Undergraduate Topics in Computer Science

'Undergraduate Topics in Computer Science' (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems, many of which include fully worked solutions.

The UTiCS concept relies on high-quality, concise books in softback format, and generally a maximum of 275–300 pages. For undergraduate textbooks that are likely to be longer, more expository, Springer continues to offer the highly regarded Texts in Computer Science series, to which we refer potential authors.

More information about this series at http://www.springer.com/series/7592

Duncan Buell

Fundamentals of Cryptography

Introducing Mathematical and Algorithmic Foundations

Springer

Duncan Buell
Department of Computer Science
and Engineering
University of South Carolina
Columbia, SC, USA

ISSN 1863-7310 ISSN 2197-1781 (electronic)
Undergraduate Topics in Computer Science
ISBN 978-3-030-73491-6 ISBN 978-3-030-73492-3 (eBook)
https://doi.org/10.1007/978-3-030-73492-3

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

I taught in a computer science department. Our cryptography class, which I just finished teaching for the fourth time, is for upper-level undergraduates (juniors and seniors) and also for graduate credit. This is, I think, similar to what is available in a large number of universities in the United States. Our course is also cross-listed with the mathematics department, and we usually have a combination of math and computer science students taking the course.

The computer science students know how to program, but they are often not so good at the math. In contrast, the math students can usually do the math but don't have the same experience in programming. I have normally accepted programming assignments in any of Python, C++, or Java. I have pointed out that Python provides multiprecise integer arithmetic as a default, and have pointed the students using C++ to the `gmp` package and the students using Java to the `BigNum` package. I have also encouraged students to work in groups on the programming assignments, allowing groups to combine both the mathematical understanding and the ability to code up the algorithms and heuristics presented.

I have assumed some minimal background in asymptotic analysis of algorithms. I think I have kept the big-$\mathcal{O}(*)$ to a minimum, and I think virtually all the needed discrete math (modular arithmetic, groups, rings, and such) actually appears in this text.

I think most students in the United States see at least the rudiments of matrix reduction in high school; I have tried to make the linear algebra limited to what most students might actually see in high school; and I have not used any linear algebra beyond the mechanical process of matrix reduction.

Although there is a certain background in groups and rings, I have tried to make that as straightforward as possible. One advantage of this material is that it isn't pure theory; one can actually see concrete examples of the theorems and, I would hope, understand that the theory is mostly just a way of discussing the material and not the end in itself.

It is, in fact, precisely because this material does and should appear in both departments that I decided to write this book. There are several good crypto books that are suitable for a course that is just mathematics. There is at least one that is suitable for a lower-level course targeting a more general audience.

Cryptography, as done in this century, is heavily mathematical, but it also has roots in what is computationally feasible. What I would hope this text does, that I think other texts do not do, is balance the theorems of mathematics against the feasibility of computation. This is the essence of my including the material of Chap. 8 in the text. My background is mathematics. My experience (except for a first year in a math department after my doctorate) is in computer science departments, with 15 years at a research lab doing high performance computing research in support of the National Security Agency. My "take" on cryptography, including my time with NSA, is that it is something one actually "does", not a mathematical game one proves theorems about. There is deep math; there are some theorems that must be proved; and there is a need to recognize the brilliant work done by those who focus on theory. But at the level of an undergraduate course, I think the emphasis should be first on knowing and understanding the algorithms and how to implement them, and also to be aware that the algorithms must be implemented carefully to avoid the "easy" ways to break the cryptography.

Exercises

Most of the exercises in the later chapters are programming problems. This is largely due to the difficulty in posing realistic problems that can be done entirely by hand. As for computational resources: Many of the exercises, and many of the necessary exercises, require some computation. There are many tools that can be used. The (ancient, but certainly feasible) Unix tool `bc` will do the multiprecise arithmetic necessary for many of the computations and can be scripted using something like

```
bc < myscript.txt
```

to redirect from standard input. Some students will prefer Matlab® or Mathematica or Maple. Others may find SageMath a better tool. The key for some of the simpler problems is that students use the software as a sophisticated calculator but do the algorithms themselves rather than just calling functions built by someone else. The key for the genuine programming problems is that students learn the details of the algorithms, at least for small examples that would not require multiprecise arithmetic packages. Following the classic advice to "make it right before you make it better", if students can code single precision examples that work, then most of the work to code multiprecision examples will only be the transition from single to multiple precision, and the tracing of the single precision example can be used as ground truth for whether the multiple precision version has bugs.

Acknowledgements

It would be inappropriate for me not to thank my most recent cryptography class for which the first version of this was their textbook, and the previous classes for which earlier versions were written. I received great feedback from the most recent class, some corrections, and some suggestions for what should be changed and what had to be changed in order to become more readable. Everyone who teaches will complain about some of the students, but I have been fortunate to teach and work with a number of superstars; I am grateful to have had such students.

Columbia, SC, USA Duncan Buell
February 2021

Contents