Competitive Programming in Python

Want to kill it at your job interview in the tech industry? Want to win that coding competition? Learn all the algorithmic techniques and programming skills you need from two experienced coaches, problem-setters, and judges for coding competitions. The authors highlight the versatility of each algorithm by considering a variety of problems and show how to implement algorithms in simple and efficient code. What to expect:

* Master 128 algorithms in Python.
* Discover the right way to tackle a problem and quickly implement a solution of low complexity.
* Understand classic problems like Dijkstra’s shortest path algorithm and Knuth–Morris–Pratt’s string matching algorithm, plus lesser-known data structures like Fenwick trees and Knuth’s dancing links.
* Develop a framework to tackle algorithmic problem solving, including: Definition, Complexity, Applications, Algorithm, Key Information, Implementation, Variants, In Practice, and Problems.
* Python code included in the book and on the companion website.

Christoph Dürr is a senior researcher at the French National Center for Scientific Research (CNRS), affiliated with the Sorbonne University in Paris. After a PhD in 1996 at Paris-Sud University, he worked for one year as a postdoctoral researcher at the International Computer Science Institute in Berkeley and one year in the School of Computer Science and Engineering in the Hebrew University of Jerusalem in Israel. He has worked in the fields of quantum computation, discrete tomography and algorithmic game theory, and his current research activity focuses on algorithms and optimisation. From 2007 to 2014, he taught a preparation course for programming contests at the engineering school École Polytechnique, and acts regularly as a problem setter, trainer, or competitor for various coding competitions. In addition, he loves carrot cake.

Jill-Jênn Vie is a research scientist at Inria in machine learning. He is an alumnus from École normale supérieure Paris-Saclay, where he founded the algorithmic club of Paris-Saclay (CAPS) and coached several teams for the International Collegiate Programming Contest (ICPC). He published a book in theoretical computer science to help students prepare for prestigious French competitive exams such as Grandes Écoles or agrégation, and directed a television show “Blame the Algorithm” about the algorithms that govern our lives. He is part of the advisory board of the French Computer Science Society (SIF), itself a member of the International Federation for Information Processing (IFIP).
Competitive Programming in Python

128 Algorithms to Develop Your Coding Skills

CHRISTOPH DÜRR
CNRS, Sorbonne University

JILL-JÊNN VIE
Inria

Translated by Greg Gibbons and Danièle Gibbons
# Contents

<table>
<thead>
<tr>
<th>Preface</th>
<th>page ix</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong> Introduction</td>
<td></td>
</tr>
<tr>
<td>1.1 Programming Competitions</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Python in a Few Words</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Input-Output</td>
<td>13</td>
</tr>
<tr>
<td>1.4 Complexity</td>
<td>17</td>
</tr>
<tr>
<td>1.5 Abstract Types and Essential Data Structures</td>
<td>20</td>
</tr>
<tr>
<td>1.6 Techniques</td>
<td>28</td>
</tr>
<tr>
<td>1.7 Advice</td>
<td>37</td>
</tr>
<tr>
<td>1.8 A Problem: ‘Frosting on the Cake’</td>
<td>39</td>
</tr>
<tr>
<td><strong>2</strong> Character Strings</td>
<td></td>
</tr>
<tr>
<td>2.1 Anagrams</td>
<td>42</td>
</tr>
<tr>
<td>2.2 T9—Text on 9 Keys</td>
<td>43</td>
</tr>
<tr>
<td>2.3 Spell Checking with a Lexicographic Tree</td>
<td>46</td>
</tr>
<tr>
<td>2.4 Searching for Patterns</td>
<td>48</td>
</tr>
<tr>
<td>2.5 Maximal Boundaries—Knuth–Morris–Pratt</td>
<td>49</td>
</tr>
<tr>
<td>2.6 Pattern Matching—Rabin–Karp</td>
<td>56</td>
</tr>
<tr>
<td>2.7 Longest Palindrome of a String—Manacher</td>
<td>59</td>
</tr>
<tr>
<td><strong>3</strong> Sequences</td>
<td></td>
</tr>
<tr>
<td>3.1 Shortest Path in a Grid</td>
<td>62</td>
</tr>
<tr>
<td>3.2 The Levenshtein Edit Distance</td>
<td>63</td>
</tr>
<tr>
<td>3.3 Longest Common Subsequence</td>
<td>65</td>
</tr>
<tr>
<td>3.4 Longest Increasing Subsequence</td>
<td>68</td>
</tr>
<tr>
<td>3.5 Winning Strategy in a Two-Player Game</td>
<td>70</td>
</tr>
<tr>
<td><strong>4</strong> Arrays</td>
<td></td>
</tr>
<tr>
<td>4.1 Merge of Sorted Lists</td>
<td>73</td>
</tr>
<tr>
<td>4.2 Sum Over a Range</td>
<td>74</td>
</tr>
<tr>
<td>4.3 Duplicates in a Range</td>
<td>74</td>
</tr>
<tr>
<td>4.4 Maximum Subarray Sum</td>
<td>75</td>
</tr>
</tbody>
</table>
### Contents

