Book Reviews of papers by or inspired by Knuth

Edited by William Gasarch
Department of Computer Science
University of Maryland at College Park
College Park, MD, 20742
email: gasarch@cs.umd.edu

Welcome to this selection of book reviews from the SIGACT NEWS Book Reviews Column. In this document we reprint (reprint? sounds so quaint!) reviews of the following books that are either by or inspired by Donald Knuth.

1. **Selected Papers on Computer Science** by Donald E. Knuth. Reviewed by Samir Khuller. This is a collection of articles on computer for a wide audience.

2. **Stable Marriage and its Relation to Other Combinatorial Problems: An Introduction to Algorithm Analysis** by Donald Knuth. Reviewed by Tim McNichol. This book uses the stable marriage problem as motivation to look at some mathematics of interest. It would be useful for undergrads; however, for a serious study of matching there are more advanced and more up-to-date books available.

3. **A=B** by Marko Petkovšek, Herbert S. Wilf, and Doron Zeilberger. Reviewed by Vladik Kreinovich. Recall that Exercise 1.2.6.63 of Knuth Volume 1 is *Develop computer programs for simplifying sums that involve binomial coefficients.* This book solves the problem completely. The math is stunning, the algorithms actually work, and code is available in MAPLE or MATHEMATICA.

4. **Selected Papers on Analysis of Algorithms** by Donald E. Knuth. Review by Timothy H. McNicholl. This is a collection of papers by Donald Knuth on the Analysis of Algorithms. The papers take the reader through a large variety of mathematical techniques in a motivated way. **Selected Papers in Discrete Mathematics** by D. Knuth. Review by Carlos A.S. Oliveira. This book is a compilation of work published (plus some unpublished papers) by the author.

5. **The Art of Computer Programming Volume 4, Fascicles 2, 3, and 4** by Donald Knuth. Review by John Rogers. Volume 4 is actually out– at least in parts. I would describe it with one word: Knuthian. John Rogers, the reviewer, has more words.

6. **The Art of Computer Programming: Volume 4A** by Donald E. Knuth. Review by John D. Rogers. Say you want to list out all of the elements of $\{0,1\}^n$ quickly. How would you do it? How would you measure quickness? Say you wanted to list them out so that every strings differs from the prior one in only one bit (a Gray Code). Then how would you do it? This book is about how to quickly list out elements of a set. Given the author it has lots of historical context and mathematics of interest. Here is hoping we will one day see Volume 4B.

7. We review four volumes of collected papers by Donald Knuth.

   (a) **Selected Papers on Discrete Mathematics** by Donald E. Knuth. Review by Daniel Apon.

   (b) **Selected Papers on Design of Algorithms** by Donald E. Knuth. Review by Daniel Apon

   (c) **Selected Papers on Fun & Games** by Donald E. Knuth. Review by William Gasarch.

   (d) **Companion to the Papers of Donald Knuth** by Donald E. Knuth. Review by William Gasarch.

8. Joint review of **Algorithmic Barriers Falling: P=NP?** and **The Essential Knuth**. The author is Edgar Daylight and Donald Knuth. The reviewer is William Gaarch These are interviews with Donald Knuth. As the father of algorithmic analysis it is worth hearing his prospective on computer science.

Book Review for
"Selected Papers on Computer Science
Author of Book: Donald E. Knuth
Author of Review: Samir Khuller

"Selected papers on computer science" by Donald E. Knuth [**?**] is a collection of lectures and published papers aimed at a wide audience. A detailed narrative, replete with illustrations and examples, this collection focuses on three principle themes: the centrality of algorithms to computer science, the importance of combining theory and practice, as well as the historical development of the field.

Addressing the first of these issues, Knuth argues that computer science is essentially about algorithms and algorithmic thought. As he puts it, "Algorithms are the life-blood of computer science ... the common denominator that underlies and unifies the different branches". Defining algorithms as concepts that exist independent of programs, even though programming languages are used to express algorithms, Knuth states that "a person well trained in computer science knows how to deal with algorithms, how to construct them, manipulate them, understand them, analyze them". In fact, in Knuth's view, an attempt to comprehend mathematical concepts as algorithms results in a far deeper understanding than might ordinarily result. However, even as Knuth underscores the importance of algorithms as a "general purpose mental tool" he simultaneously identifies them as a point of difference between computer science and mathematics. Through a series of examples, he demonstrates that while mathematicians are content once they prove that a certain object that satisfies certain properties *exists*, a Computer Scientist tries to learn how to *compute* this object. Such a viewpoint, has led to fundamental advances in Computer Science[1]. Simpler proofs, as well as alternative proofs and a better understanding of mathematical structures can be achieved when one puts on a "Computer Science hat".

The second theme that Knuth focuses on is the importance of combining theory and practice, something that in his opinion can have a significant payoff for the field as a whole. As he puts is "the world gets better theoretical tools as it gets better practical tools". In this context, Knuth specifically points to his experiences in developing METAFONT, and argues that this

[1]In my opinion, Edmonds classic paper on matching [**?**] which developed a polynomial time algorithm for maximum matching, is another such example, where the search for an efficient algorithm to find a perfect matching led to the definition of polynomial time computability. Hall's theorem gives necessary and sufficient conditions for the existence of a perfect matching, but does not immediately yield an algorithm to find one.

very practically oriented project generated many interesting new problems and thus had a very profound impact on his research. Some of these problems are discussed in Chapters 6–9. Some sample problems are: solving the K-centers problem on a line (a minor modification of the Frederickson and Johnson [**?**]) algorithm solves this problem), digitizing straight lines, hyphenating words etc.

Pervasive throughout the book, are historical snippets and comments about the historical development of computer science and the impact of the Persian scholar Abu 'Abd Allāh Muhammad ibn Mūsā al-Khwārizmī, whose textbook on arithmetic had a significant influence for about 500 years. Chapters 11-12 take the reader through a grand tour of ancient Babylonian number systems, Algorithms, Von Neumann's first computer program. A chapter that I really enjoyed reading was on Knuth's experiences with programming the IBM 650 and its nuances.

Knuth also devotes a chapter to the contributions of George Forsythe (Foresight?) whose leadership and dedication was to a large extent responsible for establishing the Computer Science Department at Stanford, as well as in establishing Computer Science as a discipline.

Include is a list of the chapters in the book.

List of Chapters

0) **Algorithms, Programs and Computer Science**

1) **Computer Science and its Relation to Mathematics**

2) **Mathematics and Computer Science: Coping with Finiteness**

3) **Algorithms**

4) **Algorithms in Modern Mathematics and Computer Science**

5) **Algorithmic Themes**

6) **Theory and Practice I**

7) **Theory and Practice II**

8) **Theory and Practice III**

9) **Theory and Practice IV**

10) **Are Toy Problems Useful?**

11) **Ancient Babylonian Algorithms**

12) **Von Neumann's First Computer Program**

13) **The IBM 650: An Appreciation from the Field**

14) **George Forsythe and the Development of Computer Science**

15) **Artistic Programming**

Opinion

I must admit that I hesitated before taking on the task of writing a book review, since (a) I had never done this kind of thing before, (b) I thought that reading an entire book to comment on would take a huge amount of time. After reading the first half of the book, on a pleasant sunday afternoon, I was very glad that I took on this task. This book is written for a very broad audience, and the writing style makes reading it a very stimulating experience. The book is accessible to a reader with some knowledge of computer science or mathematics (an undergraduate degree in computer science certainly suffices), but is excellent reading for graduate students and faculty as well.

The book [**?**] in the usual Knuth style, pays a lot of attention to detail, and is fascinating reading for students of Computer Science, interested in the historical development of the field; especially those who think the field is simply about the *Web* and *Java*! In all, I found the discussion immensely satisfying and further reinforced my belief that algorithms are extremely useful and play a role in many unusual and unexpected situations.

References

[1] J. Edmonds, "Paths, trees and flowers", *Canad. J. of Math.*, 17 (1965), pp. 449–467.

[2] G. N. Frederickson and D. B. Johnson, "Finding kth paths and p-centers by generating and searching good data structures", *Journal of Algorithms*, 4 (1983), pp. 61-80.

[3] D. E. Knuth, *Selected papers on Computer Science*, Cambridge Univ. Press, (1996).

Review of
Stable Marriage and its Relation to Other Combinatorial Problems:
An Introduction to Algorithm Analysis[2]
Author: Donald E. Knuth
Published by American Mathematical Society in 1996
$19.00, Softcover
ISBN number 0-821-80603-3

Reviewer: Timothy H. McNicholl

1 Overview

This book is a translation of the revised and corrected edition of *Marriages stables et leurs relations avec d'autres problèmes combinatories* published in 1976 by *Les Presses de l'Université de Montréal* . It is based on a series of lectures given by the author at the Centre de recherches mathematicques. The purpose of the book is to give an informal introduction to algorithm analysis in which the main ideas are introduced by example rather than by theorems.

The chief example used is the stable marriage problem which can be defined as follows. A finite set of 'men' is given along with a set of 'women' of the same cardinality. In addition, each person is assumed to have a ranking of the members of the opposite sex. A *matching* is a bijection between the two sets. We think of a matching as a set of n monogamous marriages. An *unstable* matching is one in which there is a man X and a woman Y such that X ranks Y higher than his spouse and Y ranks X higher than her spouse. The methods used to find the mean number of steps required by this algorithm are then applied to an algorthm to find the shortest distances between a node and all the other nodes in a graph. An algorithm for storing information in a table by hashing is then considered and the results obtained are used to develop a lower bound on this mean in the case where the women all have the same ranking of the men. The asymptotic value of this mean as a function of the number of men is then obtained using a probabalistic method.

2 Summary of Contents

The material is presented in seven 'lectures'. The first of these defines the stable marriage problem in a suitably informal style (no mention is made of linear orders and the like; rather, rankings are presented via matrices). Examples are given to illustrate stable and unstable matchings as well as to demonstrate that many 'societies' have several stable matchings. This raises the issues as to whether some stable matchings are better than others and whether there exists an 'optimal' stable matching. These issues are addressed in the problem set for this chapter via the following provocative exercise: show that the matching obtained by marrying each man to the highest ranked of all brides he can obtain in *some* stable matching is a stable matching. This matching, which is 'male-optimal' by definition, turns out to be 'female-minimal' in the sense that no woman can do any worse than she does in this matching.

Lecture 2 presents an algorithm for obtaining a stable matching. Although this terminology is not used in the text, the algorithm could be called the 'courtship algorithm' since it mimics how marriages are formed in many societies. The algorithm is mercilessly presented in pseudo-ALGOL, but its operation can be summarized as follows. Each man continues to make marriage proposals until he becomes permanently engaged. Only one man is allowed to make proposals at any point in time, and whenever a man makes a proposal, he does so to the woman he ranks highest out of all those who have not previously rejected or 'dumped' him. A woman rejects a proposal only if she is already engaged to a man she likes better than the suitor; otherwise she dumps her current fiance in order to accept the proposal. It is shown that the algorithm eventually terminates and that when it does the matching obtained is stable. Furthermore, it is shown that the matching obtained is the male-optimal matching discussed in the exercises for lecture 1. An example is laborously worked thorugh in order to illustrate the operation of the algorithm. More insight into the algorithm's machinations could have been conveyed if an informal presentation of the main ideas involved had preceeded the pseudo-code.

The number of propsosals made during the course of the algorithm provides a good estimate of the number of steps needed by the algorithm, and the former quantity is analyzed in lecture 3. It is shown that the mean number of proposals made when there are n men and n women and the preference matrix of the women is fixed is no more than $(n-1)H_n+1$ where H_n is the sum of the first n terms of the harmonic series. Only an upper bound is obtained since the problem is simplified by assuming that the men

have 'partial amnesia'. Namely, whenver they make a proposal, they can only remember the last woman who has dumped or rejected them whenever they are making a proposal. As a result each man possibly makes some redundant proposals and hence and upper bound is obtained. The assumption of partial amnesia allows the problem to be almost completely reduced to that of determining the probability that a coupon collector who already posseses m of n coupons next obtains a coupon already in her collection.

In the fourth lecture, these techniques are applied to Dijkstra's algorithm for finding the shortest distances from a node in a graph to each of the other nodes in the graph. The application is made via an analogy between the nodes in the graph and the women in the marriage algorithm. The logic is glossed over, but it is demonstrated that the mean number of steps in Dijkstra's algorithm is no more than the mean number of proposals in the marriage algorithm.

