect at  $(0,0)$  only, and zig-zags as in Fig. 5.13 move away. **20.**  $f(x) = 3x - 2x^2$  is a parabola with high point  $(\frac{3}{4}, \frac{9}{8})$ , so if  $P_0 = \frac{3}{4}$ , then  $P_1 = \frac{9}{8}$ . **23.** Multiply Eq. (1) by  $r$  and substitute  $s$  once for  $(1+r)$ . Now what? **25.**  $z_n = q_n + \frac{1}{2}s$ . **27.**  $3^\alpha = 4$  so  $\alpha = (\log 4)/\log 3 \approx 1.26$ . **29.**  $a_n = a_0^{(2^n)}$ . **31.**  $p_n = (10 \cdot 3^n)/(5 \cdot 3^n - 1)$ . **33.**  $2(\cos^2 x) - 1 = \cos(2x)$ , so  $p_n = \cos(2^n x_0)$ .

## Section 5.11

**1. a)**  $2n-1$  **2. a)**  $(n+2)2^n$  **b)**  $(n+4)2^n$  **c)**  $(n+2k)2^n$ ; prove by induction. **5. a)**  $\binom{n}{k-1}$ . **6. a)**  $-1/(n(n+1))$ . **8. a)**  $n(n-1)$  **c)**  $(n-1)(n-2)(2n-3)/6 = \frac{1}{6}(2n^2 - 9n^2 + 13n - 6)$  **f)**  $\frac{1}{2}3^n$  **9. b)**  $(k+1)_k = (k+1)_{(2)}$  so  $\sum = \frac{1}{3}(k+1)_{(3)} \Big|_1^{n+1} = \frac{1}{3}(n+2)_{(3)} - \frac{1}{3}2_{(3)} = \frac{1}{3}(n+2)(n+1)n$ . **10. c)**  $k_{(m+1)} \Big|_0^n = n_{(m+1)}$ . **12.**  $\Delta^0 a_n = a_n$ ,  $\Delta^k a_n = \Delta^{k-1} a_{n+1} - \Delta^{k-1} a_n$ . **14. b)** The key statement is  $p(x) = \sum_{j=0}^k (1/j!) \Delta^j a_m (x-m)_{(j)}$ . **15.**  $\Delta \binom{x}{k} = \binom{x}{k-1}$  (no factor comes out). Then  $p(x) = \sum \Delta^j a_0 \binom{x}{j}$ . **17. b)**  $-n + 3n_{(2)} + n_{(3)} = n^3 - 2n$ . **19.** Theorem 6 finds  $p(x) = x$ , which agrees with  $f(x) = \lfloor x \rfloor$  at all  $x \in N$ . **20. b)** If the difference table for  $a_0, a_1, \dots, a_k$  doesn't reach a row of 0's, then below the single-entry lowest row create a row of 0's. This is consistent with the table above it and Theorem 6 applies. **22. b)**  $-2x_{(-3)}$ . **23. a)**  $\sum_{k=1}^n k_{(-2)} = -1k_{(-1)} \Big|_1^{n+1} = 1 - \frac{1}{n+1}$ . **24. a)**  $\Delta^2 a_n + \Delta a_n - 2a_n = 0$ . **27.**  $a_{n+k} = \sum_{j=0}^k \binom{k}{j} \Delta^j a_n$  (induction). **29. b)**  $(n-1)n$  **e)**  $\Delta n(n+1) = 2(n+1)$ . **30. d)**  $A(2n-1) - B$  **f)**  $kn^{(k-1)}$  **32.**  $\sum_{k=p}^q a_k = \nabla^{-1} a_k \Big|_{p-1}^q$ . **33.**  $x^{(-k)} = 1/[x(x-1)(x-2) \cdots (x-k+1)]$ . **36.**  $\Delta(\Delta^{-1} a_n) = a_n$ .

## Supplementary Problems: Chapter 5

**1.**  $F_{-n} = (-1)^n F_{n-2}$ . **3.**  $a_n = 2^{(1+2+\dots+n)}$ . Theorem 1 does not apply since  $a_{n+1}$  not uniquely defined when  $a_{n-1} = 0$ , which won't occur if  $a_1, a_2$  nonzero. **6.** Your difference equation will be nonhomogeneous. First find a particular solution as with single-indexed difference equations. Solution:  $2\binom{n}{k} - 1$ . **8.**  $p = P/\sum_{i=1}^{420} m_i(1+r)^{-i}$ . To see why the difference equation is correct, multiply by  $N$ :  $NP_n = (NP_{n-1})(1+r) - (Nm_n)p$ . Thus the amount held at time  $n$  is the amount at the previous time with interest, minus payments to those annuitants still alive. **9.** The characteristic polynomial must have roots  $(a \pm \sqrt{b})/2$  so the difference equation is  $c_n = ac_{n-1} + (\frac{a^2-b}{4})c_{n-2}$ . By assumption the coefficients are integers. It's easy to check that  $c_0$  and  $c_1$  are integers for both sequences. Thus every  $c_n$  is an integer. **11.**  $2F_n - 1$  **12.**  $C_{n,k} = C_{n-1,k} + C_{n-2,k}$ , and  $C_{k,k} = C_{k+1,k} = 1$  (except  $C_{1,0} = 0$ ). These are Fibonacci recursions (only the first index varies), so  $C_{n,k} = F_{n-k}$  (except  $C_{n,0} = F_{n-2}$ ). **15.** Very hard to prove directly, but its just a special case of the formula in [14]. **16.** General solution:  $A(1+\sqrt{2})^n + B(1-\sqrt{2})^n + C$ , so all possibilities mentioned possible, but explosive growth ( $A \neq 0$ ) most likely. **19. a)**  $r_n = 2^{n-1}$ . **c)** For  $n \geq 1$ ,  $r_n = F_{n-1}$ . **21. b)**  $c = d/(-1 + \sum c_j)$ , so the denominator must be nonzero. **22. a)**  $k2^{k-1}$ , i.e.,  $\frac{1}{2}n \log n$  for

$n = 2^k$ . The recursion is  $m_k = 2m_{k-1} + 2^{k-1}$ . **b)**  $2^{k+1} - k - 2 = 2n - 2 - \log n$     **25.** For  $0 \leq k \leq n$ ,  $a_{n,k} = \binom{n}{k}/2^n$ .

**28. a)** The roots  $r_1, r_2$  are complex numbers with modulus  $\sqrt{c}$  and arguments  $\pm\theta$ , where  $\theta = \tan^{-1} \sqrt{(4c/b)-1}$ . The basic solutions are  $r_1^n = c^{n/2}e^{in\theta}$  and  $r_2^n = c^{n/2}e^{-in\theta}$ . Show that every real-valued linear combination of these sequences is a real linear combination of  $\sin n\theta$  and  $\cos n\theta$ . **29. a,b)**  $A = (ar+b)/[c(r-s)]$ ,  $B = (as+b)/[c(s-r)]$ ; since  $r \neq s$ , expansion is always possible. **c)** (i)  $1/(x-3) + 1/(x-2)$     **30. b)** (i)  $7/3(x-2) - 13/2(x-3) + 37/6(x-5)$     **31. a)** Multiplying by  $(x-r)^j$  and letting  $x=r$  doesn't get rid of all but one constant. **b)** Method of [30a] still creates algebraic identity. For (i):  $-4/(x-1) + 4/(x-2) - 3/(x-2)^2$ .    **32. b)** (i):  $1/11(x-3) - 2(x+1)/11(2x^2 - 4x+5)$     **33. a)** Putting right hand side over common denominator will not give numerator polynomial of degree of  $P(x)$ . **b)** Long division results in remainder with numerator degree less than denominator degree. **c)** (i):  $2 - (10x+9)/(x^2 + 3x + 7)$ .

## Chapter 6

### Section 6.1

**1. a)** Out of 10,000 people, 92 of the 100 carriers will test positive, as will 396 (4%) of the 9900 noncarriers. If you test positive, you have only  $92/(92+396) \approx .189$  probability of being a carrier. **3. #** dollars = # birds; make the coin biased. **5.** If you stay with your original choice, what is the probability you will win? If you switch and the car was not behind the door you chose originally, what is the probability you will win?

### Section 6.2

**1. b)** {1, 4}    **2. a)** At least one die has two dots:  $\{(2, 1), (1, 2), (2, 2), (2, 3), (3, 2), (2, 4), (4, 2), (2, 5), (5, 2), (2, 6), (6, 2)\}$ ; one twice as many as other:  $\{(2, 1), (1, 2), (4, 2), (2, 4), (6, 3), (3, 6)\}$ ; both same:  $\{(1, 1), \dots, (6, 6)\}$ ; sum is 5:  $\{(1, 4), (4, 1), (2, 3), (3, 2)\}$ ; Difference is 3:  $\{(1, 4), (4, 1), (2, 5), (5, 2), (3, 6), (6, 3)\}$ .    **3.** Done in Example 3. **4.** The atoms are the possible triplets of tosses.  $E = \{\text{HHH}, \text{HHT}, \text{THH}, \text{THT}\}$ .    **5. a)**  $A \cup B \cup C \sim (A \cap B \cap C)$  **8.** Since  $0 \leq |A| \leq |S|$ ,  $0 \leq |A|/|S| \leq 1$ ;  $|\emptyset|/|S| = 0/|S| = 0$ ,  $|S|/|S| = 1$ ; if  $A \cap B = \emptyset$ , then  $|A \cup B| = |A| + |B|$ , so  $\frac{|A \cup B|}{|S|} = \frac{|A|}{|S|} + \frac{|B|}{|S|}$ .    **10.**  $\binom{2+6-1}{2} = 21$ ; ind. balls in dist. urns.    **11. a)**  $E_{ij}$ : you choose  $i$ , Monty Hall chooses  $j$ , for  $i, j = 1, 2, 3$ ,  $i \neq j$ . **b)**  $E_{12}, E_{13}: 1/6$ ;  $E_{23}, E_{32}: 1/3$  (MH would always choose 3 if you chose 2);  $E_{31}, E_{21}: 0$     **c)** Stick: 1/3; switch: 2/3    **d)** If car is not under original door, it *must* be under the door you switch to (why?). **15. 3**  $\Rightarrow \Pr\{a \cup b\} = \Pr(a) + \Pr(b) \Rightarrow$  Eq. (5) by induction; (5)  $\Rightarrow \Pr(A \cup B) = \sum_{e \in A \cup B} \Pr(e)$  equals (if  $A \cap B = \emptyset$ )  $\sum_{e \in A} \Pr(e) + \sum_{e \in B} \Pr(e) = \Pr(A) + \Pr(B)$ .    **17.**  $\sim A \cap \sim B = \sim(A \cup B)$ , so  $\Pr(\sim A \cap \sim B) = 1 - \Pr(A \cup B)$ . Now use Theorem 2.    **19.**  $B$  and  $A - B$  are disjoint, and  $A \cup B = B \cup (A - B)$ , so  $\Pr(A \cup B) = \Pr(B) + \Pr(A - B) = \Pr(B) + \Pr(A) - \Pr(A \cap B)$ .    **21. a)**  $\Pr(A) = \frac{1}{2}$ ,  $\Pr(B) = \frac{1}{3}$ ,  $\Pr(C) = 1 - \frac{1}{2} - \frac{1}{3} = \frac{1}{6}$ , so odds against  $C$  are 5:1. **c)**  $ps + qr + 2qs: pr - qs$     **23. a)** Two cases: Both  $A$  and  $B$  finite or at least one infinite. **c)**  $A_i = \{i\}$ .

