Olympiads in Informatics, 2018, Vol. 12, 159–166

DOI: 10.15388/ioi.2018.13

Grading Systems for Algorithmic Contests

Ágnes ERDŐSNÉ NÉMETH[1,2], László ZSAKÓ[3]

[1]*Batthyány High School, Nagykanizsa, Hungary*
[2]*Doctoral School, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary*
[3]*Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary*
e-mail: erdosne@blg.hu, zsako@caesar.elte.hu

Abstract. Whether you teach programming or you are a competitive programmer yourself, one of the main questions is how to check the correctness of solutions. There are many different types of automatic judge systems both online and offline; you have to choose the appropriate one for the situation. In this paper, we discuss the types of judges and the possible feedbacks given by them. We also give an overview of the methods used by some well-known online sites from a pedagogical point of view. The technical issues of different grading systems are omitted.

Keywords: teaching programming, grading on algorithmic contests, automatic grading systems, ACM ICPC, IOI.

1. Overview

There are many papers about the online judges from a technical viewpoint: how these systems can be set up, what the problems with measuring the running times are, how to make good test files, etc. (Skupas, 2010; Maggiolo, 2012; Mares, 2009). There are papers from the point of view of a jury: is the solution correct, is it readable or not, (Skupiene, 2010), how can the solution's quality be measured, what are the differences between an accepted submission and a provably correct solution, what are the requirements of a correct task description. There are questions partly answered about manual grading in informatics contests (Pohl, 2008). There are papers about specific grading systems, like USACO and UVa. (Kolstad, 2007; Revilla *et al*., 2008)

We think, all participants of the teaching and competing process (teachers, jury, contestants, students, companies) agree in the positive role of scientific, and especially informatics contests. According to Manev *et al.* (2009), the main reasons:

- It is more attractive for talented pupils when education lines up well with and is for the purposes of the participating in competitions.
- It is necessary to teach students to compete as early as possible.

In many countries there is a large gap between the knowledge and skills developed in the regular curriculum and the ones needed for algorithmic contests. So teaching

programming and also preparing for algorithmic contests is mostly an out-of-school activity. It needs more independent work from students and needs more specific attitudes from teachers.

A didactically and methodologically interesting question is how to check the solutions submitted by students to a given problem, while teaching programming to them; how to enable them to check their own solutions while practicing; and how to grade solutions in algorithmic contests. All three of these situations require different approaches.

2. Grading Systems Overview

There are many online and offline grading systems for evaluating the correctness of programs written by competitive programmers. These systems are very similar in the sense that they all use a set of test data files uploaded by task-setters, performing black-box testing. The programmers' code is tested automatically, by a code-checker – not by a human being – and contestants have to write their code accordingly. For each problem, there is a set of one or more input files and a set of corresponding correct output files or a specific judge tool which checks the output of the uploaded program. Each input file is according to strict specifications described in the problem statement. The program is run on each of those input files. In most cases, the output generated by the student's program must match the correct output exactly, to be accepted. The solution, in order to be evaluated as correct, needs to produce correct output, and run within specified time and memory constraints.

The test data should check

- The complexity of the algorithm used.
- The behaviour of the code on border cases.
- The memory consumption.
- The running time efficiency (constant) of the implementation.

The difference between the systems is the method of ranking, the grouping of test data files, and the feedback given.

In this paper, we discuss the different methods of automatic judging, the differences in the feedback, the question of using individual or grouped test cases, and the methods adopted by well-known online practice sites and online competitions. We concentrate on algorithmic tasks only, and we do not discuss the specialities of interactive and output only tasks. We compare different methods from a pedagogical viewpoint and concentrate on didactical aspects: which systems can be used for different age groups and in different phases of learning, practicing and competing.

3. Types of Grading

There are four different types of grading models, each named after the best-known contest or situation it is used in.

3.1. *ACM-Style Grading*

On ACM ICPC contests, only the perfect and optimally efficient solution is worth any points. When uploading a solution, two cases are possible:

- Accepted: your program is perfect, it runs within the time limit, doesn't exceed the memory limit and for every input it prints a correct output.
- Not accepted: something went wrong, the judging system gives back the first error code.

Usually the correct solution is worth one point.

In many ACM ICPC tournaments, the number of teams reaching a given score decreases exponentially in terms of the score.

Possibilities for differentiation between teams with the same number of correct solutions:

- Defining the difficulty of each task: X_i points for the correct solution of task i (instead of one point for all tasks).
- Assigning a solution time to every task: measuring the elapsed time for every task solved by a team or person. The solution times are summed to give a time penalty. In this case, competitors must recognize which tasks they can solve quickly (easy tasks) and which ones they can solve slowly (difficult tasks). The optimal strategy is to solve easy tasks first, and leave more time-consuming tasks to the end. For example, in the case of a 10-minute and a 50-minute task, starting with the easier one, the solution time is 10+60 minutes, while starting with the more difficult one, the solution time is 50+60 minutes.
- Sometimes there is a penalty for wrong submissions too: e.g. +20 minutes or $-X_i$ points for every incorrect solution submitted. In this case, the contestants are advised to test their solution offline at first.

This type of grading is acceptable only for older and well-practiced contestants (especially university students) who are:

- Strong enough to endure failure.
- Have enough routine to see what went wrong.
- Well-trained enough to be able to repair and to rethink their solutions again and again.

Apart from the ACM ICPC contest, the same grading is used in CodeChef Cook-off contests, CodeForces rounds and CSAcademy (with full feedback).

3.2. *IOI-Style Grading*

The test files are grouped at IOI. Test cases awaiting a solution with the same asymptotic complexity (e.g. O(n), O(n*log n), $O(n^2)$, $O(n^3)$, $O(2^n)$) or logical complexity (e.g. problem restricted to an easier subproblem) are grouped together and worth a predetermined amount of partial points. The judge gives the partial points only if all the tests in the

group are passed, so only if the solution with the expected efficiency is perfect (or the solution solves the subproblem covered by the group of test cases).

The problem with the idea is that it contradicts the modern pedagogical principle of appreciating all the performance, rewarding all positive achievements. A fairness problem may arise as to whether a perfect $O(2^n)$ solution is better than an O(n) solution that does not test a single special case and hence receives 0 points. But it is acceptable for such an international competition where the competitors are well-prepared and have advanced knowledge.

Apart from IOI, CodeChef Lunchtime uses this method for contests too.

3.3. *National Olympiads*

The modern pedagogical principle says that all performance and all success should be rewarded. The individually evaluated separate test cases give this sense of achievement to all contestants.

There must be targeted test cases prepared for:

- Default values described in a task description.
- Extreme values of the problem domain and range.
- Different methods.
- Typical errors.
- Large quantities of data.

The number of points achievable by different solutions – according to the abovementioned metrics – has to be decided in advance. The distribution of test cases has to be designed in such a way, that a good balance of different types of test cases is kept, and hence the judge assigns the desired scores to different partial solutions.

When practicing, this is the best method if the online judge gives back a very detailed, exact feedback for all test cases.

The national competitions, partly university and high school exams, the practice sites of online judges – including USACO, COCI, UVa and HackerEarth – also use this method.

3.4. *Offline Judging*

When somebody begins to learn programming, it is important to develop the right coding style and formatting; and all the small successes should be evaluated and appreciated. Beginners need clear, accurate, detailed (usually verbal) feedback about their work.

Checking the code on a computer is a wide-ranging activity for all competitors. Writing code on paper is crucial in sitting exams without computers.

When somebody wants to hack another person's solution, or wants to learn new methods, they must be able to read and understand other people's code.

When there is a penalty for wrong uploads, the importance of offline judging is crucial – everyone has to learn to read code and to create test data. Everybody has to test their solution for one or two given sets of test data and must learn to make their own test cases for different approaches.

