Olympiads in Informatics, 2007, Vol. 1, 1–8 1
© 2007 *Institute of Mathematics and Informatics, Vilnius*

Polish Olympiad in Informatics – 14 Years of Experience

Krzysztof DIKS, Marcin KUBICA, Krzysztof STENCEL
Institute of Informatics, Warsaw University
Banacha 2, 02-097 Warszawa, Poland
e-mail: {diks,kubica,stencel}@mimuw.edu.pl

Abstract. This paper presents the organization of the Polish Olympiad in Informatics, together with tasks preparation process and evaluation. It is a result of over 14 years of experience in organization of programming contests for high-school pupils. We believe, that although described procedures are rather widely know, their rigorous implementation is the key to organization of a successful programming contest.

Key words: ?, ?, ?.

1. Introduction

Polish Olympiad in Informatics (POI) originates from a smaller contest, called *Contest in Informatics*, and after evolving for a couple of years, it gained status of the national olympiad in 1993. It is addressed to high-school students, however middle-school pupils can also take part in it. The detailed organization rules can be found in (XIII Olimpiada Informatyczna, 2006) or on the web page of POI: `http://www.oi.edu.pl/`.

POI consists of three stages. In each stage, a number of tasks of algorithmic nature are presented to contestants. Solution of each task is either a computer program, or computed data. The supported programming languages are: C, C++ and Pascal.

The first stage is usually organized in October and November. It gathers over a thousand of contestants. Among them, about 40 contestants are from middle-schools. The contestants are presented five or six tasks, that should be solved at home within a month and sent for evaluation. About 360 contestants are qualified to the second stage of competition.

The second stage is organized in six regional centers, located at universities cooperating with POI and takes three days. The first day is a preparation day – the contestants have to solve one or two tasks during a three hours session, however the results are not counted in the competition. The objective of the preparation day is twofold: first, the contestants can get familiar with the environment, second, the organizers can verify that everything is ready. During the second and the third day, the contestants have to solve two or three tasks during five hours sessions. The solutions are collected from all the centers and evaluated. Preliminary results are usually known a couple of hours after the competition is finished.

The results are approved in two weeks, what gives time to process possible contestants complaints. About 70 contestants are qualified to the third stage of competition.

The final, third stage is organized in one place and takes five days. Traditionally, it is organized in Sopot, a small city at the Baltic shore. Similarly to the second stage, there are three competition days. The first day is also a preparation day, when the contestants have to solve one or two tasks in three hours. Each of the other two competition days the contestants have to solve three tasks in five hours. The preliminary results are known just after the competition. The fourth day is a leisure day for contestants, and for organizers it is reserved for processing possible complaints. On the last day, there is an awarding and closing ceremony.

POI loosely follows international tradition in medal allocation. There are three categories of medals: golden, silver and bronze. The number of medallists rather not exceeds half of the number of finalists, and the ratio between the number of golden, bronze and silver medals is close to 1:2:3, but it is not a rule.

Each year, POI requires preparation of approximately 17 tasks. Tasks preparation is a continuous process. Before each stage of the competition, 5 to 7 tasks for this stage are selected from the pool of about twelve tasks ready for the competition.

The next section of this paper presents the tasks preparation process, from a task idea to a moment when the task is ready to be used. In the following section the evaluation process is described.

2. Tasks Preparation

The main objective of the tasks preparation is to assure good quality of tasks. But what does it mean? What makes a good task? We should take into account the following aspects:

- Task formulation – it must be clear, comprehensive and not too long.
- Differentiation of contestants' skills – there should exist many ways of solving the task, of different difficulty; moreover, it should be possible to distinguish these solutions by testing.
- Thoroughness of analysis – task analysis should take into account wide spectrum of solutions, covering all ways of solving the task accessible to the contestants, and different programming languages and usage of STL (where necessary).
- Thoroughness of testing – tests should distinguish correct and incorrect solutions; they also should distinguish different classes of solutions, regardless of the programming language used to implement them (and possible usage of STL).
- Correctness – all example programs should obey input/output specifications and should produce correct outputs; if necessary, an output checker is also needed.

