Olympiads in Informatics, 2012, Vol. 6, 212–217

Theoretical Tasks on Algorithms; Two Small Examples

Willem van der VEGT
Dutch Olympiad in Informatics, Windesheim University for Applied Sciences
PO Box 10090, 8000 GB Zwolle, The Netherlands
e-mail: w.van.der.vegt@windesheim.nl

Abstract. In the Dutch Olympiad in Informatics theoretical subtasks are used to tests some of the skills needed for algorithmic design. The results were somewhat discouraging. An analysis for future use of theoretical tasks is performed.

Key words: informatics olympiad, programming competition, task design.

1. Introduction

The second round of the Dutch Olympiad in Informatics (Dutch, 2012) is very selective; we want to identify the top ten students, but we also intend to offer a fair competition in which contestants can show what they are able to do in a few hours. It is usually created with two or three background stories, each leading to several subtasks (van der Vegt, 2009).

The system of subtasks also makes it easier to slip in a more theoretical subtask. Once we asked contestants to create a sample input file that could produce a specified output for a given algorithm. We only asked for this file, not for a program or a description how to find it. At the IOI we call this an output only task; we had just an output only subtask. In another contest a game was played in which you could get stuck. One of the subtasks was to output the minimal number of moves that was at least possible, whatever the initial position of the game, and however badly the game might be played. We even predicted the output for a specific case and asked the contestants to explain this strange output in a few words. In this case, they had to deliver a plain text file. Of course this file was examined and graded manually.

In this year's second round one of the background stories had to do with set partitioning. There were eight subtasks; four of them were rather easy programming tasks, two were very hard programming tasks and the final two were tasks where contestants had to find a sample set of integers, fitting with the problem statement.

This paper addresses the choice for such theoretical tasks, founded in the aims of a programming contest like the informatics olympiad. We will also discuss the results of last year's theoretical questions, and formulate a few recommendations for the future use of this kind of subtasks.

2. The Aims of a Programming Contest

Programming contests are of course about programming. What sense does it make to ask more theoretical questions, even questions that can be solved without using a computer or a computer program?

The goals of the IOI are to bring together, challenge, and give recognition to young students from around the world who are the most talented in informatics (computer science), and to foster friendship among these students from diverse cultures (IOI website, 2012). The competition tasks are of algorithmic nature; however, the contestants have to show such basic IT skills as problem analysis, design of algorithms and data structures, programming and testing.

This year's Call for Tasks specified this: IOI tasks are typically focused on the design of efficient, correct algorithms. Input and output are to be kept as simple as possible (Call for Tasks, 2012).

One of the central words in these goals is algorithm. The IOI Syllabus (IOI-syllabus, 2009) uses a quote to elaborate this word: "Algorithms are fundamental to computer science and software engineering. The real-world performance of any software system depends only on two things: (1) the algorithms chosen and (2) the suitability and efficiency of the various layers of implementation. Good algorithm design is therefore crucial for the performance of all software systems. Moreover, the study of algorithms provides insight into the intrinsic nature of the problem as well as possible solution techniques independent of programming language, programming paradigm, computer hardware, or any other implementation aspect" (ACM, 2001).

These remarks immediately raise the next question. How to test the design of algorithms?

3. Black-Box Testing and Alternatives

One of the problems of testing algorithms in a programming contest is that it is usually done by black box testing. The organizer presents a set of test data and checks the output of the submitted program. Success at the test is an indication of a proper designed and implemented algorithm. Constraints on memory use and run time are means for testing efficiency of the program. But this way of testing demands a lot of your test data.

Forišek (2006) discusses the suitability of programming tasks for automated evaluation. Several interesting tasks are definitely not suitable for IOI-style automatic testing, like planar graph coloring or substring search. Some of the other evaluation methods that are suggested are pen-and-paper evaluation, supplying a proof and code review (white-box testing).

Pohl (2008) gives an overview of the disadvantages of black-box testing. In Bundeswettbewerb Informatik, the German Olympiad in Informatics, manual grading is performed. Contestants need to write a description of the solution approach and to choose their own examples for test input. For each task a set of grading criteria is developed; grading is a joint effort of a team of jury members.

