The Book Review Column[1]

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

Welcome to the Book Reviews Column. We hope to bring you at least three reviews of books every month. In this column three books are reviewed.

1. **Algorithms on strings, trees, and sequences: computer science and computational biology** by Dan Gusfield. Reviewed by Gary Benson. This is an encyclopedic book on algorithms in computer science motivated by biological applications and (gasp!) actually used in such applications.

2. **Verification of Sequential and Concurrent Programs** by Krzysztof R. Apt and Ernst-Rüdiger Olderog. Reviewed by Anish Arora. This is a book on verification that is (according to the review) suitable as a text in a grad course.

3. **Algorithms and Programming: Problems and Solutions** by Alexander Shen. Reviewed by Jerry James. This is a quirky book of, as the title suggests, problems and solutions. The review suggests it might be used by a teacher as a source of problems and to refine ones own skills.

Review of **Algorithms on strings, trees, and sequences: computer science and computational biology** [2]

Book authored by Dan Gusfield
Publisher: Cambridge University Press, Cambridge, England, 1997
Other Info: $59.95, Hardcover, 0-521-58519-8, 534 pages
Reviewed by: Gary Benson[3]

Modern molecular biology, focusing on the hunt for structural and functional information about proteins and genes, has received an enormous boost from the field of computer science. Once it was realized that properties of newly discovered molecules could be inferred from similar, previously analyzed molecules, a boom in the use of databases, comparison algorithms and computer searches commenced. An unfamiliar molecule is first sequenced to determine the order of its DNA or protein subunits – four nucloetides in the case of DNA, 20 amino acids in the case of proteins. The molecule is then represented as a string of letters, one for each subunit, and from this alphabetic proxy much information may be gleaned. Sequence comparison algorithms have been around for less than 30 years and sophisticated methods for database searches were developed only within the last 15, yet today, it is inconceivable to start the analysis of a biological sequence without a BLAST or FASTA search of GenBank or Swiss-Prot or any of the other myriad, annotated, crosslinked databases maintained in research centers and national labs throughout the world.

One of the first cases where biological function was hinted by a database search involved the unexpected near identity between fragments of the Simian sarcoma virus (SSV), which causes

[3]Department of Biomathematical Sciences, Box 1023, Mount Sinai School of Medicine, New York, NY 10029-6574

cancer in monkeys, and platelet-derived growth factor (PDGF) which stimulates cell division. The match was found in the early 80's when PDGF was compared to sequences in a nascent protein database that already contained the SSV sequences [3]. The discovery provided a theoretical basis for the mechanism of transformation of normal to cancerous cells by the virus. An errant protein, coded for by the viral genome, stimulates unchecked growth in the cells.

In *Algorithms on strings, trees, and sequences: computer science and computational biology*, Dan Gusfield, recounts this and other examples in which database searching and comparison techniques have been keys to unraveling perplexing biological puzzles. Gusfield, a professor of computer science at UC Davis and an active researcher in the field of algorithms, has produced a rigorously written treatise on pattern matching and sequence-comparison algorithms, sprinkled with illustrations, anecdotes and potential applications from the subbranch of computer science that has come to be called computational molecular biology.

This lengthy book is divided into four major parts. Part I covers the traditional algorithmic area of pattern matching. Part II is rather unusual and can best be described as a homage to suffix trees, a clever data structure for efficiently matching substrings in a text. Part III covers dynamic programming methods which have become the hallmark of sequence comparison algorithms. Part IV covers a variety of topical problems in molecular biology that are related in one way or another to comparisons of biological sequences.

Exact pattern matching, the problem of locating a known pattern string (word, phrase or fragment) in a text string (document) has a long history in computer science. Originally motivated by the need for text searching operations in the first word processors, the problem and its variants have been studied extensively. In Part I, Gusfields presents the now classical algorithms by Boyer-Moore, Knuth-Morris-Pratt, Aho-Corasick and others, including a type of grand-unification of these methods based on his own simplified algorithm for preanalyzing the pattern.

