Informatics in Education, 2006, Vol. 5, No. 1, 113–124

Some Ways to Improve Olympiads in Informatics

Mārtiņš OPMANIS

Institute of Mathematics and Computer Science, University of Latvia
29 Raina Boulevard, Riga, LV-1459 Latvia
e-mail: martins.opmanis@mii.lu.lv

Received: March 2006

Abstract. The paper describes some possible ways how to improve Olympiads in Informatics. Tasks in Olympiads are small models of programming tasks in software industry and in the limited amount of competition time contestants need to complete several software production phases – coding, testing and debugging. Currently, only coding effort is adequately graded, but grading of other activities may be improved. Ways to involve contestants in overall testing process are investigated and ways to improve solution debugging process are described. Possible scoring schemas are discussed. In International Olympiads tasks with real numbers are quite rare. Possible reasons are investigated and a way how to return such tasks back to competition arena is suggested.

Key words: olympiads in informatics, testing, debugging, competition tasks, grading.

1. Introduction

International Olympiad in Informatics (IOI) together with International Olympiads in Mathematics, Physics, Chemistry and Biology is one of "big" worldwide olympiads for high school students. IOI is held since 1989 and is annual event. IOI'2005 in Nowy Sacz (Poland) (IOI, 2005) was attended by 272 contestants from 72 countries.

Following IOI format also previous level olympiads use a similar format – they are individual, in-present competitions with one or two competition rounds. Tasks are algorithmically oriented and solutions are graded automatically. For contestants from Latvia, to participate on IOI, successful participation on 4–5 previous OI levels with increasing difficulty level (school, region, national, selection, Baltic) is necessary.

There are several well-known regional Olympiads in Informatics: Baltic Olympiad in Informatics (BOI, 2004; BOI, 2005), Central European Olympiad in Informatics (CEOI, 2005), Balkan Olympiad in Informatics (Balkan, 2005).

In this paper by mentioning Olympiads in Informatics (OI) without more precise explanation any competition in the hierarchy of Olympiads, starting with lower level and ending with the top level – International OI (IOI) will be assumed.

2. Format of OI

2.1. *Basic Features of OI*

Olympiads in Informatics are high school student competitions which have the following basic features:

- **Individual.** Contestant does not get support and help from teammates and other persons as well as from such sources as Internet, books, etc. There is a lot of other competitions where several contestants work together as a team. In fact, teamwork is more close to real work in industry. However, due to historical reasons for OI this feature still remains unchanged and changes are not planned.
- **In-present**. Nearly every international OI now has its satellite online competition in which every interested person may participate and solve the same task set in the same time limits as registered participants do. If competition has serious formal impact (such as team selection for higher-level OI), such competitions are made "semi-online" by adding supervisors and a restriction that contestants are allowed to work only in a specified workplace (still remote from the main competition place).
- **One or two competition rounds, five hours each**. Almost all OI follow this standard. Each competition round is held in a separate day (and in this sense "competition round" and "competition day" may be used as synonyms), there may be a leisure day in-between. In Latvia, the first two OI levels (school and region) are one day competitions, but higher levels OI (Latvian, Baltic and International) are held as two round competitions. Traditionally, three tasks per round are given. It is assumed that the best students must be able to solve all three tasks.
- **Tasks are mainly algorithmic oriented.** The main goal set for students is finding and implementing a correct and efficient algorithm solving the given problem. Tasks including heuristics and some presence of "luckiness", as a rule, are avoided or at least including such tasks in problem set usually raises serious discussions. In the last years three kinds of tasks are accepted in the OI:
  1. Tasks where solution – a computer program must be submitted (if not mentioned otherwise, by "task" a task of this kind will be assumed),
  2. Tasks, where a set of input files is given and the corresponding output files must be submitted (this will be discussed briefly later in this paper),
  3. Reactive tasks, where a given library must be used (will not be discussed in this paper).
- **Solutions are tested automatically** on a previously prepared test set and correctness is determined by checking equivalence of results or checking by the grader. Feedback from grading system gives to contestant possibility to know wether the submitted solution compiles and if so, does it give correct answer when given test examples are used as input data.

