ISSN 1822-7732

INTERNATIONAL OLYMPIAD IN INFORMATICS
INSTITUTE OF MATHEMATICS AND INFORMATICS
INTERNATIONAL FEDERATION FOR INFORMATION PROCESSING

OLYMPIADS IN INFORMATICS

Volume 3 2009

Selected papers of
the International Conference joint with
the XXI International Olympiad in Informatics
Plovdiv, Bulgaria, August 8–15, 2009

OLYMPIADS IN INFORMATICS

ISSN 1822-7732

`http://www.mii.lt/olympiads_in_informatics`

Olympiads in Informatics, Volume 3, 2009

2009.06.30. 13 leidyb. apsk. l.
Tiražas 200 egz. Užsakymas Nr. 1748

Printed by Printing house “Mokslo aidai”, Goštauto 12, 01108 Vilnius, Lithuania

Olympiads in Informatics, 2009, Vol. 3

Foreword

OLYMPIADS IN INFORMATICS is an annual refereed journal that provides an international forum for presenting research in teaching and learning informatics through competition. The journal is substantially connected with the conference held during the International Olympiad in Informatics (IOI). The papers in this volume are being presented during IOI'2009 in Plovdiv, Bulgaria.

What a difference 20 years makes

In May 1989, when the teams were gathering for the first IOI (also in Bulgaria), the 80486 microprocessor had just been released on the world and Tim Berners-Lee's "Information Management: A Proposal" was being distributed to CERN's management. That processor was available at 25 Mhz; the one this introduction is being written on is running at 2.5 Ghz. That proposal was the seed of the World Wide Web, which has become so ubiquitous that access has become a basic utility, along with water and power, in many countries.

And what of the IOI?

The foundation of the IOI, thanks to the efforts of Professor Blagovest Sendov and the support of UNESCO, saw the emergence of a second phase of international science olympiads. The 50s and 60s saw Mathematics, Physics and Chemisty flourish. It was to be another 20 years before another olympiad, the IOI, came into existence. Since then there has been an explosion of interest, from Astronomy to Earth Sciences, from Biology to Linguistics.

The 13 countries who participated in 1989 have grown to around 80. The contest has run every year since its inauguration, has been held in 19 different countries across 5 different continents. Contestants from earlier years are now lead delegations. Leaders who come for 'just one year' find themselves coming back year after year. The IOI community – the IOI family – continues to grow, and long may it continue.

The contest itself has been polished over the years, but is still fundamentally faithful to the original vision. The number of questions may have changed and their difficulty tuned, but contestants today require the same skills as those 20 years ago. Machines may have hundreds or thousands of times the speed or memory, but it is a testimony to the types of problems set that, with perhaps a little tweak here and there, the challenges from the contest's history remain a challenge. In a world that demands 'bigger' and 'faster', the IOI demands 'smarter'.

And what of the wider community?

National olympiads have become an established part of the educational system in many countries. You need only read the papers in these proceedings to see how many students we reach and have reached. Some contests are run with governmental support and guidance, some independently. In all cases opportunities, which might not otherwise exist, are given to students. Not just to those who know they are interested in informatics, but often at a junior level where such an interest can be inspired.

In this year, when the IOI returns to its birthplace, we should think back over the last 20 years. Think of the thousands of students who have been given the wonderful opportunity of attending an IOI, and of the hundreds of thousands (if not more) who have been touched across the world by national endeavours. Think of the teams of volunteers who have given their time and resources to making the contests the success that they are. Organisers who work tirelessly, often for no reward or even recognition, time and time again.

Thanks are due to everyone who has contributed to this volume and the IOI conference. In particular, we would like to thank Prof. Krassimir Manev and the Bulgarian organisation of this year's IOI for giving us the opportunity to host the conference.

Editors

Olympiads in Informatics, 2009, Vol. 3, 3–16

Using Item Response Theory to Rate (not only) Programmers

Michal FORIŠEK
Department of Informatics, Faculty of Mathematics, Physics and Informatics
Comenius University
Mlynská dolina, 842 48 Bratislava, Slovakia
e-mail: forisek@dcs.fmph.uniba.sk

Abstract. We show how Item Response Theory (IRT) can be used to define a new type of rating system, one that is especially suitable for programming competitions (and other types of competitions where difficulty of competitions varies between rounds). We show some useful theoretical properties of this rating system, including the ability to argue about hardness of past competition tasks, and about the precision of contestants' skill estimates. We also define an objective method of comparing different rating systems. In the final section of the paper we apply our methods on real competition data.

Key words: item response theory, ranking, rating, programming competitions.

1. Overview

In this section we provide an overview of topics relevant to this article:
– research of scoring and ranking in programming competitions,
– research in the area of rating systems,
– Item Response Theory.

