BULETINUL INSTITUTULUI POLITEHNIC DIN IAŞI
Publicat de
Universitatea Tehnică „Gheorghe Asachi” din Iaşi
Volumul 64 (68), Numărul 1, 2018
Secția
ELECTROTEHNICĂ. ENERGETICĂ. ELECTRONICĂ

AN EVALUATION SYSTEM FOR CONTESTS AND CLASSROOMS

BY

ADRIAN ALEXANDRESCU*

"Gheorghe Asachi" Technical University of Iaşi
Faculty of Automatic Control and Computer Engineering

Received: February 19, 2018
Accepted for publication: March 23, 2018

Abstract. Algorithms and programming represent the foundation needed by every computer science student. During faculty, there are many exams which imply solving problems and implementing algorithms, and there are also various contests that require programming skills. This paper takes a look at what is required to manage a programming contest and at existing solutions for determining the problem-solving skills of students. Afterwards, a novel distributed system is proposed for evaluating the quality of problem solutions in contest and classroom environments. The system components offer a high degree of extensibility while also providing sufficient choices for evaluating the capacity of a student to develop algorithms in a contest/classroom setting.

Key words: algorithm evaluation; programming contest; evaluation system; e-learning.

2010 Mathematics Subject Classification: 68N01, 68Q99.

1. Introduction

In computer science, probably the most important asset is the ability to understand a problem, to figure out a solution to that problem and to write a computer program to solve it. In more abstract terms, solving a computer science problem consists of implementing an algorithm and running that algorithm on input data in order to obtain as output the solution to that problem. This ability is tested in programming competitions, in which students compete

*Corresponding author: *e-mail*: avalexandrescu@gmail.com

to solve one or more problem in a given time.

There are several programming competitions but the two most important are the International Olympiad in Informatics (IOI) ("International Olympiad", 2017) for secondary school students and the ACM International Collegiate Programming Contest (ACM-ICPC) ("The ACM-ICPC", 2017) for university students. Other contests that span over multiple days and involve algorithmic puzzles include Google Code Jam ("Google Code Jam", 2017), Facebook Hacker Cup ("Facebook Hacker Cup", 2017) or Amazon TechO(n) Challenge ("Amazon TechO(n)", 2017).

Obtaining good results at programming contests requires a lot of practice. In order for the students to prepare for these contests, there are various websites that have large problem sets and that offer the possibility of evaluating the quality of a solution for a given problem. There are situations (*e.g.*, ACM-ICPC) when teams of students compete in contests and are guided by a coach. Usually, the preparation consists of discussions between the coach and the students regarding specific topics and solving problems related to those techniques and methods.

The research presented in this paper focuses on presenting a system that can be used to improve the student's algorithmic and programming abilities by offering an environment in which the student can solve programming problems from different categories, follow tutorials and explanations regarding various programming techniques, participate in contests, and propose problems for other contests in an easy-to-use interface.

The proposed solution takes the best features of existing somewhat similar solutions, combines them to offer a larger problem set and flexibility in the scoring and ranking system, and also provides important practical applications such as the author sandbox and the use of the system in a classroom setting.

2. Problem Statement

2.1. *Programming Contests*

A programming contest can be seen from two points of view: the organizer and the participant. The goal of any contest is to determine a ranking between the participants. This is also the case for programming contests in which users try solve a set of problems in a limited amount of time. Usually, the source code for each problem is uploaded online where it is evaluated and scored. This job is performed by online judges, i.e., online applications that handle all the contest aspects.

From an organizer's perspective, there are several steps that have to be taken:

1. Choose an online judge,
2. Select the problem set,
3. Specify the contest conditions (*e.g.*, start- and end-times, scoring and ranking method),
4. Start the contest,

5. Show the final ranking.

Choosing an online judge depends on the characteristics of the contest. There are several good choices but each of them have their pros and cons, which are discussed in the next sections of this paper.