2.2. *Competition Round*

One OI's competition round is performed in the following way:

- At the beginning, the contestant receives **task descriptions**, where the task together with **input** and **output** data formats is described. **Limits** of task data range and time limits for one test execution also are given. Task description also contains an **example** input and the corresponding output. Maximum number of points per task is **100**.
- Created solutions must be **submitted** to grading system where they are compiled and executed on simple test cases (usually the same as given in task description as examples). This process usually is called **pre-testing**. If output of the program is correct, the submission receives the status "**accepted**", otherwise it is rejected.
- **Last submitted** solution version with status "accepted" is considered as the **final version** of the solution.
- After completion of competition round, the final version is tested on the **full test set**. This process usually is called **final testing**. Every test in this set has some number of **points** which are awarded to contestant if this test passes **with correct result in the given time limit**. Otherwise solution receives no points for this test.

3. Testing and Debugging

In the last years automated testing becomes de-facto standard in OI. Despite the impossibility to prove correctness of programs using test runs only (Verhoeff, 2006) and exclusion of several classes of tasks (like theoretical essays and proofs), at the same time, automated testing radically increases comparison objectivity of submitted solutions. It is impossible to imagine smooth grading in international arena without such a procedure.

However, there still are some deficiencies:

- The author of the test set (same as etalon solution) is jury. It is possible, that test set may be in some sense "misbalanced", especially if the number of tests is too low (I'm not speaking about trivial problems). For example, on IOI'99 (IOI, 1999) problems "Traffic lights" and "A strip of land" had only one or two simple test cases (from ten), leading to low scores and giving no reasonable result distribution. As the result, the 50% rule was invented (in task descriptions additional limits were stated, fulfilling which 50% of points were guaranteed), which led to surprising results on IOI'05 (IOI, 2005) when several contestants had chosen the relatively safe way to get half of points by solving task in these "light" limits and not attempting the "hard" ones.

Technical mistakes also took place on previous IOI's. Disagreement between task description and some test case (due to changed formulation, variable limits or simply typo) is possible. As a rule, this is discovered by some contestant or delegation leader only after investigation of testing results. Such mistakes are corrected afterwards and all submitted solutions are re-tested on the new test set.

- From the "industrial" point of view final version of submitted task solution is expected to be the "release version" in terms of software industry. In OI a working version of a weaker algorithm is better than a perfect implementation containing one typo. Such programs may be called "one wrong symbol" programs. At 11th International Olympiad in Informatics contestant got 0 points for task "Flatten" program where replacing "N" to "M" in a "for" loop made the solution worth of 100 points (IOI, 1999). On the other hand, rarely a contestant may feel lucky, when solution is not penalized due to "gaps" in used test set. For example, on 4th Central European Olympiad in Informatics a contestant got 90% of score for task "Shooting Contest" despite the fact that he implemented a solution for square areas only instead of rectangular ones mentioned in the task description (CEOI, 1997).

After a competition round for each of tasks there are:

- one test set, made by jury;
- $N + 1$ solution (N contestant solutions and the (best possible) jury solution).

OI grading systems allow very fast testing of all N submitted solutions even if there are hundreds of them. To be more precise, by distributing computing power between several computers it is possible to test solutions during competition immediately after their submissions.

During preparing solution contestant together with preparing program itself is forced to test his solutions (assuming non-trivial tasks, where correctness can not be checked without serious testing). Currently, grading system allows only checking correctness of program only on simple example test cases. Program's ability to run on bigger input data may be checked only by creating such big test cases which leads to necessity write one more program for generating big test cases. The value of such "internal" or "local" testing allows for contestant himself either have a feeling that his solution is correct (if program solves all constructed test cases) or shows incapability to write such program (there is a test case which can not be solved by the written program).

Till now, only program writing effort is graded adequately. Self-testing effort is partially thrown away. For example, if contestant constructs such a test case which can't be solved by program, it is clear, that program is not correct, but there is no way how to check other contestant's solutions on this particular test case.

Test sets made by contestants (or grading of testing capabilities in the other form) may be included in grading process and total number of points could be calculated as a sum of points for the program and points for the test set.

4. One Test Set or Multi Test Sets?

If there would be allowed to submit also test cases prepared for "internal" testing the end of competition round, then for each task there would be also $N + 1$ test sets (one from jury).

