A COMPUTATIONAL INTRODUCTION TO NUMBER THEORY AND ALGEBRA

A COMPUTATIONAL INTRODUCTION TO NUMBER THEORY AND ALGEBRA

VICTOR SHOUP

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press
The Edinburgh Building, Cambridge CB2 2RU, UK

Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9780521851541

First published 2005

Printed in the United Kingdom at the University Press, Cambridge

A catalog record for this book is available from the British Library

ISBN-13 978-0-521-85154-1 hardback
ISBN-10 0-521-85154-8 hardback

ISBN-13 978-0-521-61725-3 paperback
ISBN-10 0-521-61725-1 paperback

Contents

Preface

Number theory and algebra play an increasingly significant role in computing and communications, as evidenced by the striking applications of these subjects to such fields as cryptography and coding theory. My goal in writing this book was to provide an introduction to number theory and algebra, with an emphasis on algorithms and applications, that would be accessible to a broad audience. In particular, I wanted to write a book that would be accessible to typical students in computer science or mathematics who have a some amount of *general* mathematical experience, but without presuming too much *specific* mathematical knowledge.

Prerequisites. The mathematical prerequisites are minimal: no particular mathematical concepts beyond what is taught in a typical undergraduate calculus sequence are assumed.

The computer science prerequisites are also quite minimal: it is assumed that the reader is proficient in programming, and has had some exposure to the analysis of algorithms, essentially at the level of an undergraduate course on algorithms and data structures.

Even though it is mathematically quite self contained, the text does presuppose that the reader is comfortable with mathematical formalism and has some experience in reading and writing mathematical proofs. Readers may have gained such experience in computer science courses such as algorithms, automata or complexity theory, or some type of "discrete mathematics for computer science students" course. They also may have gained such experience in undergraduate mathematics courses, such as abstract or linear algebra—these courses overlap with some of the material presented here, but even if the reader already has had some exposure to this material, it nevertheless may be convenient to have all of the relevant material easily accessible in one place, and moreover, the emphasis and perspective here

will no doubt be different than in a typical mathematics course on these subjects.

Structure of the text. All of the mathematics required beyond basic calculus is developed "from scratch." Moreover, the book generally alternates between "theory" and "applications": one or two chapters on a particular set of purely mathematical concepts are followed by one or two chapters on algorithms and applications—the mathematics provides the theoretical underpinnings for the applications, while the applications both motivate and illustrate the mathematics. Of course, this dichotomy between theory and applications is not perfectly maintained: the chapters that focus mainly on applications include the development of some of the mathematics that is specific to a particular application, and very occasionally, some of the chapters that focus mainly on mathematics include a discussion of related algorithmic ideas as well.

In developing the mathematics needed to discuss certain applications, I tried to strike a reasonable balance between, on the one hand, presenting the absolute minimum required to understand and rigorously analyze the applications, and on the other hand, presenting a full-blown development of the relevant mathematics. In striking this balance, I wanted to be fairly economical and concise, while at the same time, I wanted to develop enough of the theory so as to present a fairly well-rounded account, giving the reader more of a feeling for the mathematical "big picture."

The mathematical material covered includes the basics of number theory (including unique factorization, congruences, the distribution of primes, and quadratic reciprocity) and abstract algebra (including groups, rings, fields, and vector spaces). It also includes an introduction to discrete probability theory—this material is needed to properly treat the topics of probabilistic algorithms and cryptographic applications. The treatment of all these topics is more or less standard, except that the text only deals with commutative structures (i.e., abelian groups and commutative rings with unity)—this is all that is really needed for the purposes of this text, and the theory of these structures is much simpler and more transparent than that of more general, non-commutative structures.

The choice of topics covered in this book was motivated primarily by their applicability to computing and communications, especially to the specific areas of cryptography and coding theory. For example, the book may be useful for reference or self-study by readers who want to learn about cryptography. The book could also be used as a textbook in a graduate

or upper-division undergraduate course on (computational) number theory and algebra, perhaps geared towards computer science students.