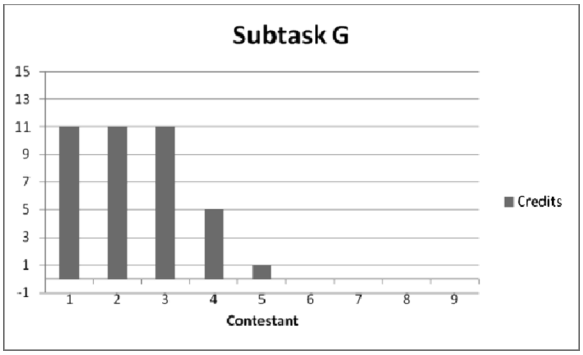

Fig. 1. Results for subtask “Strictly Inpartitionable”.

One of the aspects of designing algorithms is to be able to analyze the boundaries of a solution space. With the theoretical questions we use at the Dutch Olympiad of Informatics we want to check whether our contestants have a real understanding of the problem. We try to look into their thinking process, not by examining their algorithms, by one step earlier. We want to find out if they have a proper image of the problem their algorithms should have to deal with.

4. Two Small Problems

The task Fair Division (Dutch, 2012) used sets of different integer values. Most subtasks asked for a partition of such a set in such a way that the sum of the values of all subset were the same. All the values n_i were chosen with $0 < n_i < 1000$. For the theoretical subtasks the number of values is at most 30.

Subtask G introduces the term Strictly Inpartitionable. Contestants are asked to give an example of a set with different values, for which it is impossible to partition it in 2 or more subset with equal sums, in other words a strictly inpartitionable set. It should be a set with as many values as allowed (30 if possible) and the highest value should be minimal. Partial credits were allowed for solutions that did not meet these both conditions.

Out of 22 contestants, only 9 submitted a solution. And only 5 of them received any credits. 11 out of 15 was the maximum achieved.

What went wrong? Some of the contestants were not able to deliver solutions for all subtasks, so they did not enter any solution. Two contestants submitted a set with much larger numbers; they were rejected. Two others submitted a wrong solution. Two contestants submitted a solution with a set of far less than 30 values; they received a few points. All three contestants with 11 points entered a set with 30 different values, using the same way of thinking. They took the number 1, 2, 3..29 and the final number exceeded 435, the sum of the integers from 1 till 29. This way they were sure that the set was in no way partitionable; their 30th value was too large to fit in one of the subsets.

The solution the organizers had thought of had a much smaller highest value. We looked for the set in which the sum of all values was a prime number. For us it was

obvious that such a set is strictly inpartitionable. Since the sum of the integers from 1 till 30 is 465, we looked for the smallest prime number above 465. This number is 467. So if we remove 29 from the row and add 31, we have the optimal solution. One of the contestants must have been thinking in this direction, but he skipped 30 instead of 31, so his solution was partitionable in two subsets.

Subtask H introduced Easy Partitionable Sets. Contestants are asked to give an example of a set with different values, for which it is possible to partition it in m subsets with an equal sum, for all $2 \leqslant m \leqslant k$, maximizing the value of k. Partial credits were given for solutions with $k > 3$.

This is of course a very difficult task. Even checking the submissions is horrible and time consuming. We expected to get at least some solutions. But only three contestants submitted. One of the submissions was rejected, because the set was not partitionable in three subsets. The second contestant created the set {1, 119, 2, 118, 3, 117, 4, 116, 5, 115, 6, 114, 7, 113, 8, 112, 9, 111, 10, 110, 11, 119, 120}. It was easy to create 12 building blocks of subsets with a sum of 120. By combining these subsets, partitions in 3 and 4 subsets with equal sums could be made. A partition for $m = 5$ is not possible. This contestant has got 3 points for his solution out of a maximum of 35. The best result was for a young contestant with a background in the Mathematics Olympiad. He received 15 points submitting {1, 2, 3, 4, 5, 6, 7, 8, 9,10, 11, 12, 13, 14, 15}. His solution reached the k-value of 6.