Testing every solution on such a large amount of tests will lead to deeper testing of each particular solution. To mention it once again - technically it can be done in reasonable time limits (even if it will be couple of hours instead of seconds). Existing solution

submission system may be improved by adding the possibility to submit test sets and some checker must be written to validate tests.

4.1. *Positive Impact*

Positive features of such an approach are:

1. Testing is quite a serious part of software creation process. Efforts made in this direction also must be awarded. In OI practice there are examples when by creating good test examples contestant recovers his inability to write a correct solution. If a test set would be submitted without a working solution, it would be granted also for this. In software industry, programmers and testers are everyday opponents but we should not underestimate tester's side, because good test creation is not an easy job.
2. If contestant's solution is capable of solving $N + 1$ test set created by different authors (more than 250 on IOI), it is a more serious product than a program passing only one test set. With all respect to jury as task authors, in a particular problem, there may be some hidden traps which are not covered by jury test set, but can be discovered when tested on test sets of other contestants.

Let's assume that together with problem solution the contestant submits also test set – a set of tests, where every test case is supplied with amount of points which will be awarded if the program passes this particular test case (natural additional constraints are positive integer number of points for particular test case and the total sum equivalent to 100).

It is easy to imagine two formats of test case data:

- input file and the corresponding output file (appropriate for "unique correct output" tasks);
- input file and a program producing output file (for "non-unique correct output" tasks).

It is obvious that in the second case a wrong program leads to incorrect test set, while in the first case it is possible to construct correct test cases without a proper solution program. For example, this could be done by a correct, but inefficient solution. Submitting input and output files in non-unique correct output case fits in category "solution of open input task" and is not subject of analysis in this paper.

4.2. *Drawbacks*

Unfortunately, there are some:

1. Only "good" tests are worth of scoring. Moreover — we are looking for good test sets, not only particular tricky test cases. If points will be granted simply "for tests", then there is a potential danger to get a lot of simple tests or generated hard ones without any shadow of inspiration. It is hard to formulate criteria for a "good test set". However, some simple rules can be stated:
   - perfect solution must be given 100 points;

- partially correct solution must not be given 100 points;
- partially correct solution must be given non-zero amount of points (this is very slippery point, because “one wrong symbol” program mentioned above will get 0 on nearly all test sets and this is not fault of the test set);
- non-efficient (slow) solution must not be given 100 points;
- non-efficient, but correct solution must be given some non-zero amount of points (this rule corresponds to the current IOI 50% rule);
- wrong program must not be given significant amount of points (how to separate wrong programs from partially correct programs?).

2. What if some task is so hard, that all contestants from programmers revert to testers and submit only test sets? Would this be acceptable or not?
3. It is not obvious how points must be given for separate test cases. One possible approach is to execute all programs on all original tests (avoiding executing the same test more than once by excluding repeating tests during test merge process) and normalizing score to 100/maximum_possible_amount_of_points.

However, in this case, if all contestants submit too easy or too hard test sets, then the results can be seriously biased. Solution to this could be awarding a part of points (say, the famous 50%) according to tests prepared by jury and, the remaining part – according to tests prepared by contestants.

4.3. *Testing Tournaments*

Another possible way could be to look at the testing process as a great tournament, where each contestant program fights with every contestant’s (even with himself) test set. The case of three contestants is presented in Table 1.

In the given example (Table 1) contestant A got 300 points against 295 got by Jury, and Contestant C failed on some of his own tests. If tests are not submitted (Contestant B), then all participants get equivalent number of points. It can be either 0 (there were no test cases, on which to test the solution) or 100 (no test case is failed) – this will not

Table 1

Example of testing tournament for three contestants

		Tests submitted by:			
		Jury	Contestant A	Contestant B	Contestant C
Program submitted by:	Jury	**Passed all tests (100)**	Passed all tests (100)	Tests not submitted (0)	Passed partially (95)
	Contestant A	**Passed all tests (100)**	Passed all tests (100)	Tests not submitted (0)	Passed all tests (100)
	Contestant B	**Passed all tests (100)**	Passed all tests (100)	Tests not submitted (0)	Passed partially (55)
	Contestant C	**Passed partially (90)**	Passed partially (95)	Tests not submitted (0)	Passed partially (85)

Table 2

Score change for different grading schemas

	Points	
	Using existing grading schema	Using all submitted test sets
Jury	**100**	295/300 = 98.3
Contestant A	**100**	300/300 = 100
Contestant B	**100**	255/300 = 85
Contestant C	**90**	270/300 = 90

influence relative standings of contestants – only the absolute amount of received points will change.