Since this is an introductory textbook, and not an encyclopedic reference for specialists, some topics simply could not be covered. One such topic whose exclusion will undoubtedly be lamented by some is the theory of lattices, along with algorithms for and applications of lattice basis reduction. Another such topic is that of fast algorithms for integer and polynomial arithmetic—although some of the basic ideas of this topic are developed in the exercises, the main body of the text deals only with classical, quadratic-time algorithms for integer and polynomial arithmetic. As an introductory text, some topics just had to go; moreover, there are more advanced texts that cover these topics perfectly well, and these texts should be readily accessible to students who have mastered the material in this book.

Note that while continued fractions are not discussed, the closely related problem of "rational reconstruction" is covered, along with a number of interesting applications (which could also be solved using continued fractions).

Using the text. Here are a few tips on using the text.

- There are a few sections that are marked with a "(∗)," indicating that the material covered in that section is a bit technical, and is not needed elsewhere.
- There are many examples in the text. These form an integral part of the text, and should not be skipped.
- There are a number of exercises in the text that serve to reinforce, as well as to develop important applications and generalizations of, the material presented in the text. In solving exercises, the reader is free to use any *previously* stated results in the text, including those in previous exercises. However, except where otherwise noted, any result in a section marked with a "(∗)," or in §5.5, need not and should not be used outside the section in which it appears.
- There is a very brief "Preliminaries" chapter, which fixes a bit of notation and recalls a few standard facts. This should be skimmed over by the reader.
- There is an appendix that contains a few useful facts; where such a fact is used in the text, there is a reference such as "see §An," which refers to the item labeled "An" in the appendix.

Feedback. I welcome comments on the book (suggestions for improvement, error reports, etc.) from readers. Please send your comments to

victor@shoup.net.

There is also web site where further material and information relating to the book (including a list of errata and the latest electronic version of the book) may be found:

`www.shoup.net/ntb`.

Acknowledgments. I would like to thank a number of people who volunteered their time and energy in reviewing one or more chapters: Siddhartha Annapureddy, John Black, Carl Bosley, Joshua Brody, Jan Camenisch, Ronald Cramer, Alex Dent, Nelly Fazio, Mark Giesbrecht, Stuart Haber, Alfred Menezes, Antonio Nicolosi, Roberto Oliveira, and Louis Salvail. Thanks to their efforts, the "bug count" has been significantly reduced, and the readability of the text much improved. I am also grateful to the National Science Foundation for their support provided under grant CCR-0310297. Thanks to David Tranah and his colleagues at Cambridge University Press for their progressive attitudes regarding intellectual property and open access.

New York, January 2005 *Victor Shoup*

Preliminaries

We establish here a few notational conventions used throughout the text.

Arithmetic with ∞

We shall sometimes use the symbols "∞" and "$-\infty$" in simple arithmetic expressions involving real numbers. The interpretation given to such expressions is the usual, natural one; for example, for all real numbers x, we have $-\infty < x < \infty$, $x + \infty = \infty$, $x - \infty = -\infty$, $\infty + \infty = \infty$, and $(-\infty) + (-\infty) = -\infty$. Some such expressions have no sensible interpretation (e.g., $\infty - \infty$).

Logarithms and exponentials

We denote by $\log x$ the natural logarithm of x. The logarithm of x to the base b is denoted $\log_b x$.

We denote by e^x the usual exponential function, where $e \approx 2.71828$ is the base of the natural logarithm. We may also write $\exp[x]$ instead of e^x.

Sets and relations

We use the symbol $\emptyset$ to denote the empty set. For two sets A, B, we use the notation $A \subseteq B$ to mean that A is a subset of B (with A possibly equal to B), and the notation $A \subsetneq B$ to mean that A is a proper subset of B (i.e., $A \subseteq B$ but $A \neq B$); further, $A \cup B$ denotes the union of A and B, $A \cap B$ the intersection of A and B, and $A \setminus B$ the set of all elements of A that are not in B.

For sets $S_1, \ldots, S_n$, we denote by $S_1 \times \cdots \times S_n$ the **Cartesian product**

of $S_1, \ldots, S_n$, that is, the set of all n-tuples $(a_1, \ldots, a_n)$, where $a_i \in S_i$ for $i = 1, \ldots, n$.

We use the notation $S^{\times n}$ to denote the Cartesian product of n copies of a set S, and for $x \in S$, we denote by $x^{\times n}$ the element of $S^{\times n}$ consisting of n copies of x. (We shall reserve the notation S^n to denote the set of all nth powers of S, assuming a multiplication operation on S is defined.)

