Undergraduate Topics in Computer Science

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

More information about this series at http://www.springer.com/series/7592

Antti Laaksonen

Guide to Competitive Programming

Learning and Improving Algorithms Through Contests

Springer

Antti Laaksonen
Department of Computer Science
University of Helsinki
Helsinki
Finland

ISSN 1863-7310 ISSN 2197-1781 (electronic)
Undergraduate Topics in Computer Science
ISBN 978-3-319-72546-8 ISBN 978-3-319-72547-5 (eBook)
https://doi.org/10.1007/978-3-319-72547-5

Library of Congress Control Number: 2017960923

Printed on acid-free paper

This Springer imprint is published by the registered company is Springer International Publishing AG part of Springer Nature.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The purpose of this book is to give you a comprehensive introduction to modern competitive programming. It is assumed that you already know the basics of programming, but previous background in algorithm design or programming contests is not necessary. Since the book covers a wide range of topics of various difficulty, it suits both for beginners and more experienced readers.

Programming contests already have a quite long history. The *International Collegiate Programming Contest* for university students was started during the 1970s, and the first *International Olympiad in Informatics* for secondary school students was organized in 1989. Both competitions are now established events with a large number of participants from all around the world.

Today, competitive programming is more popular than ever. The Internet has played a significant role in this progress. There is now an active online community of competitive programmers, and many contests are organized every week. At the same time, the difficulty of contests is increasing. Techniques that only the very best participants mastered some years ago are now standard tools known by a large number of people.

Competitive programming has its roots in the scientific study of algorithms. However, while a computer scientist writes a proof to show that their algorithm works, a competitive programmer *implements* their algorithm and submits it to a contest system. Then, the algorithm is tested using a set of test cases, and if it passes all of them, it is accepted. This is an essential element in competitive programming, because it provides a way to *automatically* get strong evidence that an algorithm works. In fact, competitive programming has proved to be an excellent way to learn algorithms, because it encourages to design algorithms that really work, instead of sketching ideas that may work or not.

Another benefit of competitive programming is that contest problems require *thinking*. In particular, there are no spoilers in problem statements. This is actually a severe problem in many algorithms courses. You are given a nice problem to solve, but then the last sentence says, for example: "*Hint*: modify Dijkstra's algorithm to solve the problem." After reading this, there is not much thinking needed, because you already know how to solve the problem. This never happens in competitive programming. Instead, you have a full set of tools available, and you have to figure out *yourself* which of them to use.

Solving competitive programming problems also improves one's programming and debugging skills. Typically, a solution is awarded points only if it correctly solves all test cases, so a successful competitive programmer has to be able to implement programs that do not have bugs. This is a valuable skill in software engineering, and it is not a coincidence that IT companies are interested in people who have background in competitive programming.

It takes a long time to become a good competitive programmer, but it is also an opportunity to learn a lot. You can be sure that you will get a good general understanding of algorithms if you spend time reading the book, solving problems, and taking part in contests.

If you have any feedback, I would like to hear it! You can always send me a message to ahslaaks@cs.helsinki.fi.

I am very grateful to a large number of people who have sent me feedback on draft versions of this book. This feedback has greatly improved the quality of the book. I especially thank Mikko Ervasti, Janne Junnila, Janne Kokkala, Tuukka Korhonen, Patric Östergård, and Roope Salmi for giving detailed feedback on the manuscript. I also thank Simon Rees and Wayne Wheeler for excellent collaboration when publishing this book with Springer.

Helsinki, Finland Antti Laaksonen
October 2017

Contents