4.5 Query for the Minimum of a Range—Segment Tree 75  
4.6 Query the Sum over a Range—Fenwick Tree 77  
4.7 Windows with $k$ Distinct Elements 80

5 Intervals 82  
5.1 Interval Trees 82  
5.2 Union of Intervals 85  
5.3 The Interval Point Cover Problem 85

6 Graphs 88  
6.1 Encoding in Python 88  
6.2 Implicit Graphs 90  
6.3 Depth-First Search—DFS 91  
6.4 Breadth-First Search—BFS 93  
6.5 Connected Components 94  
6.6 Biconnected Components 97  
6.7 Topological Sort 102  
6.8 Strongly Connected Components 105  
6.9 2-Satisfiability 110

7 Cycles in Graphs 113  
7.1 Eulerian Tour 113  
7.2 The Chinese Postman Problem 116  
7.3 Cycles with Minimal Ratio of Weight to Length—Karp 117  
7.4 Cycles with Minimal Cost-to-Time Ratio 120  
7.5 Travelling Salesman 120  
7.6 Full Example: Menu Tour 121

8 Shortest Paths 124  
8.1 Composition Property 124  
8.2 Graphs with Weights 0 or 1 126  
8.3 Graphs with Non-negative Weights—Dijkstra 127  
8.4 Graphs with Arbitrary Weights—Bellman–Ford 130  
8.5 All Source–Destination paths—Floyd–Warshall 132  
8.6 Grid 133  
8.7 Variants 135

9 Matchings and Flows 138  
9.1 Maximum Bipartite Matching 139  
9.2 Maximal-Weight Perfect Matching—Kuhn–Munkres 145  
9.3 Planar Matching without Crossings 151  
9.4 Stable Marriages—Gale–Shapley 153
Contents

9.5 Maximum Flow by Ford–Fulkerson 155
9.6 Maximum Flow by Edmonds–Karp 158
9.7 Maximum Flow by Dinic 159
9.8 Minimum s – t Cut 162
9.9 s – t Minimum Cut for Planar Graphs 163
9.10 A Transport Problem 165
9.11 Reductions between Matchings and Flows 165
9.12 Width of a Partial Order—Dilworth 167

10 Trees 171
10.1 Huffman Coding 172
10.2 Lowest Common Ancestor 174
10.3 Longest Path in a Tree 178
10.4 Minimum Weight Spanning Tree—Kruskal 179

11 Sets 182
11.1 The Knapsack Problem 182
11.2 Making Change 184
11.3 Subset Sum 185
11.4 The k-sum Problem 187

12 Points and Polygons 189
12.1 Convex Hull 190
12.2 Measures of a Polygon 193
12.3 Closest Pair of Points 195
12.4 Simple Rectilinear Polygon 198