Describing the tasks preparation process, we will emphasize requirements needed to achieve the above objectives.

The tasks preparation process consists of the following phases: review of task ideas, formulation, analysis, verification, and calibration.

2.1. *Reviewing Task Ideas*

Initially we need just a task idea. It only has to define the algorithmic problem to be solved. When reviewing the idea, we should answer the following questions:

- Can the task be formulated in a short and comprehensive way? If it is too complicated or requires explanation of many terms, then it is not suitable for a competition.
- Is the task a 'handbook' one? If so, then it would rather test knowledge of a particular algorithm/technique, than creativity. Hence it is not appropriate.
- Is the task unique, to the best knowledge of the reviewer?
- Can it be solved in a polynomial time? If not, then it is rather not possible to evaluate it in a reasonable time. However, exceptions are possible.
- Are there many ways of solving the task, with different difficulties and different (time) complexities? If not, then probably the task is not appropriate for a contest, or it may not be possible to distinguish different classes of solutions.
- Can it be solved by a high-school student? There is no universal answer to this question. Our requirements are a little bit higher than those defined in (Verhoeff *et al.*, 2006). The expected knowledge is covered by most general handbooks on algorithms, e.g., (Cormen *et al.*, 1989) (skipping more advanced chapters) covers it all.

2.2. *Task Formulation*

In the task formulation, all elements missing in the task idea should be added. In particular: a short story can be added to make the task more attracting. The language should be simple. One should avoid complex sentences. All new notions should be introduced before they are used. Greek symbols should be avoided. If a coordinate system is needed, then the standard one should be used. Other detailed guidelines can be found in (Verhoeff *et al.*, 2006).

Input and output specifications must be precise – limits on the data sizes can be left undefined. Preferably, the output should be short, hard to guess (e.g., not a yes/no answer but rather some integer) and unequivocally determined by the input. However, this last requirement is not crucial. The task formulation should contain an example, preferably with a picture. The task should fit on one or two pages. Three pages is the absolute limit. The task formulation should be also accompanied with a short description (one or two paragraphs) of author's solutions – it will be taken into account during the analysis.

2.3. *Task Analysis*

Task analysis is the most time-consuming part of its preparation. The outcome of the analysis should consist of: a document summarizing the analysis, a number of programs and tests. Also, all missing elements in the task formulation (e.g., limits on the data sizes) should be defined.

The analysis document is an internal document, so it can be written in a professional language. The analysis document should discuss different solutions: the optimal solution (within contestants' scope), other possible solutions and a couple of incorrect solutions that could be expected. It should discuss all the solutions proposed by the author of the task, but by no mean should it be limited to these solutions. Of course not all incorrect solutions can be foreseen, but a good sample is valuable.

The solution descriptions for pupils are prepared post factum. However, if they are to be distributed during the competition, they should be prepared together with the analysis document.

All correct solutions should be implemented both in C/C++ and Pascal. Moreover, if application of STL is relevant, such solutions should be implemented in C++ using STL and in C (not using STL). The rationale is that we need to know the actual running time of these solutions. For other solutions, e.g., incorrect ones, just one implementation is enough, since they should produce incorrect outputs.

In case of batch (i.e., typical) tasks, the set of 10 to 20 tests should be prepared. The primary objective of tests is to distinguish correct and incorrect solutions. However they should also distinguish all the classes of correct solutions, that are of different difficulty. Tests should put stress on the asymptotic time-cost rather than absolute running time. As a rule of thumb, solutions up to twice slower then model solutions should score the full points. Moreover, the result of testing should not depend on the choice of the programming language, or usage of STL. Some version of IOI-50% rule can be applied – 30%–60% of points should be allocated to correctness tests. In other words, correct but not efficient solutions (however running in a reasonable time) should score 30%–60% of points. The rest of points should be granted for efficiency. Such a thoroughness of testing could be hard to achieve. The usual solution is to increase the data sizes. However, the amount of available RAM and expected testing time can limit it.

