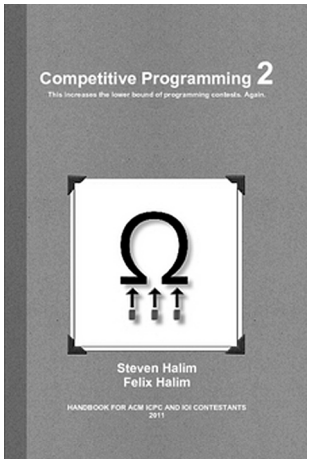
Olympiads in Informatics, 2012, Vol. 6, 226–228

REVIEWS, COMMENTS



Competitive Programming 2

Author: Steven HALIM, Felix HALIM
Publishing house: Lulu
Country: United Kingdom
Year of edition: 2011
Language: English
Number of pages: 262

There are many ways in which students can be prepared for programming contests and indeed training more generally in algorithms. Two of the more obvious areas are exposure to a wide variety of different algorithms and techniques, and practice by solving a good number of such problems. The former is important, not only at the initial levels where a student is introduced to the fundamental building blocks (e.g. dynamic programming or depth-first search) but at the progressive levels where a knowledge of more sophisticated methods gives a useful, if not essential, toolbox. Algorithmic programming contests have changed significantly since the 80s and, as this book mentions in passing, what were once the deciding problems are now basic requirements. The need for practice should be clear. Not only does it give the student the opportunity to confirm they understand the algorithms, it develops skills in finding appropriate algorithms, appreciating the different (test) cases an algorithm needs to solve, seeing ways in which multiple algorithms and data-structures can be combined and, from a purely contest perspective, helps in increasing a student's speed and accuracy.

Competitive Programming 2 is an excellent resource for both exposure and practice. It is the second edition (the first edition is just *Competitive Programming*) of this book, which grew originally from a similarly titled course that has been taught at the National University of Singapore since 2009. Its page count puts it at over 70% larger than the first edition. Readers of that edition, considering whether to also buy that edition can find a detailed comparison of the two editions (and the planned third edition) at the authors website [1]. The book is available in both A4 and A5 formats and an electronic version should be available by the time you read this. The A4 copy was review and worked well

as a format. This reviewer has heard from a colleague that the A5 edition is too small, although has not personally seen a copy at that size.

The format throughout the book is generally a brief overview of a given topic, followed by multiple algorithms, exercises, a long list of categorised problems to practise, and a brief conclusion. Starting with the categorised problems, this is a wonderful resource. Around 1300 problems appear in the book (almost exclusively from the University of Valladolid [2] online judge) and the bulk of these appear in these lists. These problems, which appear after an algorithm or group of related algorithms, typically have multiple sections each of which list a good number of problems and also highlights three *'must try'* problems. For example, the graph traversal problem list contains 56 problems (18 must try problems) over 6 sections: Just Graph Traversal; Flood Fill / Finding Connected Components; Topological Sort; Bipartite Graph Check; Finding Articulation Points / Bridges; Finding Strongly Connected Components. The ad-hoc section towards the front of the book contains 160 problems. Superb!

The description of algorithms through the book is a little more variable and varies from very detailed to incredibly terse. In general there is an okay amount of detail; enough for a motivated student to get the idea of the algorithm and start to play with it (or find details elsewhere) but not enough for the book to stand alone as a text for the weaker students. In fairness, the book does not set out to become another introductory algorithm textbook – the prerequisites suggests that it expects its audience to have passed a "basic data structures and algorithms course" – although contrary to this the very detailed sections tend to be the more basic material, for example in the dynamic programming explanation. When it goes to the other extreme it is often because of tendency to give the algorithm through code rather than explanation or just to drop in the name so that the algorithm is ticked-off.

The style used for each algorithm varies but the following are typical items that appear. There is often a motivating problem, discussion of specific tasks, examples of using the algorithm for other algorithmic tasks, exercises (as distinct from tasks, these are of the type more typical in a normal algorithm text; hints and solutions appear for many of these exercises), examples, discussions on complexity and short biographies of the people involved. Example code appears frequently in the text, and is available in both C++ and Java forms online. The coverage of the book is good, and focuses on what is used in programming contests, with the broad chapter headings covering pretty much what you would expect. The focus on contests works well and enables the book to offer focused advice on choosing an algorithm; weighing up the ease of implementation against the bounds specified in the problem or those generated by a specific sub-problem in a given solution.