13 Rectangles 200
13.1 Forming Rectangles 200
13.2 Largest Square in a Grid 201
13.3 Largest Rectangle in a Histogram 202
13.4 Largest Rectangle in a Grid 204
13.5 Union of Rectangles 205
13.6 Union of Disjoint Rectangles 212

14 Numbers and Matrices 214
14.1 GCD 214
14.2 Bézout Coefficients 214
14.3 Binomial Coefficients 215
14.4 Fast Exponentiation 216
14.5 Prime Numbers 217
14.6 Evaluate an Arithmetical Expression 218
Contents

14.7 System of Linear Equations 221
14.8 Multiplication of a Matrix Sequence 225

15 Exhaustive Search 227
15.1 All Paths for a Laser 227
15.2 The Exact Cover Problem 231
15.3 Problems 237
15.4 Sudoku 238
15.5 Enumeration of Permutations 240
15.6 Le Compte est Bon 243

16 Conclusion 245
16.1 Combine Algorithms to Solve a Problem 245
16.2 For Further Reading 245
16.3 Rendez-vous on tryalgo.org 246

Debugging tool 247
References 248
Index 251
Preface

Algorithms play an important role in our society, solving numerous mathematical problems which appear in a broad spectrum of situations. To give a few examples, think of planning taxi routes given a set of reservations (see Section 9.12); assigning students to schools in a large urban school district, such as New York (see Section 9.4); or identifying a bottleneck in a transportation network (see Section 9.8). This is why job interviews in the IT (Information Technology) industry test candidates for their problem-solving skills. Many programming contests are organised by companies such as Google, Facebook and Microsoft to spot gifted candidates and then send them job offers. This book will help students to develop a culture of algorithms and data structures, so that they know how to apply them properly when faced with new mathematical problems.

Designing the right algorithm to solve a given problem is only half of the work; to complete the job, the algorithm needs to be implemented efficiently. This is why this book also emphasises implementation issues, and provides full source code for most of the algorithms presented. We have chosen Python for these implementations. What makes this language so enticing is that it allows a particularly clear and refined expression, illustrating the essential steps of the algorithm, without obscuring things behind burdensome notations describing data structures. Surprisingly, it is actually possible to re-read code written several months ago and even understand it!

We have collected here 128 algorithmic problems, indexed by theme rather than by technique. Many are classic, whereas certain are atypical. This work should prove itself useful when preparing to solve the wide variety of problems posed in programming contests such as ICPC, Google Code Jam, Facebook Hacker Cup, Proligen, France-ioi, etc. We hope that it could serve as a basis for an advanced course in programming and algorithms, where even certain candidates for the ‘agrégation de mathématiques option informatique’ (French competitive exam for the highest teacher’s certification) will find a few original developments. The website tryalgo.org, maintained by the authors, contains links to the code of this book, as well as to selected problems at various online contests. This allows readers to verify their freshly acquired skills.

This book would never have seen the light of day without the support of the authors’ life partners. Danke, Hương. Merci, 智子. The authors would also like to thank the students of the École polytechnique and the École normale supérieure of Paris-Saclay, whose practice, often nocturnal, generated a major portion of the
material of this text. Thanks to all those who proofread the manuscript, especially René Adad, Evripidis Bampis, Binh-Minh Bui-Xuan, Stéphane Henryot, Lê Thành Dũng Nguyễn, Alexandre Nolin and Antoine Pietri. Thanks to all those who improved the programs on GitHub: Louis Abraham, Lilian Besson, Ryan Lahfa, Olivier Marty, Samuel Tardieu and Xavier Carcelle. One of the authors would especially like to thank his past teacher at the Lycée Thiers, Monsieur Yves Lemaire, for having introduced him to the admirable gem of Section 2.5 on page 52.

We hope that the reader will pass many long hours tackling algorithmic problems that at first glance appear insurmountable, and in the end feel the profound joy when a solution, especially an elegant solution, suddenly becomes apparent.

Finally, we would like to thank Danièle and Greg Gibbons for their translation of this work, even of this very phrase.

Attention, it’s all systems go!