Jean Gallier

Discrete Mathematics

December 11, 2010

Springer

To my family, especially Anne and Mia, for their love and endurance

Preface

The curriculum of most undergraduate programs in computer science includes a course titled *Discrete Mathematics*. These days, given that many students who graduate with a degree in computer science end up with jobs where mathematical skills seem basically of no use,[1] one may ask why these students should take such a course. And if they do, what are the most basic notions that they should learn?

As to the first question, I strongly believe that *all* computer science students should take such a course and I will try justifying this assertion below.

The main reason is that, based on my experience of more than twenty-five years of teaching, I have found that the majority of the students find it very difficult to present an argument in a rigorous fashion. The notion of a proof is something very fuzzy for most students and even the need for the rigorous justification of a claim is not so clear to most of them. Yet, they will all write complex computer programs and it seems rather crucial that they should understand the basic issues of program correctness. It also seems rather crucial that they should possess some basic mathematical skills to analyze, even in a crude way, the complexity of the programs they will write. Don Knuth has argued these points more eloquently than I can in his beautiful book, *Concrete Mathematics*, and I do not elaborate on this any further.

On a scholarly level, I argue that some basic mathematical knowledge should be part of the scientific *culture* of any computer science student and more broadly, of any engineering student.

Now, if we believe that computer science students should have some basic mathematical knowledge, what should it be?

There is no simple answer. Indeed, students with an interest in algorithms and complexity will need some discrete mathematics such as combinatorics and graph theory but students interested in computer graphics or computer vision will need some geometry and some continuous mathematics. Students interested in databases will need to know some mathematical logic and students interested in computer architecture will need yet a different brand of mathematics. So, what's the common core?

---

[1] In fact, some people would even argue that such skills constitute a handicap!

As I said earlier, most students have a very fuzzy idea of what a proof is. This is actually true of most people. The reason is simple: it is quite difficult to define precisely what a proof is. To do this, one has to define precisely what are the "rules of mathematical reasoning" and this is a lot harder than it looks. Of course, defining and analyzing the notion of proof is a major goal of mathematical logic.

Having attempted some twenty years ago to "demystify" logic for computer scientists and being an incorrigible optimist, I still believe that there is great value in attempting to teach people the basic principles of mathematical reasoning in a precise but not overly formal manner. In these notes, I define the notion of proof as a certain kind of tree whose inner nodes respect certain proof rules presented in the style of a natural deduction system "a la Prawitz." Of course, this has been done before (e.g., in van Dalen [6]) but our presentation has more of a "computer science" flavor which should make it more easily digestible by our intended audience. Using such a proof system, it is easy to describe very clearly what is a proof by contradiction and to introduce the subtle notion of "constructive proof". We even question the "supremacy" of classical logic, making our students aware of the fact that there isn't just one logic, but different systems of logic, which often comes as a shock to them.

Having provided a firm foundation for the notion of proof, we proceed with a quick and informal review of the first seven axioms of Zermelo–Fraenkel set theory. Students are usually surprised to hear that axioms are needed to ensure such a thing as the existence of the union of two sets and I respond by stressing that one should always keep a healthy dose of skepticism in life.

What next? Again, my experience has been that most students do not have a clear idea of what a function is, even less of a partial function. Yet, computer programs may not terminate for all input, so the notion of partial function is crucial. Thus, we carefully define relations, functions, and partial functions and investigate some of their properties (being injective, surjective, bijective).

One of the major stumbling blocks for students is the notion of proof by induction and its cousin, the definition of functions by recursion. We spend quite a bit of time clarifying these concepts and we give a proof of the validity of the induction principle from the fact that the natural numbers are well ordered. We also discuss the pigeonhole principle and some basic facts about equinumerosity, without introducing cardinal numbers.

We introduce some elementary concepts of combinatorics in terms of counting problems. We introduce the binomial and multinomial coefficients and study some of their properties and we conclude with the inclusion–exclusion principle.

Next, we introduce partial orders, well-founded sets, and complete induction. This way, students become aware of the fact that the induction principle applies to sets with an ordering far more complex that the ordering on the natural numbers. As an application, we prove the unique prime factorization in $\mathbb{Z}$ and discuss gcds and versions of the Euclidean algorithm to compute gcds including the so-called extended Euclidean algorithm which relates to the Bezout identity.

Another extremely important concept is that of an equivalence relation and the related notion of a partition.

As applications of the material on elementary number theory presented in Section 5.4, in Section 5.8 we give an introduction to Fibonacci and Lucas numbers as well as Mersenne numbers and in Sections 5.9, 5.10, and 5.11, we present some basics of public key cryptography and the RSA system. These sections contain some beautiful material and they should be viewed as an incentive for the reader to take a deeper look into the fascinating and mysterious world of prime numbers and more generally, number theory. This material is also a gold mine of programming assignments and of problems involving proofs by induction.