### Section 6.3

**1. a)**  $.6 \times .5 \neq .4$  so No.    **b)**  $.8, 2/3$     **4.** By Bayes  $\Pr(A|B) = \frac{(.2)(.8)}{(.2)(.8) + (1-.2)(.3)} = .4$ .    **7. a)** With subscripts  $t, y$  for today and yesterday,  $\Pr(G_y|G_t) = \frac{\Pr(G_t|G_y)\Pr(G_y)}{\Pr(G_t|G_y)\Pr(G_y) + \Pr(G_t|B_y)\Pr(B_y)} = .7$ .    **9. a)**  $\Pr(A|B)\Pr(B) = .2$     **c)** Not unique; set  $\Pr(A \cap B) = .2$ ,  $\Pr(\sim A \cap B) = .3$ , then (i) set  $A = A \cap B$  (so that  $\Pr(A \sim B) = 0$ ), and (ii) set  $A = (A \cap B) \cup \sim B$  (so that  $\Pr(A \sim B) = 1$ ).    **12.**  $\Pr(\text{Dem}|\text{Wrote}) = .5$  (changed)    **13. a)** H (Pr=1/2), TH (1/4), TTH (1/8), TTT (1/8)    **b)**  $7/8, 1/4, 1/4$     **14.** Let  $F$  = penny fair,  $H$  = get 10 heads; by Bayes  $\Pr(\sim F|H) = \frac{10^{-6} \cdot .1}{10^{-6} \cdot .1 + (1 - 10^{-6})(1/2)^{10}} \approx .001$ .    **16. a)** Space  $S$  = all 11-long HT-sequences, equiprobable.  $E_1$  = all atoms of  $S$  with exactly 8 H's in first 10 positions;  $E_2$  = all atoms with H in 11th position.  $\Pr(E_2|E_1) = 1/2$ .    **b)** 10-long sequences;  $\Pr\left(\frac{9}{7}\right) \div \left(\frac{10}{8}\right) = 4/5$ .    **c)** Space is all 10-long HT-sequences.  $\Pr = \frac{4}{1} \div \frac{5}{2} = 2/5$ .    **18.** Let  $C$  = cheated,  $R$  = last 10 right. Use Bayes; if  $\Pr(C) = .001$ , then  $\Pr(C|R) = .001/(.001 + .999(\frac{1}{5})^{10}) \approx 99.99\%$ . If  $\Pr(C) = .0001$ , then  $\Pr(C|R) \approx 99.90\%$     **20. a)**  $F$ : 6 events with  $b_1$  first, 6 with  $b_2$  first.  $N$ : 6 events with  $b_1$  second, 6 with  $b_2$  second.    **21. a)** Sample space  $\{(B, B), (B, G), (G, B), (G, G)\}$ , where first entry is sex of older child; equiprobable measure. Then  $B_o$  = older child a boy =  $\{(B, B), (B, G)\}$ ;  $B_y$  = younger child a

boy =  $\{(B, B), (G, B)\}$ . So  $\Pr(B_y | B_o) = \Pr(B_y \cap B_o) / \Pr(B_o) = 1/2$ . **c)** Same sample space and measure. Let  $B_i$  = first child introduced is a boy =  $\{(B, B), (B, G), (G, B)\}$ . So  $\Pr(\text{other a boy} | B_i) = \Pr(\text{both boys}) / \Pr(B_i) = 1/3$ .

**23. b)** Root has degree 2, children of root have degree 3; each meal has  $\Pr = 1/6$ . **d)**  $\frac{1/6}{2/6} = 1/2$  (independent events).

**24. d)**  $\Pr(W \cap P) / \Pr(P) = 0/.25 = 0$ . **25.**  $\frac{2}{3}, \frac{2}{3} - \frac{1}{4} = \frac{5}{12}$ ; Yes **26.**  $\frac{5/12}{2/3} = \frac{5}{8} \leq \Pr(B|A) \leq 1$ ;  $\frac{5}{9} \leq \Pr(A|B) \leq \frac{8}{9}$ .

**28.** Atoms given by lists of courses as in (5); probability of each atom: 1/36. **29. b)** With  $M$  for Math,  $D$  for discrete,  $C$  for continuous,  $L$  for Language,  $G$  for German,  $R$  for Russian,  $S$  for Scientific and  $Re$  for Regular, atoms are  $MC, MD, LF, LGRe, LGS, LRRe, LRS$ . **d)**  $\Pr(S|L) = \Pr(GS|L) + \Pr(RS|L) = 1/4 + 1/4 = 1/2$ .

**30. a)**  $\prod_{n=30}^{49} p_n$  (note upper bound) **c)**  $(\prod_{n=30}^{64} p_n)(1-p_{65})$  **e)**  $(\prod_{n=60}^{64} p_n)(1 - \prod_{n=65}^{69} p_n)$  **32. a)**  $l_{50}/l_{30}$

**c)**  $(l_{65} - l_{66})/l_{30}$  **e)**  $(l_{65} - l_{70})/l_{60}$  **35.**  $\Pr(\sim T | \sim P) = \frac{\Pr(\sim T)\Pr(\sim P | \sim T)}{\Pr(\sim T)\Pr(\sim P | \sim T) + \Pr(T)\Pr(\sim P | T)} = (.9925)(.96)/[(.9925)(.96) + (.0075)(.08)] \approx .99937$ .

**37.**  $\Pr(T|P) \approx .802$ ;  $\Pr(\sim T | \sim P) \approx .986$ . **40. a)** To get  $B$  and  $C$  after  $A$  (LHS), first you have to get  $B$  after  $A$  and then  $C$  after  $B$  and  $A$  (RHS). **b)** LHS:  $\frac{\Pr(A \cap (B \cap C))}{\Pr(A)}$ ; RHS:  $\frac{\Pr(C \cap (A \cap B))}{\Pr(A \cap B)}$ .

**41.** Proof for  $\sim A$  and  $B$ :  $\Pr(B) = \Pr(A \cap B) + \Pr(\sim A \cap B)$ , so  $\Pr(\sim A \cap B) = \Pr(B) - \Pr(A \cap B) = \Pr(B) - \Pr(A)\Pr(B) = (1 - \Pr(A))\Pr(B) = \Pr(\sim A)\Pr(B)$ . **43.** Condition 1): If  $\emptyset \subset C \subset A$ , then  $0 \leq \Pr(C) \leq \Pr(A)$  [see 20, Section 6.2], so  $0 \leq \Pr_A(C) \leq 1$ . Condition 3): If  $C \cap D = \emptyset$ , then  $\Pr(C \cup D) = \Pr(C) + \Pr(D)$ , so divide by  $\Pr(A)$  to get  $\Pr_A(C \cup D) = \Pr_A(C) + \Pr_A(D)$  **45. 1)**: Since  $0 \leq \Pr(A \cap B) \leq \Pr(A)$ , then  $0 \leq \Pr(A|B) = \Pr(A \cap B) / \Pr(A) \leq 1$ .

3): If  $B, C$  disjoint, so are  $B \cap A$  and  $C \cap A$ . Thus  $\Pr((B \cup C) \cap A) = \Pr(B \cap A) + \Pr(C \cap A)$ . Now divide by  $\Pr(A)$ .

## Section 6.4

**1. a)** 1/90 **b)** 11/90 **4. a)**  $e^{-2}2^2/2! = 2e^{-2} \approx .271$ . **b)**  $5e^{-2} \approx .677$ . **c)**  $1 - 3e^{-2} \approx .594$ . **6.** Sample space: 0, 1, 2, 3 for number of heads;  $\Pr(0, 1, 2, 3) = .064, .288, .432, .216$ ; binomial with  $p = .6$ . **7. a)**  $1 - F(x)$

**c)**  $F(x_j) - F(x_i)$  **e)**  $F(x_j) - F(x_{i-1})$  **8. a)**  $B_{10,1/6}$  **c)** Poisson **e)** Negative Binomial (order 1),  $p = 6/36$ .

