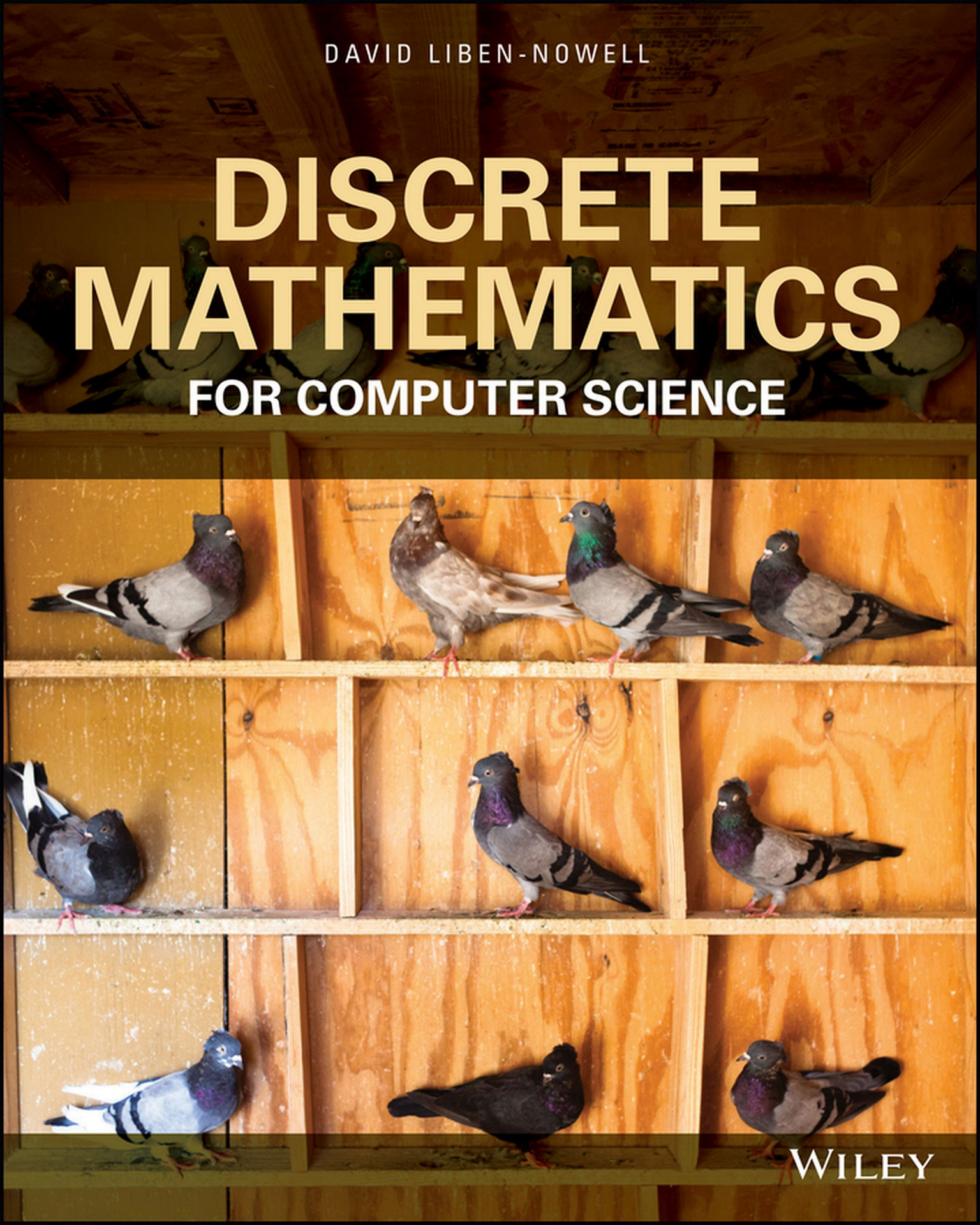
DAVID LIBEN-NOWELL

DISCRETE MATHEMATICS

FOR COMPUTER SCIENCE



WILEY

David Liben-Nowell

Department of Computer Science

Carleton College

Discrete Mathematics for Computer Science

or

(A Bit of) The Math that Computer Scientists Need to Know

VP AND EDITORIAL DIRECTOR	Laurie Rosatone
SENIOR DIRECTOR	Don Fowley
ACQUISITIONS EDITOR	Linda Ratts
EDITORIAL MANAGER	Gladys Soto
CONTENT MANAGEMENT DIRECTOR	Lisa Wojcik
CONTENT MANAGER	Nichole Urban
SENIOR CONTENT SPECIALIST	Nicole Repasky
PRODUCTION EDITOR	Rajeshkumar Nallusamy
PHOTO RESEARCHER	Billy Ray
COVER PHOTO CREDIT	© slobo/Getty Images, Inc.

This book was set in TeXGyrePagella 10/12 by SPi Global and printed and bound by Strategic Content Imaging.