The fifth lecture considers the searching of a table via hashing. It is shown in a straightforward manner that when there are m items in a table with n storage blocks, then the mean number of steps required to place the next item is

$$\frac{n+1}{n+1-m}.$$

This result is then used to obtain the mean number of steps in the marriage algorithm in the special case where all the women have the same ranking of the men. It is shown that in this case the mean number of proposals when there are n men is $(n+1)H_n - n$. It is conjectured that this is a lower bound on the mean number of proposals when the preference matrices for both sexes are arbitrary. The asymptotic value of this mean is then obtained using a probabilistic approach and is shown to be $nH_n + O(\log^4\ n)$.

The sixth lecture gives an exact implementation of the marriage algorithm in ALGOL along with a recursive algorithm for finding all stable marriages for given sets of men and women and associated preference matrices. Lecture 7 presents a list of open problems some of which have been solved since the book was first published and some of which have not.

3 Opinion

The text provides many interesting examples and problems for students to study while learning the topic of algorithm analysis. However, the writing is too terse at some key points for this book to be used as the main text for a course in this subject. Perhaps this is due to the murkiness endowed by

repeated translation (the lectures were given in english to french students; they were later translated into french and a translation into english is the subject of this review). But a professor willing to fill in some of the gaps could use this book as a supplementary text.

A more modern book on the subject which the serious resercher might want to consult is *The Stable Marriage Problem* by Gusfield and Irving (MIT Press, 1989).

Review of $A = B$
Author of Book: Marko Petkovšek, Herbert S. Wilf, and Doron Zeilberger
Author of Review: Vladik Kreinovich
A.K. Peters, Ltd., Wellesley, MA, 1996, 212 pages

Often, the problem of finding the computational complexity of a certain algorithm can be reduced to computing a combinatorial sum – i.e., a sum like this:

$$\sum_k \binom{2n-2k}{n-k}\binom{2k}{k}. \tag{1}$$

This is often true when we are interested in the worst-case complexity, and this is even more frequently true when we are looking for the average-case complexity. There are so many combinatorial sums in these problems that even writing them down is difficult: suffice it to say that Donald E. Knuth has spent so much time making these formulas look right in his famous *The Art of Computer Programming* books that after Volume 3, he decided to take a break from these books and develop an automated way to type such formulas – an effort which eventually led to TEX and its dialects.

Even teaching a computer to *write down* all these formulas is so tough a task that it requires a multi-year effort – well, teaching a computer to *derive* these formulas is even a greater challenge. Knuth himself formulated this challenge in the very first volume (*Fundamental Algorithms*) of his book, as Exercise 1.2.6.63: *Develop computer programs for simplifying sums that involve binomial coefficients.* Knuth gave this problem the rating of 50 – meaning that it is one of the toughest problems in the book. It was tough. For several decades, it avoided solution. The good news is that after 30 years of extraordinary efforts (largely the efforts of this book's authors), this problem is largely solved: there are algorithms, there are effective computer programs which make handling the cumbersome combinatorial sums unbelievably easy. This book, proudly starting with the formulation of the original Knuth's problem and with an enthusiastic preface by Knuth himself, is a culmination of these efforts. This book is about the problem, the history of its solution, the resulting algorithms, and finally – about the programs.

Actually, there are *two* problems here. The *original problem* is when we have an expression for the sum, and we would like to have a more compact expression for it. Usually, when we cannot immediately find an answer, a natural idea is to compute this sum for $n = 0, 1, 2, \ldots$, and try to guess the answer. Very often, the answer is easy to guess, and then we run into a *second problem:* to prove that this guess is indeed a correct formula for all n. For example, for the above sum (1), Knuth computed it for $n = 0, 1, 2, 3,$

and came up with the values $1, 4, 16, 64$. A natural guess is that for every n, the desired sum is equal to 4^n, but proving that this sum is indeed always equal to 4^n turned out to be hard.

In general, the second problem is as follows: given two combinatorial expressions A and B, check whether they always coincide, i.e., whether $A = B$ (this is where the unusual book title comes from).

One of the main reasons why these problems turned out to be so hard is that not only did we not have efficient *programs* for checking such equalities, but researchers were not even sure that this problem was algorithmically solvable at all. Indeed, we want to check formulas of the type $\forall n(A(n) = B(n))$, with a universal quantifier running over all natural numbers. For computable functions $A(n)$ and $B(n)$, checking such formulas is equivalent to solving a halting problem – an algorithmically undecidable task.

However, in spite of the fact that checking such equalities is impossible for *general* computable functions, Knuth believed that it may be possible for a *specific* class of computable functions – functions which contain sums of combinatorial expressions. This hope and optimism came naturally from decades of frustration. This phrase sounds somewhat paradoxical, so let me clarify it. Few (maybe none) computer science theoreticians are fond of handling combinatorial sums (which appear again and again in complexity problems). These sums are challenging, tough, often require a lot of effort – sometimes as much (or even more) effort than the computer science part of the problem – but the reward and recognition for solving a combinatorial problem is usually much-much smaller than for solving the computer science problem. Why? Because the solution is usually very technical, rather short, not very exciting, usually – a clumsy complicated combination of simple ideas and transformations rather than than a stroke of genius. This combinatorial part is very frustrating, but in this very frustration lies hope: since all these sums seem to be computed by using the same ideas, maybe these ideas are indeed sufficient, and a computer can take care of finding the appropriate combination of these ideas? This hope turned out to be true.

The history of the resulting algorithms starts with Carl Friedrich Gauss (1755-1837), a genius who was recognized, in his time, as the King of Mathematicians. Probably everyone heard the (apocryphal) story how a ten-year old Gauss, when asked to find the sum $1 + 2 + \ldots + 100$, came up with a general formula $1+2+\ldots+n = n(n+1)/2$. The trick that he used to design this formula – grouping together the first and the last term, the second and the second from last, etc. – can actually help to add up an arbitrary arithmetic progression. It is less known (and true) that Gauss kept an interest in similar sums for his entire life. He found many interesting formulas for

specific sums $\sum_k t(k)$ – and he was also looking for *general* formulas of this type.

The two most well-known cases in which we have an explicit formula for the sum are *arithmetic* and *geometric* progressions. A geometric progression $t(k) = \text{const} \cdot q^k$ can be described by the condition that the ratio of every two consecutive terms is a constant: $t(k+1)/t(k) = q$. For an arithmetic progression $t(k) = a + b \cdot k$, a similar ratio is a rational function:

$$\frac{t(k+1)}{t(k)} = \frac{a + b \cdot (k+1)}{a + b \cdot k}.$$

It is therefore natural to define a *generalization* of geometric and arithmetic progressions by considering series in which the ratio $t(k+1)/t(k)$ is a rational function:

$$\frac{t(k+1)}{t(k)} = \frac{P(k)}{Q(k)}$$

for some polynomials $P(k)$ and $Q(k)$. Gauss called such series *hypergeometric*. Example of hypergeometric series include Taylor series for most elementary functions: e.g., $\exp(x) = \sum_k t(k)$, where

$$t(k) = \frac{x^k}{k!},$$

and the ratio $t(k+1)/t(k) = x/(k+1)$ is a rational function of k. The usefulness of this notion for combinatorial sums comes from the fact that binomial coefficients are hypergeometric series: if $t(k) = \binom{n}{k}$, then $t(k+1)/t(k) = (n-k)/(k+1)$ is a rational function of k.

How can we describe the sum of such terms? In the two above cases in which we have an explicit formula for the "indefinite" sum $s(n) = t(1) + \ldots + t(n)$, the sum is either itself hypergeometric – for an arithmetic progression, or a hypergeometric term $z(n)$ plus a constant c ($s(n) = z(n) + c$) – for a geometric progression. It turns out that such a term $z(n)$ can be found in many other cases. The question of when the sum $s(n) = \sum_{i=0}^{n} t(i)$ can be represented as $z(n) + c$ for some hypergeometric term $z(n)$ was analyzed, in the late 1970s, by R. W. Gosper Jr. who was working on the first symbolic computation program MACSYMA. Gosper developed an algorithm which, given a hypergeometric term $t(n)$, checks whether the sum is hypergeometric, and if it is, produces the corresponding hypergeometric expression for the sum. This algorithm was actually implemented in *Macsyma*. From the

mathematical viewpoint, whenever we can find such an expression $z(n)$, it is great. But do we gain anything from the *computational* viewpoint?

At first glance, if we consider *algebraic complexity* (number of arithmetic operations needed to compute a term), we do not gain anything at all. Indeed, what we do gain computationally when we know that a term $t(n)$ is hypergeometric, i.e., that the ratio
$t(k+1)/t(k)$ is a rational function of k? Computing the value of a rational function requires a finite number C of arithmetic operations, so we need C operations to compute $t(1)$ from $t(0)$, C operations to compute $t(2)$ from $t(1)$, etc., until we compute $t(n)$. So, totally, we need $O(n)$ steps to compute $t(n)$. Similarly, if we know that the sum $s(n)$ is hypergeometric, we can compute it in $O(n)$ steps.

On the other hand, even if the sum $s(n)$ is *not* hypergeometric, we can still compute it in $O(n)$ steps: indeed, we start with $t(0) = s(0)$, and then, for $i = 1, \ldots, n$, compute $t(i+1) = t(i) \cdot (P(i)/Q(i))$ (constant number of operations) and $s(i+1) = s(i) + t(i+1)$ (one more operation). The multiplicative constants in these two $O(n)$'s may be different, but it is not even clear which one is faster: on one hand, we need an extra addition to compute $s(n)$ as a sum, but, on the other hand, $s(n)$ can be hypergeometric with a more complex polynomials P and Q.

In short, from the *algebraic* complexity viewpoint, there may be no advantage in using this mathematically interesting result. However, from the viewpoint of *actual* computation time, there is a serious advantage, because actual computation can use *special functions* in addition to arithmetic operations. Specifically, if we factorize both polynomials $P(k)$ and $Q(k)$, we get the expression

$$\frac{t(k+1)}{t(k)} = \frac{(k+a_1)\cdot\ldots\cdot(k+a_p)}{(k+b_1)\cdot\ldots\cdot(k+b_q)}\cdot x$$

for some (generally complex) numbers a_i, b_j, and x. Thus, by using a gamma function $\Gamma(x)$, for which $\Gamma(x+1) = x\cdot\Gamma(x)$ and $\Gamma(n+1) = n!$ for integer n, we get

$$t(n) = c\cdot\frac{\Gamma(n+a_1)\cdot\ldots\cdot\Gamma(n+a_p)}{\Gamma(n+b_1)\cdot\ldots\cdot\Gamma(n+b_q)}\cdot x^n, \tag{2}$$

where

$$c = t(0)\cdot\frac{\Gamma(b_1)\cdot\ldots\cdot\Gamma(b_q)}{\Gamma(a_1)\cdot\ldots\cdot\Gamma(a_p)}.$$

We can also use the formula $x^n = \exp(n\cdot\ln(x))$ to compute x^n. So, if we count the application of exp and Γ functions as one step each in our

description of a generalized algebraic complexity, then we can conclude that we can compute $t(n)$ in finitely many computational steps. In this case, finding an explicit hypergeometric expression for the sum $\sum t(i)$ is extremely computationally useful: it brings the complexity down from $O(n)$ to $O(1)$ operations.

One thing deserves mentioning here: from the viewpoint of *practical* computations, the formula (2) is not very useful, because it requires us to divide two extremely large numbers (of orders $n!$) to get a number of reasonable size. This division (similarly to a more well known case of subtracting two large almost equal numbers) is not a good way to computing, because for the result to come out correct, we need to compute both the numerator and the denominator with a very high accuracy. A much better formula can be obtained if we use, instead of the actual (fast growing) gamma functions, auxiliary functions $\Gamma'(n, a_i) = \Gamma(n + a_i)/n! = \Gamma(n + a_i)/\Gamma(n + 1)$. In terms of these auxiliary functions, the expression (2) takes the easier-to-compute form

$$t(n) = c \cdot \frac{\Gamma'(n, a_1) \cdot \ldots \cdot \Gamma'(n, a_p)}{\Gamma'(n, b_1) \cdot \ldots \cdot \Gamma'(n, b_q)} \cdot n!^{q-p} \cdot \exp(n \cdot \ln(x)). \qquad (3)$$

This expression was actually proposed (in slightly different notations) by Gauss himself. Interestingly, *mathematicians* (and even the authors of the book under review) consider the introduction of the $\Gamma(n+1)$ terms in Gauss's formulas as an unnecessary complication – because it does make formulas (slightly) longer, but, as we have just seen, it make perfect *computational* sense.

