“Strutture Di Dati e Algoritmi. Progettazione, Analisi e Visualizzazione”, a Book Beating Its Own Drum

Anna Bernasconi
Dipartimento di Informatica, University of Pisa, Italy

Linda Pagli
Dipartimento di Informatica, University of Pisa, Italy

Abstract

In this contribution, we discuss the innovative approach taken by Roberto Grossi and his co-authors in their book “Strutture di dati e algoritmi. Progettazione, analisi e visualizzazione,” which aims to bridge the gap between the theoretical and practical aspects of algorithms. Unlike traditional texts that either focus on formal mathematical analysis or practical implementation, this book adopts an intermediate approach that emphasizes both programming skills and theoretical understanding. This contribution reflects our experience as instructors of the introductory algorithms course in the Computer Science degree program at the University of Pisa.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Algorithms, C-Programming, Basics of Computational Thinking

Digital Object Identifier 10.4230/OASIcs.Grossi.2025.3

Category Education

Dedicated to Roberto Grossi

The book “Strutture di dati e algoritmi. Progettazione, analisi e visualizzazione” by Pierluigi Crescenzi, Giorgio Gambosi, and Roberto Grossi was first published in 1999. Focusing on algorithms – a foundational topic in computer science – the authors chose not to simply follow in the footsteps of other renowned texts. Instead, they aimed to offer a fresh and innovative perspective, taking into account the most recent developments in algorithm research at the time.

Before this volume, there were basically two main approaches, and they were pretty different from each other. The first one looks at algorithms, their correctness, and their computational complexity from a more theoretical and formal angle, rich in mathematics. It goes through the history of classic algorithms, organizing them by topic, comparing them based on their theoretical complexity, and figuring out their computational limits. This approach uses a high-level language to highlight the key features of an algorithm without getting distracted by implementation details. A great example of this is the famous book “Introduction to Algorithms” by Cormen, Leiserson, and Rivest, which first came out in 1990 and was updated to include Stein in 2002. It has been adopted in many computer science courses around the world, including at MIT, and has become a classic in the field. Many Italian books that followed have pretty much followed this line without altering its approach.

On the other hand, the second approach concentrates solely on the implementation of algorithmic problems, with the goal of optimizing the computer’s capabilities for maximum efficiency. This approach is less formal, and the study of efficiency is based more on experimentation than on theoretical analysis. It is a practical approach focused on identifying

Pierluigi Crescenzi • Giorgio Gambosi • Roberto Grossi
Strutture di dati e algoritmi
Progettazione, analisi e visualizzazione
PEARSON
Addison Wesley
Pierluigi Crescenzi • Giorgio Gambosi
Roberto Grossi • Gianluca Rossi
STRUTTURE DI DATI
E ALGORITMI
Progettazione, analisi e programmazione
Seconda edizione
ALWAYS LEARNING
PEARSON

Figure 1 The covers of the first and second edition of the book "Strutture di dati e algoritmi. Progettazione, analisi e visualizzazione".

efficient algorithms based on the actual conditions under which they operate. Quality code and experimentation are the goals of this approach, perhaps more suited to experienced programmers.

Roberto Grossi et al have followed an intermediate approach, aimed at students who are tackling programming for the first time but are also beginning to orient themselves in theory, studying classic problems and learning to analyze solutions. Special attention is given to the code of an algorithm, which is not yet complete and is almost entirely lacking parameter testing and input/output instructions. However, it is very close to actual code (C or Java) and can be implemented with only a few minor additions. Programming skills are placed on the same level as theoretical analysis. This type of training seems particularly suitable for first-year computer science students. We, the authors of this note, were at the time instructors of the two twin courses in Algorithms in the Computer Science degree program at the University of Pisa. We were already experienced in teaching this subject, as well as in using the Cormen's book, but upon learning about the new volume, we did not miss the opportunity to adopt it, curious about the new approach and with the convenience of being able to consult directly with one of the authors in case of doubts.

The book also presents other attractive features: it offers students the opportunity to visualize, through a system called Alvie accessible online, the execution of almost all the algorithms presented in the book, with input data specified by the student. Following the execution of an algorithm as things become more complicated is a very effective teaching tool for both the student and the teacher, who can avoid the often tedious manual execution on the board. Another interesting feature is the integration of theoretical topics with practical applications, with the dual aim of providing students with a clear idea of the importance of the concepts studied and giving teachers a set of useful motivations to present the more challenging topics. These integration topics between theory and practice are referred to in the volume as *opus libri*.