Two sets A and B are **disjoint** if $A \cap B = \emptyset$. A collection $\{C_i\}$ of sets is called **pairwise disjoint** if $C_i \cap C_j = \emptyset$ for all i, j with $i \neq j$.

A **partition** of a set S is a pairwise disjoint collection of non-empty subsets of S whose union is S. In other words, each element of S appears in exactly one subset.

A **binary relation** on a set S is a subset R of $S \times S$. Usually, one writes $a \sim b$ to mean that $(a, b) \in R$, where $\sim$ is some appropriate symbol, and rather than refer to the relation as R, one refers to it as $\sim$.

A binary relation $\sim$ on a set S is called an **equivalence relation** if for all $x, y, z \in S$, we have

- $x \sim x$ (reflexive property),
- $x \sim y$ implies $y \sim x$ (symmetric property), and
- $x \sim y$ and $y \sim z$ implies $x \sim z$ (transitive property).

If $\sim$ is an equivalence relation on S, then for $x \in S$ one defines the set $[x] := \{y \in S : x \sim y\}$. Such a set $[x]$ is an **equivalence class**. It follows from the definition of an equivalence relation that for all $x, y \in S$, we have

- $x \in [x]$, and
- either $[x] \cap [y] = \emptyset$ or $[x] = [y]$.

In particular, the collection of all distinct equivalence classes partitions the set S. For any $x \in S$, the set $[x]$ is called the the **equivalence class containing** x, and x is called a **representative** of $[x]$.

Functions

For any function f from a set A into a set B, if $A' \subseteq A$, then $f(A') := \{f(a) \in B : a \in A'\}$ is the **image** of A' under f, and $f(A)$ is simply referred to as the **image** of f; if $B' \subseteq B$, then $f^{-1}(B') := \{a \in A : f(a) \in B'\}$ is the **pre-image** of B' under f.

A function $f : A \to B$ is called **one-to-one** or **injective** if $f(a) = f(b)$ implies $a = b$. The function f is called **onto** or **surjective** if $f(A) = B$. The function f is called **bijective** if it is both injective and surjective; in this case, f is called a **bijection**. If f is bijective, then we may define the

inverse function $f^{-1} : B \to A$, where for $b \in B$, $f^{-1}(b)$ is defined to be the unique $a \in A$ such that $f(a) = b$.

If $f : A \to B$ and $g : B \to C$ are functions, we denote by $g \circ f$ their composition, that is, the function that sends $a \in A$ to $g(f(a)) \in C$. Function composition is associative; that is, for functions $f : A \to B$, $g : B \to C$, and $h : C \to D$, we have $(h \circ g) \circ f = h \circ (g \circ f)$. Thus, we can simply write $h \circ g \circ f$ without any ambiguity. More generally, if we have functions $f_i : A_i \to A_{i+1}$ for $i = 1, \ldots, n$, where $n \geq 2$, then we may write their composition as $f_n \circ \cdots \circ f_1$ without any ambiguity. As a special case of this, if $A_i = A$ and $f_i = f$ for $i = 1, \ldots, n$, then we may write $f_n \circ \cdots \circ f_1$ as f^n. It is understood that $f^1 = f$, and that f^0 is the identity function on A. If f is a bijection, then so is f^n for any non-negative integer n, the inverse function of f^n being $(f^{-1})^n$, which one may simply write as f^{-n}.

Binary operations

A **binary operation** $\star$ on a set S is a function from $S \times S$ to S, where the value of the function at $(a, b) \in S \times S$ is denoted $a \star b$.

A binary operation $\star$ on S is called **associative** if for all $a, b, c \in S$, we have $(a \star b) \star c = a \star (b \star c)$. In this case, we can simply write $a \star b \star c$ without any ambiguity. More generally, for $a_1, \ldots, a_n \in S$, where $n \geq 2$, we can write $a_1 \star \cdots \star a_n$ without any ambiguity.

A binary operation $\star$ on S is called **commutative** if for all $a, b \in S$, we have $a \star b = b \star a$. If the binary operation $\star$ is both associative and commutative, then not only is the expression $a_1 \star \cdots \star a_n$ unambiguous, but its value remains unchanged even if we re-order the a_i.