Programming Contests as Complementary Activities in University Programming Courses

Julián Alarte[a], Carlos Galindo[b] and Josep Silva[c]
VRAIN, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain
{jualal, cargaji, jsilva}@upv.es

Keywords: Programming contest, Out-of-class Activity, Computer Science Subjects.

Abstract: Programming contests are events whose history goes back 50 years, soon after the appearance of computers in universities, and are traditionally associated with undergraduate computer science students. However, contests tend to be independent events, non-academic, extraneous to computer science subjects such as Data Structures or Algorithmics. For this reason, these subjects rarely use programming contests or the kind of problems posed in them, either in or outside the classroom. This work documents the implementation of a programming contest in a pilot program in the lab sessions of Data Structures & Algorithms, including the full process, proposed improvements, and a small statistical analysis of the contest's results.

1 INTRODUCTION

Programming contests (or programming competitions) are events in which a set of problems are presented to participants, to be solved efficiently with knowledge from algorithmics, data structure, mathematics, and statistics. Historically, these contests started in the 1970s, with the first editions of the ICPC[1] (International Collegiate Programming Competition), organized by ACM, an association of computer scientists. Since then, there are multiple organizations and a great number of websites dedicated to hosting and supporting these contests (Wasik et al., 2018).

These competitions are often oriented towards computer science students (at the undergraduate and masters levels), as they use many of the theoretical concepts of algorithmics and data structures that are studied at such levels (Combéfis and Wautelet, 2014; Combéfis et al., 2016; Moreno Cadavid and Pineda Corcho, 2018). There are also contests for high school students, such as the IOI[2] (International Olympiad in Informatics), organized by UNESCO since 1989.

[a] https://orcid.org/0000-0000-0000-0000
[b] https://orcid.org/0000-0002-3569-6218
[c] https://orcid.org/0000-0001-5096-0008
[1] https://icpc.global/
[2] https://ioinformatics.org/

1.1 Types of Programming Contests

We can classify programming contests in two categories, depending on how they assign points to user submissions:

Binary A program can be accepted if it passes all tests established by the contests' judges or rejected if it does not. On top of that, time and memory limits can be imposed, to avoid brute force solutions when a more efficient solution exists. Participants are rewarded by being the first to solve any given problem, and penalized by each rejected submission and by the time they take to solve a problem. This system is the one used by ICPC and associated competitions, such as SWERC[3] and other local contests, such as Ada Byron[4], a country-wide contest in Spain.

Optimization The program is accepted if the output can be parsed, but then the judge system scores the solution based on parameters like the quality of the solution, how close it is to the optimum strategy, the runtime, etc. Participants are rewarded based on the score, but the time taken to solve the problem and the number of previous attempts are not taken into account. HashCode, an annual competition organized by Google between 2014 and 2022, used this scoring system.

[3] https://swerc.eu/
[4] https://ada-byron.es/

Both kinds of contest can be useful; but, in general, in binary contests the goal is to solve each problem *as soon as possible* (as long as it passes the time and memory constraints), whereas optimization contests reward solving each problem *as best as possible*.

Given these two different approaches to problem-setting and scoring, for the programming contest of this study, we chose an optimization format, as the contest will take place outside the classroom and a binary format would give advantage to students that have more free time to submit solutions at the start of the competition.

2 PILOT STUDY: LAB PROGRAMMING CONTEST

The Bachelor's Degree in Informatics Engineering from the *Universitat Politècnica de València* contains in its second year a subject titled Data Structures and Algorithms, in which students learn about various data structures and some basic algorithms.

In the lab sessions of this subject we have run a pilot program, with a separate programming contest in each lab group. Because it is a pilot program, it lacks (a) specialized sessions in the lab schedule and (b) scoring the subject's final mark. Thus, the contest was explained during a lab session, but participants worked on it on their own time, and participation was not mandatory. To motivate students to participate, we offered the following incentives: (1) an automated test for the code they were developing as part of their lab activities (because some lab activities were part of the competition's exercises), and (2) an award for the winners, both as public recognition (a certificate) and additional material to extend their knowledge of the programming language under study[5].

3 IMPLEMENTING THE CONTEST

In this section, we describe the steps taken to prepare and execute the competition, making suggestions for future editions.

3.1 Preparing the Contest

Because the contest is not included officially in the subject's curriculum, the contest's set of problems was prepared around extending the lab activities, without any effect to the theory sessions. The lab sessions of this subject consist of writing programs using data structures and algorithms efficiently, to solve common problems such as efficiently indexing a library or handling a queue of documents to be printed to minimize wait time.