When selecting the problems there are two options: new vs. existing problems. The disadvantage of creating new problems is that several people have to write the problems, write test cases, and make sure that the statement does not leave room for interpretation and that the test cases cover all the limit situations. On the other hand, using existing problems has the advantage of having tried and tested problems, but the severe disadvantage that solutions can be found online. The latter can be overcome by restricting access only to the programming language documentation during the contest (this is not always possible).

Usually, each contest lasts two, three, four or five hours in which participants must solve five, six, nine or eleven problems, respectively, depending on the problem difficulty. The scoring for each problem and the final ranking calculation depends on the contest conditions. This aspect is further discussed in Section **3** of this paper. In most cases, the winner is the contestant who solved quickest the most problems.

2.2. *Online Judges*

An online judge is the means of organizing a programming contest in which solutions are automatically evaluated. It has several roles: managing users (contestants and administrators), managing problems, managing contests (selected problem sets and contest conditions), managing compilation and execution, and computing the contest/user ranking.

For each problem, the minimal requirements are the problem name and statement, the input and corresponding output datasets, and the solution limitations (*e.g.*, time and memory constraints). Optionally, other relevant information that can be associated with a problem is difficulty, categories (e.g., string manipulation, greedy, dynamic programming), solution source code in various programming languages, and solution description.

When a contestant uploads the source code for the solution of a problem, the online judge uses the appropriate programming language to compile and execute the solution in a sandbox environment. All of the online judges support the C and C++ programming languages, but some of them also offer support for Java, Pascal, Python and other languages. The solution is tested using the input datasets associated to each problem and the obtained output for each dataset is compared to the correct output. A feedback is usually sent to the contestant depending on the source code being complied successfully, if there were no execution errors and, for each input dataset, if the output was correct.

There are restrictions, which are imposed by the sandbox, to what the uploaded source code can do and access. The program must be written in a single file, it must not access files and directories other than the input files, it must not open network connections and it must not perform actions that disturb

the judging.

2.3. *System Characteristics*

The main characteristics of an algorithm evaluation / contest management system must be:

1. Contest creation with new/existing problems,
2. A large enough existing problem set with source code and solution description, organized by problem difficulty and type/category,
3. Solution evaluation in a sandboxed environment,
4. Loosely coupled and extensible system components,
5. Multiple scoring and ranking methods with the possibility of seamlessly adding new ones,
6. Tutorials for beginners (for each problem type, different examples with problems and also mini-contests),
7. Allowing authors to easily propose problems and communicate with each other,
8. Possibility of using the system in a classroom setting, for testing the student's ability to implement a solution to a problem in a specific programming language.

3. Analysis of Existing Online Judges

There are several existing online judges but the most highly used in the programming contests community are Codeforces ("Codeforces", 2017), Infoarena ("Infoarena", 2017), Timus ("Timus Online Judge", 2017), TJU ("TJU ACM-ICPC", 2017), UVa ("UVa Online Judge", 2017), VJudge ("Virtual Judge", 2017) and CMS ("CMS", 2017). Table 1 presents a comparison of the aforementioned online judges based on various criteria that is relevant to the development of the proposed system. The last online judge is an open source Contest Management System which has to be installed locally and does not come with any existing contests or problems, so some of the criteria do not apply and are marked with N/A.

All of the online judges support at least the C, C++, Java and Pascal programming languages, two of them support also Python and the rest support other slightly less used languages.

The focus of the current research is on allowing the creation of custom contests with, optionally, custom problems. Therefore, in Table 1, the different aspects that are more relevant to the propose system are marked with bold. An important aspect is the problem set size because all of the judges allow the submission of solutions to any of the existing problems at any time. VJudge has a large problem set size because it acts as an aggregator to multiple online judges. Basically, when one creates a contest, that person can choose any problems from a significant list of other online judges (including Codeforces,

Timus, TJU and UVa). When a solution is submitted, VJudge forwards that solution to the corresponding judge and then obtains the result. This approach, is highly effective because very few judges offer the input and output tests, but it has the disadvantage of depending on the availability of the other online judges.