This book is printed on acid free paper. ∞

ISBN: 978-1-118-06553-2 (PBK)
ISBN: 978-1-119-07073-3 (EVALC)

Library of Congress Cataloging in Publication Data:

Liben-Nowell, David, author.
Title: Discrete mathematics for computer science / by David Liben-Nowell.
Description: Hoboken, NJ : John Wiley & Sons, 2017. | Includes index. |
Identifiers: LCCN 2017025007 (print) | LCCN 2017035974 (ebook) | ISBN 9781119397199 (pdf) | ISBN 9781119397113 (epub) | ISBN 9781118065532 (pbk.)
Subjects: LCSH: Computer science—Mathematics.
Classification: LCC QA76.9.M35 (ebook) | LCC QA76.9.M35 L53 2017 (print) | DDC 004.01/51—dc23
LC record available at https://lccn.loc.gov/2017025007

The inside back cover will contain printing identification and country of origin if omitted from this page. In addition, if the ISBN on the back cover differs from the ISBN on this page, the one on the back cover is correct.

To MDSWM, with never-ending appreciation, and in loving memory of my grandfather, Jay Liben, who brought more joy, curiosity, and kvetching to this world than anyone else I know.

Contents

List of Computer Science Connections

Chapter 4: Proofs

Chapter 5: Mathematical Induction

Chapter 6: Analysis of Algorithms

Chapter 7: Number Theory

Chapter 8: Relations

Chapter 9: Counting

Chapter 10: Probability

Chapter 11: Graphs and Trees

Acknowledgements

> Would thou hadst less deserved,
> That the proportion both of thanks and payment
> Might have been mine! only I have left to say,
> More is thy due than more than all can pay.
>
> William Shakespeare (1564–1616)
> The Scottish Play

To everyone who has helped, directly and indirectly, with everything over these last years—these words cannot adequately convey my thanks, but at least they're a start: *thank you!*

I owe special thanks to a very long list of generous and warm people—many more than I can mention here—for advice and kindness and support, both technical and emotional, as this book came into being. For those whom I haven't named by name, please know that it's only because I have gotten such great support from so many people, and I hope that you'll consider this sentence the promise that, when we next see each other, the first round's on me. While I'm leaving out the names of the many people who have helped make my life happy and fulfilling while I've been working on this book, I do want to give specific thanks to a few people:

I want to thank my colleagues—near and far, including many who are not just colleagues but also dear friends and beloved family members—for their wisdom and patience, for answering my endlessly annoying questions, and for conversations that led to examples or exercises or bug fixes or the very existence of this entire book (even if you didn't know that's what we were talking about at the time): Eric Alexander, Tanya Berger-Wolf, Kelly Connole, Amy Csizmar Dalal, Josh Davis, Roger Downs, Laura Effinger-Dean, Eric Egge, Adriana Estill, Andy Exley, Alex Freeman, Sherri Goings, Jack Goldfeather, Deanna Haunsperger, Pierre Hecker, David Huyck, Sue Jandro, Sarah Jansen, Iris Jastram, Jon Kleinberg, Carissa Knipe, Mark Krusemeyer, Jessica Leiman, Lynn Liben, Jadrian Miles, Dave Musicant, Gail Nelson, Rich Nowell, Layla Oesper, Jeff Ondich, Sam Patterson, Anna Rafferty, Alexa Sharp, Julia Strand, Mike Tie, Zach Weinersmith, Tom Wexler, Kevin Woods, Jed Yang, and Steve Zdancewic.

I also owe my appreciation to Don Fowley, Bryan Gambrel, Beth Golub, Jessy Moor, Anna Pham, Sondra Scott, and Gladys Soto at Wiley. Thanks to Judy Brody for relentless and efficient pursuit of permissions (from many different people and publishers)

to use the quotes that appear as epigraphs throughout the book. And thanks as well to the many insightful reviewers of previous drafts of this material. So many times I got chapter reviews back and put them aside in a huff, only to come back to the reviewers' comments months later and realize that their suggestions were exactly right. (And, to be clear: blame me, not them, for the errors that I'm sure remain.)

I specifically want to thank Eric Alexander, Laura Biester, Josh Davis, Charlotte Foran, Jadrian Miles, Dave Musicant, Layla Oesper, Anna Rafferty, Jed Yang, and the Carleton CS 202 students from 2013–2017 for their willingness to work with early, and buggy, drafts of this book. And thanks to those and many other students at Carleton for their patience, and for sending their comments and suggestions for improvements—in particular: Hami Abdi, David Abel, Alexander Auyeung, Andrew Bacon, Kharmen Bharucha, John Blake, Caleb Braun, Macallan Brown, Adam Canady, Noah Carnahan, Yitong Chen, Jinny Cho, Leah Cole, Katja Collier, Lila Conlee, Eric Ewing, Greg Fournier, Andy Freeland, Emma Freeman, Samuel Greaves, Reilly Hallstrom, Jacob Hamalian, Sylvie Hauser, Jack Hessel, Joy Hill, Matt Javaly, Emily Johnston, Emily Kampa, Carlton Keedy, Henry Keiter, Jonathan Knudson, Julia Kroll, Brennan Kuo, Edward Kwiatkowski, Dimitri Lang, Tristan Leigh, Zach Levonian, Daniel Levy, Rhys Lindmark, Gordon Loery, David Long, Robert Lord, Inara Makhmudova, Elliot Mawby, Javier Moran Lemus, Sean Mullan, Micah Nacht, Justin Norden, Laurel Orr, Raven Pillmann, Josh Pitkofsky, Matthew Pruyne, Nikki Rhodes, Will Schifeling, Colby Seyferth, Alex Simonides, Oscar Smith, Kyung Song, Frederik Stensaeth, Patrick Stephen, Maximiliano Villarreal, Alex Voorhees, Allie Warren, Ben Wedin, Michael Wheatman, Jack Wines, Christopher Winter, and Andrew Yang.