**9.** Negative Binomial,  $p = 1/2$ . **11.** Define  $\Pr'$  on  $R'$  such that if  $s \in S$  maps into  $s' \in R'$  and  $\Pr$  is probability measure on  $S$ , then  $\Pr'(s') = \Pr(s)$ . **13.** For  $\lambda = 1$ ,  $k = 0, \dots, 5$ , the values of  $e^{-\lambda} \lambda^k / k!$  are .368, .368, .184, .061, .015, .003. For  $\lambda = 4$ ,  $k = 2, \dots, 7$ , the values are .147, .195, .195, .156, .104, .060. **15.** For  $p = .3$ ,  $k = 1, 3, 5, 7, 9$ , the values of  $(k-1)(1-p)^{k-2} p^2$  are 0, .126, .123, .091, .059. For  $p = .5$ ,  $k = 0, 2, 4, 6, 8, 10$ , the values are 0, .25, .187, .078, .027, .009. **17.** Show using symmetry that  $\Pr(|X| < x) = 2F(x) - 1$ ; interpretation holds over entire domain of distribution. **18. b)** If  $\Pr(a) = \epsilon > 0$ , then, by continuity, for all  $x$  in some interval around  $a$ ,  $\Pr(x) \geq \epsilon/2$ . But there are infinitely many points in any interval so the sum of the probabilities at all these points would be  $> 1$ . Contradiction. **20.**  $B_{n,p}$  with  $p = r/(r+b)$  **22. a)**  $\frac{1}{10} \cdot 1$  [pick defective first] +  $\frac{9}{10} \cdot \frac{1}{9} = \frac{1}{5}$ . **b)** 7/15 **c)** Bayes says  $(\frac{1}{4})(\frac{7}{15}) / [(\frac{1}{4})(\frac{7}{15}) + (\frac{3}{4})(\frac{1}{5})] = \frac{7}{16}$ . **24. a)** Down if  $r < 2$ ;  $\Pr = .376$ . Same if  $r = 2$ ;  $\Pr = .302$ . **b)** Up, .370; same, .218; down .411. Less likely to exactly break even; see also [33]. **26.** .474 **28. a)** For  $k = 0, 1, 2$  the approximations are .368, .368, .184; the actual values (to 3 decimals) are .366, .370, .185. **30.**  $f_n(k, j) = \binom{n}{k, j, n-k-j} p^k q^j r^{n-k-j}$  [see trinomial coefficient, Section 4.6]. **32. a)** For  $k \leq n-1$ ,  $r_n \geq 1$  so  $a_k \leq a_{k+1}$  so sequence  $a_1, \dots, a_n$  is non-decreasing; similarly if  $k \geq n$ , sequence is decreasing. **c)** Assume  $0 < p < 1$ ; then  $r_k = \frac{n-k}{k+1} \frac{p}{1-p}$  is a decreasing function of  $k$ ; need first  $k$  such that  $r_k < 1$ , which is  $k = \lfloor (n+1)p \rfloor$ . **33. a)** For  $p = .5$ ,  $n = 10, 20, 50$ , the values of  $B_{n,.5}(n/2) = \binom{n}{n/2} .5^n$  are .246, .176, .112 (getting smaller). **c)** For  $p = .5$ ,  $n = 10, 20, 50, 100, 200$  the values of  $\sum_{k=-2}^2 B_{n,.5}(\frac{n}{2} - k)$  are .891, .737, .520, .383, .276 (getting smaller). **36.** Substitute Stirling's Formula into  $B_{n,p}(np) = \frac{n!}{(np)!(nq)!} p^{np} q^{nq}$  and simplify. Whatever constant  $p$  is,  $[1/\sqrt{2\pi np(1-p)}] \rightarrow 0$  as  $n \rightarrow \infty$ . This shows that the probabilities in [33abc] go to 0. **37.** For instance,  $\Pr(X+Y=4) = \Pr(X=1)\Pr(Y=3) + \Pr(X=2)\Pr(Y=2) + \Pr(X=3)\Pr(Y=1) = 3(\frac{1}{6} \cdot \frac{1}{6}) = \frac{1}{12}$ . So, if  $n$  is the number of ways to get sum of  $k$  on two dice,  $\Pr(X+Y=k) = n/36$ . **39. b)**  $\Pr(X_{1,p} + X_{1,p} = k) = \sum_{i=0}^{k-1} pq^{i-1} p q^{k-i-1} = \Pr(X_{2,p} = k)$ . **41.**  $x^2$  **43.** Eq. (14)  $\Rightarrow$  Eq. (15): (15) is the special case of (14) with  $A = \{x\}$ ,  $B = \{y\}$ . Converse:  $\Pr(X \in A \cap Y \in B) = \sum_{x \in A, y \in B} \Pr(X=x \cap Y=y) \stackrel{(15)}{=} \sum_{x \in A, y \in B} \Pr(X=x) \Pr(Y=b) = \sum_{x \in A} \sum_{y \in B} \Pr(X=x) \Pr(Y=b) = \sum_{x \in A} \Pr(X=x) \sum_{y \in B} \Pr(Y=b)$  [factor] =  $\Pr(X \in A) \Pr(Y \in B)$ .

## Section 6.5

1.  $E(X_A) = \sum x \Pr(X_A=x) = 0 \cdot \Pr(X_A=0) + 1 \cdot \Pr(X_A=1) = \Pr(A)$ .    3.  $\bar{x} = (x_1 + x_2 + \dots + x_n)/n$  where  $x_i = a + (i-1)\frac{b-a}{n-1}$ . Using summation formulas,  $\bar{x}$  simplifies to  $(a+b)/2$ .    5. a)  $-33.5\text{¢}$  b)  $-2\text{¢}$  (using 37¢ as the cost of a stamp)    8.  $w = 1/p$ .    10. a)  $A: -20\text{¢}$ ;  $B: -10\text{¢}$ ;  $C: -20\text{¢}$ ;  $D: -5\text{¢}$ . b)  $-13\frac{3}{4}\text{¢}$  c) If you split your dollar into bets of  $p, q, r, s$  on  $A, B, C, D$ , then your net return, respectively, if  $A, B, C$ , or  $D$  wins is  $2p-1, 3q-1, 4r-1, 9.5s-1$ . Setting these equal, one finds  $(p, q, r, s) = (114, 76, 57, 24)/271$ , and the guaranteed “winnings” are  $-43/271$ .    12. a) Ann  $2/3$ , Bill  $-2/3$ . b)  $\$2/3$  for each toss c) No payment; it's a fair game.    14. Let  $f(i)$  be the probability your  $i$ th guess is right. Then  $f(1) = f(2) = f(3) = 1/3$ . For instance,  $f(2) = \frac{2}{3} \cdot \frac{1}{2}$  (wrong on first guess  $\times$  right on second). So  $E(f) = 2$ .    15. c)  $\frac{1}{5}(1) + \frac{4}{5}(\frac{1}{4}) = \frac{2}{5}$  point. e) For a),  $\frac{1}{4}(1) + \frac{3}{4}(-\frac{1}{4}) = \frac{1}{16}$ . For b),  $\frac{n}{16}$ .    16.  $\sum_{k=0}^{n-1} \frac{n}{n-k} = n(1 + \frac{1}{2} + \dots + \frac{1}{n})$ .    17. \$1000 gain    19. Certificate: \$1000 certain gain; Bond: \$1100 expected gain.    21.  $E(X)$  is a constant, and for any constant  $c$ ,  $E(c) = c$ . So  $E(X-E(X)) = E(X) - E(E(X)) = 0$ .    22.  $\sum k \binom{n}{k} p^k q^{n-k}$  breaks into a sum of three terms:  $q \sum k \binom{n-1}{k} p^k q^{(n-1)-k}$ ,  $p \sum (k-1) \binom{n-1}{k-1} p^{k-1} q^{(n-1)-(k-1)}$ , and  $p \sum \binom{n-1}{k-1} p^{k-1} q^{(n-1)-(k-1)}$ .    25. Let  $S_n = p \sum_{k=1}^n k q^{k-1}$ . Then  $S_n = S_{n-1} + \frac{p}{q} n q^n$ . By Section 5.7 the solution is associated with the polynomial  $(r-1)(r-q)^2$ , so if  $q \neq 1$  the solution is of the form  $S_n = A + (B+Cn)q^n$ .    28.  $X_i$  satisfies the first-order negative binomial distribution, so  $E(N_{n,p}) = nE(N_{1,p}) = n/p$ .    31.  $\sum_{k \geq 0} k \lambda^k e^{-\lambda} / k! = \lambda e^{-\lambda} \sum_{k \geq 1} \lambda^{k-1} / (k-1)! = \lambda e^{-\lambda} e^\lambda = \lambda$ .    33. Both  $= k(1+n)/2$ . Let  $Z_i$  be the number on the  $i$ th ball picked. In both schemes, each  $Z_i$  is uniformly distributed, and  $X_k = \sum Z_i$  (with  $Z$ 's independent),  $Y_k = \sum Z_i$  (with  $Z$ 's not independent). But independence does not affect the mean of a sum.    35. If  $i < j$  then  $\Pr((X_i=1) \cap (X_j=1)) = \Pr(X_j=1) = q^{j-1}$ , but  $\Pr(X_i=1) \Pr(X_j=1) = q^{i+j-2}$ .    37. By definition:  $0 \cdot \frac{1}{8} + 1 \cdot \frac{3}{8} + 2 \cdot \frac{3}{8} + 3 \cdot \frac{1}{8} = \frac{3}{2}$ . By Eq. (16): For each atom,  $\Pr(e) = \frac{1}{8}$ . If we list the atoms alphabetically, HHH, HHT, ..., TTH, TTT, then  $E(X) = (3+2+2+1+2+1+1+0)/8 = 3/2$ .    39.  $X$  takes on values 1 through 6 with equal probability;  $(X-4)^2$  takes on values 0, 1, 4, 9 with probabilities  $\frac{1}{6}, \frac{2}{6}, \frac{2}{6}, \frac{1}{6}$ . By Eq. (17),  $E[(X-4)^2] = 0 \cdot \frac{1}{6} + 1 \cdot \frac{2}{6} + 4 \cdot \frac{2}{6} + 9 \cdot \frac{1}{6} = \frac{19}{6}$ . By Eq. (18),  $E[(X-4)^2] = (9+4+1+0+1+4)/6 = 19/6$ .    41.  $E(X|A) = \sum_{x \in \text{Range}(X)} x \Pr_A(X=x) = \sum_x x \Pr(\{X=x\} \cap A) / \Pr(A) = \sum_{e \in A} X(e) \Pr(e) / \Pr(A)$ .    43. a)  $7/2$  b)  $(1+4+9+16+25+36)/6 = 91/6 \neq (7/2)^2$ . c)  $(0^2 + 0 \cdot 1 + \dots + 1 \cdot 0 + 1 \cdot 1 + \dots + 6 \cdot 6)/36 = 49/4$ , which does equal  $(7/2)^2$  ( $X, Y$  independent).