The suffix tree is a data structure representing all the suffixes in a string. For example, the suffixes in *data* are $\{data, ata, ta, a\}$. Many exact pattern matching problems are easily solved once we have a suffix tree in hand. In Part II, Gusfield examines three separate algorithms for suffix tree construction, those of Ukkonen, Weiner, and McCreight. Also included is the lowest common ancestor algorithm, which, when applied to a suffix tree, greatly increases its usefulness.

Originally perceived as a rather difficult data structure, suffix trees continue to appear much more frequently in theoretical algorithms than in implemented algorithms. It is obviously Gusfield's hope that his book will popularize the use of suffix trees; he includes over 40 pages of potential applications. Unfortunately, suffix trees have a deserved reputation for complexity. Gusfield himself acknowledges this by including a section on implementation details (the only such section in the book). Here we learn of the various difficulties presented by issues of alphabet size, excessive space requirements, the need to preserve suffix links (used to build the tree, discarded in some cases, retained in others), etc., the cumulative effects of which tend to make an implementation problem specific.

Gusfield's attempt to present suffix trees as a nostrum for many problems in computational biology is strained. Given the types of mutations and errors typically found in molecular sequence data, exact-matching methods are significantly more complicated than dynamic-programming methods. Even off-the-shelf heuristics such as BLAST and FASTA would be preferred and are being used to great effect. Consider a recent clustering by similarity of almost 18,000 protein sequences from several newly established bacterial and yeast genomes [5]. Similarity was computed using a BLAST variant and although the comparisons may have taken weeks to run, the effort was certainly more efficient than building a special-purpose suffix-tree program the development of which would have taken much longer.

In Part III, Gusfield presents the aforementioned techniques for divining the similarity of two protein or nucleic acid sequences. To compute sequence similarity, one first assumes a model of the types of mutations that modify sequences. The problem, then, is to find an optimal way in which one sequence can be transformed into the other using the allowed mutations. Equivalently, one can find an optimal *alignment* of the sequences, that is, a way of lining them up so that similar parts are paired. Alignment is typically solved by a method called dynamic programming. This is a straightforward and flexible calculation and Gusfield describes the basic dynamic programming algorithms including Needleman-Wunsch, Smith-Waterman and their many variants. Here we also find methods for aligning multiple sequences and too brief descriptions of the database searching programs BLAST and FASTA.

Optimality of alignment depends not just on the types of mutations, but also on a scoring scheme applied to those mutations. Gusfield does a good job of explaining how choices for substitution scores and gap penalties heavily influence the resulting alignment. Descriptions of the PAM and BLOSUM substitution score matrices are brief, but informative and a variety of gap penalty functions are explored. Included is a section on parametric alignment which is a way of visualizing the stability of alignments as the scoring scheme changes in a systematic way. This is an interesting problem and the focus of some of Gusfield's own research, but oddly, the description of his algorithm lacks the leisurely pace and depth of detail it deserves.

Other contemporary problems in molecular biology including genome mapping, sequence assembly and evolutionary tree reconstruction have prompted the development of more complex algorithms. These methods are surveyed in Part IV.

Considered as a whole, this book, which runs to just over 500 pages, is clearly written but wordy and very technical. Much of the book is written as though it were a journal article. Dozens of algorithms are included, and almost all are accompanied by extensive proofs of correctness, and time and space complexity. Some of the proofs display the lucid elegance which is Gusfield's strength as a theoretician, but many others are merely long and complicated. Helpful figures are frequently provided and thoughtfully positioned on the same page as the proof they illustrate. Proofs and algorithms are often co-mingled, with no final pseudocode summary, an unfavorable situation for those interested in implementations. One sad example, given the emphasis on suffix trees, is Ukkonen's algorithm which is presented in three parts spread over 10 pages. Internet accessible implementations for many of the algorithms, written in C, are promised, but as of this writing, (December 1997) only one program for parametric alignments is available.

Gusfield clearly has devoted much time to developing exercises. Each chapter ends with a rich collection of questions which explore the various algorithms and seem designed to motivate thinking about alternate ways of addressing problems. This book will deservedly serve as a source of homework and exam problems for years to come.