This book would not have been possible without the support of Carleton College, not only for the direct support of this project, but also for providing a wonderfully engaging place to make my professional home. When I started at Carleton, my friends and family back east thought that moving to Minnesota (the frontier!) was nothing less than a sign that I had finally lost it, and I have to admit that I thought they had a point. But it's been a fabulous place to have landed, with great friends and colleagues and students—the kind who don't let you get away with anything, but in a good way.

Some of the late stages of the work on this book occurred while I was visiting the University of Cambridge. Thanks to Churchill College and the Computer Laboratory, and especially to Melissa Hines and Cecilia Mascolo, for their hospitality and support.

And my thanks to the somewhat less formal host institutions that have fueled this writing: Brick Oven Bakery, Cakewalk, Goodbye Blue Monday, Tandem Bagels, The Hideaway (Northfield, MN); Anodyne, Blue Moon, Bull Run, Caffetto, Common Roots, Espresso Royale, Isles Bun & Coffee, Keen Eye, Plan B, Precision Grind, Reverie, Spyhouse, Sebastian Joe's, The Beat, The Nicollet, The Purple Onion, Turtle Bread Company, Uncommon Grounds, Urban Bean (Minneapolis, MN); Ginkgo, Grand Central, Kopplin's (St. Paul, MN); Collegetown Bagels (Ithaca, NY); Slave to the Grind (Bronxville, NY); Bloc Eleven, Diesel Cafe (Somerville, MA); Lyndell's (Cambridge, MA); Tryst (Washington, DC); Hot Numbers, Espresso Library (Cambridge, England); and various Starbucks, Caribous, and Dunn Brothers.

And, last but certainly not least, my deepest gratitude to my friends and family for all your help and support while this project has consumed both hours and years. You know who you are, and I hope you also know how much I appreciate you. *Thank you!*

David Liben-Nowell
Northfield, MN
May 2017

PS: I would be delighted to receive any comments or suggestions from readers. Please don't hesitate to get in touch.

Credits

This book was typeset using LaTeX, and I produced all but a few figures from scratch using a combination of PSTricks and TikZ. The other figures are reprinted with permission from their copyright holders. The illustrations that open every chapter were drawn by Carissa Knipe (`http://carissaknipe.com`), who was a complete delight to work with—both on these illustrations and when she was a student at Carleton. I took the photograph of a house in Figure 2.48 myself. Figure 4.5 (the Therac-25 diagram) is reproduced from Nancy Leveson's book *Safeware: System Safety and Computers* with permission from Pearson Education. Figure 4.27 (a poem proving the undecidability of the Halting Problem) is reproduced with permission from Geoffrey K. Pullum. Figure 5.22 (triangulations of a rabbit) is reproduced from a paper by Tobias Isenberg, Knut Hartmann, and Henry König with permission from the Society for Modeling and Simulation International (SCS). Figure 11.15 (a map of some European train routes) is reproduced with permission from RGBAlpha/Getty Images.[1]

[1] Nancy Leveson. *Safeware: System Safety and Computers*. Pearson Education, Inc., New York, 1995; Tobias Isenberg, Knut Hartmann, and Henry König. Interest value driven adaptive subdivision. In *Simulation and Visualisation (SimVis)*, pages 139–149. SCS European Publishing House, 2003; and Geoffrey K. Pullum. Scooping the loop snooper: A proof that the halting problem is undecidable. *Mathematics Magazine*, 73(4):319–320, 2000. Used by permission of Geoffrey K. Pullum.

For their kind permission to use quotes that appear as epigraphs in sections throughout the book, thanks to:

Kurt Vonnegut, p. 102. Excerpt from *Hocus Pocus* by Kurt Vonnegut, copyright ©1990 by Kurt Vonnegut. Used by permission of G. P. Putnam's Sons, an imprint of Penguin Publishing Group, a division of Penguin Random House LLC. All rights reserved. Any third party use of this material, outside of this publication, is prohibited. Interested parties must apply directly to Penguin Random House LLC for permission.

Pablo Picasso, p. 203. ©2017 Estate of Pablo Picasso / Artists Rights Society (ARS), New York. Reprinted with permission.

Laurence J. Peter, p. 317. Reprinted with permission of the estate of Laurence J. Peter.

Carl Sagan, p. 331. From *Broca's Brain: Reflections on the Romance of Science*, ©1979 Carl Sagan. Reprinted with permission from Democritus Properties, LLC.

Peter De Vries, p. 349. Copyright ©1967 by Peter De Vries. Reprinted by permission of Curtis Brown, Ltd. All rights reserved.

1
On the Point of this Book

In which our heroes decide, possibly encouraged by a requirement for graduation, to set out to explore the world.