Table 1
Comparison of Seven Online Judges Based on Various Criteria (Values in Bold are Relevant to the Proposed System)

	Codeforces	**Infoarena**	**Timus**	**TJU**	**UVa**	**VJudge**	**CMS**
# supported languages	15	4	11	4	5	5	7
Periodical contests	Yes (~3/week)	Limited access	Yes (~3/year)	Limited access	Yes (~10/year)	No	N/A
Problem set size	~3,500	~1,900	~1,100	~3,100	>5,000	**>>10,000**	N/A
Custom contests	Requires approval	**Yes**	No	**Yes**	No	**Yes**	**Yes**
Custom problems	Requires approval	Not anymore	No	No	No	No	**Yes**
Access to problem source code	**Yes (after contest)**	**Partial**	No	No	No	No	N/A
Access to problem input/output	**Yes (after contest)**	**Partial**	No	No	No	No	N/A
Solution description	**Partial**	**Partial**	No	No	No	No	N/A
Categorized problems	**Yes**	No	**Yes**	No	No	No	No
Allows questions	Yes	Yes (forum)	Yes	No	No	Yes	Yes
Scoring system	Adaptive	Time, Points	Time	Time	~Time	Time	Point, Tokens
Special features	**Solution hacking, API**	**Tutorials, Problem difficulty (partial)**	**Problem difficulty**	**Contest password**	**uDebug**	**Contest Aggregator, Contest password**	**Runs only on own server**

Ideally, for each problem, it is helpful to have, besides the problem statement, the solution source code, the input/output datasets, a solution description and the types of techniques used in solving that problem. Unfortunately, this is seldom the case. Obtaining that information can be done either by contacting the administrators of the online judges and asking for the data, or by relying on the programming community to try to reverse engineer the input/output datasets from a correct solution to a problem.

Regarding the scoring system, there are three techniques that stand out: adaptive, time and points. In the first two methods, scores are given only if the solution passes all the tests (input/output sets). On the other hand, the points scoring system allows partial scores.

- *Adaptive scoring*: the number of points scored depends on how many people solved that particular problem. The fewer the people solved it, the higher the score,
- *Time scoring*: a penalty is computed for each solved problem as the number of minutes elapsed from the start of the contest to the moment when the solution was accepted plus 20 multiplied by the number of failed attempts,
- *Points scoring*: for each test or group of tests that pass, a fixed amount of points is given; usually, if a solution passes all the tests it receives 100 points.

In time scoring, the first ranking criteria is the number of solved problems and then the contestants are sorted in ascending order by penalty. In the other two scoring methods, the ranking is obtained by sorting in descending order the sum of the problem scores.

Another scoring aspect is having tokens, which are used at IOI. Basically, the contestant sees only partial scores, and, in order to see the full scores for a problem, he must spend a token. The tokens are generated at fixed time intervals. Also, the ranking in some cases can be frozen, i.e., in the last hour the ranking is not updated anymore to keep the suspense.

One important feature that can be included in the proposed system is solution hacking. During the contest and after locking their submitted solution, participants have the possibility of looking at other contestants' solutions and add testcases for which that solution would be wrong. If this is the case, penalties and bonuses can be applied to the final score.

4. Proposed Solution

4.1. *System Description*

The proposed solution is a system for evaluating algorithms and for organizing contests, which encompasses all the characteristics from Section 2.3. This is achieved by expanding on the techniques used by existing online judges and by creating a community in which people can post tutorials regarding different programming topics and in which authors can share their knowledge to create better problems.

A starting point for the system presented in this paper is the Contest Management System (CMS) discussed in the previous section and described in (Maggiolo & Mascellani, 2017; Maggiolo *et al.*, 2017). Compared to CMS, the proposed solution reduces the number of required services and web servers and optimizes communication between the components by means of triggers instead of periodically making requests. In order to have access to a large problem set, the system employs the method used by VJudge and allows the creation of contests that have problems from other online judges. The main difference is that, once a problem from another judge is used and solved, it is added in the

database and it is flagged for input/output test creation. This way, contestants who solved that problem have the opportunity to contribute to the system by adding test cases and by providing a solution description. Each contributor earns reputation points which can be used to unlock certain features, but this aspect is left for future research.