The encyclopedic nature of *Algorithms on strings, trees, and sequences* is both its greatest strength and biggest fault. Pattern matching is usually given only cursory coverage in algorithmic textbooks and Gusfield's book is one of a number of recent issues that have sought to fill the void [4, 2, 6]. Regrettably, Gusfield has included too much. He provides essentially no winnowing out of the less important proofs, the most egregious example being the nine pages devoted to Richard Cole's proof that the Boyer-Moore algorithm for exact string matching runs in linear time [1]. This thoroughness is a boon for those who already possess expert knowledge and can appreciate such a complete accounting in a single volume (the bibliography alone is a superior resource). However, the sheer complexity of the presentation is bound to limit its accesibility for those who are interested, but lack the technical competence to follow the details. Gusfield claims his book is directed at 'the research level professional in computer science or a graduate student or advanced undergraduate

student in computer science' and that it should serve as 'both a reference and a main text for courses in pure computer science and for computer science-oriented courses on computational biology'. I suspect, though, that most students will find this book too detailed and too complicated for their needs.

Nonetheless, Gusfield's book is an important summary of the state of the art in pattern matching and an indicator of the importance biological problems have assumed among many researchers. It will undoubtedly draw new researchers to problems in biology and will hopefully encourage them to question the importance of the problems they endeavor to solve. As Gusfield advises in his epilogue: 'focus more on the biological quality of a computation and not exclusively on speed and space improvements...learn real biology, talk extensively to biologists, and work on problems of known biological importance'. Good advice and especially apt for those hoping to make an impact in this exciting field.

References

[1] R. Cole. Tight bounds on the complexity of the Boyer-Moore pattern matching algorithm. *SIAM J. Comput.*, 23:1075–1091, 1994.

[2] M. Crochemore and W.Rytter. *Text Algorithms.* Oxford University Press, New York, 1994.

[3] R.F. Doolittle, M. Hunkapiller, L.E. Hood, S. Devare, K. Robbins, S. Aaronson, and H. Antoniades. Simian sarcoma virus *onc* gene *v-sis*, is derived from the gene (or genes) encoding a platelet-derived growth factor. *Science*, 221:275–277, 1983.

[4] J. Setubal and J. Meidanis. *Introduction to computational molecular biology.* PWS, Boston, 1997.

[5] R.L. Tatusov, E.V. Koonin, and D.J. Lipman. A genomic perspective on protein families. *Science*, 278:631–637, 1997.

[6] M. Waterman. *Introduction to Computational Biology: Maps, Sequences and Genomes.* Chapman and Hall, London, 1995.

Review of: **Verification of Sequential and Concurrent Programs**

by Authors: Krzysztof R. Apt and Ernst-Rüdiger Olderog
Publisher: Springer-Verlag New York, Graduate Texts in Computer Science
$44.95, Hardcover, Second Edition, 1997, 364pp, ISBN 0-387-94896-1

Reviewed by Anish Arora, The Ohio State University

Specification and verification of programs is increasingly being taught to undergraduate and graduate computer science students. Courses along these lines enable students to understand and reason about programs as formal objects. Especially when students deal with concurrent programs do they appreciate that program correctness —while not necessarily deep in the sense of mathematics— can be hard. There are many possibilities to consider and intuitive arguments of correctness are often wrong, hence there is need for program verification.

We find that this beautifully written and smoothly flowing textbook should serve as a fine candidate for teaching graduate-level and possibly upper-level undergraduate courses on, or with a component on, program verification.

Overview The present edition of the book abridges and reorganizes its prior edition, and is now much better suited for its use as a course text. Its coverage continues to be broad: One part begins with deterministic sequential programs and then systematically scales up to parallel programs (with or without shared memory, and with or without synchronization). A second part begins with nondeterministic sequential programs and then analogously moves on to distributed (message passing) programs. Lastly, fairness (both strong and weak) is addressed.