Why You Might Care

> Just because some of us can read and write and do a little math, that doesn't mean we deserve to conquer the Universe.
>
> Kurt Vonnegut (1922–2007)
> *Hocus Pocus* (1990)

This book is designed for an undergraduate student who has taken a computer science class or three—most likely, you are a sophomore or junior prospective or current computer science major taking your first non-programming-based CS class. If you are a student in this position, you may be wondering why you're taking this class (or why you *have* to take this class!). Computer science students taking a class like this one sometimes don't see why this material has anything to do with computer science—particularly if you enjoy CS because you enjoy programming.

I want to be clear: programming is awesome! I get lost in code all the time—let's not count the number of hours that I spent writing the code to draw the fractals in Figure 5.1 in LaTeX, for example. (LaTeX, the tool used to typeset this book, is the standard typesetting package for computer scientists, and it's actually also a full-fledged, if somewhat bizarre, programming language.)

But there's more to CS than programming. In fact, many seemingly unrelated problems rely on the same sorts of abstract thinking. It's not at all obvious that an optimizing compiler (a program that translates source code in a programming language like C into something directly executable by a computer) would have anything important in common with a program to play chess perfectly. But, in fact, they're both tasks that are best understood using *logic* (Chapter 3) as a central component of any solution. Similarly, filtering spam out of your inbox ("given a message m, should m be categorized as spam?") and doing speech recognition ("given an audio stream s of a person speaking in English, what is the best 'transcript' reflecting the words spoken in s?") are both best understood using *probability* (Chapter 10).

And these, of course, are just examples; there are many, many ways in which we can gain insight and efficiency by thinking more abstractly about the commonalities of interesting and important CS problems. That is the goal of this book: to introduce the kind of mathematical, formal thinking that will allow you to understand ideas that are shared among disparate applications of computer science—and to make it easier for you to make your own connections, and to extend CS in even more new directions.

How To Use This Book

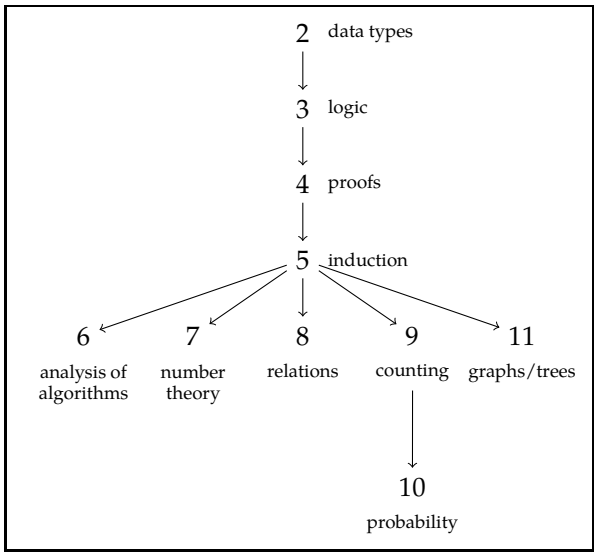
> Read much, but not many Books.
>
> Benjamin Franklin (1706–1790)
> *Poor Richard's Almanack* (1738)

The brief version of the advice for how to use this book is: *it's your book; use it however you'd like.* (Will Shortz, the puzzle editor of *The New York Times*, gives the analogous advice about crossword puzzles when he's asked whether Googling for an

answer is cheating.) But my experience is that students do best when they read actively, with scrap paper close by; most people end up with a deeper understanding of a problem by trying to solve it themselves *first*, before they look at the solution.

I've assumed throughout that you're comfortable with programming in at least one language, including familiarity with recursion. It doesn't much matter which particular programming language you know; we'll use features that are shared by almost all modern languages—things like conditionals, loops, functions, and recursion. You may or may not have had more than one programming-based CS course; many, but not all, institutions require Data Structures as a prerequisite for this material. There are times in the book when a data structures background may give you a deeper understanding (but the same is true in reverse if you study data structures after this material). There are similarly a handful of topics for which rudimentary calculus background is valuable. But knowing/remembering calculus will be specifically useful only a handful of times in this book; the mathematical prerequisite for this material is really algebra and "mathematical maturity," which basically means having some degree of comfort with the idea of a mathematical definition and with the manipulation of a mathematical expression. (The few places where calculus is helpful are explicitly marked.)

There are 10 chapters after this one in the book. Their dependencies are as shown at right. Aside from these dependencies, there are some occasional references to other chapters, but these references are light. If you've skipped Chapter 6—many instructors will choose not cover this material, as it is frequently included in a course on Algorithms instead of this one—then it will still be useful to have an informal sense of O, Ω, and Θ notation in the context of the worst-case running time of an algorithm. (You might skim Sections 6.1 and 6.6 before reading Chapters 7–11.)



I've tried to include some helpful tips for problem solving in the margins throughout the book, along with a few warnings about common confusions and some notes on terminology/notation that may be helpful in keeping the words and symbols straight. There are also two kinds of extensions to the main material. The "Taking it Further" blocks give more technical details about the material under discussion—an alternate way of thinking about a definition, or a way that a concept is used in CS or a related field. You should read the "Taking it Further" blocks if—but only if!—you find them engaging. Each section also ends with one or more boxed-off "Computer Science Connections" that show how the core material can be used to solve a wide variety of (interesting, I hope!) CS applications. No matter how interesting the core technical material may be, I think that it is what we can *do* with it that makes it worth studying.