In most combinatorial sums used in the analysis of algorithm complexity, the sum goes over all the values of the index for which the corresponding binomial coefficients are different from 0. We can therefore describe the desired sums as *definite* sums $\sum_k t(k)$, where k runs over all the integers. If the corresponding indefinite sum $s(n) = \sum_{k=0}^{n} t(k)$ is hypergeometric, then we can also compute the corresponding definite sum. But what if the indefinite sum is *not* hypergeometric (and often it is not)? In many of such cases, the definite sum is either hypergeometric itself, or is a sum of several hypergeometric term. Chapter 8 describes an algorithm which, given an integer r, checks whether a definite sum $\sum_k t(k)$ can be represented as a sum of r (or fewer) hypergeometric terms (and explicitly produces these terms if they exist). This algorithm, first proposed in Petkovšek's 1991 Ph.D. dissertation, capitalizes on the techniques previously proposed by the two other authors

of the book.

In some cases, the desired sum $\sum t(k)$ is not hypergeometric, so we need a larger class of functions to describe such sums. The main reason why such a class is needed is that the class of hypergeometric functions is not algebraically closed: while the *product* of two hypergeometric functions is (trivially) also hypergeometric, the *sum* is often not: one can easily check that even the sum $t(n) = a_1^n + a_2^n$ of two geometric progressions is not hypergeometric.

In this particular non-hypergeometric sum, the first term $t_1(n) = a^n$ is a solution to the following first-order difference equation with constant coefficients: $(S - a_1)t_1 = 0$, where $(St)(n) \stackrel{\text{def}}{=} t(n+1)$; similarly, the second term $t_2(n) = a_2^n$ is a solution to $(S - a_2)t_2 = 0$, hence the sum is a solution to the following *second-order* difference equation with constant coefficients: $(S - a_1)(S - a_2)t = 0$, i.e., $t(n+2) = (a_1 + a_2) \cdot t(n+1) - a_1 \cdot a_2 \cdot t(n)$.

In general, each hypergeometric term $t_i(k)$ is, by definition, a solution to the first-order difference equation with *rational* coefficients $t_i(k+1) = (P_i(k)/Q_i(k)) \cdot t_i(k)$. Therefore, the sum $t(k) = t_1(k) + \ldots + t_r(k)$ of such terms satisfies a higher-order difference equation with rational coefficients:

$$t(k+r) = \frac{P_1'(k)}{Q_1'(k)} \cdot t(k+r-1) + \ldots + \frac{P_r'(k)}{Q_r'(k)} \cdot t(k). \tag{4}$$

Functions satisfying such equations (4) are called *holonomic*. The set of holonomic functions is closed under addition, multiplication, indefinite addition, etc. A natural algorithmic definition of such a function starts with functions which are solutions of equations (4), and then allows addition, multiplication, etc.

There exist algorithms for checking whether two given holonomic functions (of one or several variables) coincide or not.

The programs in MAPLE and MATHEMATICA implementing all above algorithms can be downloaded from the book's website http://www.cis.upenn.edu/ wilf/AeqB.html. Moreover, the entire book can be downloaded for free (the visionary publisher agreed to it!). For those who want to study the book seriously: there are several minor typos – which are all mentioned on the book's website.

It is worth mentioning, that for several variables, the problem of checking whether $A = B$ quickly becomes computationally intensive. This is not surprising since one can show that satisfiability can be described in these terms. Namely, if we have a propositional formula with v Boolean variables $z_1, \ldots, z_v$, then we can define n auxiliary holonomic functions $a_i(n_i), 1 \le i \le v$, as follows: $a_i(0) = 1$ and $a_i(n_i + 1) = -a_i(n_i)$ (hence, $a_i(n_i) = (-1)^{n_i}$).

Constants are holonomic functions, and sums and products of holonomic functions are holonomic, so for every i, the function

$$s_i(n_i) = \frac{1}{2} \cdot (a_i(n_i) + 1)$$

is also holonomic. We assign, to every Boolean variable z_i, the corresponding function $f^{[z_i]} = s_i(n_i)$, to its negation, the function $f^{[\bar{z}_i]} = 1 - f^{[z_i]}$; to a disjunction, the function $f^{[a \vee b]} = f^{[a]} + f^{[b]} - f^{[a]} \cdot f^{[b]}$, and to conjunction, $f^{[F_1 \& F_2]} = f^{[F_1]} \cdot f^{[F_2]}$. As a result, to the original Boolean formula F, we assign a holonomic function $f^{[F]}(n_1, \ldots, n_v)$ such that this function f is identically 0 if and only is not satisfiable. Thus, checking equalities for thus defined holonomic functions is *NP-hard*.

Moreover, we can add $\sum_{n_i=0}^{1}$ instead of $\exists z_i$ and $\prod_{n_i=0}^{1}$ instead of $\forall z_i$ and thus get a holonomic function equivalent to an arbitrary *quantified satisfiability* problem – hence, we can prove that the corresponding problem is *PSPACE-hard*.

In this review, I have only mentioned the computational complexity of the problems and algorithms, and I said nothing about a beautiful mathematical theory behind them, a theory for which Herbert Wilf and Doron Zeilberger received a prestigious Steele prize from the American Mathematical Society. I just want to mention that a large part of the algorithms for computing a definite sum $f(n) = \sum_k F(n,k)$, where $F(n,k)$ is a hypergeometric function of both variables, is based on the possibility to find a "dual" hypergeometric expression $G(n,k)$ for which either

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k) \tag{5}$$

– then $f(n) = \text{const}$ – or a more sophisticated equality hold for some polynomials $a_j(n)$

$$\sum_{j=0}^{J} a_j(n) \cdot F(n+j,k) = G(n,k+1) - G(n,k) \tag{6}$$

– in this case $f(n)$ is a holonomic function satisfying the difference equation

$$\sum_{j=0}^{J} a_j(n) f(n+j) = 0.$$

There is an algorithm which, given F, always returns G and a_i. The left-hand side of (6) can be unraveled via unraveling several consequent first-order difference operators – like a telescoping fishing rod. This is how Zeilberger called his algorithm – *creative telescoping*.

The equality (5) is as simple and as beautiful as its continuous analog

$$\frac{\partial F}{\partial x} = \frac{\partial G}{\partial y}$$

which is a part of a definition of a complex analytical function $F(x, y) + \mathrm{i}G(x, y)$, and the consequences of the simple equality (5) are as deep, unexpected, beautiful (and as computationally helpful) as those of complex analysis.

The book also contains an interesting history, from the pioneer 1945 algorithm by Sister Mary Celine Fasenmyer (who happened to die, at the age of 90, in the exact same year in which this book was published), to algorithms and results of Gosper, Zeilberger, Wilf, and Petkovšek.

Although the book is very technical, it is written in a very popular and understandable way. It starts with the basics, it gently guides the reader through the programs, through the formulas, and through the numerous examples. Because of the book's unusual level of understandability, when it was published in 1996, it was selected as a Main Selection of the Library of Science – an honor rarely reserved for technical books.

I just love this book, and I hope y'all will too.

A review of **Selected Papers on Analysis of Algorithms**
Author of Book: Donald E. Knuth
Author of Review: Timothy H. McNicholl
Trade Paperback, February 2000, 540 pages
Cambridge University Press

This book contains an excellent collection of Donald Knuth's papers on the analysis of algorithms. In addition, there are a few papers on number theory. The chronological range of the essays begins with Knuth's first published paper on algorithms in 1963 ('Analysis of Length of Strings in a Merge Sort') and continues until the present day ('Linear Probing and Graphs', 1998). The range of topics, although mostly confined to the analysis of algorithms, is vast. There are papers on game algorithms, algorithms related to common mathematical problems (greatest common divisors, factoring, ...), and algorithms related to computer science problems. There are even a few philosophical and etymological essays. It is this wide range of topics that makes this a hard book to review. To focus on any one aspect of this book would give short-shrift to the others. I therefore decided to focus this review on **why I believe every reader of SIGACT news should buy this book**. I divided my reasons up into categories as follows.

Historical essays (the evolution of ...) The evolution of current terminology and fundamental unifying ideas can be found in these essays. For example, the currently accepted use of Ω, O, Θ is laid down in 'Big Omicron and Big Omega and Big Theta'. Interestingly, the earliest use of Ω turns out to be a paper by Hardy and Littlewood [HL]. Their definition of $\Omega(f(n))$ is not the same as Knuth's, but is similar (it requires the key inequality to hold for infinitely many n instead of all sufficiently large n). Knuth makes the case that his definition is more appropriate for computer science, and that since Hardy and Littlewood's definition is not widely used, there is no danger in his re-definition of Ω conflicting with current usage. It should also be noted that with Knuth's definition we get the identity $\Theta = O \cap \Omega$.

The essays 'A terminological proposal' and 'Postscript about NP-hard problems' trace the evolution of the term **NP-hard**. In the first paper, a number of terms for this class are proposed such as 'Herculean', 'arduous', 'formidable'. A poll is taken among SIGACT readers to determine which is most acceptable. The term **NP-hard** is not on this list, but is a write-in. Its consonance with other terminology and descriptive power win out over democratic considerations.

Other essays trace the emergence of the key ideas in algorithm analysis such as average-case complexity, asymptotic values, and generating func-

tions. For example, the author's first paper in the subject, 'Length of Strings in a Merge sort', published in 1963, examines the average length of strings that occur in a kind of merge sort (which doesn't look at all like the merge sort we teach our students today). Many of the early papers examine algorithms that are stated in a very machine-like context. With the evolution of Algol, it becomes possible to state them in a purer fashion so that their mathematical essence shines through. 'Mathematical Analysis of Algorithms', published in 1972, clearly articulates the basic kinds of questions asked such as asymptotic behavior of average case complexity and lower bounds on algorithmic solutions of a certain kind to a given problem. The difficulties with the latter kind of problem are also lucidly described. In fact, this essay would make excellent reading for a beginning student in algorithm analysis. Some of the history of random tree algorithms can be found in 'A trivial algorithm whose analysis isn't'.

Philosophical essays Certainly, the most provocative essay in this book for readers of SIGACT news is 'The Dangers of Computer Science Theory.' The question is posed , 'Has theoretical computer science contributed anything of value to programmers?' The essay then goes on to catalog many hilarious misapplications of theory to practice from a not-so-random number generator to a class of sorting algorithms for which bubble sort is optimal. Of course, this only shows that there have been a significant number of blunders, not that there haven't been any successes. *E.g.* as a programmer I am much better off for knowing that quicksort is usually better than selection sort unless I expect the input lists to be 'nearly sorted'. The preceding essay in the book, 'Mathematical Analysis of Algorithms', also provides a partial answer. Namely, when engaging in the analysis of algorithms we sharpen those mental faculties we will use in everyday programming tasks. What his examples **do** certainly demonstrate is the danger of taking theory as gospel before empirical testing to see how well our models reflect the real world.

Methodological essays Knuth does an amazing job of relating **how** he approaches problems rather than merely recording a highly polished solution which to the reader seems to come out of the blue. An essay that illustrates this is 'An Analysis of Alpha-Beta Pruning.' Here, a very subtle mistake involving conditional probability is intentionally left in one of the derivations, whose conclusion then turns out to be wrong. The discovery of the mistake through empirical testing of the results, and its correction are then described. The paper is thus more instructive than had the final correct version appeared by itself.

It's just plain fun to read. I will briefly discuss three essays I found most delightful for the problems they explored and the range of mathematics they employed.

(1) 'Linear probing and graphs'.

A surprising connection is found between a pure computer science problem and a problem of graph theory. The situation unfolds as follows. Consider a hashing table with m table slots and n keys and where the hashing is done by linear probing. Let d be the **total displacement** of the resulting table. That is, the sum over all keys k of the displacement between k's initial probe and where it winds up in the table. Let $F_{m,n}(x) = \Sigma x^d$ where the sum ranges over all m^n hash sequences. Thus, the coefficient of x^d in this sum is the number of hash sequences that yield displacement d. Let $F_n(x) = F_{n+1,n}(x)$. Now, let $C_{m,n}$ be the number of connected graphs on n vertices and m edges. It is shown that

$$F_n(1+w) = \sum_{i=0}^{\infty} w^i C_{n+i,n+1}.$$

Unfortunately, this beautiful and surprising result is demonstrated through primarily algebraic means. It would be nice to find a proof that was more combinatorial in nature.

(2) 'The subtractive algorithm for greatest common divisors'.