The problem set for the contest consists of exercises already present in the lab sessions, with an additional requirement to optimize their implementation. In this way, students that followed the activities day-to-day could participate in the contest with no further effort. However, students that wanted to invest more time in the contest could consider the efficiency of their solutions, to try to obtain better results. Although one of the goals of the subject is for students to produce efficient solutions to problems, this is not graded in exams, so students tend to gloss over it. In this case, the programming contest makes students pay attention to something that is not evaluated as part of the subject, and pushes them to consider the cost of the solutions they write.

Last but not least, the resources to start the competition were the course management software used for the subject (Sakai[6], in our case), so that students can submit solutions as if it were a homework assignment, and an announcement establishing the rules of the competition.

3.2 Running the Contest

Even though this event uses computers and every step of the way can be automatized (as we will see later), this contest was evaluated through a more manual process. Every day, we downloaded the last submission from each participant and we ran a judge program that checks for common errors and times the executions of the solutions provided. This judge generates tables ranking the participants according to how many test cases they were able to solve, and, when tied, by relative efficiency. All this information is shown to all participants through public web pages[7] that were updated daily. This update was mostly automatic (except for downloading the files, starting the judge and then publishing the web page file produced by the judge).

The most time-consuming part of the process was preparing personalized feedback for each student. The goal of this was to motivate each student to im-

[5] Available at https://mist.dsic.upv.es/teaching/csedu2024/finalists.pdf.

[6] https://www.sakailms.org/

[7] An example of these web pages can be found at https://mist.dsic.upv.es/teaching/csedu2024/example-leaderboard.html and https://mist.dsic.upv.es/teaching/csedu2024/example-table.html.

Table 1: Example runtime for student-submitted solutions.

Student	Test 0	Test 1	Test 2
A	1.2 s	5.3 s	10 s
B	1.5 s	4.8 s	11 s
C	4.8 s	4.9 s	10 s
Minima (u_i)	1.2 s	4.8 s	10 s

prove their solution, explaining the more complex errors that students encountered, which given the subject's level, they were not expected to be able to interpret correctly.

As is common in programming contests, each round of results was publicly available immediately, so that participants could see prompt feedback on their efforts and compare their current position and that of their competitors, but the last round of results was kept hidden, and only shown at the end of the contest.

3.2.1 Problem Evaluation Formula

The formula used to compute the aforementioned relative efficiency is the following:

$$t_r = \sum_{i=0}^{N} \frac{t_i - u_i}{u_i}$$

where N is the number of test cases, t_i is the absolute time that the student's program took to the i-th test case, and u_i is the time that the fastest student-submitted solution took to solve i.

Table 1 shows an example competition with three test cases (0, 1, 2) and three students (A, B, and C). Applying the formula would obtain the following t_r for each student:

$$t_r^A = \frac{1.2 - 1.2}{1.2} + \frac{5.3 - 4.8}{4.8} + \frac{10 - 10}{10} = 10.4\%$$

$$t_r^B = \frac{1.5 - 1.2}{1.2} + \frac{4.8 - 4.8}{4.8} + \frac{11 - 10}{10} = 35.0\%$$

$$t_r^C = \frac{4.8 - 1.2}{1.2} + \frac{4.9 - 4.8}{4.8} + \frac{10 - 10}{10} = 302.1\%$$

We can see that the obvious winner is A, who achieved the best result on Tests 0 and 2, and obtained a relative slowdown of 10% on Test 1. On the other side, C is strongly penalized due to their solution to Test 0 taking 4 times as much as A's. This formula was designed to award students that get close to the best solution found, while penalizing those that stray too far from the best solution in each test case. Additionally, because it is a relative measurement, we can compare test cases with disparate runtimes (e.g., Test 2 takes 10 times as long as Test 0).

Table 2: Average marks in theory and lab exams grouped by contest participation.

Part.?	Theory	Lab	Count
Yes	7.67 ± 0.66	8.75 ± 0.67	18
No	6.51 ± 0.75	6.72 ± 1.27	24
All	7.03 ± 0.53	7.80 ± 0.76	42

3.3 Results and Awards

In every competition and contest, one of the most important events is the presentation of results and awards. In programming contests, this event is even more important, as organizers typically explain the problems that the participants have faced, and show what the optimal solution would look like.

We took advantage of this moment to analyse common mistakes made by students and show what the optimized programs looked like. We also gave students general tips to write efficient code with the data structures we were using.

Before presenting these results, we had to analyse the winner's submissions, to check that they complied with the rules of the contest (as there were some conditions that could not be checked automatically by the judge), and a general reading of all the solutions sent, to give group feedback.

4 STATISTICAL RESULTS

This pilot program was implemented in two lab groups of the subject, during the last three weeks of the first half of the semester (March 2023). It encompassed 42 students, of which 18 made at least one submission (43%). Given the low effort required to participate (upload three files with exercises that were completed on previous lab sessions), we expected a higher participation rate.

