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.