What This Book Is About

> All truths are easy to understand once they are discovered; the point is to discover them.
>
> Galileo Galilei (1564–1642)

This book focuses on *discrete* mathematics, in which the entities of interest are distinct and separate. Discrete mathematics contrasts with *continuous* mathematics, as in calculus, which addresses infinitesimally small objects, which cannot be separated. We'll use summations rather than integrals, and we'll generally be thinking about things more like the integers ("$1, 2, 3, \ldots$") than like the real numbers ("all numbers between π and 42"). Because this book is mostly focused on non-programming-based parts of computer science, in general the "output" that you produce when solving a problem will be something different from a program. Most typically, you will be asked to answer some question (quantitatively or qualitatively) and to justify that answer—that is, to *prove* your answer. (A *proof* is an ironclad, airtight argument that convinces its reader of your claim.) Remember that your task in solving a problem is to persuade your reader that your purported solution genuinely solves the problem. Above all, that means that your main task in writing is communication and persuasion.

Be careful; there are two different words that are pronounced identically:

discrete, adj.: individually separate and distinct.

discreet, adj.: careful and judicious in speech, especially to maintain privacy or avoid embarrassment.

You wouldn't read a book about discreet mathematics; instead, someone who trusts you might quietly share it while making sure no one was eavesdropping.

There are three very reasonable ways of thinking about this book.

View #1 is that this book is about the mathematical foundations of computation. This book is designed to give you a firm foundation in mathematical concepts that are crucial to computer science: sets and sequences and functions, logic, proofs, probability, number theory, graphs, and so forth.

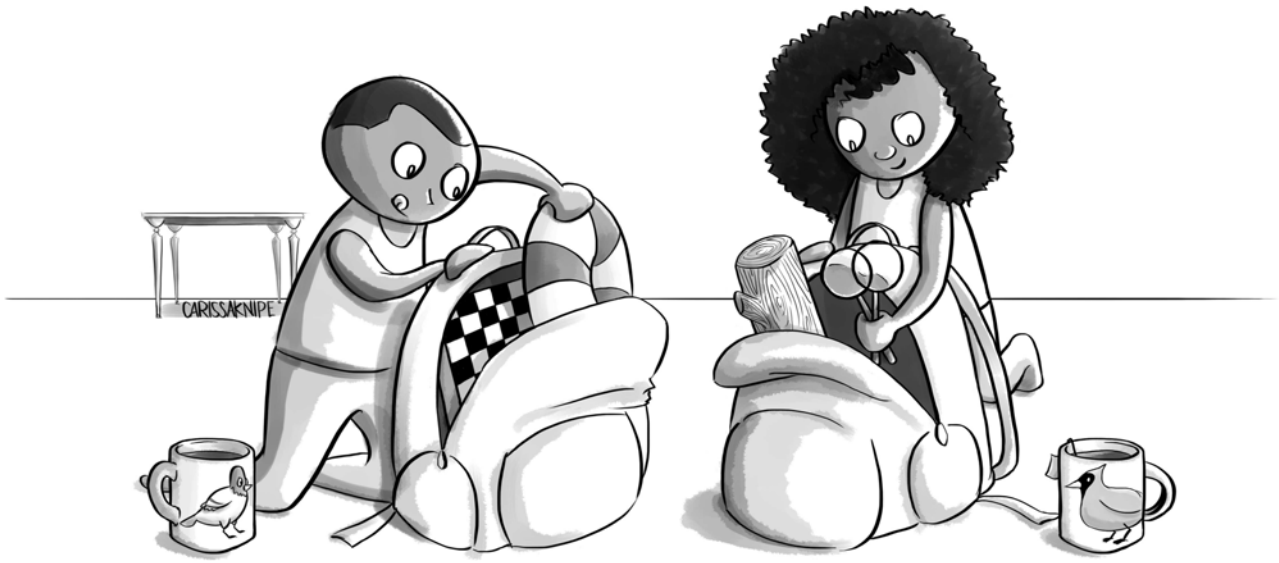
View #2 is that this book is about practice. Essentially no particular example that we consider matters; what's crucial is for you to get exposure to and experience with formal reasoning. Learning specific facts about specific topics is less important than developing your ability to reason rigorously about formally defined structures.

View #3 is that this book is about applications of computer science: it's about error-correcting codes (how to represent data redundantly so that the original information is recoverable even in the face of data corruption); cryptography (how to communicate securely so that your information is understood by its intended recipient but not by anyone else); natural language processing (how to interpret the "meaning" of an English sentence spoken by a human using an automated customer service system); and so forth. But, because solutions to these problems rely fundamentally on sets and counting and number theory and logic, we have to understand basic abstract structures in order to understand the solutions to these applied problems.