All judges perform black-box testing only, but we think that in teaching programming and preparing for contests, white-box testing must also be used.

4. Test Data

The quality of the task description and the test cases is crucial for the quality of a contest or an online practice site. The evaluation summary presented in online judge systems should be easy to read and to understand.

Some problems can be easily solved using heuristic algorithms, some can be solved on a subset of test cases by generally incorrect algorithms, still receiving very good evaluation scores (Forišek, 2006), so it is very important, that test cases used by the judge have been carefully designed and checked as widely as they can be. Wide testing is limited by available resources and confidentiality (for live contests).

The speciality of CodeForces Educational Rounds is that the results that are obtained by the end of a round are preliminary, and after contest, there is a twenty-four-hour-period of open hacks, when every visitor may try to hack any complete solution to a problem from the round, and all successful hacks are added to the official test set and every solution is re-tested on the now extended test set. This method makes the test cases better after the contest, and it widens the group of possible test makers. This is good practice and it may be worthwhile to adapt it for other contests.

5. Feedbacks

In all automatic judging system after writing a program on the contestant's machine, the source code is uploaded to the judging server. The source code is compiled and run on the server. The automatic judge tests it with some inputs and outputs (string comparison) or with a specific judge tool (for more complicated problems). After the process, the server gives back simple or detailed feedback.

The simple feedback is: "accepted" or "not accepted" (in that case it gave back the first error message).

The detailed feedback given back is one the following. We show the notation used by almost all sites (HackerEarth, SPOJ, TopCoder, CSAcademy…), the one used by USACO, and the one used by CodeChef, respectively, in brackets for each possible feedback. Sometimes feedback is given for all the test cases, sometimes just till the first error.

- Accepted (AC, *, ): the program is correct. It produced the right answer within the time and memory bounds.

- Wrong Answer (WA, x, ): correct solution not reached for the inputs. This may include empty or missing output as well.
- Compile Error (CE, c, ): the compiler could not compile the uploaded program. Warning messages are ignored. Usually the compiler output messages are reported on the screen.
- Runtime Error (RE, !, ): the uploaded program failed during execution (segmentation fault, floating point exception…). Sometimes the exact cause is not reported to the user, sometimes it is specified: SIGSEGV: segmentation fault, SIGABRT: fatal error, SIGXFSZ: output is too large, NZEC: non-zero exit code, SIGFPE: floating point error.
- Time Limit Exceeded (TL–TLE, t, ): the program did not terminate within the time limit. This error does not give information about whether the program would have reached the correct solution or not.
- Memory Limit Exceeded (ML–MLE, !, -): the uploaded program tried to use more memory than the judge allows. This is hard to separate from Runtime Errors for technical reasons, hence some judges do not report this.

Some systems also give other verdicts (UVa):

- Output Limit Exceeded (OL–OLE): The program tried to write too much information. This usually occurs if it goes into an infinite loop.
- Submission Error (SE): The submission is not successful. This is due to some error during the submission process, or data corruption.
- Presentation Error (PE): The program outputs are correct, but outputs are not presented in the correct way. However, usually the judging systems ignore extra white spaces, like '\n','\t'.
- Restricted Function (RF): The uploaded program is trying to use a function which may be harmful to the system.
- In Queue (QU): The judge is busy and cannot attend the submission. It will be judged as soon as possible.
- Cannot Be Judged (CJ): The judge doesn't have test input and outputs for the selected problem.

6. Conclusions

Our opinion is that partial scoring and detailed feedback is a must at the beginning of the learning process of programming. White-box testing with verbal feedback about the coding style is an optimal case for future work.

While practicing, detailed feedback is also a must. But instead of a simple "Wrong Answer" feedback, the detailed error signals are very useful. For example, if the task is about finding the shortest way in a graph, the type of error would be textually in the feedback, like:

- Wrong path length.
- The given path does not have minimal length.

- An invalid vertex is on the given path.
- One vertex turns up several times on the given path.
- The end of the given path is not the end of the expected path.
- The given sequence of vertices is not a path in the graph.

If the error is noticeable, then the judge should give the solution back with the cause of the error.

Our opinion – supported by our practice with high school and university students too – is, that feedback of tests case by case is a must on exams and on national Olympiads level as well.

When students are well-practiced on international level – like IOI and ACM – is the first level at which test cases in groups are acceptable. These students' knowledge can be measured well through the carefully selected tasks with adequate points worth, well grouped test cases.

From pedagogical viewpoint the ACM-style evaluation, differentiating based on elapsed time is good for easy, have-to-solve tasks only, not for making contests for high school students.

References

CodeChef, not-for-profit educational initiative by Directi. `https://www.codechef.com`
CodeForces online task archive and contest site. `http://codeforces.com`
COCI Croatian Open Competition in Informatics. `http://hsin.hr/coci/`
Cormack, G. (2006). Random factors in IOI 2005 test case scoring. *Informatics in Education*, 5(1), 5–14.
CSAcademy educational platform. `https://csacademy.com/`
Forišek, M. (2006). On the suitability of programming tasks for automated evaluation. *Informatics in Education*, 5(1), 63–75.
HackerEarth: Be a better programmer! `https://www.hackerearth.com/`
Horváth, Gy. (2014). A programozási versenyek szerepe az oktatásban. *INFOÉRA Konferencia 2014.*
Kolstad, R., Piele, D. (2007). USA Computing Olympiad (USACO). *Olympiads in Informatics*, 1, 105–111.
Manev, K., Sredkov, M., Bogdanov, T. (2009). Grading systems for competitions in programming. *Proceedings of the XXXVIII. Spring Conference of the Union of Bulgarian Mathematicians, 2009.*
Mares, M. (2009). Moe–Design of a Modular Grading System. *Olympiads in Informatics*, 3, 60–66.
MESTER online task archive and judge system. `https://mester.inf.elte.hu`
Maggiolo, S., Mascellani, G. (2012). Introducing CMS: A Contest Management System. *Olympiads in Informatics*, 6, 86–99.
Pohl, W. (2008). Manual Grading in an Informatics Contest. *Olympiads in Informatics*, 2, 122–130.
Revilla M.A., Manzoor, S., Liu R. (2008). Competitive learning in informatics: the UVa on-line judge experience. *Olympiads in Informatics*, 2, 131–148.
Skupas, B. (2010). Feedback Improvement in Automatic Program Evaluation Systems. *Informatics in Education*, 9(2), 229–237.
Skupiene, J. (2010). Improving the Evaluation Model for the Lithuanian Informatics Olympiads. *Informatics in Education*, 9(1), 141–158.
SPOJ Sphere Online Judge. `http://www.spoj.com/`
TopCoder Algorithms&Analytics. `https://www.topcoder.com`
USACO USA Computing Olympiad open contest and training pages. `http://usaco.org/`
UVa Online Judge. `https://uva.onlinejudge.org`
Verhoeff T.(2008). Programming task Packages: Peak exchange Format, *Olympiads in Informatics*, 2, 192–207.

Á. Erdősné Németh teaches mathematics and informatics at Batthyány Lajos High School in Nagykanizsa, Hungary. A lot of her students are in the final rounds of the national programming competitions, some on CEOI and IOI. She is a PhD student in the Doctoral School of Faculty of Informatics, *Eötvös Loránd* University in Hungary. Her current research interest is teaching computer science for talented pupils in primary and secondary school.

L. Zsakó is a professor at Department of Media & Educational Informatics, Faculty of Informatics, Eötvös Loránd University in Hungary. Since 1990 he has been involved in organizing of programming competitions in Hungary, including the CEOI. He has been a deputy leader for the Hungarian team at IOI since 1989. His research interest includes teaching algorithms and data structures; didactics of informatics; methodology of programming in education; teaching programming languages; talent management. He has authored more than 68 vocational and textbooks, some 200 technical papers and conference presentations.