If necessary, tests can be grouped – a solution is granted points for a group of test only if it passes all the tests from the group. Grouping should be used when the correct result could be 'guessed' with high probability, or more than one test is needed to distinguish correct from incorrect solutions.

Tests can be prepared in form of files or a generating program can be provided. The latter option is especially useful for generating huge efficiency tests. If such a program uses random number generator, then it should also set the random seed, so that it always generates exactly the same set of tests. Tests should be accompanied with a program verifying their correctness. Such a program should verify all conditions stated in the task, what is sometimes quite not trivial.

If the task is an output-only one, then the set of tests should be prepared in a similar way, however we cannot control the running time. Contestants can even use different programs to solve different tests. Usually we cannot measure efficiency of contestants' solutions. However in this type of tasks, the running time is not so crucial, or the competition time is a sufficient limit. So, we can skip the requirement to implement all correct solutions in all supported programming languages.

If the task is an interactive one, then we have to provide modules that should be compiled with contestants programs, in all supported programming languages. There should be two versions of such modules: one provided during the competition and one for the evaluation. The first version should just allow testing contestants' solutions. The second one should implement a couple of 'strategies'. Different interacting strategies and different initial configurations correspond to tests. The module used for evaluation should also be secured against reverse-engineering attacks or misuse. It can be divided into a separate process and simple communication module that is compiled with contestants solution. Then, the separate process is protected by the operating system. Moreover it can be implemented in just one programming language. The communication can be done via standard input output. However it should contain verification/check-sum codes, to prevent contestant's code form interacting with such a process.

2.4. *Verification*

The main goal of the verification phase is to assure correctness. The two main ways to achieve it is thorough inspection of the analysis document and model solutions, and cross-checking the model solutions and tests. The inspection should cover: task formulation, analysis document, model solutions and programs for test generation and verification. An independent model (but not necessarily optimal) solution should be implemented. The result given by such a solution should agree with those produced by the model solutions. Also, a program verifying correctness of tests has to be implemented. All model and incorrect solutions should be evaluated on all the tests, and it should be possible to distinguish classes of solutions of interest.

2.5. *Calibration*

All other phases of tasks preparation can be done in advance. However it is not possible to define the actual time-limits without knowledge of the hardware used to evaluate solutions. And this is usually known just before the competition. Hence, the calibration of time-limits must be done as soon, as the hardware used for evaluation is installed.

3. Evaluation

In POI solutions to all tasks are graded automatically. A special software system has been created for this purpose. It has been evolving from 1992. First it was written for MS-DOS, then ported to Windows NT, thrown away and totally rewritten, and then finally ported to Linux when it got the current shape of a full-fledged web system known as SIO (Information System of Olympiad[1]). Contestants submit their solutions to the SIO system which grades them and provides results. In most cases the full publication of the results is delayed until the end of the competition. Only the result of the test case from the task description is publicized immediately.

[1]The suspected permutation of letters is phantom. The abbreviation comes from the Polish name of the system.

Most tasks require writing a program which is to read the input data and produce a result. Such a task will be called a *batch task*. A small set of test cases is prepared for each batch task. Each submitted solution is run for each test case. The SIO system measures time and kills the program if the real time of the run exceeds the time limit twice. The time limit does not concern the real running time but the system time plus user time of the process. We allow doubled time limit for safety, but only the process running time counts. If the process time exceeds the time limit, the solution will always get no points for the given test case. The SIO also controls the security. It will stops a solution, if it executes a forbidden system call (e.g., forks and network routines) or opens any file (solutions to batch tasks are reading from the standard input and are requested to send the result to the standard output).

If for a test case the solution does not exceed the time limit, terminates with the exit code 0 and provides a correct answer, the solution will get points for this test case. In all other cases, the solution will get no points for this test case.

There are two problems with such a grading procedure. POI is proud of finding good yet simple solutions to them. The first problem is the discontinuity of the function which maps running times to points. The second problem consists in outputs which are relatively frequent in the result space (e.g., the answer ‘NO’ in a task which requires a specific sequence of numbers as output).

The time-to-points function is flat from zero to the time limit (its value is the maximum number of points for the test case). Then, it instantly drops to zero and stays flat until infinity. Apparently, this function is not continuous in the point of the time limit. If a solution fluctuates over the time limit, its result will vary much in subsequent evaluations. In order to avoid it, we decided to make the time-to-point function continuous. In POI this function is flat from zero to the halved time limit. Then it linearly descends to zero in the point of time limit. It is now linear and if the running time of a solution fluctuates, the number of point will never change rapidly. The final required remark here is that time limits are set in such a way that the model solutions always terminate long before the time-to-point function starts its descend.

The problem of answers frequent in the result space has been eventually adopted by the IOI, but it has been invented for POI. The solution to this problem is as simple as the linear time-to-point function. Each test case consists of a number of test runs. Each test run is connected with specific input data. Thus, running each test case means running a solution program for each test run separately. A solution will get points for a test case only if it solves all test runs correctly. Let us consider an example task where the answer is a sequence of integers with some properties, but if such a sequence does not exist, the program must output one word ‘NO’. Of course we don’t want to award any points to a solution program which always prints ‘NO’. Thus, all test runs for which the correct answer is ‘NO’ are grouped in test cases together with test runs which do produce sequences of numbers. This way programs which solve nothing will get nothing.

Last but not least we should mention *errare humanum est*. Indeed, the grading procedure is configured and maintained by humans. Particularly, it concerns preparation of test data. Thus, before the decisions on the qualification to the next round or the medal allocation are made, contestants have access to the preliminary grading of their solution on all test cases. These days it is easy, since we have the Internet. All this information is available on contestants' accounts in the SIO system. The contestants can appeal (of course through SIO). Yet after all this appeals are concluded, the results are verified by those who are the most interested. The decisions on medals and qualifications are made using these strongly verified results.

4. Conclusions

Running an annual programming contest is a never-ending job. We have described the organization of the Polish Olympiad in Informatics, with focus on tasks preparation and evaluation. We hope, that it can be fruitful to organizers of other contests.

References

Cormen, T.H., C.E. Leiserson and R.L. Rivest (1989). *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company.

Kanarek, P., and A. Iwanicki (Eds.) (2006). *XIII Olimpiada Informatyczna*. Komitet Główny Olimpiady Informatycznej.

Verhoeff, T., G. Horváth, K. Diks and G. Cormack (2006). A proposal for an IOI syllabus. *Teaching Mathematics and Computer Science*, **IV**(I).

K. Diks (1956), PhD Hab. in computer science, associate professor at Warsaw University, director of Institute of Informatics of Warsaw University, chairman of Polish Olympiad in Informatics, member of IOI-IC since 2001, former chairmen of IOI'2005 in Nowy Sacz, CEOI'2004 in Rzeszów and BOI'2001 in Sopot, Poland. His research interests are: algorithms and data structures, parallel and distributed computing, and graph theory.

M. Kubica (1971), PhD in computer science, assistant professor at Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, Warsaw University, scientific secretary of Polish Olympiad in Informatics, former IOI-ISC member and former chairman of Scientific Committees of IOI'2005 in Nowy Sacz, CEOI'2004 in Rzeszów and BOI'2001 in Sopot, Poland. His research interests focus on combinatorial algorithms and computational biology.

K. Stencel (1971), PhD in computer science, at the moment works at the Faculty of Mathematics, Informatics and Mechanics of Warsaw University. His research interests are connected with non-relational databases. From 1995 he has been the chairman of the jury of Polish Olympiad in Informatics. He was also the chair of jury at CEOI'97, BOI'2001, CEOI'2004 and IOI'2005.