Algorithms for finding the greatest common divisor are some of the oldest in existence. They demonstrate that our discipline has origins that pre-date the modern computer. In this paper, an amazingly simple algorithm due to the ancient Greeks and older than Euclid's algorithm is examined. The algorithm works as follows. Given a pair of distinct numbers, subtract the smaller from the larger. Replace the larger with the difference. Repeat until the two numbers are the same. This common value is the greatest common divisor of the two original numbers. Let $S(n)$ be the average number of steps performed by this algorithm on pairs of the form (m, n) where $1 \leq m \leq n$. (It is assumed that all such values of m are equally likely.) It is shown that

$$S(n) = 6\pi^{-2}(\log(n))^2 + O(\log(n)(\log\log(n))^2).$$

Average-case results are known for the Euclidean algorithm. Amongst these, the one that is most comparable to the above result is that the average amount of steps performed by Euclid's algorithm on two numbers that are

both $\leq n$ is is $O((\log n)^2)$ [Kn2]. Hence the subtraction algorithm is (surprisingly) of the same order of magnitude.

(3) 'The toilet paper problem'.

This somewhat tongue-in-cheek paper examines the following problem. Assume that a bathroom stall has two toilet paper dispensers, either one of which can be accessed at any time. It is likely that at any time, one roll will have more toilet paper than the other. Assume that the probability that a person will choose from the smaller roll is q, and let $p = 1 - q$ be the probability that a person will choose from the larger roll. Assume also that each person uses the same amount of toilet paper. Let n be the length (in number of sheets) of a roll of toilet paper, and let $M_n(q)$ be the average number of sheets left on the larger roll when the smaller roll empties. For fixed q, the asymptotic behavior of $M_n(q)$ is analyzed. From such a whimsical problem, much mathematics arises! The Catalan number make an appearance since they are related to the number of ways to get from one toilet-paper state to another. Let $C(z)$ be the generating function for the Catalan numbers. That is,

$$C(z)\Sigma_{n1}^{\infty}\binom{2n-2}{n-1}\frac{1}{n}z^n.$$

Let $M(z)$ be the generating function for $M_n(q)$. It is shown that

$$M(z)\frac{z}{(1-z)^2}\left(\frac{q-C(pqz)}{q}\right).$$

Asymptotic values for $M_n(p)$ are then derived. If $p \neq q$ then if r is any number greater than $4pq$ we have:

$$M_n(p) = \begin{cases} p/(p-q) + O(r^n) & \text{if } q < p \\ ((q-p)/q)n + p/(q-p) + O(r^n) & \text{if } q > p. \end{cases}$$

If $p = q$, then

$$M_n(p) = 2\sqrt{\frac{n}{\pi}} - \frac{1}{4}\sqrt{\frac{1}{\pi n}} + O(n^{-3/2}).$$

I close this review by proposing a problem based on the toilet paper problem. Instead of assuming that everyone uses the same amount of toilet paper, assume that different people use different amounts of toilet paper in a range $[a, b]$ subject to a continuous probability distribution f (it might be objected that we are exchanging one unreasonable assumption for another

since no one uses $\frac{1}{2}$ a sheet or π sheets of paper) and that this probability is independent of whether a person chooses from the little or big roll. Proceed to investigate $M_n(q)$. A solution will certainly fill an important gap.

1. [HL] G. H. Hardy and J. E. Littlewood, Some problems of Diophantine approximation, **Acta Mathematica 37**(1914), 155 - 238.

2. [Kn2] Donald E. Knuth, **Seminumerical Algorithms**, Volume 2 of **The Art of Computer Programming** (Reading, Massachusetts Addison-Wesley, 1969).

Review of[3]

Selected Papers in Discrete Mathematics
by D. Knuth
Published by CSLI (Center for the Study of Language and Information Publication
Paperback, 286 pages, $72.00
$42.00 on Amazon used

Review by Carlos A.S. Oliveira
School of Ind. Engineering and Management, Oklahoma State Univ.
coliv@okstate.edu

1 Introduction

The book "Selected Papers in Discrete Mathematics" is a compilation of work published (plus some unpublished papers) by Donald E. Knuth, in the area of discrete mathematics. Clearly, the author does not need any introduction, and is well known by his authoritative work in areas such analysis of algorithms and digital typography. However, more than this, Knuth is a great example of good expositor and writer. Thus, even if your interests are not directly related to the more mathematical areas of computer science, it is greatly rewarding to read this book as a way of learning how to write good papers.

The selection and ordering of the papers in this volume seems to have been performed more by subject than chronological order. Although there is not a clear indication of such a division (which can be seen as a minor organizational issue), one can identify from the table of contents the major topics that have been addressed in the papers forming the book. Initially, for example, there are a couple of papers related to the issue of notation, then exposition papers on history of discrete mathematics. Other than the table of contents, a good way to find topics covered in the book is checking the comprehensive index.

2 Topics Covered

The book is extensive, with 812 pages, index included, of mostly mathematical topics. Thus, I will not make a review of every paper, since this would be boring to readers and probably not worthwhile the effort (one can check the preface for a brief description of the contents of each chapter). I will

instead describe the main areas that are covered by the book and provide examples based on the papers on each area. If you have interest, keep in mind that there is much more that was not discussed here.

The first chapter of the book describes how computers can be useful as a tool for helping the development of mathematics. The paper, written in 1965, serves roughly as a statement of the type of mathematics the author is interested in. The use of computers in discrete mathematics has only increased since the time the paper was written, thus it is interesting to compare the results stated there with what we have today.

The first major topic discussed in the book is related to the use of notation in discrete mathematics, and how they can influence the discovery and development of useful results. One of the interests of Knuth has been the establishment of good mathematical notation, especially to be used in his series "The Art of Computer Programming." Some of the notations discussed in these papers are:

- the so called *Iverson's convention*, where square brackets around an expression are used to return a value 1 if and only if the expression is true. This can be useful to simplify summations and other types of formulas used in discrete mathematics;

- a different convention for writing Stirling numbers, where a Stirling number of the first type is written as $\begin{bmatrix} a \\ b \end{bmatrix}$, and of the second type as $\begin{Bmatrix} a \\ b \end{Bmatrix}$. This leads to much simplification in many of the formulas including Stirling numbers, and new insights as well. For example, the simple result
$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} -b \\ -a \end{bmatrix}$$
can be better appreciated only due to this improved notation, as explained in the paper;

- the notation $[F(z)]G(z)$ for the vector product of two polynomial functions is shown to lead to easier manipulation of this type of operations in Chapter 3.

The next topic discussed in the book is history of discrete mathematics. There it is included an interesting paper about the work of Johann Faulhaber on summations of powers. As the paper describes, it is not exactly

known what method he used (at the time mathematicians where more interested in results than in proofs), but he accomplished amazing computational achievements for the time. A puzzle proposed by Faulhaber is solved using computers, showing that he did correct computations for very large numbers considering the time (he died in 1635). Another paper (Chapter 5) discusses the work of Thomas Harriot, who did impressive discoveries in the area of discrete mathematics in the early 17th century. Another paper of historical interest is Chapter 18, since it was the first paper published by Knuth in discrete mathematics. It presents amusing work on number representation, despite some inaccuracies that are pointed out in the postscript. (This shows that, by the way, even expert researchers can make mistakes early in their careers.)

Matrices are important objects in discrete mathematics. They are constantly used as a representation means as well as main objects of study. Many chapters in the book are related to matrices. For example, Chapter 6 describes a very interesting result by Egorychev (1980), proving a conjecture of van der Waerden (made in 1926) about the permanent of doubly stochastic matrices. The concept of *Pfaffian* of a matrix, a generalization of determinant, is discussed in the next chapter, together with a history of the discovery and use of Pfaffians. *Combinatorial matrices* are the subject of Chapter 9. A combinatorial matrix is a type of matrix expressed in general as the combination $bJ+(a-b)I$, where a and b are scalars, J is the matrix of ones, and I is the identity matrix. The chapter introduces some properties of this type of matrix.

Graph theory is another fundamental topic in discrete mathematics, thoroughly explored in this book. Topics such as spanning trees and graph enumeration are discussed in connection with diverse problems. A first paper (Chapter 8) presents the well know *theta function*, introduced by Lovász as a graph parameter whose value is always between the clique number and the chromatic number of a graph. The paper is a nicely written introduction and survey of the concept of theta function and its related characterizations. Chapter 10 describes properties of the so called *Aztec diamond* graphs, such as the number of spanning trees in such graphs. The spectra of matrices representing the tensor product of graphs is discussed in Chapter 11. The number of subtrees of a digraph is the topic of Chapter 12. A generalization of a theorem of Cayley (1889), giving the number of directed trees in a graph, is described in the next chapter (13). The generalization involves graphs with an associated coloring of nodes, such that some rules concerning the orientation and the colors adjacent to arcs must be enforced. The proof amounts to defining a correspondence between graphs and integer se-

quences and using this correspondence to calculate the number of directed trees. Later in the book (Chapter 24) interesting work is presented on the representation of strongly connected directed graphs.

The next major topic covered in the book is manipulation of polynomials. A computationally oriented paper appears in Chapter 15, showing how to use properties of convolution polynomials using the *Mathematica* package. "Polynomials involving the floor function" is a paper devoted to identities and general properties of polynomials, where variables are modified using the floor operator. Numerous applications in computer science use the floor function, and therefore it is important to understand how this operation can be manipulated when used on polynomials.

Discrete mathematics also plays an integral role (no pun intended) in modern algebra, for example in the study of finite fields and groups. Work in this area is presented in some of the chapters. Topics such as finite fields (Chapters 19 and 20), projective planes over semifields (Chapters 20 and 21), and grupoids (Chapter 22) are treated in full depth.

Recurrence relations in general appear on Chapters 37 through 39. In the first paper, linear recurrences with constant coefficients are discussed. In the second paper, a recurrence of the type

$$M(n) = \begin{cases} g(0) & \text{if } n = 0 \\ g(n) + \min_{0 \le k < n}(\alpha M(k) + \beta M(n-k)) & \text{for } n > 0 \end{cases}$$

is shown to be convex in some cases, being efficiently computable. In the third paper, a recurrence related to trees in random graphs is studied.

The last two papers are a real "tour de force", discussing ideas from the important area of random graphs. This area is concerned with the properties of graphs whose edges are selected randomly, according to some distribution, and was championed by Erdös and Rényi. Some of the basic results in random graphs state that, after a specific threshold is achieved by the probability distribution function, most graphs will have some specific property such as connectivity, for example. The second last paper in the book describes the properties of initial cycles formed in a evolving random graph (i.e., a graph in which edges are being added according to a probability distribution). For example, one such property of interest is the distribution of cycle lengths. The last paper, which is the largest in the book (150 pages, filling up a whole issue of the *Random Structures and Algorithms* journal) describes results related to one of the most fascinating phenomena in random graphs: the appearance of a giant component during the random process of adding edges.

Other topics appear in different parts of the book. For example, matroids (Chapters 26 and 27), permutations and partitions (Chapters 28 and 30 to 35), coding theory (Chapter 29), and number theory (Chapter 36).

3 Conclusion

As seen above, "Selected Papers in Discrete Mathematics" is a book that covers a long range of material of interest to people working in combinatorics, theoretical computer science, optimization, and other related areas. However, it must be said that although some of the papers cover basic concepts, the book is not suitable for a beginner to learn any of these topics. There are too many, sometimes very distant topics being discussed, such as random graphs and algebra.

I would say that the audience for the book comprises two groups. First, researchers working with discrete mathematics, which may want to have a good collection of papers in one of the areas covered. A second group that would benefit from the book are professionals interested in how to write good papers, and willing to learn at least a little of mathematics.

This said, the book is well deserving the price. The content is written in an authoritative way. Suffices to say that these papers have been published in some of the best journals in the area, and they were written and throughly revised by the same author of "The Art of Computer Programming."

Review[4] of
The Art of Computer Programming
Volume 4, Fascicles 2, 3, and 4
Author of Book: Donald E. Knuth
Author of Review: John Rogersh
Pearson Education (Addison-Wesley), 2005
399 pages, Softcover

1 Introduction

DePaul University, where I teach, offers a Master of Science degree in Computer Science. Most of the students entering this program have very little background in CS and we help them fill this in by offering undergraduate CS courses tailored to older (and so more motivated) students.

Teaching the course in discrete mathematics is both a pleasure and a challenge. A pleasure because it is a real joy to present some of the gems of our field to people who have never encountered the mathematics we rely on. Pulling out Euclid's GCD algorithm and Euler's cycle finding algorithm and discussing the wonderful elegance of each reminds me of why I entered this field.

The challenge comes from connecting the material to practice. Many of these students come from the banking, investing, consulting, or trading firms located steps away from our downtown campus and many are working programmers. For most, the MS is a professional degree so connecting Euclid's algorithm to public key cryptography helps them understand why grasping the fundamentals of number theory or combinatorics or graph theory (and the more difficult algorithms and proofs to come in later courses) is relevant to practice.