1.1. *Programming Competitions*

For a few years the International Olympiad in Informatics (IOI) community is concerned about the accuracy of the testing, scoring and ranking process. Several publications that research various aspects of this problem include (Cormack, 2006; Cormack *et al.*, 2006; Forišek, 2004; Forišek, 2006; Opmanis, 2006; Van Leeuwen, 2005; Verhoeff, 2006; Yakovenko, 2006).

A publication particularly relevant to the topic of this paper is (Kemkes *et al.*, 2006) where Kemkes *et al.* use Item Response Theory to analyze scoring used at IOI 2005, and in particular the impact of the proportion of "easy" and "hard" test cases on the relevancy of the competition results. Based on the results of the analysis, new scoring methods with better discrimination are suggested.

(We would like to note that similar research has recently been conducted for other competitions as well, for example see (Gleason, 2008) for an analysis of two mathematical competitions.)

However, we would like to note that while these publications use IRT only passively, as a tool for analysis of tasks only. In parts of this paper, we will use IRT as an active tool – not only to rate tasks and participants, but also to make estimates and predictions.

1.2. *Rating Systems*

The idea of a rating system has been studied for several decades. Competitiveness is a part of our human nature, and when there is competition, there is the need to rate and/or rank the competitors. Also, the need for rating is often encountered in educational systems, and many other areas.

In this context, *rating* means assigning a vector of properties to each subject (i.e., contestant), and *ranking* means arranging the subjects into a linear order according to some set of criteria. Usually, ranking is the goal, and rating represents possible means to achieve this goal.

The first rating systems were reward-based: A good performance was rewarded by granting the subject rating points. The main advantage of these rating systems was their simplicity. Due to this reason, such rating systems are still used in many popular sports, such as tennis and Formula 1.

These systems are usually designed so that their discrimination ability is highest among the top subjects, and rapidly decreases as the skill of the subjects decreases. Moreover, the reward values are usually designed ad-hoc, and the rating systems usually have little to no scientific validity.

One of the first areas to adopt a more scientific-based rating (and thus ranking) system was chess. The United States Chess Federation (USCF) was founded in 1939. Initially, USCF used the Harkness rating system (Harkness, 1967). This was a reward-based system determined by a table that listed the reward size as a simple function of the difference between the players' current ratings.

After discovering many inaccuracies caused by this system, a new system with a more solid statistical basis was designed by Arpád Elő, and first implemented in 1960. For details of this rating system see (Elo, 1978).

The Elo rating system was based on the following set of assumptions:

- The performance of a player in a game is a normally distributed random variable.
- For a fixed player, the mean value of his performance remains constant (or varies negligibly slowly).
- All players' performances have the same standard deviation.

The importance of the Elo model lies in bringing in the statistical approach. Even though subsequent research showed that each of these assumptions was flawed, and accordingly changes to the rating system were made, the currently used rating system in chess is still called the Elo rating system in Arpád Elő's honor.

An excellent overview of various rating systems in chess can be found in (Glickman, 1995). For a discussion of some problems of the currently used rating system, see (Sonas, 2002).

The next important step in the history of rating systems was the Glicko system (Glickman, 1999) that, in addition to calculating ratings, calculated also the rating deviation for

each of the players – a value that measures the accuracy of the given player's rating. This system was later amended into the Glicko-2 system (Example of the Glicko-2 system) that also computed the rating volatility – a value that measures how consistent a player's performances are. Glickman's rating systems were designed to handle situations where each event is a comparison of a pair of subjects.

In recent years, Glickman's models were generalized to handle situations when events involve more than two subjects. Notable advances in this direction include Microsoft's recently published TrueSkill™ ranking system (Herbrich and Graepel, 2006; Herbrich *et al.*, 2007; Dangauthier *et al.*, 2008a), and TopCoder's rating algorithm (TopCoder Inc., 2008).

All these rating systems are incremental: given the previous estimates and new results, they compute a new set of estimates using Bayesian inference. Hence they are usually called Bayesian rating systems.

1.3. *Item Response Theory*

In practice we often encounter a situation when a variable we might be interested in can not be measured directly. For example, this situation is often encountered in *psychometrics*, when trying to measure aspects such as knowledge, abilities, attitudes, and personality traits.

The key approach to these situations is to model the measured attribute as a hidden, *latent variable*. In contrast to visible attributes, such as height and weight, these latent variables can not be observed or determined by direct measurement. However, these variables can be estimated from the results of appropriate tests.

The first and to date most common approach is currently known as the Classical test theory (CTT). As a gross oversimplification, we may state that in the CTT the test is scored, and the subject's score is used to estimate the latent ability. The main goal of CTT is to construct the tests in such a way that the *reliability* and *validity* of the test are maximized.

In recent years, CTT has been superseded by a more sophisticated approach, the Item Response Theory (IRT). The main difference is that IRT models include not only the latent variables we try to measure, but also *item parameters* (such as difficulty of a question in an IQ test). In general, IRT brings a greater flexibility and provides more sophisticated information than CTT did.

IRT is ideally suited for our setting, as in programming competitions the tasks indeed have various difficulty. We are interested both in determining the item parameters (i.e., discuss task difficulty) and in using this additional information to make better estimates of the contestants' abilities.

In this section we give an overview of the areas of IRT that are relevant to our article. For a more general overview of the areas related to IRT, we strongly recommend (Partchev, 2004). A reader interested in a deeper background in IRT is advised to pursue this topic further in the excellent monography (Baker and Kim, 2004).

The 2-Parameter Logistic Model

First of all, we will assume that the latent ability we are interested in (e.g., the ability to solve programming competition tasks) is a scalar, i.e., that it can be described by a single real number. For each contestant c, we will denote their ability score as $\theta(c)$, or just θ if c is fixed.

As stated above, in IRT we take into consideration the individual test items. More precisely, we assume that each item comes with an inherent *item characteristic function*. This is a function that maps the subject's ability score θ to the probability that the subject answers the given item correctly.

We model programming tasks using the 2-parameter logistic model (2PL model). In this model, each item is described by two parameters: its difficulty b and its discrimination a. The item characteristic function in this model is given in (1).

$$Pr(\theta, a, b) = \frac{1}{1 + \mathrm{e}^{-a(\theta - b)}}. \tag{1}$$

Estimating Parameters

One of the most common methods how to estimate the unknown parameters (be it task parameters, subjects' abilities or both) is the maximum likelihood method.

Let X be a random variable X with a parametrized probability distribution function f_y. We measured the random variable X and got the result x. The *likelihood* $L(y, x)$ of a given parameter value y is defined as the conditional probability $Pr[X = x|y]$. (Note that this is **not** the probability of y being the true parameter.) Our estimate of the unknown parameters will be the estimate for which the likelihood function is maximized.

For example, consider the case where a subject was given n tasks. For each of these tasks we know its 2PL parameters a_i and b_i. We also know whether the subject solved each of the tasks. Formally, let $s_i = 1$ if the ith task was solved correctly, $s_i = 0$ otherwise. The values s_i are usually called the *response pattern*. Then the likelihood function of an ability estimate θ is the function:

$$L(\theta) = \prod_{i=1}^{n} Pr(\theta, a_i, b_i)^{s_i} \cdot \big(1 - Pr(\theta, a_i, b_i)\big)^{1 - s_i}. \tag{2}$$

Fisher Information and Error of Measurement

Whenever we observe a random variable, this observation gives us information that we can use to make a better estimate of the hidden parameters. This statistical version of information was first formalized by Sir Robert Fisher.

Intuitively, we can define information as the reciprocal of the precision with which the parameter could be estimated. Formally, let X be a random variable such that its probability distribution depends on a parameter θ. Let $L(\theta, x)$ be the likelihood function. Then the *score* V is the partial derivative with respect to θ of the natural logarithm of the likelihood function, and then the *Fisher information* is the expected variance of the score – or equivalently (as the expectation of the score is always zero) Fisher information is the expectation of the square of the score. Informally, this definiton corresponds to the steepness of the log-likelihood function in the vicinity of its maximum.

In the 2PL model, it can be computed that the Fisher information function of a single item with parameters a, b can be simplified to:

$$\mathcal{I}(\theta) = a^2 Pr(\theta, a, b)\big(1 - Pr(\theta, a, b)\big). \tag{3}$$

Given an ability estimate $\hat{\theta}$, the variance of this estimate can be estimated as the reciprocal of the test information function at that point:

$$Var(\hat{\theta}) = \frac{1}{\mathcal{I}(\hat{\theta})}. \tag{4}$$

The standard error of measurement (SEM) is defined as the square root of the variance

$$SEM(\hat{\theta}) = \sqrt{\frac{1}{\mathcal{I}(\hat{\theta})}}. \tag{5}$$

In the case of the 2PL logistic model, we get

$$SEM(\hat{\theta}) = \sqrt{\frac{1}{\sum_{i=1}^{n} a_i^2 Pr(\theta, a_i, b_i)(1 - Pr(\theta, a_i, b_i))}}. \tag{6}$$

2. Towards an IRT-Based Rating System

In this section we present our work directed towards designing a IRT-based rating system. First, we describe the setting for which we want to construct the system – in other words, the assumptions we make. These assumptions do hold for common programming competitions. We then discuss some properties any IRT-based rating system must have, and finally provide a brief description of our proof-of-concept implementation.

2.1. *Assumptions*

Our rating system is designed for competitions that consist of multiple rounds, each round consists of multiple items, and various items can have various degrees of difficulty.

We assume that the ability that determines the success in solving the items is a scalar.

Additionally, we assume that the setting is not antagonistic (contestants do not directly influence the performance of other contestants), and that there is no guessing – in other words, as ability decreases, the probability of solving a task converges to zero.

For the part of our research that is presented in this paper, we also assume that the abilities of subjects are invariant in time.

2.2. *Necessary Properties of IRT-Based Rating Systems*

The basic idea behind using IRT in a rating system is simple – the ability estimates and the task parameter estimates will be computed at the same time, as the maximum likelihood estimate given the observed response patterns.

However, the situation is not so easy in practice. We will now show that we need to enforce at least two additional restrictions.

The first issue is the symmetry of the item characteristic function. If we multiply all estimates (both for the abilities and for the task parameters) by -1, the likelihood of the estimate will not change. We need to break this symmetry somehow, and enforce that the positive direction represents higher ability level / task difficulty.

The second issue that needs to be addressed is the actual existence of the maximum likelihood estimate.

Consider the following simple example: Let $C = \{c_1, \ldots, c_{n+1}\}$ be the set of contestants in a round, and let $T = \{t_1, \ldots, t_n\}$ be a set of tasks in the round. Sey $s_{i,j} = [i > j]$, i.e., contestant i solved all tasks j for which $i > j$.

There is a clearly defined linear order on this set of contestants – for any pair of contestants, the set of tasks one of them solved is a strict superset of the set of tasks the other one solved. However, it can easily be proved that the maxima of the likelihood function are of the following form: For any integer x we can set $\theta(c_y) = -\infty$ for $y \leqslant x$ and $\theta(c_y) = \infty$ otherwise, set all $a(t_y) = 1$, set $b(t_y) = -\infty$ for $y < x$, $b(t_y) = \infty$ for $y > x$ and set $b(t_x)$ arbitrarily. This completely fails to reflect the linear order.

There is only one solution to both of these issues – we need to restrict the estimates to bounded intervals. Note that we are free to pick the exact bounds, as we are not influencing the results in any way by doing so, we are just defining the scale.

In our case with the 2PL model, we opted to restrict θ and b to the same interval $[-\beta, \beta]$, and to restrict a to the asymmetric interval $[-\alpha/10, \alpha]$. The rationale behind the restriction to a is to enforce that most tasks have positive a, i.e., the probability of solving the task increases with increasing ability.

(Note that in practice we can occassionally have tasks where the probability of solving the task actually slightly decreases with increasing ability. This is why we allow slightly negative values of α. On the other hand, if we are getting many tasks with $a < 0$, this is usually a sign that the ability we are measuring has nothing to do with the results.)

For the example presented above, after we enforce the restrictions, there is just one global maximum of the likelihood function – the abilities are uniformly distributed along $[-\beta, \beta]$, task difficulty parameters b are uniformly distributed between these, and all task discrimination parameters are α. This precisely corresponds to the linear order we described above.

2.3. *Implementation Details*

Our proof-of-concept implementation used the values $\alpha = \beta = 10$. We compute the maximum likelihood estimate of all the parameters numerically. In order to reduce the running time, we used the observation that the parameters for each task can be estimated only from the (current estimate of the) subjects' abilities, i.e., independently from the other tasks. Hence we implemented a bootstrapping algorithm that alternately computes a new estimate for all the subjects' abilities and a new estimate for all the task parameters until sufficient convergence is reached.

3. Comparing Rating Systems

In situations where we have multiple rating systems, it is only natural to ask which of them is *better*. The word *better* can have multiple meanings, the most natural one (but by far not the only one) being "which of them estimates the true latent ability more precisely". In this section we give our answer to the question: Is there an objective method how to compare rating systems?

It is obviously impossible to compare rating systems directly, as we are not able to measure the latent ability.

However, there is a natural way out. Having a model and a set of rating estimates should enable us to predict the outcome of a future rated event. The more accurate predictions we can make, the more trustworthy the pair (rating system, prediction algorithm) is.

In other words, while we can not directly compare rating systems, we can compare rating systems accompanied by prediction algorithms.

Still, the previous paragraphs leave one open question: what exactly makes a prediction more accurate? The answer is not unique. Moreover, the answer to this question is actually what we should start with in practice. In the simplest setting with two-player antagonistic matches, the focus is usually on predicting the winner. (Or, more precisely, each player's probability of winning.)

The setting with multiple subjects allows for a much richer spectrum of possible goals. To name some: predicting a subject's score, a subject's placement, estimating their probability of placing among top K subjects, etc.

As one possible example, we will now focus on the last goal mentioned above: given a round with N known participants and an integer K, we want to predict the probabilities that each of them will place among the top K in the round.

3.1. *Predicting Success in a Bayesian Rating System*

In this section we will show an algorithm to predict the probabilities of placing in the top K using a Bayesian rating system. The presented algorithm is **not** our original work – however, it is only informally known in the public domain, we we were not able to assign authorship to a particular author.

In the models used in the Bayesian rating systems the performance of each subject is modeled as a normally distributed random variable.

In this section we will assume that for each contestant i the rating system computed the estimate of the mean μ_i and the variance σ_i^2 of this random variable. (This is for example the case with the TopCoder ratings, where the subject's volatility computed by the rating system is an estimate of the standard deviation σ of this normal variable.)

This model is great when it comes to computing the expected placement of a subject – this is simply one plus the sum of probabilities that the other subject performs better.

However, the situation is much worse when we actually need to predict the probabilities. The only known algorithm in this case is a Monte Carlo randomized simulation: We

simulate a sufficient number of rounds. For each round, we generate the actual values of all the performances, and sort them to find the top K subjects.

We would like to stress several major disadvantages of this prediction algorithm:

- Its convergence is slow – obviously the number of simulation steps necessary to achieve precision ε (with high probability) is at least linear in $1/\varepsilon$.
- It does not easily generalize for multiple round tournaments. It is easily seen that the time complexity necessary to predict the outcome of d consecutive rounds within some fixed ε precision grows exponentially with d.
- It can only predict relative order of the performances, not their actual values.

3.2. *Predicting Success in the IRT-Based Rating System*

In this section we present the algorithm we designed to predict the probabilities of placing in the top K in our IRT-based rating system.

The main idea of our algorithm is that, similarly as in the previous case, we will simulate multiple rounds and take the average of the results.

First, we will start by generating the task parameters for the round. (In practice, the best way to do this is by randomly sampling a pool of known past tasks.) Given the set of tasks, we will use binary search to find the correct estimate for the threshold, i.e., the expected number of solved tasks needed to be in the top K.

Given some threshold, we can, for each of the subjects, compute the probability of solving at least that many tasks. The sum of these probabilities gives us the expected number of people that will cross this threshold. If this number is less than K, we need to lower the threshold, otherwise we need to raise it.

The needed subproblem (given a subject, a set of tasks and a threshold, compute the probability of reaching it) can be solved using dynamic programming – for each x and y, we compute the probability of solving exactly y out of the first x tasks.

Some of the advantages of this approach.

- Much faster convergence in practice – if we know the approximate task parameters beforehand, already the first round will give us a reasonable approximation of the answer.
- The approach generalizes to multiple round tournaments nicely.
- We can predict absolute performances. This can be useful e.g. to predict the number of contestants that will not be able to solve any tasks.

3.3. *Using the Standard Error of Measurement*

At this point we would like to make one technical note.

For Bayesian rating systems handling newcomers poses a significant challenge. The existing rating systems usually use some kind of a provisional ad-hoc approach. For example, the TopCoder rating system assigns newcomers a slightly above-average rating of 1200, and the rating change formulas are set so that they enable large rating changes for the first few rounds.

The prediction algorithm as described above uses the ability estimates as the exact truth in order to make the predictions. A much better way is to also use the standard error of measurement that is provided by our chosen model. In this way, we can, for example, solve the newcomer problem in a clean and systematic way – for any newcomer, the number of attempted tasks is small, hence the information function returns a small value, hence the standard error of measurement will be high.

The prediction algorithm presented in 3.2 can be modified to include this information. Given a contestant c and a task t, we will compute the probability that c solves t using the assumption that c's actual ability is a normally distributed random variable with mean equal to our rating estimate, and standard deviation equal to the computed standard error of measurement. In this case, the predicted number of tasks c solves out of a set T of tasks can be expressed as

$$PNT(c,T) = \frac{1}{SEM(c)\sqrt{2\pi}} \sum_{t \in T} \int_{-\infty}^{\infty} \exp\left(-\frac{(x-\theta(c))^2}{2SEM(c)^2}\right) \frac{1}{1+\mathrm{e}^{-a(x-b)}}\,\mathrm{d}x. \quad (7)$$

Evaluating the Predictions

Suppose that we predicted the vector of probabilities $(p_1, \ldots, p_N)$, and the actual outcome is $(s_1, \ldots, s_N)$, where $s_i = 1$ if contestant i placed Kth or better, and $s_i = 0$ otherwise. How to measure how good the prediction was?

Our definition will be based on the following experiment: Imagine that we, in a sequence, throw N biased coins, where coin i will fall heads with probability p_i. In this experiment, we can easily compute the probability of getting the outcome $(s_1, \ldots, s_N)$: it is $\prod_{i=1}^{N} p_i^{s_i}(1-p_i)^{1-s_i}$.

Equivalently, this formula can be seen as the likelihood that the actual probabilities before the round were $(p_1, \ldots, p_N)$, given that the outcome was $(s_1, \ldots, s_N)$. And this is almost exactly how we'll define the quality of the prediction.

For computational reasons, we prefer to use the log-likelihood function in our definition instead. Given the results $(s_1, \ldots, s_N)$, we define the quality of a prediction $(p_1, \ldots, p_N)$ as

$$Q(p_1, \ldots, p_N) = \sum_{i=1}^{N} s_i \log p_i + \sum_{i=1}^{n} (1-s_i)\log(1-p_i). \quad (8)$$

4. Evaluation on Real Life Data

Data Sets Used for Analysis

We tested our ideas on two separate data sets. One data set we used included 88 tasks used in two years of Slovak national programming competitions, and the scores of over 300 contestants solving them. All of these tasks used partial scoring on the scale 0 to 10, with any correct solution scoring at least 4, and any efficient solution scoring at least 7 points.

Our basic model was adapted to this setting as follows: For each task, we have three items. For each contestant, the first of these is set as solved iff she scored at least 3 points, the second iff she scored at least 6 points, and the third item is considered solved iff she scored at least 9 points.

The second data set included 560 tasks used in TopCoder competitions between 2006-05-09 and 2008-02-16, inclusive, and the performances of over $12\,000$ contestants on these tasks.

Sanity Check

We used our proof-of-concept implementation of the rating system described in Section 2.3 to estimate the ratings of the contestants and the task parameters for the tasks in the first data set. A natural sanity check at this point is to examine whether the computed estimates give us a sufficient approximation of the real data. This check was performed as follows:

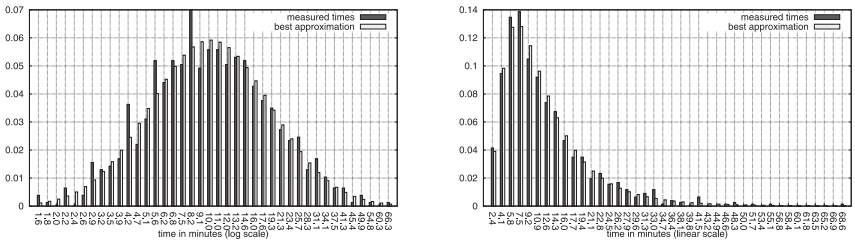
We divided the ability range into 30 equally large buckets. For each task, we divided the contestants that attempted to solve it into these buckets, based on their ability estimate θ. For each bucket i we now computed two values: The actual total score x_i its contestants achieved, and their expected total score y_i. The value x_i is simply computed by summing the corresponding input data, while y_i is computed based on the estimated parameters of the given task and on their individual ability estimates. Note that under ideal conditions the values x_i and y_i would be identical for all i, meaning that our model matches the original data perfectly.

Once we computed all x_i and y_i, we proceeded to compute the weighted correlation coefficient of the vectors (x_i) and (y_i), where the weight of each element was the number of contestants c_i in the corresponding bucket i.

A summary of the resulting correlation coefficients is tabulated in Table 1. We see that for 64 out of 88 tasks ($\sim$73%) the correlation coefficient exceeds 0.9, and for half of these even exceeds 0.98, which is excellent. The three most significant outliers occurred in 2007/08, and all three were difficult tasks that only 10 contestants attempted to solve, and almost nobody did.

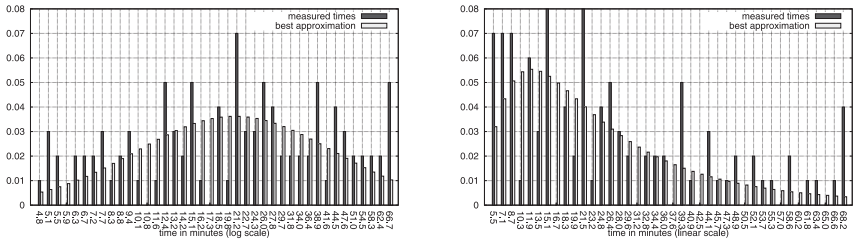
Table 1
Distribution of correlation coefficients between predicted and actual scores

ρ range	year 2006/07	year 2007/08
[0.99, 1.00]	9	12
[0.98, 0.99)	6	5
[0.95, 0.98)	9	9
[0.90, 0.95)	7	7
[0.75, 0.90)	12	8
[0.00, 0.75)	1	2
[−1.00, 0.00)	0	1



(a) log scale on x axis (b) linear scale on x axis

Fig. 1. Solving times for PalindromeDecoding match the lognormal distribution at 99% confidency level. Sample size $N = 772$.



(a) log scale on x axis (b) linear scale on x axis

Fig. 2. Solving times for Caketown match the lognormal distribution at 98% confidency level. Smaller sample size $N = 100$.

Solving Time

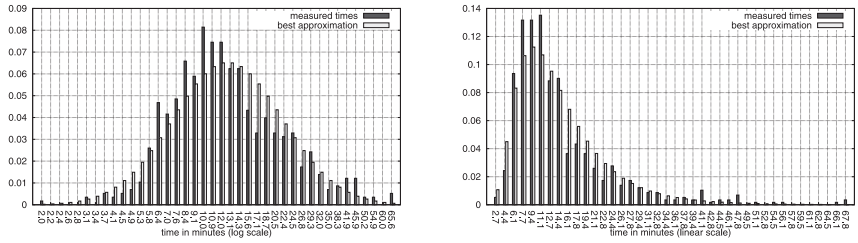
After we processed the data from the TopCoder competition using our IRT-based rating system, we managed to discover that the random variable giving the solving time for a particular task usually has a log-normal distribution.