In the end, of course, all three views are right: I hope that this book will help to introduce some of the foundational technical concepts and techniques of theoretical computer science, and I hope that it will also help demonstrate that these theoretical approaches have relevance and value in work throughout computer science—in topics both theoretical and applied. And I hope that it will be at least a little bit of fun.

Bon voyage!

2
Basic Data Types



In which our heroes equip themselves for the journey ahead, by taking on the basic provisions that they will need along the road.

2.1 Why You Might Care

> It is a capital mistake to theorize before one has data.
>
> Sir Arthur Conan Doyle (1859–1930),
> *A Scandal in Bohemia* (1892)

This chapter will introduce concepts, terminology, and notation related to the most common data types that recur throughout this book, and throughout computer science. These basic entities—the Booleans (True and False), numbers (integers, rationals, and reals), sets, sequences, functions—are also the basic data types we use in modern programming languages. Essentially every common primitive data type in programs appears on this list: a Boolean, an integer (or an `int`), a real number (or a `float`), and a string (an ordered sequence of characters). Ordered sequences of other elements are usually called *arrays* or *lists*. If you've taken a course on data structures, you've probably worked on several implementations of sets that allow you to insert an element into an unordered collection and to test whether a particular object is a "member" of the collection. And functions that map a given input to a corresponding output are the basic building blocks of programs.

Virtually every interesting computer science application uses these basic data types extensively. *Cryptography*, which is devoted to the secure storage and transmission of information in such a way that a malicious third party cannot decipher that information, is typically based directly on integers, particularly large prime numbers. A ubiquitous task in *machine learning* is to "cluster" a set of entities into a collection of nonoverlapping subsets so that two entities in the same subset are similar and two entities in different subsets are dissimilar. In *information retrieval*, where we might seek to find the document from a large collection that is most relevant to a given query, it is common to represent each document by a vector (a sequence of numbers) based on the words used in the document, and to find the most relevant documents by identifying which ones "point in the same direction" as the query's vector. And functions are everywhere in CS, from data structures like hash tables to the routing that's done for every packet of information on the internet.

In this chapter, we'll describe these basic entities and some standard notation that's associated with them. Some closely related topics will appear later in the book, as well. Chapter 7, on number theory, will discuss some subtler properties of the integers, particularly divisibility and prime numbers. Chapter 8 will discuss relations, a generalization of functions. But, really, every chapter of this book is related to this chapter: our whole enterprise will involve building complex objects out of these simple ones (and, to be ready to understand the more complex objects, we have to understand the simple pieces first). And before we launch into the sea of applications, we need to establish some basic shared language. Much of the basic material in this chapter may be familiar, but regardless of whether you have seen it before, it is important and standard content with which it is important to be comfortable.

2.2 Booleans, Numbers, and Arithmetic

> Everything you can imagine is real.
>
> Pablo Picasso (1881–1973)

We start with the most basic types of data: *Boolean* values (True and False), *integers* ($\ldots, -2, -1, 0, 1, 2, \ldots$), *rational numbers* (fractions with integers as numerators and denominators), and *real numbers* (including the integers and all the numbers in between them). The rest of this section will then introduce some basic numerical operations: absolute values and rounding, exponentiation and logarithms, summations and products. Figure 2.1 summarizes this section's notation and definitions.

2.2.1 Booleans: True and False

The most basic unit of data is the *bit*: a single piece of information, which either takes on the value 0 or the value 1. Every piece of stored data in a digital computer is stored as a sequence of bits. (See Section 2.4 for a formal definition of sequences.)

We'll view bits from several different perspectives: 1 and 0, on and off, yes and no, *True* and *False*. Bits viewed under the last of these perspectives have a special name, the *Booleans*:

Booleans are named after George Boole (1815–1864), a British mathematician, who was the first person to think about True as 1 and False as 0.

Definition 2.1 (Booleans)
A Boolean value *is either True or False.*

The Booleans are the central object of study of Chapter 3, on logic. In fact, they are in a sense the central object of study of this entire book: simply, we are interested in making true statements, with a proof to justify why the statement is true.

2.2.2 Numbers: Integers, Reals, and Rationals

We'll often encounter a few common types of numbers—*integers*, *reals*, and *rationals*:

Definition 2.2 (Integers, Reals, and Rationals)

- *The* integers, *denoted by* $\mathbb{Z}$, *are those numbers with no fractional part:* 0, *the positive integers* ($1, 2, \ldots$), *and the negative integers* ($-1, -2, -3, \ldots$).
- *The* real numbers, *denoted by* $\mathbb{R}$, *are those numbers that can be (approximately) represented by decimal numbers; informally, the reals include all integers and all numbers "between" any two integers.*
- *The* rational numbers, *denoted by* $\mathbb{Q}$, *are those real numbers that can be represented as a ratio* $\frac{n}{m}$ *of two integers* n *and* m, *where* n *is called the* numerator *and* $m \neq 0$ *is called the* denominator. *A real number that is not rational is called an* irrational *number.*

The superficially unintuitive notation for the integers, the symbol $\mathbb{Z}$, is a stylized "Z" that was chosen because of the German word *Zahlen*, which means "numbers." The name *rationals* comes from the word *ratio*; the symbol $\mathbb{Q}$ comes from its synonym *quotient*. (Besides, the symbol $\mathbb{R}$ was already taken by the reals, so the rationals got stuck with their second choice.)