The chosen style of programming is a standard imperative one: program statements are allowed to perform skips, assignments, in-sequence actions, conditionals, iterations, and guarded commands; by way of contrast, subtypes, functions, and procedures are not considered. Programs may be composed in parallel, in which case their concurrent computation is simply interpreted as the nondeterministic interleaving of the computations of the component programs. (Nondeterminism implies that no assumption is made about the relative speed of execution of the component programs, which motivates the need for separate discussion of fairness. Interleaving implies that concurrent execution of actions is assumed to be equivalent to their serial execution in some order, which enables one to readily scale up reasoning from sequential to concurrent programs.) Shared memory is dealt with by allowing certain statements in component shared-memory programs to be atomic, i.e., indivisible, and to even await some condition. Distributed memory is dealt with à la CSP, by allowing a send to or a receive from a channel in statement guards and assuming that communication is synchronous.

The chosen style of verification is syntax-driven and axiomatic: proof systems are given to verify Hoare-style assertions about the input/output behavior of programs. Assertions for both partial and total correctness are considered. The reader will recall that a Hoare-triple $\{p\}S\{q\}$ is true in the sense of partial correctness if every *terminating* computation of S that starts at a state where p holds terminates at a state where q holds. By way of contrast, $\{p\}S\{q\}$ is true in the sense of total correctness if every computation of S that starts at a state where p holds terminates at a state where q holds. This style of verification may be contrasted with those based on temporal logic (cf. the book on UNITY by Chandy and Misra, and the books by Manna and Pnueli) or on denotational semantics.

Each class of programs is discussed in a uniform manner —in terms of its syntax, operational semantics, proof system (with arguments of soundness and, in one case, of completeness relative to the assertion language), and case studies in program verification/derivation— which makes navigating even the toughest parts of the book tractable. But perhaps the main reason this book flows smoothly is because of its use of transformations. Transformations are used to give the semantics of a new programming concept in terms of a known one. For example, parallel programs with fine-grain atomicity are transformed into equivalent coarse-grain ones, programs with disjoint parallelism into equivalent sequential ones, and distributed programs into equivalent nondeterministic ones. Thus, the reader gets an extra handle for understanding the new concept and, moreover, is accorded the reuse of known proof rules for reasoning about the new concept.

Opinion We mention several points worth considering in teaching with this book. For one, although the book is self-contained, students' familiarity with predicate calculus, proof systems, and operational semantics is an essential prerequisite. As another, covering this book fully in a semester —let alone a quarter— is unlikely. One choice here might be to focus the discussion of the

proof systems on either their soundness/completeness aspects or their detailed application in case studies, but not on both. As far as fairness is concerned, this reviewer would be quite content to cover only weak fairness. Yet another point is providing exposure to specification/verification styles other than that given in this book, if only to expose students to the appreciation that multiple styles are just as warranted for specification/verification as are they are for programming.

A more substantial point is how to tie in the verification ideas taught in this book with other topics in the computer science curricula. In other words, reinforcing these ideas in other courses, especially practice-oriented ones, is essential for enabling students to apply the ideas in their future careers. One option here, which has worked well in the case of this reviewer, is to integrate verification ideas into systems courses such as operating systems and distributed systems. Preliminary experience shows that laboratory exercises involving the machine-assisted verification of system components give students insight into system behavior and confidence in system design.

Review of *Algorithms and Programming: Problems and Solutions*
by Alexander Shen
Published by Birkhäuser Boston, 1997
Hardcover,$42.50, 264 pages
ISBN number 0-817-63847-4

Review by
Jerry James
Department of Computer Science
University of California at Santa Barbara
jamesj@acm.org

1 Overview

This book contains a collection of problems and their solutions. Most of the problems are of the type that would be encountered in a course on data structures or compilers. Each chapter presents a series of problems running around some central theme, and gives solutions to those problems, or hints that should lead to solutions. In some cases, multiple solutions are given to one problem, to illustrate varying solution techniques. This is often done to show solutions with various time or space complexities. The problems cover a range of difficulty, usually beginning with the simple and ending with the difficult. Many are appropriate for undergraduate classes in data structures and compilers, but some will prove challenging for students in graduate-level classes on these topics.