One encounters that same pleasure in Don Knuth's latest additions to his collection of volumes titled "The Art of Computer Programming." At the same time, he also meets the challenge of practical relevance.

2 Summary

Before discussing that pleasure and challenge, let me review the scope of this multi-decade project, as described on Knuth's web page dedicated to it (`www-cs-faculty.stanford.edu/~knuth/taocp.html`). So far, he has

[4]

completed the first three volumes and parts of the fourth volume of what is planned to be a seven-volume set. As many know, volume 1 deals with fundamental algorithms, volume 2 with seminumerical algorithms, and volume 3 with sorting and searching.

Volume 4 is concerned with combinatorial algorithms and will eventually appear in three (or maybe four) books. Continuing the chapter numbering of the first three volumes, this one will contain chapters 7 and 8. Volume 4A will take up sections 7.1 ("Zeroes and Ones"), 7.2 ("Generating All Possibilities"), and perhaps also 7.3 ("Shortest Paths"). The work reviewed here comprises subsection 7.2.1 ("Generating Basic Combinatorial Patterns").

Knowing that, even in retirement, it will take time to complete this opus, Knuth has decided to issue portions in *fascicles*, an old word referring to published installments of a longer work. In this case, each fascicle is a soft-covered book containing, respectively, 128, $128 + 23$, and $128 - 8$ pages. To date, there are four fascicles, each with its own table of contents and index. Fascicles 0 and 1 are scheduled to appear in 2008. Fascicle 1 of Volume 1 presents Knuth's MMIX machine and is not reviewed here.

Fascicles 2, 3, and 4 cover subsection 7.2.1, which in turn contains seven sub-subsections:

- 7.2.1.1: Generating all n-tuples
- 7.2.1.2: Generating all permutations
- 7.2.1.3: Generating all combinations
- 7.2.1.4: Generating all partitions
- 7.2.1.5: Generating all set partitions
- 7.2.1.6: Generating all trees
- 7.2.1.7: History and further references

The first two appear in fascicle 2, the next three in fascicle 3, and the last two in fascicle 4.

It may seem that this form puts a reader *in media res* (which it does!) but each fascicle has its own table of contents and index and is written so that it can stand pretty much on its own. In the text there are references to other portions of the set but most are to chapters already published and, in those cases where a reference is to something not yet available, it does not greatly impair understanding.

Knuth issued these fascicles to explain how to *generate* basic combinatorial patterns. He does not call this enumeration as this implies creating a list of objects. One usually wishes to visit the objects one by one so as to determine whether each possesses some property. Of course, being able to generate the objects one-by-one in some kind of order means being able to enumerate them. But his emphasis on generation often requires a more subtle algorithmic approach.

He moves from the simplest objects, showing how to generate all n-bit strings, to the complex, showing how to generate all n-node trees. He follows this with a 25-page history of the subject.

Most reading this review need no explanation of the objects being generated so, rather than discussing each unit, I will take as a typical example section 7.2.1.2, generating all permutations on n items.

Knuth begins by reviewing what a permutation is and then launches immediately into algorithm L (for lexicographic), which generates in lexicographic order the permutations of an ordered multiset. He traces the algorithm back to 14th century India and 18th century Germany and briefly analyzes the work L does to pass from one permutation to its successor.

Referring back to Gray code sequences in the previous (and incredibly thorough) section, where he presents algorithms that generate each n-bit string from the previous by changing only one bit, he lays out algorithm P, which generates each permutation from the previous by performing one swap of adjacent elements.

Why is this called algorithm P? That's answered in its analysis, when Knuth reveals that this is the procedure used to carry out the "plain changes" method of ringing a set of church bells. Need a reference? There are two, pointing to 17th century English texts, followed by a brief digression into other change ringing sequences.

After a short detour into using permutation generation to solve alphametic puzzles ("`SEND + MORE = MONEY`" is the best known), Knuth launches into a general framework that makes use of the group theoretic properties of sets of permutations.

We next see algorithm G (a general permutation generator that uses a Sims table), algorithm X (lexicographic generation with restricted prefixes), algorithm H (dual permutation generator), algorithm C (generation by cyclic shifts), and algorithm E (generation using Ehrlich swaps). Each time, potentially unfamiliar terms (e.g., Sims table, Ehrlich swap) are explained and references provided.

Topological sorts come next. In this case, restrictions are placed on which order pairs of elements may appear in a permutation. Algorithm V

deals with that.

Knuth ends with a caution about generating permutations, noting that one should consider whether that's the best way to solve the problem, even in the case of small instances of NP-complete problems.

To test and improve the reader's understanding he includes 112 exercises, each one scored according to its difficulty, along with solutions. The exercises are important as they often expand on some of the material. In fact, of the 69 pages devoted to this topic, 33 are taken up with exercises and solutions.

3 Opinion

Who would benefit from this book? One obvious audience is computer scientists. Speaking as one of those, I enjoyed these fascicles. Knuth is teaching me the way I try to teach my students in that introductory discrete math class, which is to convince the audience that the notion of generating combinatorial objects is intrinsically interesting AND it derives from and leads to other areas of mathematics AND it connects to pursuits outside computer science AND people from widely different times and places have wrestled with and sometimes solved associated problems AND it can be applied to solve real-world problems today. This is science writing for computer scientists.

Who else? One could use this in a course devoted to combinatorial generation. But it would also find a place in a more general algorithms course as a supplemental text. Although not written as a textbook, the presentation and the exercises give it a solid pedagogical feel. And the price is right: Each fascicle costs around US$20 so students shouldn't suffer sticker shock.

Working programmers tackling these problems will certainly welcome the diversity of approaches to solving them. I program very little but, when I need to, it's usually to test hypotheses about combinatorial objects by looking at small examples. I have found the material here very useful in generating all partially ordered sets on 10 or fewer elements in order to get an idea of the properties of poset games. I will note that Knuth relies heavily on branching in his algorithms. This allows them to be expressed succinctly and with a focus on the steps making changes to an object but it does mean that the algorithms do not leap off the page into a language like Java without rearranging some code.

The real value in this work comes from its scholarly attention to detail. Knuth recognizes and acknowledges the many and varied people and writ-

ings that provided him with algorithms, methods, variations, digressions, inspiration. Sometimes in computer science we forget that although our discipline is hardly 60 years old we owe a debt to intellectual forebears going back many centuries. Knuth honors that debt and the text is all the richer for it.

I will end the review with a mention of the last section, the one on the history of combinatorial generation. This is good stuff. Starting with the *I Ching*, it sweeps along to the rhythmic structure of ancient Indian poetry and the metrical structure of classical Greek poetry and from there to a variety of places, objects, and people. On this sojourn he shows how authors in the past have attempted combinatorial generation in a multitude of settings and ultimately makes the point that it's not easy to get right.

But Knuth gets it right. The algorithms work, the text flows easily, the references and explanations entice, and the exercises challenge and teach. In 1999, the journal *American Scientist* named the original three volumes of *The Art of Computer Programming* one of the century's best monographs in physical science. These fascicles continue to uphold that selection.

The Art of Computer Programming:Volume 4A
Author of Book: Donald E. Knuth
Author of Review: John Rogers
Pearson Education (Addison-Wesley), 2011
899 pages, Hardcover

1 Introduction

Early in my days as a graduate student, I had to jump through a set of hoops known as "qualifying exams." These were written tests covering various areas of Computer Science and administered on a Saturday in one of the department's classrooms. On the warm day I took them the classroom happened to face a small quadrangle also bordered by a building with a set of bells. At 9am two dozen keyed-up graduate students began taking the exam. At 9:05 a series of peals rang out across the quad. We all looked up, out the windows, and then at our proctor. Knowing that he was powerless to stop this clangor he looked at us for a moment and then returned to reading his book.

Now it's certainly annoying enough to have bells ringing while one is trying to recall the definition of "Bélády's anomaly" but that wasn't the worst part. As the bells rang I realized that each sequence was a subtle variation on the one before. Wasn't the one just rung identical to the previous one but with a single swap? In fact, hasn't the same bell figured in the last three swaps? Then the real danger struck me. I needed all of my intellectual energy and focus to pass this exam. With difficulty I ignored the bells and their elusive but definite patterns. Amazingly I did so and forged ahead, passing the exam (which put me in place to jump through the next set of hoops, the "oral exams", as in "oral surgery" but a lot less fun).

I soon discovered that we had been the unwilling audience of a performance known as a *change ringing*. But it wasn't until I read Knuth's latest volume in his series on "The Art of Computer Programming" (TAOCP) that I learned how much more that ringing had to do with the Computer Science I enjoy than Bélády and his anomaly.

Change ringing sequences are an example of combinatorial generation: an algorithmic way of listing, one by one and without repetition, each object from a set of combinatorially defined objects. This raises many questions, among them:

- In what order are the objects listed?

- Are there some requirements we want to impose in the transition from one object to the next?
- Is the listing meant to be exhaustive?
- If not exhaustive, is there an ordering that allows the algorithm to skip efficiently over the objects we don't want listed?

A very simple example is listing all bit strings of length n. There is the usual way of listing them lexicographically, starting with all zeroes and ending with all ones, in which case there are transitions from one value to the next requiring the flipping of many bits. If we apply the requirement of flipping exactly one bit for each transition we come to the well-known *Gray code ordering*. For the change ringing mentioned above, the ringers need to generate all permutations on the values 1 through n. Using the *plain changes* sequence requires that each transition swap exactly one pair of adjacent values.

2 Summary

"Combinatorics is the study of the ways in which discrete objects can be arranged into various kinds of patterns." So begins the introduction of this volume's first chapter. Knuth continues with more precise versions of the questions I asked above:

1. Are there any arrangements that conform to a given pattern?
2. Can such an arrangement be found quickly?
3. How many such arrangements are there?
4. Can all such arrangements be visited systematically?
5. Which arrangements optimize the value of some objective function?

This volume proceeds to answer these questions about several kinds of combinatorial objects, including bit strings, boolean expressions, mixed radix numerals, combinations, permutations, and trees.

Continuing the chapter numbering from the first three volumes, this one contains only Chapter 7 (Combinatorial Searching) which, in turn, contains two sections: 7.1 "Zeros and Ones" and 7.2 "Generating All Possibilities". Section 7.1 is now complete but 7.2 currently only contains the sub-section

"Generating Basic Combinatorial Patterns"; two sub-sections, one on basic backtracking and one on efficient backtracking, are yet to appear, as are sections 7.3 to 7.10 ([**?**]).

Most of sub-section 7.2.1 appeared in the fascicles numbers 2, 3, and 4 ([**?**]) issued a few years ago and reviewed here previously ([**?**]). It contains:

- 7.2.1.1: Generating all n-tuples
- 7.2.1.2: Generating all permutations
- 7.2.1.3: Generating all combinations
- 7.2.1.4: Generating all partitions
- 7.2.1.5: Generating all set partitions
- 7.2.1.6: Generating all trees
- 7.2.1.7: History and further references

As the titles of the sub-sub-sections indicate Knuth's emphasis is on generating sequences instead of enumerating them. The difference is that, to him, enumeration implies creating a list of all of the objects of a certain size. Instead, his algorithms focus on the transitions: Given a combinatorial object, here is a way to generate the next one. Some of the algorithms take into account an objective function that indicates which objects not to generate.

These algorithms are expressed in pseudo-code and sometimes implemented in the assembly language for Knuth's MMIX machine ([**?**]).

In each sub-section Knuth provides a thorough treatment of that kind of combinatorial generation. He includes not only the technical and mathematical aspects but also sometimes surprising cultural and historical references. From the history of our own discipline we learn that using a Gray code provided a way to address disk sectors so that at sector boundaries the next sector address is only one bit different than the previous address. Generating all permutations arose in determining the change ringing sequences mentioned above, something a number of writers from the past struggled with. Listing classical Hindi and Greek poetic meters and enumerating the hexagrams of the *I Ching* relied on generating bit sequences. Many puzzles require the solver to determine the right way to traverse a sequence of combinatorial representations of the state of the puzzle.

True to the spirit of puzzle and problem solving, every sub-section ends with many exercises, ranging in difficulty from questions requiring a few

seconds reflection to open problems requiring extensive research. And true to the spirit that these should support and extend the material in the main body of the text there are 300 pages of solutions. Knuth cares passionately about well-crafted exercises and their value in extending and reinforcing the main presentation.