There is simply not enough data to see if a mathematical problem analysis was performed by the contestants. If a set is m-partitionable, then m needs to be a divider of the sum of the set. If a set is easy partitionable for all values of m till a maximum of k, then k needs to be a common multiple of all integers for 2 to k. So it is a good idea to start with a set with as a sum the least common multiple of 2 to k. For $k = 8$ this least common multiple is 420. If you want to have a set with 30 values, the average value should be 14. We used this for our sample solution to get {13..27, 29..43} as an easy partitionable set for all m from 2 to 8.

Table 1
Best contestant solution for subtask H

2-partition	3-partition	4-partition	5-partition	6-partition
{1..5, 7..11}	{1..4, 6..9}	{1, 14, 15}	{1, 5, 6, 12}	{1..4, 10}
{6, 12..15}	{5, 10, 12, 13}	{2..4, 6..8}	{2..4, 7, 8}	{5, 15}
	{11, 14, 15}	{5, 12, 13}	{9, 15}	{6, 14}
		{9, 10, 11}	{10, 14}	{7, 13}
			{11, 13}	{8, 12}
				{9, 11}

5. Evaluation

With 5 and only 2 partial solutions in a field with 22 contestants, these two theoretical tasks did not deliver what they were developed for. In a discussion with contestants and fellow organizers, we found a few reasons.

1. These two subtasks were 2 out of 14 subtasks in a three hour contest. Only 6 contestants succeeded in submitting over 10 of these 14 subtasks.
2. These two subtasks were at the end of the task description. If contestants were not able to write a program for subtask F or even subtask E, they simply did not start looking at these two final subtasks.
3. Theoretical tasks are out of the comfort zone of the contestants. They are used to writing programs, rather than solving problems without using a computer program.
4. Contestants are not used to think about properties of numbers. No solutions used the notion of prime numbers. The least common multiple was not at hand in their mind.

The question remains whether these kind of questions can help to learn more about the ability of contestants to design proper algorithms. The constraints in subtask G were posed to guide the thinking of contestants in the direction of prime numbers. This idea failed. Some of the contestants however used another property of the natural numbers to reduce the maximum value within the set. With subtask H, we expected to see that contestants create a small working example and extend it.

The main problem we encountered is that asking for solution for theoretical questions is a very implicit way to find out about the skills on algorithms and design. Reducing the total number of subtasks and reordering the subtasks within the task description can improve the participation. Combining more theoretical questions with writing a program can enlarge the involvement with the subtask. And of course, focusing on the goals of the contest is still needed to design future tasks.

References

ACM/IEEE-CS Joint Curriculum Task Force (2001). *Computing Curricula 2001: Computer Science*, December. http://www.acm.org/sigcse/cc2001/

Call for Tasks (2012). `http://www.ioi2012.org/competition/call-for-tasks/`.

Dutch Olympiad in Informatics (2012). `http://www.informaticaolympiade.nl` (in Dutch only).

Forišek, M. (2006). On the suitability of programming tasks for automated evaluation. *Informatics in Education*, 5, 63–76.

IOI-syllabus (2009). `http://people.ksp.sk/~misof/ioi-syllabus/`.

IOI-website (2012). `http://www.ioinformatics.org`.

Pohl, W. (2008). Manual grading in an informatics contest. *Olympiads in Informatics*, 2, 122–130.

Van der Vegt, W. (2009). Using subtasks. *Olympiads in Informatics*, 3, 144–148.

W. van der Vegt is teacher's trainer in mathematics and computer science at Windesheim University for Applied Sciences in Zwolle, the Netherlands. He is one of the organizers of the Dutch Olympiad in Informatics and he joined the International Olympiad in Informatics since 1992. He was involved in the IOI-workshops on tasks in Dagstuhl (2006, 2010) and Enschede (2008). Currently he is one of the team leaders of the Netherlands at the IOI.