2 Summary of Contents

Chapter 1 covers problems related to variables, expressions, and assignments. The first section, "Problems without arrays," begins with very simple problems, such as computing a^n. It provides progressively harder problems, such as computing Fibonacci numbers and several variations on Euclid's algorithm for finding the greatest common divisor. It concludes with some algebraic and numerical problems, e.g., finding the length of the period of $1/n$ for some natural number n. The second section, "Arrays," begins with simple functions on arrays, such as array assignment, determination of maximum elements, and reversal of elements. Some problems dealing with polynomials and sets are covered next, and the section ends with several algebraic problems, such as computing

inverse permutations. The third section, "Inductive functions," contains such problems as finding the maximal length of a sequence of integers that is a subsequence of two given sequences.

Chapter 2 deals with combinatorial problems. The first section, "Sequences," requires finding all sequences of integers satisfying some property. The second section, "Permutations," contains a single problem: find all permutations of $1, \ldots, n$. The third section, "Subsets," has problems involving the finding of all subsets satisfying some property. "Partitions," begins with the problem of finding all representations of a positive integer n as a sum of positive integers. The remaining problems deal with representations of partitions as sequences. The fifth section, "Gray codes and similar problems," first has the reader prove the theory behind Gray codes, that it is possible to list all 2^n strings of length n over $\{0, 1\}$ such that adjacent strings differ in exactly one bit. Two variations on this theme are then presented, the last being generation of permutations of $1, \ldots, n$ such that adjacent permutations differ by a single transposition. Consider all sequences composed of n 1s and n -1s such that the sum of any prefix is nonnegative. The number of such sequences is called the *Catalan number* . Section six, "Some remarks," applies techniques developed previously in the chapter to problems related to Catalan numbers. The chapter ends with Section seven, "Counting," in which the number of elements of some sat that satisfy some predicate are counted. This section culminates with a proof that the Catalan number is $\binom{2n}{n}/(n+1)$.

Chapter 3 contains problems relating to tree traversal and backtracking. It contains only two sections. The first section is "Queens not attacking each other; position tree traversal;" the second is "Backtracking in other problems." The first section introduces the idea of tree search, and shows how the leaf nodes correspond to board positions. The reader is asked to show that a program generates all possible configurations of n queens on an $n \times n$ board such that no queen can attack another, to show that the program terminates, and to show that several variations on tree traversal order are correct. The section ends with problems on the complexity of the solution and the correctness of an optimization. The final section briefly shows how to apply the same techniques to the generation of sequences satisfying a property.

Chapter 4 is dedicated to sorting algorithms. The first section is "Quadratic algorithms." It asks the reader to produce very simple $O(n^2)$ sorting algorithms. The solutions provided show bubble sort (although, strangely, the name is not used) and insertion sort. The second section, "Sorting in $n \log n$ operations," contains a single question: find an $O(n \log n)$ sorting algorithm. The solution to this problem is atypically long, showing the operation of both merge sort and heap sort. Quicksort, which runs in expected $O(n \log n)$ time, is mentioned in passing. Section three, "Applications of sorting," shows that various problems of maxima- and minima-finding can be solved by sorting. Section four, "Lower bound for the number of comparisons," shows that any comparison-based sorting algorithm has complexity at least $O(\log_2 n!)$. It then shows that some sorting algorithms that can take advantage of the internal structure of the data to be sorted can improve on this bound. Four algorithms are developed, each working on an idea similar to radix sort (although radix sort itself is not mentioned). Section five, "Problems related to sorting," seems to be similar in spirit to section 3. That is, the problems in this section deal with finding minima, maxima, and the kth element in some order.

Chapter 5 is entitled, "Finite-state algorithms in text processing." It consists of two sections, "Compound symbols, comments, etc." and "Numbers input." The idea is to process text files (program files are used as examples in the text) using very simple mechanisms. For example, remove all occurences of some string, or change all occurences of some string s to another string t. The second section presents various problems related to the parsing of strings representing numbers.

Chapter 6 has problems for several fundamental data types. It begins with stacks. The chapter opens by showing how a stack of bounded size can be implemented with Pascal arrays.

The readers is asked to decide whether a given file has balanced and properly nested parentheses and braces and to implement multiple stacks using one array. The second section addresses queue and deque implementations. Various constraints are given, and the reader is asked to produce an implementation meeting those constraints. For example, implement a bounded deque using an array so that all operations take $O(1)$ time. Various problems that can be solved with queues or deques are covered as well, such as finding the convex hull of n points in $O(n)$ time, when the points are sorted by (x, y) coordinates (with one coordinate more significant than the other). Section 3, "Sets," begins by asking for set implementations satisfying various time and space complexity constraints on the operations. It closes with a set-oriented problem on reachability in graphs. The chapter ends with a section on priority queues, in which the reader is asked to provide two implementations meeting time complexity constraints.