With respect to results, 50% of participants managed to solve all 18 test cases, and from the other half, two thirds solved more than 80% of the test cases. If we compare the average marks obtained in the exams for the lab sessions, we obtain Table 2, which also shows error margins with 95% confidence. The average lab mark for students who participated is above 2 points higher[8] than those who did not take part. This result suggests a link between better results and participation in the contest. However, we cannot imply causation one way or the other, as doing well on the contest requires doing well on the lab sessions, and vice-versa.

[8] In a system where 0 is the worst qualification and 10 the best.

On the other hand, if we pay attention to the marks obtained in the theoretical part of the exam, marks are overall lower than the lab exam's, but we can still see a difference of 1.2 points. This margin is almost half compared to the one in the lab exam, which suggests a small causal effect, given that the gap is widened in the area the contest was supposed to push students to do better.

We also analysed the marks obtained against the number of test cases solved, but we did not find a statistically significant correlation.

Other studies have found a small change in absolute grade and grade distribution, but, most importantly, an improvement in self-reported metrics such as perceived difficulty of the subject, familiarity, proactivity in class and effort dedicated (Bandeira et al., 2019). Furthermore, participation in programming contests is seen as productive and career-building (Raman et al., 2018).

5 RECOMMENDATIONS FOR FUTURE EDITIONS

Preparing and managing a programming contest can be a daunting task, but it does not have to be.

The first obstacle is designing and selecting the problem set, and the test cases to determine whether a solution is valid or not. Problems can be sourced from exercise sets from the subject itself, more complicated versions of those exercises, or new problems altogether; according to the level of complexity desired and how much time we want the students to dedicate to it.

Then, the type of contest and scoring system must be established. This includes automating as much of the submission and scoring process as possible. When organizing a binary contest, software like DOMjudge[9] can be very useful, as it is freely available and allows the organizers to setup a full competition (Kinkhorst, 2014), including judging submissions automatically as they are received and generating a live ranking. For binary contests with partial scores (DOMjudge only gives pass/fail results), Contest Management System (CMS)[10] is a viable alternative. There are many other judge software suites (Wasik et al., 2018), as any large enough competition adapts or creates a software judge to fit their needs. Both DOMjudge and CMS may not be flexible enough, as they judge based on outputs for a given input, and cannot evaluate behaviour like reading and writing files, so we recommend, if possible, adapting the judge to the contest's pedagogical requirements (Bowring, 2008). Regardless, a fully automated judge is most desirable, because it gives feedback to participants instantly, letting them make submissions whenever suits them best, and allowing them to issue fixes to incorrect submissions.

Lastly, there must be some incentive for students to participate, such as including its result in the grading of the subject or by giving out prizes for the winner(s).

It is not necessary to generate personalized feedback for each participant, but giving feedback and hints regarding problems that are specially difficult can be motivating for participants who have not yet comprehended it. In the end, problems that are unreachable to all generate a lack of interest in the event.

The contest must also be monitored for the entire duration, as unexpected failures can appear in test cases, for which updated test cases should be promptly loaded to the judge system and a notification be given to participants.

6 CONCLUSIONS

This work describes the implementation of a programming contest in a second-year subject of the Bachelor's Degree in Informatics Engineering. It details the implementation of a pilot study contest and gives recommendations for organizing future events. The pilot study included 18 participants across two lab groups. The results show that participants achieved much better results in later exams than students who did not take part, which points to a possible use of contests and competitions as reinforcement or additional activities outside the classroom. However, the results are not conclusive enough due to the small sample size. It is our intention to repeat this experiment in larger groups.

REFERENCES

Bandeira, I. N., Machado, T. V., Dullens, V. F., and Canedo, E. D. (2019). Competitive programming: A teaching methodology analysis applied to first-year programming classes. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–8.

Bowring, J. F. (2008). A new paradigm for programming competitions. *SIGCSE Bull.*, 40(1):87–91.

Combéfis, S., Beresnevičius, G., and Dagienė, V. (2016). Learning programming through games and contests: Overview, characterisation and discussion. *Olympiads in Informatics*, 10:39–60.

[9]https://www.domjudge.org/

[10]https://cms-dev.github.io/

Combéfis, S. and Wautelet, J. (2014). Programming trainings and informatics teaching through online contests. *Olympiads in Informatics*, 8:21–34.

Kinkhorst, T. (2014). Domjudge at amrita.

Moreno Cadavid, J. and Pineda Corcho, A. F. (2018). Competitive programming and gamification as strategy to engage students in computer science courses. *Revista Espacios*, 39:11–23.

Raman, R., Vachharajani, H., and Achuthan, K. (2018). Students motivation for adopting programming contests: Innovation-diffusion perspective. *Educ Inf Technol*, 23:1919—-1932.

Wasik, S., Antczak, M., Badura, J., Laskowski, A., and Sternal, T. (2018). A survey on online judge systems and their applications. *ACM Computing Surveys*, 51:1–34.