4.2. System Achitecture

The proposed system has four main components: System Web Server, Data Manager, Dispatcher and Worker, which communicate by means of WebSockets and simple Sockets. Due to the fact that the communication uses triggers and is based on the Observer design pattern, there is a need for a permanent connection between the four components, which, in turn, removes the need for a heartbeat pattern, as it was the case at the CMS:

There is a single access point in the system and that is the System Web Server (SWS). It has four views depending on the user permissions:

- Admin view: user, problem and contest management,
- Author view: sandbox in which author can test the proposed problems,
- Contestant view: view of an ongoing contest
- Observer view: ranking of an ongoing or finished contest

All the business logic is coordinated by the Data Manager (DM). It handles the communication between the SWS and the Dispatcher, and stores all the information in a database (optionally, it stores it also in a network file system). The Dispatcher's role is to coordinate the submission evaluation and to give tasks to the workers. The worker compiles and executes each submission against the corresponding datasets, and returns the submission score. The worker also has a logic that allows it to kill a task that exceeded the specified time limit. It is important that each worker processes each submission sequentially so that the execution time is measured properly.

An example of a communication flow between the contestant and the system is presented in Fig. 1. The contestant submits a solution to the SWS, which sends it to the DM who stores it in the database and notifies the dispatcher that there is a new submission to be evaluated (TASK_SUBMITTED event). The dispatcher chooses a worker and sends the submission. After the worker evaluates the submission, it sends the results back to the dispatcher, which sends it to the DM to be stored in the database. The DM triggers a TASK_COMPLETE event, and the SWS listens to it and sends the score to the contestant.

This architecture design was chosen so that the Dispatcher-Worker components can be replaced with any traditional task processing logic because a task can be viewed as the process of using a specific compiler, then executing the code on multiple data sets and comparing the resulting output with the correct output. The Dispatcher, in its simplest form uses a round robin task distribution algorithm, but other load balancing techniques can be applied.

Moreover, the dispatcher can contain fault tolerance and replication logic, and it can calculate the execution time as the mean of the execution times on multiple workers. By using triggers and events, the Dispatcher-Worker logic can be replaced with a single Worker if there are few contestants and there are few resources at disposal.

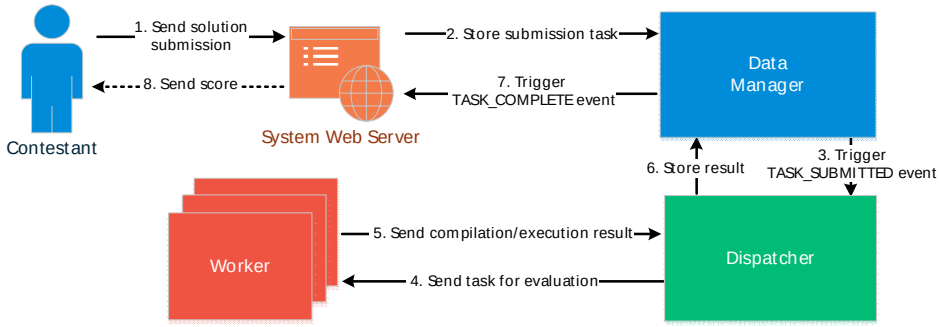

Fig.1 – Contest submission communication logic.

The Worker logic pseudocode is as follows:

1. Receive the submission id, submission source code, input/output data sets, the values for the max execution time and the max memory, the programming language name, and a scoring object.
2. Compile the source code (if it fails then return error).
3. Execute program for each input dataset and feed the output to the scoring object (if it fails then return error).
4. Send the calculated score for that submission.