In the actual OI only one ("Jury") column is taken into account.

After testing on all submitted test sets results may be essentially different. As seen in the given example (Table 2), contestants indistinguishable when using old grading schema (A and B) becomes distinguishible. Contestant, having lower rank under the old schema (C), now have higher rank. Embarrasing things can happen if jury's solution will not receive the full score (as seen in the example). Our expectation is that jury solution must pass all tests in the given time limits and contestants will not find counterexamples investigating jury source code. If there is no such solution which receives perfect score (as A in the example) this can raise question about solvability of the task as such.

Weakness of this "tournament" approach is that contestants are stimulated in submitting only tricky tests to "catch" opponents, and not balanced test sets for throughout analysis of programs.

4.4. *"Challenge" Approach*

Another version of tournament also is possible: **source code** of every contestant program which passed all jury tests is given to all contestants. Similar approach is used in the "Challenge" phase of TopCoder competition (TopCoder). The task now would be – find **one** test case which this particular ("perfect" from the viewpoint of jury!) program **does not pass** and receive some points for this case. In example given above, A and B programs would be investigated, and C can potentially raise his score by finding a test which "beats" B's program. At the end, all total scores must be normalized to 100 points and the contestant who got full score after jury's testing and did not find (or even did not try to find) any mistakes in opponent's programs may be overtaken by a diligent tester who was not perfect after jury's tests.

However, if such "challenge" phase would be included in official program of the competition, contestants may start "defend" their source code by making it unreadable. Such code is also known as "obfuscated code".

5. Testing on a Complete Test Set

For “one wrong symbol” programs, it is quite clear that in real life such mistakes are found and corrected quite easily, sometimes even in seconds. On OI there is no way how to manage this and nearly every year there are protests or begging for making exceptions in the rules to accept such solutions which, in fact, **are** really good.

One possible solution could be giving some amount of time for correction of mistakes when the results of full-range testing are announced (similar to Association for Computing Machinery (ACM) competitions). If this time is not very big (say, one hour for three tasks), there is no way to rewrite all from scratch, but is plenty of time for correcting small mistakes (and creating new ones). However, it may be hard to change existing timetables of international OI, where right after a competition round a dinner is planned and contestants can discuss their solutions so making impossible further return to solutions not breaking at the same time the idea of individual competition.

A more realistic approach could be permission to check solutions on the full jury test set. If it would show that results in all test cases are perfect, contestant would know in advance that his solution is perfect, but in case of total testing he can be sure that solution is at least “fairly good”. If the result after this “first try” is not perfect, there is time for error fixing (and making new errors). Of course, if the contestant works in a hurry and has no time to validate, then the old schema works – last submitted program which passes initial tests, qualifies for full grading and the obtained result will be final.

To avoid some “gambling” (“first try” gives some information about test set and solution is modified in such a way to pass only this particular test set) number of these “preliminary testings on a complete test set” must not be big. One, or maybe two, but no more. It will help on testing, but in contradistinction to ACM, solution can get points also if the task is not solved perfectly. To stimulate thinking before sending there can be involved a rule that the total amount of points is calculated as (points_in_first_version + points_in_final_version)/2. If there is only one submission, then these two numbers are equal. If we use this formula, a contestant who solves the task perfectly and makes only one submission, still has advantage. Of course, such counting does not exclude possibility to worsen the first result (say, 90 points after the first testing, 40 points after the final testing due to new errors leads to overall score 65).

There could be also other grading schemas: only final version result is taken into account and in case of equivalent results higher rank has the contestant who used less testings (like counting jumps in high jump competitions). In case of usage of tests made by other contestants, this “one version” approach is preferable to keeping records of several distinct program versions for each contestant.