The book, while applicable to all programming contests, is focused on the ACM ICPC and the IOI, and chooses its subject material accordingly. Where appropriate in the text the authors indicate where material is only relevant to one of these contests. The book is skewed towards the ICPC and the authors, whilst displaying a very broad knowledge of its problem set (the authors claim to have solved over 50% of the Valladolid problems), do not display similar knowledge of the IOI set (only 8 problems are discussed and only

one is pre-2008). This is not a problem in terms of presenting the material. In the terser algorithmic sections where no problems are discussed, and a good IOI task exists, it appears to be a choice of the authors since not only do good ICPC tasks exist but they are used by the authors in the corresponding problem lists. Furthermore, the bias means that the listed problems can be submitted to an online judging system which is invaluable. One does feel however, on occasion, as though the existing text has been hacked to include IOI references rather than having had it in mind from the beginning.

The majority of the book discusses algorithms (and data structures) though it does kick off with an introductory section on competitive programming. It makes a good introduction and one that might usefully be read by all students entering such contests, even those for whom most of the material in the rest of the book would be too advanced. It is debatable whether "Type Code Faster!" should have been the first tip – for the vast bulk of competitors this is not going to be an important factor – but the remaining tips are sensible and strong.

The book has two clear audiences; coaches and students. For the former the book provides a comprehensive set of categorised problems, and a far wider selection of problems that most are likely to have familiarity with themselves. The latter get the same list, as well as a good sampling of applicable algorithms. The book could be given "as is" to the more talented or motivated students, though for many the guidance of a coach along with exposure to other material would be advisable. It is not a substitute for an algorithms textbook but it is an excellent accompanying book and one which comes highly recommended.

References

1. https://sites.google.com/site/stevenhalim/
2. http://uva.onlinejudge.org

Richard Forster

Olympiads in Informatics, 2012, Vol. 6, 229–231

Teaching Programming for Secondary School: A Pedagogical Content Knowledge Based Approach

Author: Mara SAELI
Publishing house: Eindhoven University of Technology
Country: The Netherlands
Year of edition: 2012
Language: English
Number of pages: 164
ISBN: 978-90-386-3084-7

In this book (a PhD thesis) Mara Saeli reports her research about "teaching programming in secondary school with the aim of portraying the Pedagogical Content Knowledge of this subject", as is mentioned on the backside of it. The reason for a review in this journal is that it contains a meticulous description and analysis of why teaching of programming in secondary school is important, what should be taught, what problems and limitations students have while learning programming and how the teaching could be performed. All these issues have a great relevance for teachers who prepare students for Olympiads in informatics and for researchers who investigate this Olympiads. Problem solving, perhaps one of the most important aspects for Olympiad's participants turns out to be a very relevant part of the teaching and learning of programming.

The study of Saeli takes Pedagogical Content Knowledge (PCK) as a starting point. PCK is defined by Shulman (1986): it is the amalgam or combination of knowledge of a teacher of the content of a specific topic and the knowledge of the pedagogy of it: giving insights into educational matters relative to the learning and teaching of a specific topic. The PCK of a teacher starts during the education as a teacher in a particular field and grows by enhancing his theoretical background and teaching experiences. Teachers with good PCK are teachers who can transform their knowledge of a topic into something accessible for the learners.

In order to operationalize the concept of PCK for programming in secondary school, the author used the reformulation of PCK by Grossman (1990), namely the answers to the

four key questions: why to teach . . ., what to teach . . ., learning difficulties of . . ., and how to teach . . .? Saeli started to answer these questions by performing a broad literature review with respect to programming in secondary school. She found the following answers.

- Why: programming enhances students' problem solving skills and offers them a learning environment which includes aspects of different disciplines, gives opportunities to use modularity and transferability of the knowledge and/or skills, and to work with a multi-disciplinary subject.
- What: a list of concepts/aspects which a programming curriculum should include, for instance knowledge of data, instructions and syntax of a programming language, but also primitive expressions, means of combination and of abstraction, strategies, and programming sustainability which refers to the ability to create user friendly and attractive software that takes care of ethical and privacy issues.
- Learning difficulties: difficulty to instruct the machine about the solution of a problem, the tendency to converse with a computer as if it was human, the tendency the students maintain a local, limited point of view, failing to find a suitable solution.
- How: for instance offering a simple programming language so students can focus on the syntax, carefully choosing a diversity of problems in order to focus the students on algorithmic thinking.