Lastly, we liked the cover of the first edition, where little robots in the shape of a colorful, stylish personal computer, very popular in the 1990s, work together to build a treehouse and make a computer function.

Adopting a newly released book and following its framework in teaching also means seeing some limitations and finding inevitable calculation errors, printing mistakes, and even some subtler ones. For us, it was a discovery, a challenge, and also a struggle, but we believe we have prepared students well in the field of algorithms and have contributed operationally to the production of a better second edition of the volume. The students appreciated the book, and with the help of lessons and exercises, they also learned the most theoretical concepts and complicated proofs; in particular, we will mention some of the most educationally formative topics, enjoyable to teach and well-received by the students.

The book begins with difficult problems, the distinction between P and NP, and the important concept that for many interesting problems, no better solution algorithms are known than those that explore the whole set of possible solutions. These algorithms, known as *brute force*, require exponential time in the worst case with respect to the input size of the problem. To explain to students the complexity and length of an exponential algorithm that tests all possibilities, the game of Sudoku is used as an example. Sudoku emerged in the 1990s and became very popular during that time. It is a useful tool for explaining the definition of the NP class, which includes problems that can be easily verified in polynomial time. Indeed, in Sudoku, having the solution to a specific puzzle (which is published in newspapers shortly after the puzzle is proposed) and placing the numbers in the correct spots allows one to easily verify that the solution satisfies all the specified constraints. Furthermore, in solving games proposed as difficult, one proceeds exactly as in brute force algorithms, going through multiple decisions, where different values can be associated with a generic cell. By choosing one of these, one can proceed for one or more steps or even to the end, or one must go back to the most recent step where a choice was made and move on to the next choice if it exists. With a nice example that is not too large, students, almost all familiar with the game, can quickly grasp the meaning of a brute force algorithm and have a practical idea of time exponential in the dimensions of the starting grid.

In our opinion, one very educational topic introduced in the volume regarding data structures is that of variable-sized arrays. The array is studied along with its memory allocation. When the array of size d is dynamic, meaning it undergoes many insertions and deletions, care is taken to ensure that the memory space reserved for it is always adequate. In particular, if, for a new insertion, the number of elements in the array reaches its size, the array is duplicated, leading to the elements being copied into a new array of size $2d$. Conversely, if, following a new deletion, the number of stored elements becomes equal to $d/4$, the array is halved, which also involves copying the elements. It is shown that n operations of insertion or deletion in a dynamic array can be performed in $O(n)$ time. The proof is very brief and neat but not trivial; it simultaneously shows that halving when the number of elements is equal to $d/2$, a choice that seems the most natural, does not work, which helps to better understand why the choice to halve when $n = d/4$ is made. The analysis of dynamic arrays introduces the concept of *amortized complexity*: the cost of copying the array is spread over previous and subsequent operations that do not require any action, and these are sufficiently numerous to absorb the cost of copying the elements of the array.

Amortized complexity is introduced and discussed even more explicitly in the chapter dedicated to lists, where the study of self-organizing lists is explored, particularly the Move-to-Front strategy. This strategy is familiar to students because it is used in the call logs on their phones (which were not yet smart at the time). The analysis of the computational cost of this strategy is not simple and requires nearly an entire lesson. It was especially rewarding to complete the lesson without losing the students' attention, and even more so when they chose this topic as one they were eager to discuss during the oral exam.

Another innovative topic, never encountered in other similar texts, is that of recursive algorithms on binary trees. A general paradigm is presented, as an extension of divide and conquer on arrays, to be applied to binary trees. The typical divide and conquer scheme – solve recursively on the left subset of $n/2$ elements, solve recursively on the right subset of $n/2$ elements, and then recombine – applies almost directly to binary trees, replacing the subset with the left and right subtrees, respectively. This type of approach allows for very easy solutions to classic problems on trees that would otherwise be quite complicated. Students easily grasp this tool and use it to effectively solve problems that, when approached without this framework, would traditionally be challenging.