## Section 6.6

1.  $\text{Var}(X) = p(1-p)$ ,  $\sigma_X = \sqrt{p(1-p)}$ .    5.  $\text{Var}(X+Y) = E[(X+Y-\bar{x}-\bar{y})^2] = E[(X-\bar{x})^2] + E[(Y-\bar{y})^2] + 2(E(XY) - \bar{x}E(Y) - \bar{y}E(X) + \bar{x}\bar{y}) = \text{Var}(X) + \text{Var}(Y) + (\text{terms} = 0 \text{ by independence})$ .    7.  $\text{Var}(\frac{1}{n} \sum X_i) = (1/n^2) \text{Var}(\sum X_i) = \text{Var}(X)/n$ .    8. a) If  $X$  counts number of 6's,  $E(X) = 1$ ,  $\text{Var}(X) = 5/6$ ,  $\sigma_X = .913$  and  $\frac{1}{\sigma_X} = 1.095$ . If  $|X-1| < 1 = 1.095\sigma_X$ , then  $X \neq 0$ . By Chebyshev,  $\Pr(|X-1| < 1.095\sigma_X) \geq 1 - 1/(1.095)^2 = .166$ . Therefore,  $\Pr(X \neq 0)$  is at least .166.    9. b) If  $X$  counts the number of an odd number of dots,  $E(X) = 300$ , and with  $p=.5$ ,  $\text{Var}(X) = pqn = 150$ ,  $\sigma_X = 12.25$ . Actual count 21 or more away from mean is  $1.71\sigma_X$ . So  $\Pr(|X-300| > 1.71\sigma_X) \leq 1/(1.71)^2 = .342$ .    11. By implicitly assuming, as in [5], that  $X$  is independent of itself which is clearly not true.    13. Since the throws are independent, the variance of the sum equals the sum of the variances which, using Example 2, is  $35n/12$ .    15. From [1] the variance of a single Bernoulli trial is  $p(1-p) = pq$ . Since the  $n$  trials are independent, the binomial variance is  $npq$ .    17. b) Proof is precisely parallel to that of Theorem 3 by changing appropriate strict inequalities to nonstrict ones and vice versa.    19. a) Let  $X$  count the number of heads in 2 throws. Then  $\Pr(X=0, 1, 2) = \Pr(X^2=0, 1, 4) = 1/4, 1/2, 1/4$ . Therefore,  $E(X) = 1$ ,  $E(X^2) = 3/2$  and by [18],  $\text{Var}(X) = 1/2$ .    20. a) Eq. (14) contains a parameter,  $k$ , which, when set to 0, gives the result in [18]. c) Dispersion is minimum when  $k = E(X)$ , in which case it equals the variance, and increases geometrically as  $k$  moves in either direction.    23. c) You should get  $q/p^2$ .    25.  $\text{Var}(X) = p(1 - \frac{1}{p})^2$  [case 1] +  $qE([X+1-E(X)]^2)$  [case 2] =  $(q^2/p) + q(\text{Var}(X)+1)$  from [21]. Solving:  $\text{Var}(X) = q/p^2$ .

## Section 6.7

1. Probability    3. With  $89.5 \leq X \leq 110.5$ , then  $-1.15 \leq Z = \sqrt{(6/5)(X-100)/10} \leq 1.15$ . From Table 6.1, get  $\Pr(Z < 1.15) = .8749$  so  $\Pr(|Z| < 1.15) = 2(.8749 - .5) = .7498$ , very close to .7501.    5. Let  $Y = h/10000$ . Then from

(2),  $Z = (h/50) - 100$ ,  $\Pr(|Z| < .2) = .1586$ ,  $\Pr(.4 < Z < 2) = .3218$ . **7. a)** Since mean = 0, variance = 1,  $\Pr(|Z| \leq 2) = .75$ . **b)** Chebyshev holds for *any* random variable so you expect a better result when you know precisely the form of the random variable. **8. b)**  $n \geq 752$  **d)**  $n \geq 16577$  **10.** Need  $.52 - p \leq .02$ . 2 is still a good approximation to  $\sqrt{1/pq}$  for any  $p$  in the range [.4, .6] so want  $Z < 2(.52 - p)\sqrt{1000} < 1.264$ . From Table 6.1, probability is .8969, so about .9 that a majority of entire population would answer Yes. **12.** Without:  $E(X) = .8$ ,  $\text{Var}(X) = .36$ ; with:  $E(X) = .8$ ,  $\text{Var}(X) = .48$ .

## Section 6.8

**1.**  $m$  is initialized to  $x_1$ . **2. a)** For  $(a, b) = (0, 0), (0, 1), (1, 0), (1, 1)$  both sides are, respectively,  $(i-1)(j-1)/ij$ ,  $(i-1)/ij$ ,  $(j-1)/ij$ ,  $1/ij$ . **4. a)** Best 2, worst  $2n-2$ , average  $2 + \sum_{k=3}^n (3/k)$  **b)** Best  $n-1$ , worst  $2n-3$ , average  $2n-3 - \sum_{k=3}^n (1/k)$  **6.** 1/3 cm; 365/3 cm **9.** With  $Y, Z$  as in the text discussion of sequential search,  $X = Y$  when restricted to  $A$  (because  $Z = 0$ ) and  $X = k + Z$  when restricted to  $\sim A$  (because  $Y = k$ ). So  $E(X|\sim A) = k + E(Z|\sim A) = k + E_{n-k}$ . **13.** Let event  $A$  be “success on first trial”. Use Theorem 1. **15. a)** With  $a_k = A_{2k}$  we get  $a_k = 1 + a_{k-1}$ ;  $a_0 = 1$ . The solution is  $a_k = k+1$ , or  $A_n = 1 + \log n$ . This solution is high for all  $n > 1$ , because actually  $a_k < 1 + a_{k-1}$ . **b)** Let  $b_k = A_{2k-1}$ . Then  $b_k = 1 + \frac{2^k-2}{2^k-1}b_{k-1}$ , with  $b_1 = 1$ , is exact for  $n = 2^k - 1$ . **c)**  $b_k = k-1+k/(2^k-1)$  **16.** It makes multisets with two distinct elements twice as likely as multisets with only one distinct element. **18. a)**  $12 \rightarrow 312, 132, 123; 21 \rightarrow 321, 231, 213$ . **b)** Putting  $n$  in the middle wipes out all temporary maxes to its right, so the number of maxes in the derived permutation does not have a constant relation to the number in the starting permutation. (A recursion is possible if we keep track of how many permutations have  $k$  temporary maxes in the leftmost  $j$  positions, but it's complicated: If  $\text{Perm}(n, j)$  is the number of permutations of  $n$  things  $j$  at a time, one can obtain  $P_{n,k} = \sum_{j=0}^{n-1} \text{Perm}(n-1, n-j-1)P_{j,k-1}$ , and this can be reduced to the recursion in e) below.) **c)**  $12 \rightarrow 231, 132, 123; 21 \rightarrow 321, 312, 213$  **e)** The derived permutation has the same number of temporary maxes as the starting permutation if the appended digit is  $< n$ , and one more if  $n$  is appended. Thus  $P_{n,k} = (n-1)P_{n-1,k} + P_{n-1,k-1}$ . **20.**  $p_{n,k} = \frac{n-1}{n}p_{n-1,k} + \frac{1}{n}p_{n-1,k-1}$ .

## Section 6.9

**1.**  $2/3$  **2. a)**  $p_1 = \sum_{k=0}^{\infty} (5/6)^{2k} (1/6) = (1/6) \sum_{k=0}^{\infty} (25/36)^k$ . **5.**  $(p_1, p_2, p_3) = \frac{1}{91}(36, 30, 25)$ . **7.**  $E = (11/36)E + (25/36)(E+2)$ , which implies  $E = 61/11$ . The probability that Player 1 gets 6 on the first toss *given* that she wins is  $(1/6)/(6/11) = 11/36$ . The probability that she does not get a 6 on the first toss given that she wins must be complementary:  $25/36$ . **9.** Let  $p_i$  be the chance that Player  $i$  wins. Then  $p_1 = q/(1+q^2)$ , which is  $2/5$  when  $p = q = 1/2$ . Obtained from the recursion  $p_1 = qp_2 + pqp_1$  and  $p_2 = 1 - p_1$ . **11.** Some values of  $(p, pg)$  are: (.3, .099), (.4, .264), (.45, .377), (.49, .475), (.5, .5), (.51, .525), (.6, .736). **13.**  $p^3 + C(3, 1)p^3q + C(4, 2)p^3q^2$  **15.** In all three cases,  $E = 2/(p^2 + q^2)$ . **17.**  $\frac{1}{2}p_*^{21} + \frac{1}{2}pp_*^{21}$ , where  $p_* = p/(1-pq) = \sum_{k \geq 0} (qp)^k p$  is the probability that  $A$  is the first to win a point if  $A$  serves first. **19. a)**  $\sum_{k \geq 0} (qp)^k p = p/(1-qp')$ , where  $q' = 1 - p'$ . **b)**  $p'p/(1-qp')$ . Or, solve both parts simultaneously by recursion. **21.** Let  $g = (k-m)/2$ , so that  $g+(g+m) = k$ . If  $g \notin N$ , then no ways; else  $C(k, g)$  ways (choose which  $g$  games  $A$  wins). Probability is  $C(k, g)p^g q^{g+m}$ . **22.**  $P_m = 1 - \frac{m}{m+n}$ , so in Example 5,  $P_3 = 5/8$ . **24. b)**  $P_m \approx 1 - \bar{r}^n$ . **d)**  $1 - \bar{r}^{m+n} < 1$ , so  $P_m > 1 - \bar{r}^n \geq 1 - \bar{r}^{n_0}$ . For  $p = .49$  and  $n_0 = 100$ ,  $P_m \geq 1 - (.49/.51)^{100} \approx .9817$ . **25.** No. For  $P_m$  to be  $\leq .01$ , [22] shows that we need  $m/(m+n) \geq .99$ , or  $m \geq 99n$ . So the needed  $m$  increases with  $n$ . **27.** The recursion is  $E_k = 1 + pE_{k+1} + qE_{k-1}$ ,  $E_0 = E_{m+n} = 0$ . Solution:  $E_m$  where  $E_k = \frac{m+n}{(p-q)(1-r_*^{m+n})}(1-r_*^k) + \frac{k}{q-p}$ . **29. a)** Multiply top and bottom of Eq. (8) by  $p^{2n}$  and note that  $p^{2n} - q^{2n}$  is the difference of two squares. **b)** Let  $p_i$  = probability Player  $i$  is ruined. There is a one-to-one correspondence between win-loss sequences where Player 1 is ruined and those where Player 2 is ruined: switch wins and losses. Therefore  $p_1/p_2 = q^n/p^n$  (why?). Also,  $p_1 + p_2 = 1$ . **31. a)**  $N_{k,m,n} = N_{k-1,m+1,n-1} + N_{k-1,m-1,n+1}$ ;  $N_{k,0,n} = 0$  if  $k > 0$ , 1 if  $k = 0$ ;  $N_{k,m,0} = 0$  for  $k \geq 0$  and  $m > 0$ . This implies  $N_{k,m,n} = 0$  if  $m > k$ , which is convenient to use in calculations. **b)**  $N_{5,3,5}$  expands to 3:  $A$  is ruined in five steps if she wins just once in the first three and loses the next two. **c)** 41 **d)**  $\sum_{k \leq 10} N_{k,3,5}(.5)^k \approx .344$  (calculator). **33. a)** The diagonal  $p_{4,0}, p_{3,1}, p_{2,2}, p_{1,3}, p_{0,4}$ , for instance, is  $0, 1/8, 4/8, 7/8, 1$ , which is  $1/8$  times the running sum of the row  $1, 3, 3, 1$  from Pascal's Triangle. **b)** We want  $p_{N,N}$  where  $p_{m,n} = \sum_{k \geq m} \binom{m+n-1}{k} p^k q^{m+n-1-k}$ . **34. a)** The mapping truncates each scorecard in

$G'$  just after the point where  $A$  won for the  $m$ th time or  $B$  won for the  $n$ th. For each scorecard  $S$  in  $G$ , the set of scorecards in  $G'$  that map to  $S$  have together the same probability as  $S$ , because any sequence of results for the remaining points up to  $m+n-1$  is possible. **37. a)** Probability of winning the next point is greater if you have the serve whatever the probability of winning the serve. **c)**  $p_{0,m,n} = pp_{0,m-1,n} + qp_{1,m,n}$ ;  $p_{1,m,n} = pp_{0,m,n} + qp_{1,m,n-1}$ . Use these to express  $p_{0,m,n}$ ,  $p_{1,m,n}$  solely in terms of smaller cases. **39. a)** Let  $p_k$  be the probability of being or returning home if he starts at point  $k \in Z$ . So  $p_0 = 1$  (he's already there), and  $p_k = \frac{1}{2}(p_{k+1} + p_{k-1})$ . Solution is  $p_k = A + Bk$ ;  $A = 1$  since  $p_0 = 1$ ,  $B = 0$  since  $0 \leq p_k \leq 1$  for all  $k$ . So  $p_k = 1$  for all  $k$ . **b)** Infinity! Let  $E_k$  be the expected time until first being home if he starts at  $k$ . Then  $E_k = 1 + \frac{1}{2}(E_{k+1} + E_{k-1})$ ,  $E_0 = 0$ . There are no finite solutions that are always nonnegative.

## Supplementary Problems: Chapter 6

1. If everyone in the population says "I have tuberculosis", then .001 of them are correct. **2. a)** For  $m > n$ ,  $(\frac{3}{4})^{n-1}(\frac{1}{6})(\frac{11}{12})^{m-1-n}(\frac{1}{12})$ ; for  $m = n$ , 0; for  $m < n$ ,  $(\frac{3}{4})^{m-1}(\frac{1}{12})(\frac{5}{6})^{n-1-m}(\frac{1}{6})$ . **c)**  $\sum_{k \geq 1} f(k) < 1$ ; the "distribution" doesn't cover all possibilities. **3. a)** 1/2 **b)** 1/4 **c)** 1/2 **d)** 1/3 **4.**  $pg = p^4 + C(4,1)p^4q + C(4,1)p^4q^2$  [to reach 4-2 but not 2-2] +  $2C(4,1)p^3q^3p^*$  [to reach 3-3 but not 2-2 via 3-1 or 1-3] +  $C(4,2)p^2q^2p^*$  [to reach 2-2]; messy algebra to show same as Eq. (4) in Section 6.9. **5. a)** Now LWW results in a win. **b)**  $\sum_{k \geq 0} p^{\lfloor k/2 \rfloor} q^{\lceil k/2 \rceil} p^2 = \sum_{m \geq 0} (p^m q^m + p^m q^{m+1}) p^2 = (\frac{1+q}{1-pq}) p^2$ . **c)**  $P_p = p(p+P_q)$ ,  $P_q = qP_p$ . **d)** Win-by-2 is better. Algebra shows that  $\frac{1}{1-2pq} - \frac{1+q}{1-pq} = \frac{q^2(2p-1)}{(1-2pq)(1-pq)}$  and  $2p-1 > 0$  for  $p > .5$ . Thus  $\frac{1}{1-2pq} > \frac{1+q}{1-pq}$  for all  $p > .5$ . **6.**  $\Pr(X > k_0) = \Pr(\text{first } k_0 \text{ trials don't fail}) = q^{k+0}$ . So  $\Pr(X=k | X > k_0) = q^{k-1}/p/q^{k_0} = q^{(k-k_0)-1}p = \Pr(X=k-k_0)$ . **8. a)** By Eq. (18) in [38, Section 6.5],  $E(U(X)) = \sum_x U(x)\Pr(X=x) = a \sum x\Pr(X=x) + b \sum \Pr(X=x) = aE(X) + b$ . So if  $X$  represents, say, the possible monetary outcomes of investing your money in some stock, and  $Y$  represents an alternative, then  $E(U(X)) > E(U(Y)) \iff E(X) > E(Y)$ . **b)** The value of winning \$9999.75 would be relatively less and the value of losing 25¢ would be more negative. So  $n$  must be even smaller to make the contest worthwhile.
10.  $\sum_{k=51}^N \left( \prod_{n=51}^k p_n \right) 5000$ , where  $N$  is an upper bound on human age, usually set at 100. The most direct (but not efficient) translation of this formula into an algorithm is a double loop with  $P \leftarrow P * p_n$  inside the inner loop and then  $S \leftarrow S + 5000 * P$  in the outer loop. **12.**  $E(X_i^2) = p$ ,  $E(X_i X_j) = p^2$ , so  $E(B_{n,p}^2) = np + (n^2 - n)p^2$ .
14.  $E(B_{1,p}^2) = p$  and  $E(B_{n+1,p}^2) = qE(B_{n,p}^2) + pE([1+B_{n,p}]^2) = E(B_{n,p}^2) + 2pE(B_{n,p}) + p = E(B_{n,p}^2) + 2np^2 + p$ . By induction or sum formulas,  $E(B_{n,p}^2) = n(n-1)p^2 + np$ . **16.** Need to evaluate  $p^{15}(1-p)^{10}$ . Ratio for  $p = .5, .7$  is  $.5^{25}/(.7^{15} \cdot .3^{10}) = 1.063 = .515/.485$  so probability that coin is fair is .515. **18.** Since  $1/\sqrt{pq} = 2.18$  for biased coin, we wish  $2\sqrt{n}(\bar{p}-.5) = 2.18\sqrt{n}(.7-\bar{p})$  from which  $\bar{p} = .6043$ . Use equation for either coin to guarantee 95% confidence level and find  $n = 62$ . **20.** Surely any  $\bar{p} \leq .5$  implies a fair coin but also  $.5 < \bar{p} \leq .6$  would lead to acceptance of the null hypothesis that coin is fair using solution to [19]. **22. a)** 1/3 **c)** 1/6 **e)** 1/3 **g)**  $\frac{1}{9}/(\frac{1}{9} + \frac{1}{3}) = 1/4$ . **23.** From the text,  $p_A = p/(1-pq)$ ,  $p_B = p^2/(1-pq)$ . Next,  $p_A^* = p_A^2 + p_A(1-p_A)p_B^* + (1-p_A)p_Bp_A^*$ . Similarly,  $p_B^* = p_Bp_A + p_B(1-p_A)p_B^* + (1-p_B)p_Bp_A^*$ . Now solve simultaneously. **26. a)** Let  $p_i$  = probability bug gets to  $C$  from vertex  $i$  without passing  $B$ . Then  $p_A = \frac{1}{2}p_D$  and  $p_D = \frac{1}{2}p_A + \frac{1}{2}$ , so  $p_A = 1/3$ . Or  $\sum_{k \geq 0} (1/4)^k (1/4) = 1/3$  [ $k$  trips ADA followed by DC]. **b)** 4 **c)** 3 **28. a)** Each node has two children except perhaps at next to last level; if complete with  $k$  levels, there are  $n = 2^k - 1$  nodes so  $k = \log_2(n+1)$ ; if not complete at level  $k$  then ceiling gives correct value since  $2^{k-1} - 1 < n < 2^k - 1$ . **b)** Using the method of Section 5.7 with  $n = 2^k - 1$ ,  $A_n = (1/n) \sum_{i=1}^k i2^{i-1} = k - 1 + k/(2^k - 1)$ . **c)** For first  $k-1$  levels get  $nA_n = (k-2)2^{k-1} + 1$ ; at  $k$ th level there are  $n - (2^{k-1} - 1)$  nodes so  $nA_n = (k-2)2^{k-1} + 1 + k[n - (2^{k-1} - 1)]$ . Thus  $A_n = k + (k+1)/n - 2^k/n$ .