Together with visible pros there are also serious contras. There will be much more serious responsibility on jury during preparation of the final test set – it simply must be PERFECT. If usually final test set becomes “visible” only after competition round during final testing, now possible errors will have impact during the first “big submissions”. Failing in some test case due to jury mistake can mislead contestant forcing him seek for nonexistent error so wasting time or stimulating making inappropriate changes and

submitting a worse version. Also server workload must be limited to avoid overloading in last competition round minutes due to lot of requests for first try full testings.

6. Rehabilitation of Real Numbers

During the last years in international OI only a few tasks were using real (or non-integer) numbers.

As a rule, tasks with non-integer numbers are not welcome on OI's and serious analysis of possible reasons is already done (Horvath and Verhoeff, 2003). One of the technical difficulties in tasks with real number output is the impossibility of simple checking correctness of a given answer on equivalence/non-equivalence basis. Simple transformation of a task with non-integer output to a task with integer output will be demonstrated below.

In tasks with non-integer output the expected precision of answer is defined by requiring for one of the following:

- some number of correct digits after the decimal point;
- some number of correct significant digits;
- limited allowed difference between contestant's and jury's results.

In the automated Latvian OI task testing system (OLIMPS) a different approach for tasks with real number answers is used.

Let's assume a usual OI task where some input file is given and some output file containing one real number must be produced. Task description includes one of the necessary conditions for answers, given above.

In the suggested approach, the original input will be appended by additional 100 real numbers – possible answers, one of which is "correct" (the answer obtained by jury). Task section is changed in such a way that instead of original "compute and output" must be stated "compute and decide which of given possible answers is closest to your result, where "closest" means that absolute value of difference between your result and this answer among all possible answers is minimum." And instead of a real number in original formulation now one **integer** – index of the answer must be written in file. Obviously, this simplifies testing of solutions – now the answer given by program must be equivalent to the correct answer. The number 100 was chosen arbitrary just to minimize the risk of successful guessing and avoiding essential additional computations. Possible answers can be given in sorted or random order – there is no obvious advantage of any of them.

Thus the original solution program must be extended by adding code like the following Fig. 1 and instead of output of `result` in original formulation, now `i_best` must be written as answer in output file. Such improvements can be done by an average OI contestant quite easy.

But is this "modified description" the same as the original one in the sense of precision and how this precision can be managed?

If, in the possible answer set, the gap between two given answers is 2ε (assuming the same gap between any two neighbor answers), then the contestant must compute his result with precision ε to get closer to the correct answer (see Fig. 2). This corresponds

```
{“result” at this point is computed real answer}
readln(input,possible_answer);
i_best := 1;
delta_best := abs (result - possible_answer);
for i:=2 to 100 do
 begin
  readln(input,possible_answer);
  delta := abs (result - possible_answer);
  if delta<delta_best
    then
      begin
       i_best := i;
       delta_best := delta;
      end;
 end;
```

Fig. 1. Example of additional code in Pascal.

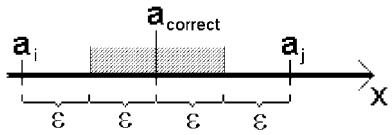

Fig. 2. Acceptance region. a_i, a_j, $a_{correct}$ – three of 100 given possible answers, $a_{correct}$ – correct answer.

directly with the third accuracy criterion given above and the value obtained by contestant must fit in the same region. The only difference is in case if correct value is minimum or maximum in the given set of possible answers. To avoid such situation, the correct answer must be strictly an inside value.

Even more – jury can test the accuracy of contestant's solution, giving similar tests with the same essential part and changing only possible answers section by increasing or decreasing gaps between values. It is a well-known practice (ACM NEERC, 2005) that the same task may be given asking for different precision of the result. In the described approach in test cases only ε must be changed.

Unfortunately, the described approach cannot be used in all tasks without a preliminary analysis. It cannot be used in tasks where the knowledge of answer values gives essentially new information. For example, if the original task asks for finding coordinates of the point common to all given triangles, the above-mentioned switching to new data format essentially decreases the level of task, because now you need just to check all possible answers without any other calculations which is not possible in the original task.