3 Opinion

For readers of this review, there are three ways to view this book. With its extensive development of each topic and the extraordinary number of exercises and their solutions it could certainly serve as a textbook, both in computer science and mathematics. Its list price is $74.99 but less expensive copies can be found.

A second way is to use it as a reference for developing algorithms and code. With the computing power now available it is not unreasonable to attack combinatorial problems by generating and understanding "small" instances, in the hope that this leads to a deeper understanding of the unbounded problem.

The third way is the one I favor most and that I articulated in my earlier review of the fascicles that contained some of this work. This is science writing for computer scientists. There is enough mathematical treatment to stimulate the theoretical parts of our brains and there is enough practical detail to keep our coding neurons firing. But it is the history and references that lift it to being a classic text, one that can send the reader off on a merry chase through time and technique.

The many algorithms presented in the book are (for the most part) written in a way to make it easier to see why the algorithm works. Quite often this means that they are not written to be easy to code. But that really isn't much of an impediment. With a little practice one will find it straightforward to translate his pseudo-code into a working program. In fact, I've found that they work well as coding exercises for CS students learning data structures.

I also enjoy the way puzzles have been worked into the narrative. Knuth has a number of quotes pointing out the importance of puzzle-solving in understanding mathematical principles. In a number of cases combinatorial problems were derived from analyzing a well-known puzzle, such as Ozanam's card arrangement puzzle mentioned on page 3 or the tiring irons or Chinese ring puzzle on page 285.

Knuth rewards a careful reader's attention in other ways as well. Sprin-

kled throughout the text are pertinent quotes that mostly amuse. For example the last printed page of the section containing exercise answers has this gem: *"I may," said Poirot in a completely unconvinced tone, "be wrong"* ([**?**]). This is followed on the next page by two quotes, one from the Marx brothers, and how often does that happen in an algorithms text?

This short review cannot even touch on the vast majority of the topics presented in this book. Section 7.1, with the simple title "Zeros and Ones", could be a book on its own, reminding us of the richness of boolean logic and algebra. Section 7.2 has already appeared as a collection of three short books. What's astonishing is that this is the work of a single author...and he's not done yet. As with Agatha Christie's skeptical detective, Knuth might be wrong here and there but his careful scholarship, attention to detail, and obvious love of the subject matter make for a very engaging read.

With respect to this volume's relationship to the first three volumes, it stands for the most part independently. One might argue that volumes 1 through 3 are more fundamental to our discipline but for some, myself included, while I might employ those earlier algorithms more often the ones in this book are the kind I enjoy more. I think the author agrees. In the preface Knuth mentions his inclination to subtitle the volume *The Kind of Programming I Like Best*. His enthusiasm for this material comes through very clearly. It makes one hope that the waiting time to volume 4B is far less than the gap between volumes 3 and 4A.

References

[Chr53] Agatha Christie. *After the Funeral*. 1953.

[Knu05] Donald Knuth. *The Art of Computer Programming, Volume 1, Fascicle 1: MMIX – A RISC Computer for the New Millennium*. Addison-Wesley, 2005.

[Knu07] Donald Knuth. *The Art of Computer Programming (TAOCP), volume 4, fascicles 2, 3, 4*. Addison-Wesley, 2007.

[Knu11] Donald Knuth. The art of computer programming (taocp). `http://http://www-cs-faculty.stanford.edu/\~uno/taocp.html`, 2011.

[Rog08] John Rogers. Review of the Art of Computer Programming, Volume 4, fascicles 2, 3, and 4. *SIGACT News*, September 2008.

Selected Papers on Discrete Mathematics
Author of Book: Donald E. Knuth
Author of Review : Daniel Apon
Center for the Study of Language and Information (CSLI), 2003
812 pages $80.00, University of Chicago Press

1 Introduction

Selected Papers on Discrete Mathematics is a 2003 collection of *forty-one* of Knuth's papers on discrete mathematics, bringing together "almost everything [he] has written about mathematical topics during the past four decades." This is the sixth entry in a nine-volume series archiving Knuth's published papers. The full series in order is: (i) *Literate Programming*, (ii) *Selected Papers on Computer Science*, (iii) *Digital Typography*, (iv) *Selected Papers on Analysis of Algorithms*, (v) *Selected Papers on Computer Languages*, (vi) the current book, (vii) *Selected Papers on Design of Algorithms*, (viii) *Selected Papers on Fun and Games*, and (ix) *Companion to the Papers of Donald Knuth*.

While not designed as a textbook, this book is *huge* and covers a number of diverse topics in great detail. Measuring nearly 800 pages before the index, the book contains something on almost every fundamental area of discrete mathematics: mathematical notation, permutations, partitions, identities, recurrences, combinatorial designs, matrix theory, number theory, graph theory, probability theory, and a dash of algebra. As if to emphasize this point, the final two papers in the book (both dedicated to Paul Erdős) comprise an intensive, nearly 200-page study of the properties of "randomly-generated (evolving) graphs" first initiated in a classic 1959 paper of Erdős and Rényi.

In the sequel, I give a chapter-by-chapter summary of the book. For lack of space, I can only briefly skim over a few chapters, but I have tried to go into more depth on some of the parts I found most interesting and rewarding. Afterward, I conclude with my opinion of the book.

2 Summary

Chapter 1 begins the book, perhaps symbolically, with a 1965 paper of Knuth and Hall's observing that computers may (in fact!) be useful for solving combinatorial problems by searching through the possible witnesses.

The chapter introduces the basic backtracking technique and discusses applications to constructing latin squares and projective planes.

Chapters 2 and 3 discuss aspects of notation that Knuth would like to promote. One of these is *Iverson's convention*, which lets us write any sum as an infinite sum without limits: if $P(k)$ is any Boolean property of the integer k, then we could write $\sum_{P(k)} f(x)$ as $\sum_k f(x)[P(x)]$. For example, the sum-of-squares for all integers k in some set S could be written as $\sum_k k^2[k \in S]$.

Chapters 4 and 5 discuss mathematical results from the early 17th century, due to Johann Faulhaber and Thomas Harriot. (There's even a picture of an engraving of Johann in 1630!) The first of the two proves a result of Johann's on sums of powers (for which Johann did not publish a proof). Namely, the r-times-repeated summation of $1^m, 2^m, ..., n^m$ is a polynomial in $n(n+r)$ times the r-fold sum of $1, 2, ..., n$, when m is a positive odd number. Of particular interest – Knuth's goal was to prove this fact only using techniques available in the early 17th century (and thus, to Johann).

A particular favorite of mine was Chapter 6, which gives a self-contained exposition of G.P. Egorychev's 1980 proof of a conjecture first made by van der Waerden in 1926. VDW's conjecture (now a theorem) states that the permanent of an $n \times n$ doubly stochastic matrix is never less than $n!/n^n$, which implies the matrix where all entries are equal to $1/n$ gives the minimum permanent for this class of matrices. The proof is given from only elementary principles, with the exception of a single, simple fact from analysis in the proof of one lemma, for which Knuth spends a little extra time giving intuition.

Having examined the permanent, Knuth turns to exploring the Pfaffian of an array of numbers in Chapter 7. Let $A = (a_{i,j})$ be a $2n \times 2n$ skew-symmetric matrix. The *Pfaffian of A* is defined as

$$\mathrm{pf}(A) = \frac{1}{2^n n!} \sum_{\sigma \in S_{2n}} \mathrm{sign}(\sigma) \prod_{i=1}^{n} a_{\sigma(2i-1),\sigma(2i)}$$

where S_{2n} is the symmetric group; that is, the set of all permutations on $2n$ elements (whose group operation is permutation composition). For the curious reader: Knuth argues that the Pfaffian is "more fundamental" than the determinant. To quote him, "a determinant is just the bipartite special case of a Pfaffian." In any case, Chapter 7 gives an identity for the product of $\mathrm{pf}(A) \cdot \mathrm{pf}(B)$ where B is a submatrix of A.

Chapter 8 discusses the Sandwich Theorem. It is NP-complete to compute $\omega(G)$, the size of the largest clique in a graph G, and it is NP-complete to compute $\chi(G)$, the minimum number of colors needed to color the ver-

tices of G. But *in polynomial time*, it is nonetheless possible to compute a real number — the Lovász number $\vartheta(G)$— that is "sandwiched" in between these two:

$$\omega(G) \leq \vartheta(G) \leq \chi(G).$$

The chapter is nearly 50 pages long, and written in an expository format, building up numerous aspects of the theory surrounding the Sandwich Theorem and Lovász's function.

Chapters 9 through 14 explore various relationships between matrices, graphs, and trees. In Chapter 13, we consider the following problem: Given a set of vertices v that each been assigned a color $c_v \in [m]$ with n_j vertices of color $j \in [m]$, how many oriented trees on these vertices with designated root exist, subject to constraints of the form "no edge connects vertices colored i and j?" While the general statement requires introducing prohibitively much notation for this review, here is an exact statement for a simple case: Suppose we have $|A|$ vertices of color a, which have directed edges only onto vertices of color b and c, and $|B|$ vertices of color b, which have directed edges only onto vertices of color a and c, and $|C|$ vertices of color c, which have directed edges only onto vertices of color c and the root. How many oriented trees of this type exist? Exactly

$$(|B|+|C|)^{|A|-1}(|A|+|C|)^{|B|-1}(|C|+|1|)^{|C|-1}(|A||C|+|B||C|+|C|^2).$$

Chapters 15 and 16 discuss properties of polynomials. The first considers families of convolution polynomials that arise as coefficients when a power series is raised to the power x. These are families of polynomials $F_0(x), F_1(x), ...$ where $F_n(x)$ has degree $\leq n$ and

$$F_n(x+y) = F_n(x)F_0(y) + F_{n-1}(x)F_1(y) + \cdots + F_0(x)F_n(y)$$

hods for all x and y and for all $n \geq 0$. For example, setting $F_n(x) = x^n/n!$ yields the binomial theorem for integer exponents.

Chapters 17 and 18 are fairly light. Chapter 17 constructs an equidistributed sequence of real numbers, and Chapter 18 presents a construction (from Knuth's college days) of a base-$2i$ number system and implements addition and multiplication.

Chapters 19 through 21 cover finite fields and semifields. Chapters 22 and 23 go further into topics in algebra. An interesting twist here is Knuth's construction of Huffman's algorithm (for finding minimum redundancy codes) from abstract algebra. In particular, he defines a *Huffman algebra* $(A, <, \circ)$, which is a linearly ordered set A with a binary operator $\circ$.

Huffman's algorithm, in this context, is given as input elements $a_1, ..., a_n$ of A and builds an expression (over the algebra) out of the elements by repeatedly applying $\circ$ to the smallest and second-smallest elements until only a single expression remains. The expression yields the corresponding Huffman code.

Chapters 24 and 25 discuss how graphs are constructable from various components. An example theorem is that the complement of a transitive closure of the complement of a transitive relation is, itself, transitive.

Chapters 26 and 27 explore the theory of matroids. The first of these, in particular, gives pseudocode from one of Knuth's attempts to generate random matroids for experimental analysis. This is also the only piece of mathematical writing that I've seen use the phrase "homomorphic image of a free erection" with a serious face.

If $x_1, x_2, ..., x_n$ are real numbers whose sum is zero, then we can always find some permutation $p(1)p(2)...p(n)$ such that each of the partial sums $x_{p(1)} + \cdots + x_{p(j)}$ is nonnegative, for $1 \leq j \leq n$. Daniel Kleitman conjectured that the number of such permutations is always at most $2n!/(n+2)$, if the x's are nonzero. Chapter 28 presents a proof of Kleitman's conjecture, but strengthened to be parameterized by the number of positive and negative x_i.

Chapter 29 presents an efficient construction of a balanced code that optimizes serial encoding and decoding. The scheme is extended to optimizations for parallel encoding and decoding.

Chapters 30 through 36 present numerous results on partitions. Here are a few examples:

In Chapter 30, the following problem (known as the Knowlton-Graham partition problem) is considered: Two parties want to communicate over a long cable containing n indistinguishable wires. They want to label the wires consistently so that both ends of each wire receive the same label, perhaps by locally re-wiring connections. How can they do this? (Using a special partition!)

A *plane partition* of n is a two-dimensional array of nonnegative integers $\{n_{i,j}\}$ for $i, j \geq 1$, for which $\sum_{i,j} n_{i,j} = n$ and the rows and columns are in nonincreasing order:

$$n_{i,j} \geq n_{(i+1),j} \text{ and } n_{i,j} \geq n_{i,(j+1)}.$$

Chapter 32 gives generating functions for various classes of plane partitions.

Chapter 34 obtains a generalization of Jacobi's triple product identity, by analogy to a similar generalization for Euler's partition identity, which

can written

$$\prod_{j=1}^{\infty}(1-q^{2j-1}z)(1-q^{2j-1}z^{-1})(1-q^{2j}) = \sum_{k=-\infty}^{\infty}(-1)^k q^{k^2} z^k.$$

Chapter 36 studies coefficients that arise in the preceding study of partitions. Interestingly (at least to me), there are connections to cyclotomic polynomials and determinants of semi-lattices from geometry of numbers.

Chapters 37, 38, and 39 discuss recurrence relations. For example, Chapter 38 investigates solutions of the general recurrence

$$M(0) = g(0), \qquad M(n+1) = g(n+1) + \min_{0\le k\le n}\left(\alpha M(k) + \beta M(n-k)\right),$$

for various choices of α, β, and $g(n)$. In many cases, $M(n)$ is shown to be a convex function (and thus much more efficiently computable).

Finally, there are the mammoth chapters – 40 and 41 – on *evolving graphs*. Nothing I say in a few sentences can quite do two hundred pages of work justice, but here is the general idea: Begin with n disconnected points. Now add edges between the points at random. Then...

How long is the expected length of the first cycle that appears? Asymptotically, what is the average length of the kth cycle that appears? What is the probability that the graph has a component with more than one cycle at the point that the number of edges passes $n/2$? When such a graph has approximately $n/2$ edges, what is the probability that the graph consists entirely of trees, unicyclic components, and bicyclic components as $n \to \infty$?

Knuth (and co-authors) obtain numerous high-quality estimates in a uniform manner, and obtain closed-form expressions of multiple constants of interest.

3 Opinion

As with all of Knuth's books, this book is written authoritatively and eloquently. Knuth's presentation is a testament to clean, formal writing; his standard format is to begin with a few paragraphs describing the topic, and then present a list clear lemmas and their proofs, followed by a main theorem and a proof connecting each of the lemmas together. Interspersed between each of the lemmas are a few paragraphs describing the purpose of each subsequent lemma. The notation and typography are beautiful and succinct. As mentioned above, some of the chapters even explicitly focus on *which notation is the most clear and useful*. I found it a delight to read.

Selected Papers on Design of Algorithms
Author of Book: Donald E. Knuth
Author of Review : Daniel Apon
Center for the Study of Language and Information (CSLI), 2010
453 pages$75.00, University of Chicago Press

1 Introduction

Selected Papers on Design of Algorithms is a compilation of twenty-seven of Knuth's technical papers focusing on the design of new algorithms. This is the seventh entry in a nine-volume series archiving Knuth's published papers. The full series in order is: (i) *Literate Programming*, (ii) *Selected Papers on Computer Science*, (iii) *Digital Typography*, (iv) *Selected Papers on Analysis of Algorithms*, (v) *Selected Papers on Computer Languages*, (vi) *Selected Papers on Discrete Mathematics*, (vii) the current book, (viii) *Selected Papers on Fun and Games*, and (ix) *Companion to the Papers of Donald Knuth*.

The papers are revised for errors, and cover a range of algorithmic topics – from combinatorics and optimization, to algebra and theorem proving, to managing error in numerical computations. To quote from the back cover of the book:

> "Nearly thirty of Knuth's classic papers on the subject are collected in this book, brought up to date with extensive revisions and notes on subsequent developments. Many of these algorithms have seen wide use — for example, Knuth's algorithm for optimum search trees, the Faller–Gallager–Knuth algorithm for adaptive Huffman coding, the Knuth–Morris–Pratt algorithm for pattern matching, the Dijkstra–Knuth algorithm for optimum expressions, and the Knuth–Bendix algorithm for deducing the consequences of axioms. Others are pedagogically important, helping students to learn how to design new algorithms for new tasks. One or two are significant historically, as they show how things were done in computing's early days. All are found here, together with more than 40 newly created illustrations."

The book is primarily an "archival" work. That is, the majority of the book proper is a sequence of (significant) papers, though on disparate topics. It is *not* particularly intended to be read from cover-to-cover: An effort was made to group papers into loosely related themes, but the book lacks a

"coherent narrative" when read in order. As a consequence, not everyone will like every paper, but on the other hand, most people will find something to enjoy in the 27 papers. Summaries of some of the chapters I enjoyed the most appear below.

2 Summary

The first chapter of the book is distinct; it is not about an algorithm, but rather about the life and work of Knuth's long-time colleague and co-author, Robert W Floyd (1936-2001). If this name doesn't ring a bell, recall at least the Floyd-Warshall algorithm for All-Pairs-Shortest-Path from your undergraduate Algorithms course! This opening chapter chronicles the professional and personal relationship between Knuth and Floyd, originating through correspondence by mail in 1962 (*mail* – not *email!*), and developing further after Knuth and Floyd both joined Stanford's computer science department in 1967 and 1968, respectively.

Chapter 5 is on dynamic Huffman coding. A Huffman tree implements a minimal-weight prefix code, which can be used as the core of a one-pass algorithm for file compression. For a letter a_j in a file, a_j's weight w_j is the number of occurrences of a_j. Huffman coding uses this weight to optimize the construction of the underlying tree. But what if the file is *streamed*? Each time a new character a_j of the file is seen, the corresponding weight w_j is incremented. This necessitates *efficiently updating* the structure of the underlying tree each time a new character arrives. This is accomplished by a clever swapping of sub-trees after each update so that the optimality of the tree's weights are maintained.

Chapter 9 covers the well-known Knuth-Morris-Pratt algorithm for fast pattern matching in strings. Here, we have a text file of n characters and a string of $m < n$ characters. The goal is to find the string within the text file, if it exists. Naïvely, this takes $O(mn)$ time, but the KMP algorithm does it in $O(m + n)$ using clever preprocessing.

Chapter 10 gives algorithms for addition machines – that is, machines that can read/write/copy numbers to and from registers as well as evaluate $\geq$, +, and -. The chapter begins with a simple $O(\log^2(x/y))$ time algorithm for computing $x \bmod y$. Surprisingly, this can be improved to $O(\log(x/y))$ by using the (unique!) representation of numbers as sums of Fibonacci numbers with pairwise difference ≥ 2, instead of their binary representations. The Fibonacci-based techniques are extended to efficient algorithms for multiplication, division, computing $\gcd(x, y)$, implementing

stacks, sorting, and computing $x^y \bmod z$.

Chapter 11, entitled "A simple program who's proof isn't," is about the subroutine for converting between decimal fractions and fixed-point binary in TEX. This is an interesting example of a general principle arising from careful examination of a finite problem. Internally, TEX represents numbers as integer multiples of 2^{-16}, and it must frequently convert a given decimal fraction with k digits of precision, written $.d_1 d_2 ... d_k$, into this format while rounding correctly. The straightforward procedure is to compute

$$N = 10^k \left(2^{16} \sum_{j=1}^{k} d_j / 10^j + 1/2 \right)$$

then letting the output be $\lfloor N/10^k \rfloor$. Unfortunately, since k may be arbitrarily large, the values N and 10^k may be too large for the computer's hardware to support. Digging into the details of the computation, however, reveals a procedure for the task that only requires TEX to maintain an array of 17 digits for the input and computes intermediate values no larger than $1,310,720 \ll 10^k$. A generalization to arbitrary combinations of precision is discussed at the end.

Chapter 23 is "Evading the drift in floating point addition" and begins by considering the following simple form of rounding after floating-point arithmetic: Suppose we are storing numbers to three significant digits and that rounding is performed by adding 5 to the fourth significant digit, then truncating to three significant digits. This is rounding to the nearest number, and breaking ties by rounding up. If $x = 400$ and $y = 3.5$, then $x + y = 404$, $(x + y) - y = 401$, $((x + y) - y) + y = 405$, and so on. This *drift* towards $+\infty$ is due to the bias of always rounding up in the case of ties. Ideally we would like to always have $((\cdots((x+y)-y)\cdots+y)-y) = x$, regardless of how many times we add-then-subtract the same number. The solution is to analyze a list of 30+ different cases covering the possible, arithmetic relationships between the numbers being added or subtracted. Sometimes we round ties up, sometimes we round ties down. On the plus side, the resulting nested depth of **if-else**'s in code is only 5 or 6, so the more complicated rounding scheme can be implemented to run efficiently and avoid the drift.

3 Opinion

Selected Papers on Design of Algorithms bears Knuth's usual eloquence in writing. The algorithms and proofs in each chapter are presented cleanly,

and pseudocode for implementing them accompanies most of the algorithms. Part of the real *charm* of this collection comes from the historical notes interspersed throughout the book. These take the form of either additional commentary attached to the end of a paper explaining how the technical topic has progressed since the writing of the paper (explaining which open problems have been solved, which haven't, and how the newer results were obtained), or a story of how the original result was obtained.

A particularly nice example from the book is the sequence of events leading up to the Knuth-Morris-Pratt algorithms for fast pattern matching in strings. One of the authors, J. H. Morris, originally invented the method while implementing a text editor for the CDC 6400 computer during the summer of 1969. His code, however, was rather complicated, and after several months other implementers of the system had "fixed" his routine into a complete mess. Independently a year later, Knuth and Pratt developed a string matching algorithm, working from a theorem of Cook that showed any language recognizable by a two-way deterministic pushdown automata in *any* amount of time, can be recognized on a RAM in $O(n)$ time. Later, Pratt described the algorithm to Morris, who recognized it as the same as his own – and the Knuth-Morris-Pratt algorithm was born.

Gems like these are probably the best reason to own the book. On the other hand, I would *not* recommend this book as a tool for research or as the primary text for a classroom. It could be used as supplemental material for an algorithms course – but in these cases, it is simpler (and cheaper) to find the specific paper you want students to read, and distribute that by itself. Alternatively, it could find some use for self-study – particularly in illuminating not just technical material (some of which is dated at this point), but *how* the human beings who developed the technical material came up with it in the first place. And of course, this book is another must-have for any Knuth-ophile who loves their copy of *The Art of Computer Programming*.

Review [5] of
Selected Papers on Fun & Games
by Donald E. Knuth
Center for the Study of Language and Information
742 pages, SOFTCOVER, $37.00, 2011
Review by William Gasarch gasarch@cs.umd.edu

1 Introduction

Recently Donald Knuth had his publisher send me five books of his Selected papers to review for my column. Knuth said that this book, *Selected Papers on Fun & Games*, was his favorite. After reading every word of its 700+ pages I can see why. The book format and the topic allows him to put in whatever he finds interesting. By contrast his book *Selected Papers on Discrete Math* has to have all papers on ... Discrete Math.

In the preface he says *I've never been able to see the boundary between scientific research and game playing.* This book also blurs the distinction between recreational and serious research, between math and computer science, and between light and heavy reading. By the latter I mean that many of the chapters are easy to read— if you skip details.

Many of the topics in the book come from things either he noticed or people told him leading to a math or CS problem of interest. Hence he is in many of the chapters as a character. I will do the same: here is a story that happened to ME but it's the KIND OF THING that happens to him quite often.

I was having dinner with my darling and two other couples. We were sitting at a rectangular table and I noticed that I sat across from my darling while the other two couples sat next to each other. I then thought: If there are n couples sitting down to eat at a rectangular table how many ways can they do it if everyone sits next to or across from their darling. I solved it and I invite my readers to do the same.

There are $7^2 = 49$ papers in this book. Its hard to find a theme that ties them together except *Stuff that Donald Knuth thinks is fun and hopefully you will too!*. That makes the book hard to summarize. Hence I will just comment on some of the chapters. I try to pick a representative sample.

One of the best features: many of the chapters have addenda that were added just recently. This is excellent since an old article needs some commentary about *what happened next?*.

[5] ©2014, William Gasarch

2 Summary

Chapter 1 is a reprint of Donald Knuth's first publication. Was it in American Math Monthly? Mathematics Magazine? Fibonacci Quarterly? No. It was in MAD magazine! Its a satirical (or is it?) way of doing weights and measures. Chapter 2,3,4, and 5 do not contain any real math or computer science. Chapter 6, *The Complexity of Writing Songs* is borderline since it uses O-notation.

Chapter 7, *TPK in INTERCAL* is about a rather odd challenge— writing programs in the language INTERCAL. This language has very few operators and what it does have is not the usual arithmetic operation. In this chapter he discusses writing a particular program in it. Challenging!

Chapter 8, *Math Ace: The Plot Thickens* considers the following. We all know how to graph such equations as $x = y^2$. But what about $|x + y| \leq 1$? This would not be a curve but a region in the plane. Which regions of the plane can you achieve this way? He has some rather complicated equations that yield very interesting shapes.

Chapter 12, *The Gamov-Stern Elevator Problem* takes a real world problem and solves it. Essentially, Gamov and Stern both thought that when they were waiting for an elevator to go down they would always have the next elevator going up, and vice versa. Let

$$p = \frac{\text{distance from our floor to the bottom floor}}{\text{distance from top floor to bottom floor}}$$

Assuming that the elevators start at a random position and cycle up and down continuously what is the probability (in terms of p) that the next elevator that gets to your floor will be going down? Knuth first solves this using hard math (Integral signs! Lots of them!). He notes that the answer is simple and writes ... *our formula is so simple it suggests that there must be a much simpler way to derive it.* He then does indeed present a much more elegant solution.

Chapters 13,14, and 15 are on Fibonacci Numbers; however, I suspect they contain much material *you* do not know (they contained much material *I* did not know). Here is a tidbit: (1) Every number can be written as a sum of Fibonacci numbers, (2) If you do not allow the use of two consecutive Fibonacci numbers then this representation is unique.

Chapter 17 is titled *Mathematical Vanity Plates* but it has much information on vanity plates in general (it's 40 pages long!). His interest in vanity plates was sparked by seeing the license plate *H65 537* and wondering if the driver knew that his plate was the fourth Fermat Prime. This chapter is

very indicative of Knuth's style: something in the real world inspires him and then he learns *everything* about it.

Chapter 23, *Basketball's Electronic Coach* is about a program he wrote as an undergraduate that helped rate how good the basketball players were and hence gave some idea of who to play when. This is impressive since (1) it was 1956-1960, way before Bill James and the Moneyball revolution, and (2) basketball would seem to be much harder than baseball to analyze this way. The article quotes various news sources about what he did, including Walter Cronkite. Knuth himself is skeptical that it really worked, though the team did improve quite a bit. We also learn that Knuth himself is 6 feet 5 inches, which makes me wonder if he would have done even more good by being on the court. Perhaps his program told him otherwise.

Chapters 40,41,42,43 are all on knight tours. On a chessboard (or an $a \times b$ grid) a knight's tour is a way for a knight to visit every square on the board. What if you insist that the tour not cross itself (Chapter 40)? What if you insist that the tour makes a beautiful Celtic pattern (a what? Chapter 41)? What if you insist the knight's tour be skinny (Chapter 42)? How do you generalize this concept (Chapter 43)?

There are articles on music, math, music&math, word games, other games, a candy bar contest that a young Donald Knuth entered, and ... I want to say *etc* but it's not clear that I have established a pattern. This makes it a delight to read but hard to write a coherent review of.

3 Opinion

As noted above, the book is a joy to read. The diversity of topics is overall a good thing, though there may be a topic you do not care for. As an extreme example he has a chapter that has the code and commentary on it for the game *Adventure*. My darling, who is a Software Engineering and has spent (wasted?) many an hour playing Adventure, is very interested in that. Myself... less so.

I predict that

$$(\forall p \in P)(\exists C \subseteq CH)[|C| \geq 30 \wedge (\forall c \in C)[L(p,c)]]$$

where

- P is the set of people who read this column.
- CH is the set of chapters of the book.

- $L(p, c)$ means that person p liked chapter c.

Companion to the Papers of Donald Knuth
Author of Book: Donald E. Knuth
Author of Review: William Gasarch
Center for the Study of Language and Information (CSLI), 2010
438 pages

1 Introduction

Companion to the Papers of Donald Knuth is really five books: (1) problems and a few solutions (30 pages), (2) essays by Donald Knuth on some topics (10 pages), (3) conversations with Donald Knuth from 1996 (157 pages) (4) Donald Knuth's list of papers and other information about him (100 pages) (5) an index to this book and to the other books of Selected papers of Donald Knuth (150 pages).

2 Summary

The problems are all problems that he posed in other journals. They vary tremendously in their scope and difficulty.

The essays are interesting and short so they make their point quickly. The essay that really gets to the heart of what makes Knuth a great researcher is *Theory and Practice and Fun* which says in just 2 pages that we should be driven by our curiosity and a sense of fun.

The conversations I will discuss in the next section of this review.

The list of papers is very nicely annotated. Most papers have brief summaries. There is a list available online `http://www-cs-faculty.stanford.edu/~knuth/vita.pdf` but it does not have the summaries. In this modern electronic age it would be good to have the papers themselves all online at some website, as has been done for Paul Erdos (`http://www.renyi.hu/~p\_erdos/`) and Ronald Graham. (`http://www.math.ucsd.edu/~ronspubs/`).

The index is useful if you have all of the other books.

3 The Conversations

Kurt Vonnegut once said (I am paraphrasing) that interviews do not work that well since the interviewer does not know what questions will lead to interesting responses. Kurt Vonnegut cut this Gordian knot by interviewing himself. The conversations in *Companion* are between DEK (Donald E Knuth) and DK, so I originally wondered if Donald Knuth did the same.

No he did not— DK is allegedly Dikran Karagueuzian, a name so contrived it has to be real [6].

The conversations are fascinating in that they tell much about Donald Knuth and about how computer science has changed. As I was reading them I noted some interesting tidbits for this review; however, that soon became impossible since there was at least one on every page. I will just mention a few.

The conversations took place shortly after Knuth won the Kyoto Prize, hence some of it is about the prize and what he will do with the money. He gave it all to various charities including helping his church get a new organ. That seems so down-to-earth I find it charming.

I vaguely knew that Donald Knuth was the first person to really apply mathematical analysis to algorithms; however, I didn't know that Knuth himself originally saw math (his major) and computer science (his job) as two distinct subjects with no intersection or interplay. The turning point was when he realized that linear probing should and could be analyzed mathematically.

I knew that Donald Knuth created TEX. I had thought that one of the reasons TEX is so much better than Troff (an older word processing system) is that TEX was written by ONE person while Troff was written by a committee. While this may be true it is also important who the one person was. Donald Knuth learned about *every single aspect of typography* while working on TEX. He consulted many of the worlds experts and hence became an expert himself. His knowledge of typesetting— from soup to nuts— is astounding. Here is a tidbit: typesetting a 4th grade arithmetic primer is *harder* than typesetting (say) Knuth Volume 3, since they often have in such primers really big numerals (I don't mean the numbers are large like 10^{100}, I mean the numerals are many inches tall).

I had often said (partially in jest) that all great scientists and mathematicians come from awful family lives. Donald Knuth is a counterexample to this. His childhood and his own life are very happy. He *was not* bored in school, he *was not* an outcast. He earned several varsity letters in sports for being a scorekeeper (not for playing) and he helped the basketball team at Case (his ugrad school) win by devising a way to tell how good a player was by using statistics. This was actually reported on by Walter Cronkite. He is still on his first marriage and has two children John and Jennifer and four

[6] Donald Knuth told me that in the hardcover version of the book there is a picture on the back cover of Donald Knuth and Dikran Karagueuzian together. Amazing what photoshop can do nowadays.

grandchildren. John is high school math teacher and Jennifer is a homemaker. Neither of them have had a CS course for fear of being compared to their father. All very normal and happy. This is all interesting because it's not— that is, it's interesting that there are no hardships or scandals. In addition he is neither boastful nor falsely modest.

A list of tidbits:

1. The term *multiset* is relatively new. Knuth saw the need for a term for a set with repeated elements and he asked Dick de Bruijn about it. De Bruijn suggested *multiset* which became the term. The only competitor was *bag* which is dying out.

2. *The Art of Computer Programming* was supposed to be a book (yes ONE book) on compiler design. But his need to get everything rigorous and right made it become what it is now.

3. Say the variable x is 5 and you do $x := x + 1$, so now x is 6. The fact that it was 5 is gone! This concept, so natural to us now, was troubling to people early on in the field, including von Neumann.

4. In 1967 computer scientists either did *Programming Languages* or *Numerical Analysis* or *Artificial Intelligence*. He had to define the very name of his field: *Analysis of Algorithms*

5. Code Optimization: The conventional wisdom is that compiled code is only 10% slower than if you did it in assembly yourself looking for optimizations. This is already an odd statement since it depends on who YOU are. Donald Knuth states that he's never seen a program that he couldn't make 5 to 10 times faster in assembly; however, he doesn't need that kind of speed.

4 Opinion

I like this book mostly for the conversations. They really give you insight into the Knuth more than a biography or even an autobiography would. Donald Knuth has been called *The father of algorithmic analysis* and hence his thoughts are worth knowing. The book also tells you how things were in the past which is interesting and good to know as we ponder the future.

Joint Review of
Algorithmic Barriers Falling: P=NP?
by Donald E. Knuth and Edgar G. Daylight
Publisher: Lonely Scholar
$20.00 Paperback, 100 pages, 2014

and

The Essential Knuth
by Donald E. Knuth and Edgar G. Daylight
Publisher: Lonely Scholar
$15.00 Paperback, 90 pages, 2014
Reviewer: William Gasarch gasarch@cs.umd.edu

1 Introduction

Both of these books are Edgar Daylight interviewing Donald Knuth. They talk on many topics including computer science, mathematics, and the history of science. Given Donald Knuth's place in our history his perspective is worth listening to. *Algorithmic Barriers Falling: P=NP?* (henceforth ALG) is more about algorithms and theory, *The Essential Knuth* (henceforth KNU) is more about Knuth.

While Knuth is making his points he seems to *not* be criticizing others. I'm not saying *he is careful to not criticize others*, I think being nice just comes naturally to him. As an example, when discussing pointer machines and RAM's he says the following:

RAM's and pointer machines are polynomially equivalent. They differ only when we make finer distinctions, like between linear time and $n\alpha(n)$ (α is the inverse of Ackerman's function). The pointer model hasn't become more popular than the RAM model, because complexity theorists are happiest with a model that makes it easiest to prove theorems.

Those guys have a right to study polynomial fuzzy models, because those models identify fundamental aspects of computation. But such models aren't especially relevant to my own work as a programmer. I treat them with respect but I don't spend too much time with them when they're not going to help me with the practical problems.

2 Very Short Summary

Since the books are short, the review will also be short. I'll just give a list of points that struck me. If you read the books then you may generate a different list.

The following points from ALG struck me:

1. Math papers shouldn't hide how they got to their results.
2. Notation actually drives research. By not using $o(n)$ Knuth forced himself to obtain sharper results.
3. Much asymptotic work has no real application to computing.
4. In the 1960's compiler research represented about 1/3 of computer science. I doubt this is meant as a precise estimate; but suffice to say that it was a large part.
5. It's important to know the history of the field since observing how others created new results may help you to create new results.
6. The history of the Knuth part of the Knuth-Morris-Pratt algorithm: Cook had shown that any set of strings recognized by a 2-way PDA has a linear-time algorithm on a RAM. Knuth went through the construction for the case of pattern matching and came up with a usable algorithm. This surprised him since he didn't think automata theory would ever lead to a simple algorithm that he couldn't come up with using just his programmers intuition.
7. Right after Cook's paper on (what is now called) the Cook-Levin Theorem there was *optimism*— all we need to do is show that SAT is in P and we'll have many other problems in P! This lasted for a few months.
8. LaTeX: while working on it he became an expert on typefaces and fonts. Each letter is somewhat complicated and involves 65 parameters.
9. Knuth thinks that $P = NP$ (yes, that $P = NP$) but that there will still be some sort of distinction within P. Problems that are NP-complete will still be hard, because we won't know explicit polytime algorithms for them — we'll only know that such algorithms exist.

The following points from KNU struck me:

1. Knuth is really a programmer at heart. He may need some esoteric piece of math for an analysis, but his real goal is faster or better programs. It is common in theory (in science? in life?) to use a simple model as a starting point for what you really want to study, but then mistake the model for reality. Knuth never fell into this trap.

2. Knuth was doing statistical analysis of sports (Basketball) way before the moneyball revolution.

3. Knuth wrote a tic-tac-toe program in the early days of computing when it required using symmetries to save space.

4. Dijkstra coined the term *the pleasantness problem* to mean the gap between what we specify (perhaps formally) what we want a program to do and what it really does.

5. Structured programming was a very big breakthrough; however, gotos are sometimes useful.

6. Knuth is bothered by the trend in History of Science to dumb down the science. If these two books were an attempt to counter that trend then they have succeeded. (Actually, even if that was not the intent then they have still succeeded.)

3 Opinion

These are a wonderful books that gives great insights from THE founder of algorithmic analysis. What is most remarkable is that he is truly a computer scientist — he will learn and come up with hard mathematics, but he never loses sight of the original goal: faster real world algorithms for real world problems.