We have included some material on lattices, Tarski's fixed point theorem, distributive lattices, Boolean algebras, and Heyting algebras. These topics are somewhat more advanced and can be omitted from the "core".

The last topic that we consider crucial is graph theory. We give a fairly complete presentation of the basic concepts of graph theory: directed and undirected graphs, paths, cycles, spanning trees, cocycles, cotrees, flows, and tensions, Eulerian and Hamiltonian cycles, matchings, coverings, and planar graphs. We also discuss the network flow problem and prove the max-flow min-cut theorem in an original way due to M. Sakarovitch.

These notes grew out of lectures I gave in 2005 while teaching CIS260, Mathematical Foundations of Computer Science. There is more material than can be covered in one semester and some choices have to be made regarding what to omit. Unfortunately, when I taught this course, I was unable to cover any graph theory. I also did not cover lattices and Boolean algebras.

Beause the notion of a graph is so fundamental in computer science (and elsewhere), I have restructured these notes by splitting the material on graphs into two parts and by including the introductory part on graphs (Chapter 3) before the introduction to combinatorics (Chapter 4). The only small inconvenience in doing so is that this causes a forward reference to the notion of an equivalence relation which only appears in Chapter 5. This is not a serious problem. In fact, this gives us a chance to introduce the important concept of an equivalence relation early on, without any proof, and then to revisit this notion more rigorously later on.

Some readers may be disappointed by the absence of an introduction to probability theory. There is no question that probability theory plays a crucial role in computing, for example, in the design of randomized algorithms and in the probabilistic analysis of algorithms. Our feeling is that to do justice to the subject would require too much space. Unfortunately, omitting probability theory is one of the tough choices that we decided to make in order to keep the manuscript of manageable size. Fortunately, probability and its applications to computing are presented in a beautiful book by Mitzenmacher and Upfal [4] so we don't feel too bad about our decision to omit these topics.

There are quite a few books covering discrete mathematics. According to my personal taste, I feel that two books complement and extend the material presented here particularly well: *Discrete Mathematics*, by Lovász, Pelikán, and Vesztergombi [3], a very elegant text at a slightly higher level but still very accessible, and *Discrete Mathematics*, by Graham, Knuth, and Patashnik [2], a great book at a significantly higher level.

My unconventional approach of starting with logic may not work for everybody, as some individuals find such material too abstract. It is possible to skip the chapter on logic and proceed directly with sets, functions, and so on. I admit that I have raised the bar perhaps higher than the average compared to other books on discrete maths. However, my experience when teaching CIS260 was that 70% of the students enjoyed the logic material, as it reminded them of programming. I hope this book will inspire and will be useful to motivated students.

A final word to the teacher regarding foundational issues: I tried to show that there is a natural progression starting from logic, next a precise statement of the axioms of set theory, and then to basic objects such as the natural numbers, functions, graphs, trees, and the like. I tried to be as rigorous and honest as possible regarding some of the logical difficulties that one encounters along the way but I decided to avoid some of the most subtle issues, in particular a rigorous definition of the notion of cardinal number and a detailed discussion of the axiom of choice. Rather than giving a flawed definition of a cardinal in terms of the equivalence class of all sets equinumerous to a set, which *is not* a set, I only defined the notions of domination and equinumerosity. Also, I stated precisely two versions of the axiom of choice, one of which (the graph version) comes up naturally when seeking a right inverse to a surjection, but I did not attempt to state and prove the equivalence of this formulation with other formulations of the axiom of choice (such as Zermelo's well-ordering theorem). Such foundational issues are beyond the scope of this book; they belong to a course on set theory and are treated extensively in texts such as Enderton [1] and Suppes [5].

Acknowledgments: I would like to thank Mickey Brautbar, Kostas Daniilidis, Max Mintz, Joseph Pacheco, Steve Shatz, Jianbo Shi, Marcelo Siqueira, and Val Tannen for their advice, encouragement, and inspiration.

References

1. Herbert B. Enderton. *Elements of Set Theory*. New York: Academic Press, first edition, 1977.
2. Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation For Computer Science*. Reading, MA: Addison Wesley, second edition, 1994.
3. L. Lovász, J. Pelikán, and K. Vesztergombi. *Discrete Mathematics. Elementary and Beyond*. Undergraduate Texts in Mathematics. New York: Springer, first edition, 2003.
4. Michael Mitzenmacher and Eli Upfal. *Probability and Computing. Randomized Algorithms and Probabilistic Analysis*. Cambridge, UK: Cambridge University Press, first edition, 2005.
5. Patrick Suppes. *Axiomatic Set Theory*. New York: Dover, first edition, 1972.
6. D. van Dalen. *Logic and Structure*. Universitext. New York: Springer Verlag, second edition, 1980.

Philadelphia, November 2010 *Jean Gallier*

Contents