Here are a few examples of each of these types of numbers:

Booleans	True and False
$\mathbb{Z}$	integers $(\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots)$
$\mathbb{Q}$	rational numbers
$\mathbb{R}$	real numbers
$[a, b]$	those real numbers x where $a \leq x \leq b$
(a, b)	those real numbers x where $a < x < b$
$[a, b)$	those real numbers x where $a \leq x < b$
$(a, b]$	those real numbers x where $a < x \leq b$
$\lvert x \rvert$	absolute value of x: $\lvert x \rvert := -x$ if $x < 0$; $\lvert x \rvert := x$ if $x \geq 0$
$\lfloor x \rfloor$	floor of x: x rounded down to the nearest integer
$\lceil x \rceil$	ceiling of x: x rounded up to the nearest integer
b^n	b multiplied by itself n times
$b^{1/n}$, or $\sqrt[n]{b}$	a number y such that $y^n = b$ (where $y \geq 0$ if possible), if one exists
$b^{m/n}$	$(b^{1/n})^m$
$\log_b x$	logarithm: $\log_b x$ is the value y such that $b^y = x$, if one exists
$n \bmod k$	modulo: $n \bmod k :=$ the remainder when dividing n by k
$k \mid n$	k (evenly) divides n
$\sum$	summation: $\sum_{i=1}^{n} x_i := x_1 + x_2 + \cdots + x_n$
$\prod$	product: $\prod_{i=1}^{n} x_i := x_1 \cdot x_2 \cdot \cdots \cdot x_n$

Figure 2.1: Summary of the basic mathematical notation introduced in Section 2.2.

Example 2.1 (Integers, reals, and rationals)
The following are all examples of integers: 1, 42, 0, and -17.

All of the following are real numbers: 1, 99.44, the ratio of the circumference of a circle to its diameter $\pi \approx 3.141592653\cdots$, and the so-called *golden ratio* $\phi = (1 + \sqrt{5})/2 \approx 1.61803\cdots$.

Examples of rational numbers include $\frac{3}{2}$, $\frac{9}{5}$, $\frac{16}{4}$, and $\frac{4}{1}$. (In Chapter 8, we'll talk about the familiar notion of the equivalence of two rational numbers like $\frac{1}{2}$ and $\frac{2}{4}$, or like $\frac{16}{4}$ and $\frac{4}{1}$, based on common divisors. See Example 8.36.) Of the example real numbers above, both 1 and 99.44 are rational numbers; we can write them as $\frac{1}{1}$ and $\frac{4972}{50}$, for example. Both π and ϕ are irrational.

Here are a few useful points relating these three types of numbers:

- All integers are rational numbers (with denominator equal to 1).
- All rational numbers are real numbers.
- But not all rational numbers are integers and not all real numbers are rational: for example, $\frac{3}{2}$ is not an integer, and $\sqrt{2}$ is not rational. (We'll prove that $\sqrt{2}$ is not rational in Example 4.21.)

Taking it further: Definition 2.2 specifies $\mathbb{Z}$, Q, and $\mathbb{R}$ somewhat informally. To be completely rigorous, one can define the nonnegative integers as the smallest collection of numbers such that: (i) 0 is an integer; and (ii) if x is an integer, then $x + 1$ is also an integer. See Section 5.4.1. (Of course, for even this definition to make sense, we'd need to give a rigorous definition of the number zero and a rigorous definition of the operation of adding one.) With a proper definition of the integers, it's fairly easy to define the rationals as ratios of integers. But formally defining the real numbers is surprisingly challenging; it was a major enterprise of mathematics in the late 1800s, and is often the focus of a first course in analysis in an undergraduate mathematics curriculum.

Virtually every programming language supports both integers (usually known as `int`s) and real numbers (usually known as `float`s); see p. 217 for some discussion of the way that these basic numerical types are implemented in real computers. (Rational numbers are much less frequently implemented as basic data types in programming languages, though there are some exceptions, like Scheme.)

In addition to the basic symbols that we've introduced to represent the integers, the rationals, and the reals ($\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$), we will also introduce special notation for some specific subsets of these numbers. We will write $\mathbb{Z}^{\geq 0}$ and $\mathbb{Z}^{\leq 0}$ to denote the nonnegative integers $(0, 1, 2, \ldots)$ and nonpositive integers $(0, -1, -2, \ldots)$, respectively. Generally, when we write $\mathbb{Z}$ with a superscripted condition, we mean all those integers for which the stated condition is true. For example, $\mathbb{Z}^{\neq 1}$ denotes all integers aside from 1. Similarly, we write $\mathbb{R}^{>0}$ to denote the positive real numbers (every real number $x > 0$). Other conditions in the superscript of $\mathbb{R}$ are analogous.

We'll also use standard notation for *intervals* of real numbers, denoting all real numbers between two specified values. There are two variants of this notation, which allow "between two specified values" to either *include* or *exclude* those specified values. We use round parentheses to mean "exclude the endpoint" and square brackets to mean "include the endpoint" when we denote a range:

- (a, b) denotes those real numbers x for which $a < x < b$.
- $[a, b]$ denotes those real numbers x for which $a \leq x \leq b$.
- $(a, b]$ denotes those real numbers x for which $a < x \leq b$.
- $[a, b)$ denotes those real numbers x for which $a \leq x < b$.