Chapter 7 is entitled, "Recursion." The first section, "Examples," shows how to use recursion to compute factorials, integral powers of real numbers, and solve the Towers of Hanoi puzzle. The second section, "Trees: recursive processing," asks the reader to find various parameters of a tree, such as its height, number of leaves, number of vertices, number of vertices at a given depth, etc. Section three, "The generation of combinatorial objects; search," contains problems relating to the generation of all sequences meeting some criterion (e.g., all representations of the positive integer n as the sum of a nonincreasing sequence of positive integers). The last section, "Other applications of recursion," shows its use in topological sorting, finding connected components of a graph, and quicksort.

Chapter 8 focuses on finding equivalent nonrecursive algorithms for various recursive algorithms. The first section, entitled "Table of values (dynamic programming)," gives a recursive algorithm for computing binomial coefficients and asks the reader to write a nonrecursive version. The time and space complexities of the two versions are then compared. The remaining problems in this section address such topics as triangulating a convex polygon, mulitplying a list of matrices, and the knapsack problem. The concept of memoization is introduced. The second section is called, "Stack of postponed tasks." Here, several recursive algorithms from chapter 7 (e.g. Towers of Hanoi) are rewritten as iterative algorithms with an explicit stack representation. The final section, "Difficult cases," asks the reader to construct nonrecursive evaluators for the function f in the following two cases:

$$\begin{aligned} f(0) &= a \\ f(x) &= h(x, f(l(x))) \quad (x > 0) \end{aligned}$$

$$\begin{aligned} f(0) &= a \\ f(x) &= h(x, f(l(x)), f(r(x))) \quad (x > 0) \end{aligned}$$

In each case, we assume that a is a constant, h, l, and r are known functions, and repeated applications of either l or r to any nonnegative integer eventually produces zero.

Chapter 9 contains graph algorithms. The first section, "Shortest paths," defines path length by edge weights, and casts all problems in terms of the Traveling Salesman problem. The reader is asked to find the minimal travel cost from a fixed city to all other cities in $O(n^3)$ time, via the Ford-Bellman algorithm, then via the improved Ford algorithm. The reader is then asked to improve this to $O(n^2)$ time via Dijkstra's algorithm. The second section is "Connected components, breadth and depth search." Besides the two search algorithms mentioned in the section name, the reader is also introduced to the notion of a bipartite graph, and asked to write a decision procedure for such graphs. The chapter closes with a nonrecursive solution to topological sort on acyclic graphs.

Chapter 10 is on pattern matching. The first section is "Simple example." True to its name, the reader is asked for an algorithm that checks for the string "abcd" in a given string. The solution employs a simple finite automaton. Section 2, "Repetitions in the pattern," notes that the simple finite automaton of the first section breaks down if we are checking for, say, "ababc" or some other string with a repetition. The user is asked to create a more sophisticated finite automaton that can cope with such repetitions. Section 3, "Auxiliary lemmas," asks for three simple proofs that are needed in the next section, "Knuth-Morris-Pratt algorithm." This latter section walks the reader through an $O(n)$ substring checking algorithm. Section 5, "Boyer-Moore algorithm," presents a simplified version of the Boyer-Moore algorithm that has time complexity $O(nm)$, where n is the length of the pattern and m is the length of the string. The reader develops this algorithm through solving a series of subproblems. Another algorithm for substring matching is given in the next section, "Rabin-Karp algorithm," where a function is computed on each substring and compared with the same function on the pattern. Only if the outputs matches are the pattern and substring compared letter-by-letter. A trick that makes this approach reasonable is explained. Finally, section seven is "Automata and more complicated patterns." Regular expressions are introduced. The reader is asked to construct several regular expressions. Then NFAs are introduced, and their equivalence to DFAs is proved. Finally, the reader shows that REs are exactly the set of languages recognized by NFAs.