The scoring object contains the logic that determines the problem score based on the obtained output and the correct output. This object implements an IScoring interface and has configurable implementations for each of the three previously discussed scoring methods.

In comparison with the CMS, the proposed solution has fewer components but they have a greater importance. At the CMS if one non-critical component fails, the other components still function but they will most likely be in a waiting state. For example, if the EvaluationService component fails then the Workers function but they have no one to give them tasks. The solution presented in the current paper removes some of the communication overhead, at the expense of a slightly lower performance in extreme cases of component failure.

4.3. *Database Structure*

The user, problem, contest and submission information is stored by the DataManager in a database or a database cluster (large scale environments). Fig. 2 shows the proposed database structure. There are several fields of type BLOB in which are stored the source code, the problem description, the input and output data, and the contest description. An alternative to storing that

information in the database is to store only the paths to the actual data on the hard drive. The user table contains typical user information and a role, which can be administrator and normal user. Any normal user can propose problems but they have to go through a validating process and that must be approved by one of the administrators. All the problem related data is stored in the task tables. Besides the aforementioned task characteristics, a task can have a difficulty and one or more categories or tags, in order to help a contest creator to properly choose the problems that have to be resolved.

In the contest table, there is a type property that encompasses aspects such as the scoring object, the contest mode (public, private or password protected) and the ranking mode (live ranking, freeze X minutes before end, or hidden). After a submission is evaluated by the Worker, the score, statusCode and statusMessage entries are filled. The latter two values, represent compilation and execution errors if this was the case.

In its current state, the database does not allow contest enrolment and user restrictions, which are features of a few online judges. Also, there is no logging system in place as is the case of the CMS. These characteristics will be added in the next version of the proposed system.

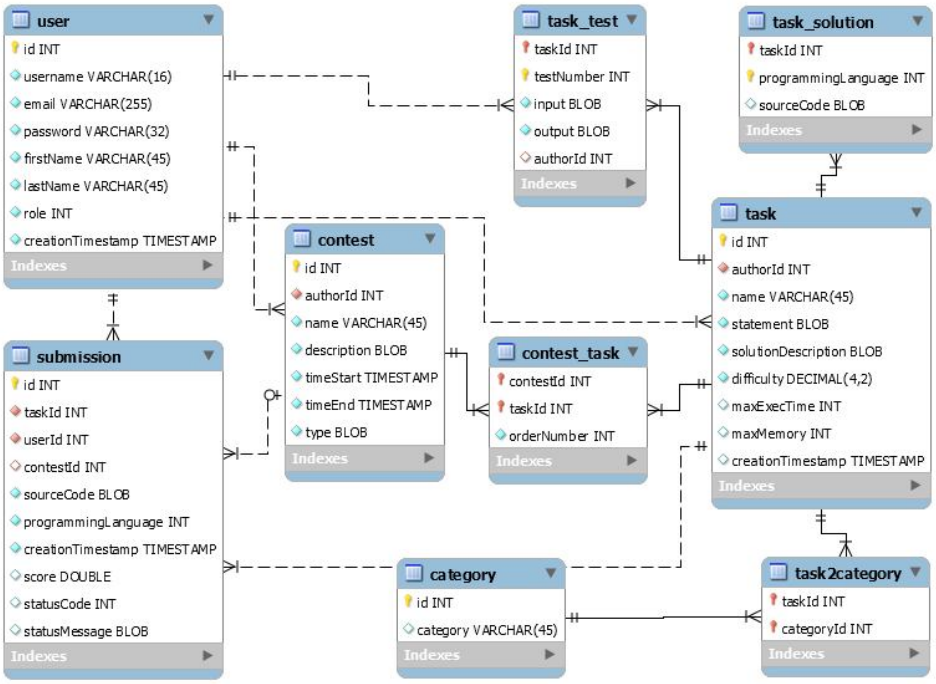

Fig.2 – Proposed database structure.