Sometimes (a, b) and $[a, b]$ are, respectively, called the *open interval* and *closed interval* between a and b. These four types of intervals are also sometimes denoted via a *number line*, with open and closed circles denoting open and closed intervals; see Figure 2.2 for an example. For two real numbers x and y, we will use the standard notation "$x \approx y$" to denote that x is *approximately equal to* y. This notation is defined informally, because what counts as "close enough" to be approximately equal will depend heavily on context.

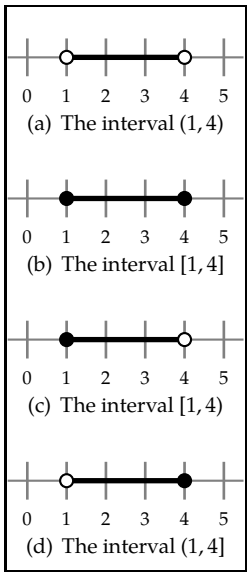


Figure 2.2: Number lines representing real numbers between 1 and 4, with 1 included in the range in (b, c), and 4 included in the range in (b, d).

2.2.3 *Absolute Value, Floor, and Ceiling*

In the remaining subsections of Section 2.2, we will give definitions of some standard arithmetic operations that involve the numbers we just defined. We'll start in this subsection with three operations on a real number: absolute value, floor, and ceiling.

The *absolute value* of a real number x, written $|x|$, denotes how far x is from 0, disregarding the *sign* of x (that is, disregarding whether x is positive or negative):

Definition 2.3 (Absolute Value)
The absolute value *of a real number x is* $|x| := \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{otherwise.} \end{cases}$

For example, $|42.42| = 42.42$ and $|-128| = 128$. (Definition 2.3 uses standard notation for defining "by cases": the value of $|x|$ is x when $x \geq 0$, and the value of $|x|$ is $-x$ otherwise—that is, when $x < 0$.)

For a real number x, we can consider x "rounded down" or "rounded up," which are called the *floor* and *ceiling* of x, respectively:

Definition 2.4 (Floor and ceiling)
The floor *of a real number x, written $\lfloor x \rfloor$, denotes the largest integer that is less than or equal to x. The* ceiling *of a real number x, written $\lceil x \rceil$, denotes the smallest integer that is greater than or equal to x.*

Note that Definition 2.4 defines the floor and ceiling of negative numbers, too; the definition doesn't care whether x is greater than or less than 0.

Here are a few examples of floor and ceiling:

Example 2.2 (Floor and ceiling)
We have $\lfloor \sqrt{2} \rfloor = \lfloor 1.4142 \cdots \rfloor = 1$, $\lfloor 2\pi \rfloor = \lfloor 6.28318 \cdots \rfloor = 6$, and $\lfloor 3 \rfloor = 3$. For ceilings, we have $\lceil \sqrt{2} \rceil = 2$, $\lceil 2\pi \rceil = 7$, and $\lceil 3 \rceil = 3$.

For negative numbers, $\lfloor -\sqrt{2} \rfloor = \lfloor -1.4142 \cdots \rfloor = -2$, and $\lceil -\sqrt{2} \rceil = -1$.

The number line may give an intuitive way to think about floor and ceiling: $\lfloor x \rfloor$ denotes the first integer that we encounter moving left in the number line starting at x; $\lceil x \rceil$ denotes the first integer that we encounter moving right from x. (And x itself counts for both definitions.) See Figure 2.3.

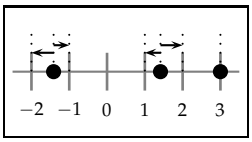


Figure 2.3: The floor and ceiling of $-\sqrt{2}$, $\sqrt{2}$, and 3.

2.2.4 *Exponentiation*

We next consider raising a number to an *exponent* or *power*.

Definition 2.5 (Raising a number to an integer power)
For a real number b and a nonnegative integer n, the number b^n denotes the result of multiplying b by itself n times:

$$b^0 := 1 \qquad \text{and, for } n \geq 1, \quad b^n := \underbrace{b \cdot b \cdots b}_{n \text{ times}}.$$

The number b is called the base *and the integer n is called the* exponent.

For example, $2^0 = 1$, $2^2 = 2 \cdot 2 = 4$, $2^5 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 32$, and $5^2 = 5 \cdot 5 = 25$.

Note again that $b^0 = 1$ for *any* base b, including $b = 0$. (The case of 0^0 is tricky: one is tempted to say *both* "0 to the anything is 0" *and* "anything to the 0 is 1." But, of course, these two statements are inconsistent. For us, the latter trumps the former, and $0^0 = 1$, as in Definition 2.5.)

Raising a base to nonintegral exponents

Consider the expression b^x for an exponent $x > 0$ that is not an integer. (It's all too easy to have done this calculation by typing numbers into a calculator without actually thinking about what the expression actually means!) Here's the definition of $b^{m/n}$ when the exponent $\frac{m}{n}$ is a rational number: