Olympiads in Informatics, 2021 Vol. 15, 143–146

DOI: 10.15388/ioi.2021.12

Reviews of Two Programming Books

Antti LAAKSONEN
University of Helsinki, Department of Computer Science
e-mail: ahslaaks@cs.helsinki.fi

In this article I review two recent competitive programming books, published in 2020, which have not yet been presented in the *Olympiads in Informatics* journal. The books are *Algorithmic Thinking* by Daniel Zingaro and *Competitive Programming in Python* by Christoph Dürr and Jill-Jênn Vie.

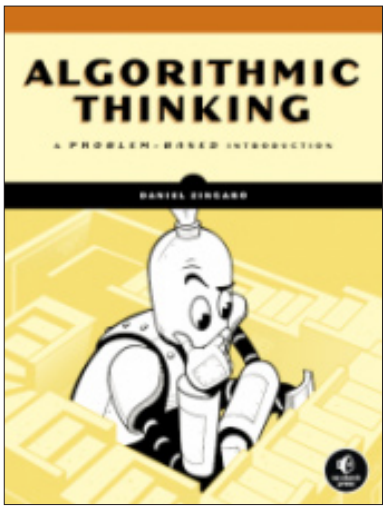
Both the books are introductory books but their approaches are different. *Algorithmic Thinking* focuses on the process of learning algorithmic problem solving and uses competitive programming problems as motivating challenges. *Competitive Programming in Python* develops skills for programming contests and job interviews, and shows how the Python language can be used in competitive programming.

Algorithmic Thinking: A Problem-Based Introduction

The book discusses data structures and algorithm design techniques, and shows how they can be used to solve problems selected from various programming contests. However, the main goal of the book is not to prepare for programming contests, but rather to teach problem solving through motivating and challenging problems. The C programming language is used throughout the book with the purpose of showing how data structures and algorithms can be implemented from scratch without using libraries.

The first chapter of the book discusses the use of hashing in algorithm design. First, the author presents a problem of classifying snowflakes and shows a quadratic algorithm for solving the problem. After sending the solution to an online judge, it turns out that the algorithm is too slow, and a better algorithm that uses hashing is developed. This is the teaching style throughout the book: first an initial algorithm is created and then it is improved step by step.

The following topics in the book are recursion and dynamic programming. Recursion is discussed in the context of tree traversal, which is first implemented using a stack and then using a recursive function. After that, the author shows how recursion can be made more efficient through memoization, which leads to dynamic programming.



Author: Daniel Zingaro
Number of pages: 408
Publisher: No Starch Press (2020)

The next two chapters focus on finding shortest paths in graphs. The book first presents a chessboard problem where an optimal sequence of moves can be found using breadth-first search. Also a more advanced problem is discussed where a special two-state breadth-first search can be used. Dijkstra's algorithm is first used to find the shortest path in a graph, and then to also calculate the number of paths with minimum length. A quadratic implementation of Dijkstra's algorithm is presented; a better implementation using a heap is included in the appendix.

Binary search is introduced using a non-standard approach which is more relevant in competitive programming: the first problem is to find an optimal solution using a function that tests whether a solution is feasible. Only after that the traditional binary search problem of locating an element in a sorted array is mentioned. Also techniques for efficiently processing static range sum queries are discussed in connection with binary search.

After that, the author presents two data structures that are based on a binary tree and have some similarities: a binary heap and a segment tree. While many competitive programmers use segment trees that are perfect binary trees, in this book the tree is built in a different way which does not require the size of the array to be a power of two. Finally, the last chapter of the book discusses the union-find data structure with union-by-size and path-compression optimizations.

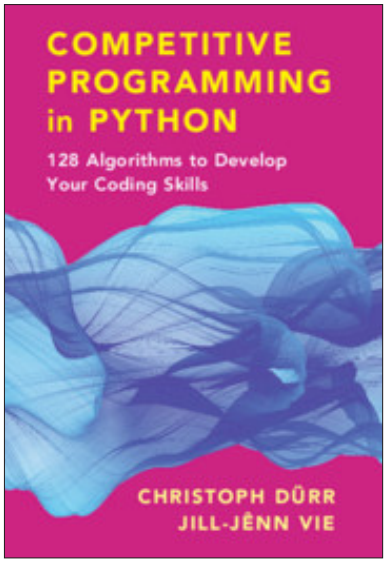
The strength of the book is that the process of discovering and improving algorithms is described in detail and various different approaches are analyzed. Compared to traditional textbooks, there are also interesting topics that are not usually covered, including the applications of binary search, calculating the number of shortest paths using Dijkstra's algorithm, and processing range queries using segment trees.