## Chapter 7

### Section 7.1

1. Because it will never terminate since  $u$  is always nonnegative but  $x$  is negative. **3. c)** Not valid; can't assume

Illinois is in USA. **4. b)** Not sound; London is not in Scotland so hypothesis not true. **c)** Not sound because not valid; hypotheses true but can't infer from them that all of England is in the UK.

## Section 7.2

**1.** I may have meat; otherwise I'll have fish. **2.** I deny that I like discrete math. **5. a)**  $P \mid Q$  is true unless both  $P$  and  $Q$  are true;  $P \downarrow Q$  is true only if  $P$  and  $Q$  are both false. **b)**  $P \not\Rightarrow Q$  is true only if  $P$  true,  $Q$  false. **c)**  $P \not\Rightarrow Q$  is true only when  $P, Q$  have different truth values. **6. a)**  $P \mid Q = \neg(P \wedge Q); P \downarrow Q = \neg(P \vee Q)$ . **b)**  $P \not\Rightarrow Q = P \wedge \neg Q; P \not\Rightarrow Q = \neg P \wedge Q$ . **7.** See answer to [6a]. **9. a)** True when  $P = \text{"4 is an integer"}$  and  $Q = \text{"4 is an even integer"}$ . False when  $P = \text{"10 is composite"}$  and  $Q = \text{"3 is a prime"}$ .  $P$  because  $Q$  not logical operator because doesn't have a truth value that depends solely on the truth values of  $P$  and  $Q$ . **10. a)**  $\neg J \quad d) \quad J \wedge M$  **g)**  $\neg J \wedge \neg M$  **j)**  $\neg J \vee \neg M$  (if both are allowed not to be here); otherwise  $\neg(J \Leftrightarrow M)$  **11. a,d)**  $R \Rightarrow C$ , **g)**  $C \Rightarrow R$  (if interpretation is clouds are sufficient for rain but really ambiguous) **12. c)**  $x = 2 \Rightarrow x^3 = 8$  **f)**  $x^3 = 8 \Rightarrow x = 2$  **i)**  $x^2 \neq 4 \Rightarrow x = 2$  or  $\neg(x^2 = 4) \Rightarrow x = 2$  **13. b)**  $(xy = 0) \Leftrightarrow ((x = 0 \vee (y = 0))$  **d)**  $x = 3 \Rightarrow x > 0$  **15.** Here are two which require four applications:  $\neg\neg P \wedge P, P \wedge \neg P \wedge P$ . **17. a)** OK as wff using Rules 1 and 3 but ambiguous since  $(P \wedge Q) \vee R$  and  $P \wedge (Q \vee R)$  have different truth tables. **c)** Not a wff **19. a)** Only  $\Rightarrow$  is not associative. **20. a)** Consider all possible symbols following  $\neg$  and show that no ambiguity could result.

## Section 7.3

**1. a)** True only when  $(P, Q, R)$  is  $(T, T, T), (T, T, F), (T, F, F)$ . **3. e)** Not a tautology **f)** Tautology **5. a)** First doesn't logically imply second. **7.** Use a truth table to show that  $P \wedge \neg P$  is always false. **9.** Show  $(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$  is a tautology;  $\neg B \Rightarrow \neg A$  is the contrapositive. **10. b)** Invalid because from what is given you don't know that Illinois is in USA. **11. b)** Invalid; if  $A, C$  true and  $B$  false, premise is true but conclusion false **12. a)** To go from  $n$  to  $n+1$ , apply induction hypothesis and then use basis case  $(\neg\neg P \Leftrightarrow P)$  **13. a)** Tautology  $\neg$  always true **c)** Tautology  $\neg$  premise always false **15. a)** Here is a procedure to generate the next row of atoms having started with all atoms  $T$ :

```

 $j \leftarrow n$ 
repeat while  $j > 0$  and  $P_j = F$  [Change F to T right to left]
   $P_j \leftarrow T; j \leftarrow j - 1$ 
endrepeat
if  $j = 0$  then end  $\leftarrow$  true [j = 0, then previous row was last]
else  $P_j \leftarrow F$  [Rightmost T changed to F]
endif

```

## Section 7.4

**1.** An assertion is normally a statement of something for which a proof is unnecessary while a hypothesis is something assumed true for the purposes of argument. **3. IS:**  $x \neq 0$ ; **OS:**  $y = 1/x$ ; **Proof:** Algorithm computes  $y$  as  $1/x$  which can be done as long as  $x \neq 0$  **5. b)** **IS:**  $w_1 \leq w_2 \leq w_3 \leq \dots \leq w_n$  (see Section 7.6 for a better way to express this). **6.** For [5c],  $1 \leq \text{Perm}(i) \leq n, i = 1, \dots, n \wedge \text{Perm}(i) \neq \text{Perm}j$  if  $i \neq j$ . **7. a)** **IS:**  $\{n \geq 0\}; i \leftarrow n; even \leftarrow \text{true}; \{2 \mid (n-i) \wedge i \geq 0\}$ ; **repeat while**  $i \geq 2; i \leftarrow i-2$ ; **endrepeat**;  $\{2 \mid (n-i) \wedge 0 \leq i \leq 1\}$ ; **if**  $i = 1$  **then**  $even \leftarrow \text{false}$ ; **OS:**  $\{(2 \mid n \wedge even = \text{true}) \vee (2 \mid n \wedge even = \text{false})\}$ ; proof needs to show only that loop invariant is true on each entry and that loop termination condition is true.

## Section 7.5

**1. a)** (i) and (ii) immediate setting  $a = 0, 1$ ; (iii) and (iv) also immediate setting  $(a, b) = (0, 0), (0, 1), (1, 0), (1, 1)$ . **2. b)**  $a(b+c) = ab+ac$  [Part (vii) of Theorem 1]. **3. b)** With  $U$  for the universe:  $U \vee A = A$ ; not a theorem. **4. a)** First part:  $a = a + 0$  [(i) in Theorem 1];  $\bar{0} = \bar{0} + 0$  [Substituting  $\bar{0}$  for  $a$ ];  $\bar{0} = 0 + \bar{0}$  [(iii)];  $\bar{0} = 1$  [(ix) with  $a = 0$ ]. Second part:  $\bar{1} = \bar{1} \cdot 1$  [(ii)];  $\bar{1} = 1 \cdot \bar{1}$  [(iv)];  $\bar{1} = 0$  [(x)]. **6. a)**  $1 + 1 = 1$  **b)** Let  $a = 0$ ; by (i,iv) of Theorem 1 (and this theorem is true in every Boolean algebra),  $0 + x = x + 0 = x$ . So if  $0 + x = 0$ , then  $x = 0$ . **9. a)**  $\bar{p} + q$  has same truth table as implication. **b)** Same truth values;  $\bar{p} + q = \bar{p} + (p + \bar{p})q = \bar{p} + \bar{p}q + pq = \bar{p}(1 + q) + pq = \bar{p} + pq$ . **11. c)** For [10b]:  $(p+r)(q+\bar{r}) = (\bar{p}+r)(\bar{q}+\bar{r})$  **12. b)**  $p(p+q) = pp+pq$  [(vii)] =  $p+pq$  [Eq. (3)] =  $p$  [12a] **d)**  $p(\bar{p}q) = (\bar{p}p)q$  [(vi)] =  $0q$  [(x)] =  $0$  [4d] **14. a)** By induction **b)**  $(p+q)(p+r) = (p+q)p + (p+q)r$  [(vii)] =  $p+pr+qr$  [(iv), [12b]], [(vii)] =  $p+qr$  [12a]; then generalize using induction; **15. a)** For each 1 in right hand column, write product

of variables, where there are 1's in the row, times the product of the complements of variables, where there are 0's in the row. **17.b)** DNF:  $pqr + p\bar{q}r + \bar{p}qr + \bar{p}\bar{q}r + pq\bar{r} + p\bar{q}\bar{r}$ ; first four terms give  $2 \times 2$  region which simplifies to  $r$ , and last two terms give  $2 \times 1$  region which simplifies to  $p\bar{r}$ , so expression simplifies to  $r + p\bar{r}$

**19.c)**  $\bar{p}\bar{q}r + \bar{p}qr + p\bar{q}\bar{r} + pq\bar{r} + pqr$    **20.**  $pq + p\bar{q}r$  or  $pq\bar{r} + pr$ ; neither is simpler   **22.a)**  $4 \times 8$  array with left and right labels as on  $4 \times 4$ ; on top duplicate  $q$  in  $4 \times 4$  and similarly on bottom for  $s$ ; add on bottom:  $t\bar{t}t\bar{t}t\bar{t}t\bar{t}$

**b)**  $2 \times 2 \times 2$  cube with  $p, \bar{p}, q, \bar{q}, r, \bar{r}$  on the perpendicular sides; symmetric in each of the three variables whereas two-dimensional map is not; easier to determine simplification also   **24.b)**  $\bar{p} + pqr + pq\bar{r}s$    **26.** Use [25] and (xii) of Theorem 1. **29.a)** Need to pay attention to 0 rows since all other terms in product are 1; for each row, sum the variables, where the row entry is 0, plus the sum of the variable complements, where the row entry is 1.

**30.a)**  $(p + \bar{q} + r)(p + \bar{q} + \bar{r})(\bar{p} + q + r)$ ; **b)** Use  $(a+b)(a+\bar{b}) = a$ . **31.** [17b] becomes  $(p+q+r)(p+\bar{q}+r) = (p+r)$ .

**34.**  $c_1 = \bar{b}_1$ ;  $c_2 = (b_1 + b_2)(\bar{b}_1 + \bar{b}_2)\bar{b}_8$ ;  $c_4 = b_1 b_2 \bar{b}_4 + (\bar{b}_1 + \bar{b}_2)b_4$ ;  $c_8 = b_1 b_2 b_4 + \bar{b}_1 b_8$ .