4.4. *System Deployment*

In its simplest form the system can be deployed to a single computer, but it is recommended that the worker runs on another computer in order not to

influence the task execution time. So, there are two scripts that have to run. One will start the SWS, the DM, the Dispatcher and the database server on one machine, and the other will start the worker on another machine. When the worker starts it will automatically register itself to the dispatcher.

The script that starts the worker also installs the prerequisite packages for compiling and executing programs in various programming languages: GNU compiler collection for C/C++, Oracle's JDK for Java, Free Pascal and Python.

5. Conclusions and Future Work

The research presented in this paper consists of:

– a study of existing online judges with their advantages and disadvantages and a short presentation of what is needed in order to have a programming contest,

– a novel and highly extensible system for managing programming contests: users, problem sets, solution evaluation and ranking,

– a tool for authors to manage their proposed problems, to better test them and to collaborate with other authors in order to increase the quality of the problems and the input/output datasets,

– a method of applying the system in a classroom environment so that the student evaluation is quick and objective,

– discussions regarding the issues that can occur in developing the proposed solution.

The presented system is currently under development and the future research will firstly consist of addressing other issues, situations, restrictions and security concerns that will inherently appear. Other future developments include improvements in the dispatcher (a better load balancing component), plagiarism detector (solutions for existing problems can easily be found online), student team support (including team/individual ranking), user reputation points which unlock access to different features (source code, test cases, solution description), or a tool for detecting the algorithm complexity by analyzing the submitted code.

REFERENCES

Maggiolo S., Mascellani G., *Introducing CMS: a Contest Management System*, Olympiads in Informatics, *6*, 86-99 (2012).

Maggiolo S., Mascellani G., Wehrstedt L., *CMS: a Growing Grading System*, Olympiads in Informatics, *8*, 123-131 (2014).

* * * *International Olympiad in Informatics*, 2017. Retrieved from http://www.ioinformatics.org/

* * * *The ACM-ICPC International Collegiate Programming Contest*, 2017. Retrieved from https://icpc.baylor.edu/

* * * *Google Code Jam*, 2017. Retrieved from https://code.google.com/codejam/

* * * *Facebook Hacker Cup*, 2017. Retrieved from https://www.facebook.com/hackercup/
* * * *Amazon TechO(n) Challenge*, 2017. Retrieved from http://challenge.amazontechon.com/
* * *Codeforces*, 2017. Retrieved from http://codeforces.com/
* * * *Infoarena comunitate informatică, concursuri de programare*, 2017. Retrieved from http://www.infoarena.ro/
* * *Timus Online Judge*, 2017. Retrieved from http://acm.timus.ru/
* * *TJU ACM-ICPC Online Judge*, 2017. Retrieved from http://acm.tju.edu.cn/toj/
* * *UVa Online Judge*, 2017. Retrieved from https://uva.onlinejudge.org/
* * *Virtual Judge*, 2017. Retrieved from https://vjudge.net/
* * * *CMS :: Main. Contest Management System*, 2017. Retrieved from https://cms-dev.github.io/

SISTEM DE EVALUARE PENTRU CONCURSURI ŞI SĂLI DE CLASĂ

(Rezumat)

Algoritmii şi programarea reprezintă baza necesară fiecărui student în domenii precum informatică şi calculatoare. În timpul facultății, multe examene şi teste implică rezolvarea unor probleme şi implementarea unor algoritmi; de asemenea, cunoştințele de programare sunt testate la diverse concursuri. Lucrarea de față prezintă ce presupune un astfel de concurs de programare şi face un rezumat al principalelor caracteristici ale soluțiilor existente de evaluare a capacității unui student de a rezolva o problemă. Principala contribuție este propunerea unui sistem nou de evaluare a calității soluției unei probleme în cadrul unui concurs sau a unei săli de clasă. Sistemul oferă un grad ridicat de extensibilitate prin independența componentelor sale. De asemenea, sistemul pune la dispoziția utilizatorului suficiente opțiuni pentru evaluarea capacității unui student de dezvoltare şi implementare a algoritmilor folosiți pentru rezolvarea anumitor probleme.