7. Conclusions

In the past decade, technical development of Olympiads in Informatics is marked by overall involving of automatic testing and systems allowing program submission together with preliminary testing. However, there are still ways how this development could be

continued. Involving contestants in testing, giving more freedom in debugging process, returning tasks with real numbers together with new kinds of tasks may be possible ways of further successful development. The described ideas should be investigated further to determine their usefulness at particular OI levels.

Acknowledgements

I would like to thank Prof. Kārlis Podnieks and Prof. Valentina Dagiene whose valuable comments helped me in writing of this paper.

References

ACM NEERC (2005). Tasks "Map Generator" and "Map Generator returns". North-Eastern European Regional ACM Programming Contest (Western subregion), Minsk, October 26, 2005. `http://www.fpmi.bsu.by/acmicpc/2005/tasks_acm_2005.zip` (accessed December 28, 2005).

BalkanOI (2005). *13th Balkan Olympiad in Informatics*. Rhodes Island, Greece, September 3–8, 2005. `http://www.adhoc.gr/boi/` (accessed February 3, 2006).

BOI (2004). *10th Baltic Olympiad in Informatics*. Ventspils, Latvia, April 21–25, 2004. `http://www.boi2004.lv` (accessed February 3, 2006).

BOI (2005). *11th Baltic Olympiad in Informatics*. Pasvalys, Lithuania, May 5–9, 2005. `http://ims.mii.lt/olimp/tin/boi2005` (accessed February 3, 2006).

CEOI (1997). Task "Shooting contest". *4th Central European Olympiad in Informatics*. Nowy Sącz, Poland, July 17–24, 1997. `http://ceoi.inf.elte.hu/probarch/97/SHO.HTM` (accessed February 3, 2006).

CEOI (2005). *12th Central European Olympiad in Informatics*. Sarospatak, Hungary, July 28–August 4, 2005. `http://ceoi.inf.elte.hu/ceoi2005/` (accessed February 3, 2006).

Horvath, G., and T. Verhoeff (2003). Numerical difficulties in pre-university informatics education and competitions. *Informatics in Education*, **2**(1), 21–38.

IOI International Olympiads in Informatics Competition Rules. `http://olympiads.win.tue.nl/ioi/rules/index.html` (accessed December 28, 2005).

IOI (1999). Task "Flatten". *11th International Olympiad in Informatics*. Antalya, Turkey, October 9–16, 1999. `http://www.ioi99.org.tr/tasks/flat` (accessed February 3, 2006).

IOI (2005). *17th International Olympiad in Informatics*. Nowy Sącz, Poland, August 18–25,2005. `http://www.ioi2005.pl` (accessed February 3, 2006).

OLIMPS Tasks "Dolāru maiņa" and "Slaloms", automated grading system "OLIMPS". `http://www.lio.lv/olimps` (accessed December 28, 2005, in Latvian).

TopCoder programming competition. `http://www.topcoder.com` (accessed February 3, 2006).

Verhoeff, T. (2006). The IOI is (not) a Science Olympiad. Paper presented at Workshop *Perspectives on Computer Science Competitions for (High School) students* in IBFI Schloss Dagstuhl, Germany, January 23, 2006. `http://www.bwinf.de/competition-workshop/RevisedPapers/14_Verhoeff_rev.pdf` (accessed February 7, 2006).

M. Opmanis is a researcher of the Institute of Mathematics and Computer Science of University of Latvia. His major interest is implementation of information technologies into education, using olympiads in informatics and other kinds of competitions in mathematics and computer science. The interests also include different aspects of programming paradigms and languages.

Galimi informatikos olimpiadų tobulinimo būdai

Mārtiņš OPMANIS

Straipsnyje aprašomi galimi informatikos olimpiadų tobulinimo būdai. Šiuo metu olimpiados užduotis rengia tik komisijos nariai. Tiriama galimybė į šį procesą įtraukti ir varžybų dalyvius. Aptariamos galimos vertinimo sistemų schemos. Aprašomi sprendimų derinimo proceso tobulinimo būdai. Informatikos olimpiadų užduotyse realieji skaičiai pasitaiko gana retai, – straipsnyje nagrinėjamos galimos to priežastys bei siūlomi tokių užduočių grąžinimo į varžybas būdai. Pateikiama ir keletas netradicinių užduočių pavyzdžių.