## Section 7.6

**2.a)**  $i = 0, 1, \dots, 16$ ; **c)** Either  $i = 0, 1, \dots, 9$  or  $j = 21, 22, \dots$  or both   **4.b)**  $(\forall i : 1 \leq i \leq m)(\forall j : 1 \leq j \leq m)$   $i \neq j \Rightarrow n_i \neq n_j$    **6.a)** Each nonzero real number has a multiplicative inverse (i.e., for each nonzero real number there is another such that the product of the two is 1). **d)** There is a number [in fact, 0] such that  $x$  plus that number is  $x$  for any  $x$ ; **g)** If two numbers are not equal, there is a number in between them. **7.a)**  $(\forall x)(\exists y)x+y=0$ ;

**d)**  $(\forall x)(x > 0 \Rightarrow (\exists y)(y > 0 \wedge y^2 = x))$ ; **g)**  $(\forall x)x \neq 2x$  [of course, not true since false for  $x=0$ ]   **9.a)** For each integer, there exists an integer one greater; **d)** For each integer, there exists an integer which is its square;

**g)** If  $i$  divides  $j$  and  $j$  divides  $k$ , the floor of the floor of  $k/j$  divided by  $i$  is the same as the floor of  $k$  divided by the floor of  $j$  divided by  $i$ ; false; for example,  $i=4, j=8, k=16$    **10.a)**  $(\exists m : m \geq 0)(\forall n : n \geq 0)[n \geq m]$ ;

**d)**  $(\forall n : 0 \leq n \leq 100)(\exists i)2^i < n < 2^{i+1}$ ; **g)**  $(\forall n)(\forall m)2|n \wedge 2|(m+1) \Rightarrow 2|mn$    **12.a)**  $(\forall x, y \in S)x \neq y \Rightarrow f(x) \neq f(y)$  or  $(\forall x, y)(S(x) \wedge S(y) \wedge x \neq y \Rightarrow f(x) \neq f(y))$ . **b)**  $(\forall x \in S)f(x) = c$  or  $(\forall x)S(x) \Rightarrow f(x) = c$ .   **13.b)** Let  $s$  be a subset of  $S$  and  $n, m$  be integers in  $N$ ;  $(\forall s \in S)(\exists n \in N)(m \in s \Rightarrow m \geq n)$ .   **14.a)** Let  $S$  be set of squares,  $R$  the set of rectangles, let  $P(r)$  be true if quadrilateral  $r$  has four right angles;  $[(\forall s \in S)(s \in R) \wedge (\forall r \in R)P(r)] \Rightarrow [(\forall s \in S)P(s)]$ .   **16.** To prove  $(\forall n \in N)P(n)$ , it suffices to prove  $P(0) \wedge P(1) \wedge (\forall n \in N)[P(n) \Rightarrow P(n+2)]$ .   **18.b)** If  $P$  is true and some  $Q(x)$  is true, then both sides are true; if all  $Q(x)$  are false and  $P$  is true, both sides are false; if  $P$  is false, both sides are true so in all cases both sides have the same truth value.   **19.b)** For [17b]:  $[\forall x(P_x \Rightarrow (Q_x \vee R_x))] \Rightarrow [\forall x(P_x \Rightarrow Q_x)] \vee [\forall x(P_x \Rightarrow R_x)]$ .   **20.a)** Not valid only if RHS can be false when LHS is true; RHS false only if all  $P_x$  true,  $Q$  false; but then LHS false also; therefore, valid. **b)** Suppose  $P_y$  is true,  $P_z$  is false,  $Q$  false; then LHS is true ( $F \Rightarrow F$ ) but RHS is false ( $P_y \Rightarrow Q$  is false); therefore, not valid.

**22.a)**  $(\forall m : 2 < m < 10)(\forall n : 1 < n < m)P(m, n)$ ; **c)**  $(\forall m : 1 \leq m \leq 5)(\forall n : m \leq n \leq 5)P(m, n) \wedge (\forall n : 1 \leq n \leq 5)P(0, n)$

**25.** Argue basis case (single quantifier) directly;  $n \Rightarrow n+1$  quantifiers: Apply induction hypothesis to first  $n$  and then use basis case   **a)** For [6a]:  $\neg(\forall x)(\exists y)(x \neq 0 \Rightarrow xy = 1) \Leftrightarrow (\exists x)(\forall y)\neg(x \neq 0 \Rightarrow xy = 1)$ .   **27.a)**  $x, y$  free; **d)** None free.   **28.** True for all nonnegative  $z$ . (All  $x, y \in R$  have a corresponding  $z$ .)

## Section 7.7

**1.** When  $j = 0$  value is smaller than all previous and is put in first position with other  $i - 1$  moved up one place which leaves outer loop invariant satisfied   **3.** Outer loop is vacuous since  $m = 1$  so algorithm does nothing   **5.a)** Whatever element is largest initially, once it starts being compared with other elements, it always “wins”; **b)** Same argument; stop when comparing second to last and next to last elements; **c)** By induction.

**7.a)** Should require  $n - 1$  passes; **b)** Stop whenever no interchanges are made during a pass.   **9.a)** Outer loop:  $0 < i \leq m - 1 \wedge (\forall j : i + 2 \leq j < m)n_j \leq n_{j+1} \wedge (\forall k : 1 \leq k \leq i + 1)n_k \leq n_{i+2}$  (with  $n_{m+1} = \infty$ ); Inner loop:  $(\forall k : 1 \leq k \leq L)n_k \leq n_{L+1}$  where  $L$  measures how far down the list the comparison has proceeded; **c)** Outer loop:  $(\forall j : 2 \leq j < m)n_j \leq n_{j+1} \wedge n_1 \leq n_2$ ; Inner loop: Replace  $L$  by  $i$  in loop invariant.

## Section 7.8

**1.a)**  $((A \Rightarrow B) \wedge \neg B) \Rightarrow \neg A$    **b)** LHS in a) has truth values F, F, F, T (with usual order of atoms), RHS has F, F, T, T, so implication true in every case.   **2.c)**  $P \Rightarrow Q$  [Premise];  $Q \Rightarrow R$  [Premise];  $\neg R$  [Premise];  $\neg Q$  [Modus tollens, second and third premises];  $\neg P$  [Modus tollens; first premise and fourth line].   **4.a)** Second rule: If  $A$  and  $B$  are true, then  $A$  is true.   **5.a)**  $P \Rightarrow Q$  is true if  $P$  is false or  $P$  and  $Q$  are true but in each of these cases

$\neg Q \implies \neg P$  is also true. **c)**  $P \implies Q$  [ Premise];  $\neg Q$  [Assumption];  $\neg P$  [Modus tollens, first premise and second line];  $\neg Q \implies \neg P$  [Conditional proof] **7.a)** Rule 1 of [4] followed by Rule 2 of [4] **c)** Assume  $A$ ; derive  $B \wedge C$  (Premise 1 and modus ponens); derive  $B$  (Rule 2 of [4]); derive  $D$  (Premise 2 and modus ponens); derive  $A \Rightarrow D$  (conditional proof). **9.** This line represents a statement “global” to anything in the subproof and, thus, is usable within the subproof. **12.** Premise; Premise; Universal instantiation; Universal instantiation; Assumption; Modus ponens, lines 3 and 5; Modus ponens, lines 4 and 6; Conditional proof, lines 5 and 7; Universal generalization. **14.**  $S \subset T$  [Premise];  $(\forall s \in S)s \in T$  [From premise];  $(\forall s \in S)s \notin \bar{T}$  [Element can't be both set and its complement];  $(\forall t \in \bar{T})t \notin S$  [Since no  $s \in S$  is in  $\bar{T}$ ];  $(\forall t \in \bar{T})t \in \bar{S}$  [If  $t \notin S$ , must be in  $\bar{S}$ ];  $\bar{T} \subset \bar{S}$  [Since each element in  $\bar{T}$  is in  $\bar{S}$ ]. **15.** For MULT IS is really  $(\forall x \in N)x \geq 0$  and OS is really  $(\forall x \in N)(\forall y \in Z)prod = xy$  and similarly for INSERTION. **17.** With  $G$  a planar graph:  $(\forall G)G$  is planar  $\implies G$  is 4-colorable [Premise];  $(\forall k)(\forall G)G$  is  $k$ -colorable  $\implies G$  has a set of  $v/k$  independent vertices [Premise];  $(\forall G)G$  is 4-colorable  $\implies G$  has a set of  $v/4$  independent vertices [Universal instantiation];  $(\forall G)G$  is planar  $\implies G$  has an independent set of at least  $v/4$  vertices [Transitivity].

## Supplementary Problems: Chapter 7

**2.** With  $P$ : The door I am pointing at is the red door and  $Q$ : Assistant tells the truth, ask: What is the truth value of  $P \Leftrightarrow Q$ ? **3.a)**  $(P \Rightarrow Q) \Leftrightarrow \neg(P \wedge \neg Q)$ ; **d)** Since  $\Leftrightarrow$  is associative, ignore parentheses; since  $\neg P$  is false when  $P$  is true and any expression with just T's and  $\Leftrightarrow$ 's is true, can't explain  $\neg$ ; for  $\vee, \wedge, \Rightarrow$  show by induction that an expression with  $P, Q$  and  $\Leftrightarrow$  has 0, 2 or 4 T's in truth table; but truth tables for  $\vee, \wedge, \Rightarrow$  each have an odd number of T's. **4.c)**  $(P \mid P) \mid (Q \mid Q)$  **5.b)**  $(P \downarrow P) \downarrow (Q \downarrow Q)$  **7.** Since Peirce's arrow is Nor, just use formulas for  $\wedge, \vee, \neg$ , in terms of Peirce's arrow. **9.** Handle negation of  $\wedge$  and  $\vee$  using De Morgan's laws; handle  $\Rightarrow$  by  $\neg(A \Rightarrow B) \Leftrightarrow A \wedge \neg B$ ; handle  $\Leftrightarrow$  by  $\neg(A \Leftrightarrow B) \Leftrightarrow (A \wedge \neg B) \vee (\neg A \wedge B)$ ;  $(\exists i)[((\exists j)P_j \wedge (\forall k)Q_{ik}) \wedge (R_i \wedge \neg S)]$ . **10.a)**  $A, B, /A * A/, /A * B/, /B * A/, /B * B/, /A * /A * A//$  and 7 others of this form, //A \* A/\*A/ and 7 others of this form **b)** All of length  $4k+1$ ; if put two such together with 2 slashes and a star, new length is also 1 more than multiple of 4 **c)** First is winning, next two are not. **11.** Sum picks out just those rows for which value is 1; see [15a, Section 7.5]. **12.** When a term in (1) is one, all terms in (2) are 0 and vice versa. **14.** Take complement twice, first using (2) and then using [28, Section 7.5]; then finally use [13a]; the result is in CNF form (why?).