To verify this, we used the Jarque-Bera normality test (Bera and Jarque, 1980), on natural logarithms of solving times. Out of the 560 tasks in our data set, for 71 the sample size was obviously too small (at most 5 contestants succesfully solved the task).

Out of the remaining 489 tasks, for 221 the null hypotesis can be accepted at 99% confidency level, and for another 163 it can be accepted at 98% confidency level.

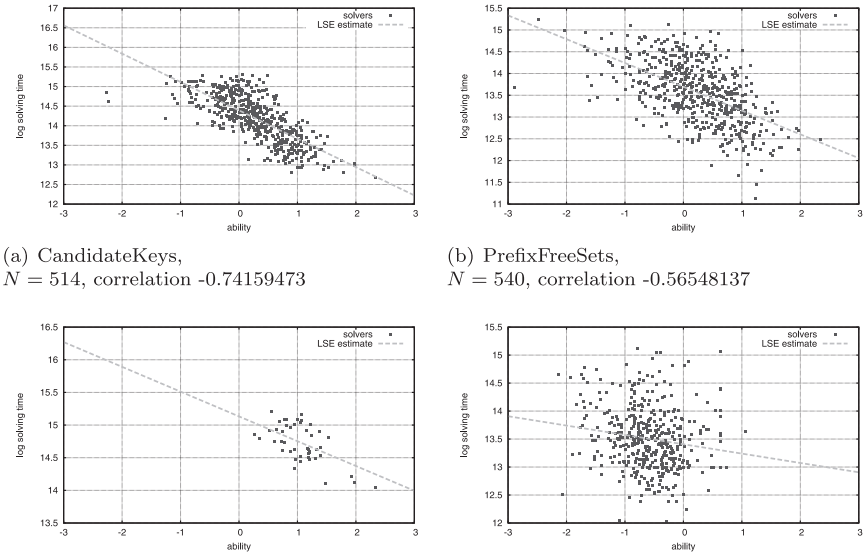
We verified the remaining 105 tasks by hand, and discovered that the main reasons for rejecting the lognormality hypotesis were mostly either small sample sizes, or it were too easy tasks where most of the solving times come from a narrow interval. Even for these cases the most probable lognormal distribution provides a reasonable approximation.

Furthermore, we note that for many (but sadly, by far not for all) tasks there is a significant correlation between the logarithm of the solving time and the ability estimate made by our rating system. Plots of the dependency for four random tasks are shown in Fig. 4.



(a) log scale on x axis (b) linear scale on x axis

Fig. 3. Solving times for RussianSpeedLimits do not match the lognormal distribution at 98% confidency level. Sample size $N = 577$. Lognormal distribution still offers a reasonable approximation.



(a) CandidateKeys, $N = 514$, correlation -0.74159473

(b) PrefixFreeSets, $N = 540$, correlation -0.56548137

(c) BasketballStrategy, $N = 47$, correlation -0.62801324

(d) VoteRigging, $N = 422$, correlation -0.15639571

Fig. 4. Correlation between abilities and log solving times – and least squares linear approximation thereof – for four tasks.

Predicting Advancement in the TopCoder Open 2008

We adapted our prediction algorithm to the TopCoder setting – by estimating the solving time using the observations presented above, we were able to compute the probability of exceeding a given score threshold. (We did ignore the challenge phase, and only computed the expected score of solving the three given tasks.)

Using this algorithm, we predicted the advancement probabilities for the first two online rounds of the TopCoder Open 2008. For the first round, the prediction given by our algorithm was slightly better as the one given by the TopCoder's own ratings and the Monte Carlo prediction algorithm (-812.26 vs. -818.85). For the second round the

Monte Carlo method gave a slightly better prediction (-372.04 vs. -394.68).[1] Hence already our simplest IRT-based rating system was able to produce results comparable to the ones given by the existing rating system.

Additionally, our superior model allowed us to compute predictions that were not possible in the previously used model. Specifically, the tournament has a rule that one has to have a positive score in order to advance. Our prediction algorithm computed that in the first round the expected number of contestants with a positive score is 872.994, which is less than the 900 advancer spots. This is almost exactly what actually happened – the actual number of advancers was 864. For the second round the algorithm correctly predicted that all 300 advancer spots will be taken.

5. Conclusion

In this paper we presented an overview of our research in the area of rating algorithms. We describe a new type of a rating system we developed using Item Response Theory, define a formal way how to compare rating systems, and use it to compare our rating system (adapted to a slighty different setting) to an existing Bayesian rating system.