In the literature these answers are not very well connected to each other. So, it was necessary to go into more detail with respect to each commonly taught topic. To this purpose Saeli organized six workshops in four different countries (Lithuania, Italy, Belgium and the Netherlands) with about five experienced informatics teachers in each workshop. Each workshop started with the question: What are the core concepts ("Big Ideas") of programming? Every participant answered this question individually. During the second part of the workshop the participants discussed one of more of the formulated Big Ideas using the following eight questions as a guideline:

1. What do you intend the students to learn about this Big Idea?
2. Why is it important for the students to know this Big Idea?
3. What else do you know about this Big Idea (and you don't intend students to know yet)?
4. What are the difficulties/limitations connected with the teaching of this Big Idea?
5. What do you think students need to know in order for them to learn this Big Idea?
6. Which factors influence your teaching of this Big Idea?
7. What are your teaching methods (any particular reasons for using these to engage with this Big Idea?
8. What are your specific ways of assessing students' understanding or confusion about this Big Idea?

Question 1 is about the *why*, question 2 about the *what*, questions 3, 4, 5, 8 about the *difficulties*, and questions 6, 7 about the *how*. This method to obtain data as a basis for portraying the PCK are based on CoRe (Content Representation), developed by Loughran *et al.* (2004) to give a narrative account providing an overview of how teachers approach the teaching of a specific topic in science in secondary school.

The results of the workshops turned out to be the following seven Big Ideas: control structures with focus on loops, data structures, arrays, problem solving skills, decomposition, parameters and algorithms. These results are in line with the research literature.

The next part of this study was to find out if and how the Dutch textbook support teachers with respect to the teaching of this seven Big Ideas. The reason for performing this and the following part of study is that the Dutch informatics teachers have almost all originally another disciplinary background and got a license for teaching informatics by a re-educating programme focused on informatics Content Knowledge, comparable to one year of a university Bachelor study in informatics. Using the developed portray of the PCK of programming as a referential framework the conclusion is, in most general terms, that the textbooks are helpful as far as the Content Knowledge is involved, but failed for the Pedagogical Knowledge.

The last part of the study was an investigation among the Dutch secondary informatics teachers: how do they assess their own PCK of programming? Here again the developed portray of the PCK was the basis for developing the online questionnaire that served as the research instrument. Saeli argued convincingly the validity of this instrument. About a quarter of all the almost 350 Dutch secondary informatics teachers filled in the questionnaire, but only 69 did so completely. (The reader should know that informatics is an elective course in the curriculum of the upper part of the Dutch secondary schools preparing to higher vocational and university education). The outcome of this study confirmed that the Dutch secondary informatics teachers have, in general, a poor PCK as a consequence of the fact that that they are re-educated teachers form other disciplines. As for their Pedagogical Knowledge, it turns out that this is sufficient, but not for extra-curricular topics – this is at least problematic for a fast developing subject as informatics. This conclusion is of course closely related to the fact that their Content Knowledge must be qualified as low. These findings are even more problematic because of the conclusion that the Dutch textbooks do not really support the teachers with respect to the Pedagogical Knowledge.

Saeli is maybe not the first researcher who tried to portray the PCK of programming, but certainly she is the first who has done so in a systematic way. The four chapters of her thesis which report her studies are in the meantime all published in peer reviewed scientific journals. Interesting for the readers of this journal is her recommendation to use tasks borrowed from Olympiads or designed analogue to Olympiad tasks in order to improve the teaching of programming in secondary schools.

References

Grossman, P.L. (1990). *The Making of a Teacher: Teacher Knowledge and Teacher Education*. New York, Teacher College Press, Columbia University press.

Loughran, J., Mulhall, P., Berry, A. (2004). In search of pedagogical content knowledge in science: developing ways of articulating and documenting professional practice. *Journal of Research in Science Teaching*, 41(4), 370–391.

Schulman, L. (1986). Those who understand: knowledge growth in teaching. *Educational Researcher*, 15, 4–14.

Bert Zwaneveld