Chapter 11 is on hashing. The first section is "Hashing with open addressing." The reader is led to discover the collision problem, and the lookup problem after a collision and a deletion. In the second section, "Hashing using lists," introduces buckets, implemented as lists, to solve the collision problem. The remainder of the section, and the chapter, is spent in an investigation into universal families of hash functions. Such functions uniformly distribute elements across the hash table. Two families of universal hash functions are identified.

Chapter 12 deals with sets represented by trees, and especially balanced trees. Section 1, "Set representation using trees," takes an unusual amount of space for this book in defining the notions of full binary tree, T-tree (a tree labeled with elements of the set T), subtree, height, and ordered T-tree. It also introduces the notion of considering a tree to be the set of all labels in the tree. It shows how to use arrays to store (not necessarily full) binary trees. The reader is asked to implement various set operations on an ordered tree. The time complexity of each operation is shown to be proportional to the height of the tree. This leads to the topic of Section 2, "Balanced trees." The reader is introduced to AVL trees, and develops the operations on them one by one. At the end, B-trees are introduced briefly.

Chapter 13 is on context-free grammars. The first section, "General parsing algorithm," defines context-free grammars and gives a few examples. The reader is then asked to give context-free grammars for a few simple languages, and show that no context-free grammar exists for languages of the form $0^k1^k2^k$. The reader is also asked to derive a general polynomial time algorithm for checking whether a string belongs to some context-free language. In section 2, "Recursive-descent parsing," recursive descent parsers are derived for several grammars. The reader is asked to identify context-free grammars that are and are not parseable by recursive descent. Section 3, "Parsing algorithm for LL(1) grammars," begins by defining the notion of a leftmost derivation and asking the reader to work through some examples. LL(1) grammars are then defined and characterized, and the reader is asked to decide whether certain grammars have an LL(1) form. The reader is then led to derive an LL(1) parsing algorithm, and a means of generating the *First* and *Follow* sets.

Chapter 14 , the final chapter, covers LR parsing. The first several problems of section 1, "LR-processes," develop the theory of LR parsing by showing the existence and uniqueness of rightmost

derivations, and showing how to construct the *Left* sets. Section 2, "LR(0)-grammars," begins by showing how shift/shift and shift/reduce conflicts arise. The reader is led to the derivation of an LR(0) parsing algorithm. Section 3, "SLR(1)-grammars," extends the results of section 2, asking the reader to write a parser for SLR(1) grammars. In section 4, "LR(1)-grammars, LALR(1)-grammars," shows how grammars can fail to be SLR(1) but still be LR(1). The reader is asked to extend the solutions derived for SLR(1) to LR(1). Then LALR(1) grammars are introduced. The reader is then asked to show that every SLR(1) grammar is LALR(1), and every LALR(1) grammar is LR(1), and to improve the LR(1) parser for LALR(1) grammars. The chapter, and book, conclude with section 4, "General remarks about parsing algorithms." This section contains no problems, but simple points the reader to automatic parser generators (yacc/bison) and the well-known compiler book by Aho, Sethi, and Ullman.

3 Style

The book is mostly written in a problem-answer style. However, the further toward the back one goes, the more sections of explanation there are. Indeed, the book seems somewhat schizophrenic in this regard, as though the author could not decide between writing a textbook and a book of problems and answers to be used as a resource for teachers. Some parts have no explanation at all, yet seem equally as difficult as parts that were almost all explanation, with only a handful of obvious problems.

4 Opinion

The book will prove useful for those who need homework or test questions for the areas covered by it. Many of the questions are formulated in such a way that producing variants on them can be done with ease. For teaching the concepts contained therein, though, I recommend one of the standard textbooks. The explanations in this book are not nearly thorough enough. Furthermore, newer data structures and algorithms are omitted. For example, B-trees are given only a cursory treatment, and variations (such as B^*-trees) are not mentioned at all.

Most of the answers followed in an obvious way from the questions. There were a few cases where the question was poorly worded, and a handful of places where the answer can be improved with little effort. Overall, though, the book is well done. I recommend it to teachers and those wishing to sharpen their data structure and compiler skills.