A basic algorithm for the sorting problem is Quick-Sort, which is super efficient in the average case, while in the worst case it is quadratic like the most basic algorithms. Since it is ultimately the most used algorithm in practice and, with some variations, is present in the libraries of many programming languages, it is important to demonstrate to students its average case cost. As is well known, proofs in this case are much more complicated than those considering the worst case, where identifying the situation that leads to the highest complexity often involves elementary mathematical study. The average case, on the other hand, requires considering and evaluating all possible inputs, often making use of probability, a topic that first-year algorithm students will tackle the following year. The text offers an alternative approach that, while not trivial, simplifies the analysis and is particularly well-suited to students.

Dynamic Programming is a rather original algorithm design technique that is typically applied to optimization problems, where the so-called principle of optimality holds. This principle states that, for the problem at hand, the optimal solution can be derived from the optimal solutions of smaller subproblems. The algorithms are generally complex and not very intuitive, to the point that sometimes it is not at all straightforward to even understand how they work; designing a solution using this technique is even more challenging and requires a deep mastery of the subject. Thus, the book provides a sort of guide for defining an algorithm based on four fundamental aspects:

1. The formulation of the problem structure that is valid for the problem in question and its subproblems.
2. The identification of elementary subproblems and the assignment of their solution values, obtained through direct inspection.
3. The definition of the recursive rule that allows the problem to be solved as a composition of the solutions of subproblems.
4. The derivation of an ordering of the subproblems for efficient computation and the storage of partial solutions in a table.

These rules, which may seem obscure at first, accompanied by various examples, provide important guidance and assistance to those who wish to tackle the definition of a dynamic programming algorithm. Students struggle with this topic, but in the end, with step-by-step guidance, they manage to master it. We can assert that the four-step technique is quite effective and provides important guidelines in the design of a dynamic programming algorithm.

The implementation of sorted dictionaries on balanced binary search trees is another topic that Roberto's book presents in a clear and effective way. The chosen balancing technique is that of 1-balanced binary search trees, known as *AVL trees* (named after the initials of the Russian authors Adelson-Velsky and Landis, who proposed them in 1962). A binary tree is 1-balanced if the heights of the two child subtrees of any node differ by at most one. The connection between being 1-balanced and having logarithmic height is not immediate and involves Fibonacci trees, which are the most unbalanced among 1-balanced trees. More specifically, a Fibonacci tree is a 1-balanced tree that has the fewest nodes for a given height. Starting with Fibonacci trees, the text walks the reader through a detailed process that uses concepts such as minimality, recursive definitions, closed-form Fibonacci numbers, and exponential growth. This all comes together to prove that a generic 1-balanced tree has a logarithmic height. After this somewhat theoretical discussion, the student is quickly rewarded with code to implement dictionary operations on AVL trees. The code is clear, comprehensive in all its stages (including the rebalancing of the structure through rotations), and practically ready to be executed.

In all algorithm books, graphs are treated as a static structure: memory representations rely on keeping all the nodes of the graph in a direct access array, which requires knowing all the nodes of the graph in advance to ensure constant time access to any node. All traversal algorithms are based on this assumption, which is never explicitly stated, allowing their complexity to remain linear in the number of nodes and edges, precisely because access to any node is achieved in constant time. Classic algorithms on graph structure also always assume that the entire graph is known a priori. Networks like the internet, naturally represented as graphs, are instead very dynamic structures, where important information must be calculated, such as degree, eccentricity, shortest paths, etc. In Roberto's book, dynamic graphs are given separate treatment, where nodes are allocated in a dictionary, that is, in a flexible structure, for example organized as a balanced tree or a hash table, which guarantees efficient access, insertion, and deletion operations, but not executable in constant time. Consequently, graph algorithms can be transformed, providing students with immediate use of all variations on the organization of a dictionary, introduced earlier, and giving instructors an inexhaustible source of problems to propose as exercises.

When the Algorithms course was accompanied by a separate course on Practical Programming Laboratory, the implementation part of basic algorithms was almost completely transferred to that course, and the Algorithms course inevitably shifted a bit more towards theory. Thus, we returned to the original course setup, resuming Cormen et al as the textbook. Nevertheless, almost all the topics mentioned above retained the approach and important concepts encountered in Roberto et al book during the lessons.

As a conclusion, let us say that Roberto is an innovator, and this attitude is reflected in all his activities: from scientific research to teaching and writing textbooks, from finding the best leavening for homemade bread to creating cooking recipes inspired by Japanese culture, and from preparing students for the *Olympiad in Informatics* to educating and protecting them as if they were his own children.