## Epilogue

### Section E.1

**1.**  $-3$  in (1) and  $1$  in (2). But if  $c_m = -1$ ,  $c_u = 1$  and  $c_p$  is unchanged, then the values are  $1$  and  $-3$ . **3.** Bottom row first, then left column, then across rest of rows from lower to higher. All initial cases done first. **4.** No. Can't tell if the pass was caused by an insertion or a deletion, and can't tell which symbol in the mismatch is the original. **5.** The lineup with no passes (main diagonal of the graph) is uniquely optimal with value 2. **7.**  $(dc_m)N_m + (dc_u)N_u + (dc_p)N_p = d(c_m N_m + c_u N_u + c_p N_p)$  and multiplying all  $V(\mathcal{S}, \mathcal{T}, \alpha)$  by  $d > 0$  does not change which is largest. **9.**  $(c_m + b)N_m + (c_u + b)N_u + (c_p + \frac{1}{2}b)N_p = (c_m N_m + c_u N_u + c_p N_p) + \frac{1}{2}b(2N_m + 2N_u + N_p) =$  previous value +  $\frac{1}{2}b(k+l)$ . Adding the same amount to all  $V(\mathcal{S}, \mathcal{T}, \alpha)$  does not change which is largest. **11.** Set  $b = -2c_p$ ; this sets  $c'_p = 0$  and ensures  $c'_m > 0$  since  $c'_m = c_m + b > 2(c_p + \frac{1}{2}b) = c'_p$ . Now multiply all the  $c'$  by  $1/c'_m$ . **13.** Horizontal edges represent a blank in  $\hat{\mathcal{T}}$ , vertical edges a blank in  $\hat{\mathcal{S}}$ . **15.** Unique answer: ACC with value  $3c_m$ . **17.** Longest directed path between any two points in the graph. **19.** Keep track of which  $i_0, j_0$  maximize  $V^*(i, j)$ . Also at each point  $i, j$ , keep track of which choice in Eq. (11) determined  $V^*(i, j)$ . Now work backwards from  $(i_0, j_0)$ , stopping when the choice in Eq. (11) was 0. **21.** Every 3-subsequence of CAGT except CAG is a longest common subsequence. **23.** Do SUBSTRINGALIGN giving mismatches and passes extremely high negative values. **25.a)** Match substring CGAG to  $\mathcal{T}$  without passes. **b)** Change the weight of all edges on the top and bottom rows to 0. **27.a)** 2 **29.b)**  $C(k, 0) = C(0, l) = 1$  because the initializations have no subcalls.  $C(k, l) = 1 + C(k, l-1) + C(k-1, l) + C(k-1, l-1)$  because each evaluation of  $V$  in the max line of STRINGALIGN-REC requires subcalls. **d)** Because the solution to  $D'_1 = 1$ ,  $D'_l = 3D'_{l-1} + 1$  is  $\frac{1}{2}(3^n - 1)$ . **31.a)** 6: 3 with no

blanks, 2 with one blank and 1 with two blanks. **c) 2 e)** For each pair  $V = 2$ , but  $V(\mathcal{S}, \mathcal{T}, \mathcal{U}) = 5$ , not 6. For instance, any multiple alignment that aligns the C's of  $\mathcal{S}, \mathcal{T}$  and the B's of  $\mathcal{S}, \mathcal{U}$  cannot align the D's of  $\mathcal{T}, \mathcal{U}$ .

## Section E.2

**2. b)**  $T(n)$  can't be shorter than the length of the longest path. **c)** Reducing  $d(j)$  for any  $j$  on a unique longest path will result in a shorter longest path, even if it is another path. **3.** Provide  $V(i, j)$  for each vertex in the graph (i.e., for all alignments of initial segments  $\mathcal{S}_i, \mathcal{T}_j$ ) and a check that they satisfy Eq. (9), Section E.1. **5.** Let  $d_j$  be the shortest distance from the start to  $v_j$ . Then iterate on  $d_j = \min\{d_i, (i, j) \in E\}$ . Unlike DIJKSTRA this only works for DAGs. **7. x** to 2, **y** to 1. **9. a)** Same as Theorem 1, replacing "vertex activity" by "edge activity". **c)** In Theorem 2, replace "vertex activity" by "edge activity" and modify the key line in PROJECTSCHED like this:  $T(j) \leftarrow \max_{i \rightarrow j} \{T(i) + d(i, j)\}$ . **10. a)** That theorem required  $a_n$  to depend on just the  $k$  previous values, whereas each  $L(j)$  (and  $T(j)$ ) may depend on any previous values. **b)** Strong induction, because this does not put any limit on which previous cases may be considered.

## Final Problems

**1. a)** Inductive step:  $n - 1$  disks to end (by earlier case of LSTOH); largest to middle;  $n - 1$  disks to other end; largest to end;  $n - 1$  disks to end. **b)** Inductive step for S1STOH:  $n - 1$  disks to wrong end; largest to right end;  $n - 1$  disks to middle (by S2STOH);  $n - 1$  disks to right end. For S2STOH:  $n - 1$  disks to middle;  $n - 1$  disks to other end (by S1STOH); largest to middle;  $n - 1$  disks to middle. **c)** By induction. **d)** LSTOH:  $p_n = 3p_{n-1} + 2$ ;  $p_1 = 2$ . S1STOH, S2STOH:  $q_n = 3q_{n-1} + 1$ ;  $q_1 = 1$ . Solutions:  $p_n = 3^n - 1$ ;  $q_n = (3^n - 1)/2$ . **e)**  $q_n = q_{n-1} + p_{n-1} + 1$ ,  $q_1 = 2$ ; same solution as d). **f)** Since the short trip solution is unique (why?) and since using LSTOH gives exactly the same result as using short trip twice, LSTOH must in fact be made up of two short trips. **g)** Basis: LSTOH puts disk on middle pole first; with  $n$  disks, first  $n - 1$  disks go to far end; then after largest in middle, putting  $n - 1$  disks back on original end is same as LSTOH with  $n - 1$  disks; by induction hypothesis at some point  $n - 1$  disks are on top of largest disk on middle pole. **3.** Can be won for all  $n$ , both to clockwise pole (CL) and counterclockwise pole (CCL). Inductive step for  $n$  rings to CL:  $n - 1$  to CCL, largest to CL,  $n - 1$  to CCL. For  $n$  rings to CCL:  $n - 1$  to CL, largest to CCL,  $n - 1$  to CL. Uniqueness of minimum sequence by induction but very subtle. Minimum # of move for CL:  $(1/4)[5 \cdot 2^n + (-2)^n] - 1$ . For CCL:  $(1/2)[5 \cdot 2^{n-1} + (-2)^{n-1}] - 1$ . **6. a)** Always  $n^2$  multiplications; at most  $n^2 + n - 2$  additions. **b)**  $(P10^m + Q)(R10^m + S) = 10^n PR + 10^m(PS + QR) + QR + QS$ . **c)**  $M_{2n} = 4M_n$ ,  $M_1 = 1$ ; for  $n = 2^k$  solution is  $M_n = n^2$ ; no improvement. **d)**  $A_{2n} = 4A_n + cn$ ,  $A_1 = 0$ ; for  $n = 2^k$   $A_n = (c/2)(n^2 - n)$ ; since  $c > 2$  more additions than standard method needed for large  $n$ . **e)** Now  $M_{2n} = 3M_n$  so solution is  $M_n = n^{\log_2 3}$ ; this is an improvement since  $\log_2 3$  is about 1.56; extra digit in  $P+Q$  or  $R+S$  can be ignored if we assume  $m$  large. **f)** Now  $A_{2n} = 3A_n + cn$  and  $A_n = c(n^{\log_2 3} - 1)$  which is better than result in d) for large enough  $n$ . **8. a)** With  $s_j = r_{k-j}$ , the recurrence  $r_{i+1} = r_{i-1} - \lfloor r_{i-1}/r_i \rfloor r_i$  for Euclid's algorithm becomes the one given. **b)** Since  $r_i < r_{i-1}$  (except perhaps in the first step, where the  $r$ 's are  $m$  and  $n$ ), and since in the last step  $r_k = 0$ , then  $Q_1 = r_{k-2}/r_{k-1} \geq 2$ . **c)** By induction;  $s_1 \geq 1 = f_1$ ; since  $Q_1 \geq 2$ ,  $s_2 \geq 2 = f_2$ ; then since  $Q_j \geq 1$ , each  $s_j \geq f_j$ ; thus  $m = s_{k+1} \geq f_{k+1}$ . Going other way, if  $(m, n) = (f_{k+1}, f_k)$ , each remainder is next smaller Fibonacci number until reaching  $(s_2, s_1) = (f_2, f_1) = (2, 1)$ . **d)** One more iteration needed. **11. a)** Pick a vertex  $v$  in  $n$ -vertex graph  $G$ ; on  $G - v$  find  $\alpha_1$  and on  $G - N(v)$  find  $\alpha_2$ ; size of maximum independent set is maximum of  $\alpha_1$  and  $1 + \alpha_2$ . **b)** Let  $g_n$  be work to find  $\alpha$  for an  $n$ -set; then  $g_n > 2g_{n-1}$ ; with  $g_1 = 1$ , this means  $g_n > 2^{n-1}$ ; since there are  $2^n$  subsets of an  $n$ -set, this method is comparable to just checking each subset. **c)** If  $v$  connected to one other vertex, then  $G - N(v)$  has at most  $n - 2$  vertices; therefore,  $g_n > g_{n-1} + g_{n-2}$  so that  $g_n > f_n$ , the  $n$ th Fibonacci number; since Fibonacci numbers grow as about  $(1.6)^n$ , this is a better bound.