The programming style of the author is clearer than that of most competitive programmers. However, some of the examples are difficult to understand due to low-level data structure manipulation and the old-fashioned way to declare all variables at the

beginning of a function. While the purpose of using C has been to implement things from scratch, the qsort function is still used, probably because it happens to be in the C standard library.

Overall, the book is clearly written, the topics are well-chosen, and the book is a good introduction to some important competitive programming techniques. The main audience of the book are probably beginners who do not have much background in problem solving and are willing to use the C programming language.

Competitive Programming in Python: 128 Algorithms to Develop your Coding Skills



Authors: Christoph Dürr, Jill-Jênn Vie
Number of pages: 264
Publisher: Cambridge University Press (2020)

The purpose of the book is to teach skills that are needed for succeeding in programming contests and job interviews. The Python programming language is used in examples, which is not a typical choice in competitive programming. The reason for using Python is that it is an easy language. However, the book also mentions that it may be difficult to get Python solutions accepted, because Python is slower than languages like C++ and Java.

The book begins with a short introduction to Python and basic concepts like time complexity, data structures and algorithm design techniques. While Python provides many useful data structures in its standard library, the book contains an alternative binary heap implementation where keys can be changed. An interesting Python trick mentioned in the book is the lru_cache decorator that automatically adds memoization to a recursive function.

The next chapters present string processing algorithms, such as the KMP algorithm and Manacher's algorithm, and dynamic programming algorithms, such as calculating

the edit distance of two strings and the longest increasing subsequence. An interesting example is a game where you have a stack of numbers and on each move you can either remove one number or x numbers from the stack, where x is the topmost number.

After that, the book discusses the classical maximum sum subarray problem and presents the segment tree and Fenwick tree data structures that are often used for processing range queries in competitive programming. The book also describes a data structure called interval tree that can be used to report all intervals that contain a given point.

There are four chapters devoted to graph algorithms. Most standard techniques in competitive programming are described, such as the DFS and BFS algorithms, finding shortest paths, topological sorting, and constructing an Eulerian tour. The book also discusses more advanced problems like the Chinese postman problem and the stable marriage problem, that are not often seen in programming contests.

The following chapters present techniques for processing trees, such as efficiently finding lowest common ancestors and the diameter of a tree, and more dynamic programming topics, such as finding an optimal way to construct a sum of coins. Also some geometric algorithms are discussed, including a randomized algorithm for finding two closest points. This algorithm should be easier to implement than the traditional divide and conquer algorithm.

After that, the book discusses algorithms for processing rectangles, including two important competitive programming problems: finding the maximum area of an empty rectangle in a grid and calculating the area of a union of rectangles. Finally, the book goes through some mathematical algorithms, such as calculating binomial coefficients and the sieve of Eratosthenes, and search algorithms like the dancing links algorithm that is famous for solving sudokus.

The book covers many topics, and it can be seen as a mix of standard introductory textbook topics, competitive programming topics, and more advanced theoretical topics. However, since the number of topics is large and many of them are only briefly described, it can be difficult to really understand them by reading the book. Fortunately, the book has a good list of additional literature and references that can be used to find more information.

Python is a good language for representing algorithms, but as mentioned in the book, it is often not a good choice in a programming contest – if it is available at all (for example, at the moment, Python is not available at IOI). Since most competitive programming books use C++, this book is suitable for someone who wants to use Python instead for practicing before participating in serious contests, or just for preparing for job interviews.

A. Laaksonen works as a university lecturer at the Department of Computer Science of the University of Helsinki. He is one of the organizers of the Finnish Olympiad in Informatics and has written a book on competitive programming. He is also a developer of the CSES online judge.