Our rating system is more general than the existing models, in that it allows us to make predictions that were not possible in the existing models. Additionally, we believe that in settings with different tasks (such as programming competitions) an advanced version of the rating system will be able to use this additional information to give significantly better predictions than the existing Bayesian rating systems.

This is a new and promising area of research with many possible practical applications. Some points that surely deserve more attention include:

- a detailed analysis of the numerical aspects of the rating system (discussing convergence and its rate);
- evaluation of this approach on data from other areas, such as sports;
- modifying our approach to address settings in which abilities change over time;
- using the computed data to argue about hardness of past competitions.

References

Baker, F.B., Kim, S.-H. (2004). *Item Response Theory*: *Parameter Estimation Techniques*. CRC. `http://edres.org/irt/baker/`

Bera, A.K., Jarque, C.M. (1980). Efficient tests for normality, homoscedasticity and serial independence of regression residuals. *Economics Letters*, **6**(3), 255–259.

Cormack, G. (2006). Random factors in IOI 2005 test case scoring. *Informatics in Education*, **5**, 5–14.

Cormack, G., Munro, I., Vasiga, T. and Kemkes, G. (2006). Structure, scoring and purpose of computing competitions. *Informatics in Education*, **5**, 15–36.

Dangauthier, P., Herbrich, R., Minka, T. and Graepel, T. (2008a). TrueSkill through time: revisiting the history of chess. In *Advances in Neural Information Processing Systems*, Vol. 20, pp. 931–938.

[1] The presented numbers are values returned by the log-likelihood function (8).

Dangauthier, P., Herbrich, R., Minka, T. and Graepel, T. (2008b). TrueSkill through time: revisiting the history of chess. In J.C. Platt, D. Koller, Y. Singer and S. Roweis (Eds.), *Advances in Neural Information Processing Systems*, Vol. 20. MIT Press, Cambridge, MA, pp. 337–344.

Elo, A. (1978). *The Rating of Chessplayers, Past and Present*. Arco Publishing.

Forišek, M. (2004). On suitability of tasks for the IOI competition. Personal communication to the IOI General Assembly.

Forišek, M. (2006). On the suitability of programming tasks for automated evaluation. *Informatics in Education*, **5**, 63–76.

Forišek, M. (2009). Vyhodnotenie reliability hodnotenia Olympiády v Informatike. In *Proceedings of Conference DidInfo 2009* (to be published).

Gleason, J. (2008). An evaluation of mathematics competitions using item response theory. *Notices of the ACM*, **55**(1).

Glickman, M.E. *Example of the Glicko-2 System*.
`http://math.bu.edu/people/mg/glicko/glicko2.doc/example.html`

Glickman, M.E. (1995). A comprehensive guide to chess ratings. *American Chess Journal*, **3**, 59–102.

Glickman, M.E. (1999). Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics*, **48**, 377–394.

Harkness, K. (1967). *Official Chess Handbook*, David McKay Company.

Herbrich, R., Graepel, T. (2006). *TrueSkill^{TM}: A Bayesian Skill Rating System*. Technical report. MSR-TR-2006-80.

Herbrich, R., Minka, T. and Graepel, T. (2007). TrueSkill(TM): A Bayesian skill rating system. In *Advances in Neural Information Processing Systems*, Vol. 20, pp. 569–576.

Kemkes, G., Vasiga, T. and Cormack, G.V. (2006). Objective scoring for computing competition tasks. In V. Dagiene and R. Mittermeir (Eds.), *Information Technologies at School*.

Opmanis, M. (2006). Some ways to improve olympiads in informatics. *Informatics in Education*, **5**, 113–124.

Partchev, I. (2004). *A Visual Guide to Item Response Theory*.
`http://www.metheval.uni-jena.de/irt/VisualIRT.pdf`

Sonas, J. (2002). *The Sonas Rating Formula – Better than Elo*?
`http://www.chessbase.com/newsdetail.asp?newsid=562`

TopCoder Inc. (2008). *Algorithm Competition Rating System*. Technical report.
`http://www.topcoder.com/wiki/display/tc/Algorithm+Competition+Rating+System`

Van Leeuwen, W.T. (2005). *A Critical Analysis of the IOI Grading Process with an Application of Algorithm Taxonomies*. Master's thesis. TU Eindhoven.
`http://www.win.tue.nl/ wstomv/misc/ioi-analysis/thesis-final.pdf`

Verhoeff, T. (1990). Guidelines for producing a programming-contest problem set. An unpublished personal note. `http://www.win.tue.nl/ wstomv/publications/guidelines.pdf`

Verhoeff, T. (2006). The IOI is (not) a science olympiad. *Informatics in Education*, **5**, 147–159.

Yakovenko, B. (2006). 50% rule should be changed. Presented at *Perspectives on Computer Science Competitions for (High School) Students*.
`http://bwinf.de/competition-workshop/RevisedPapers/6_Yakovenko2_rev.pdf`

M. Forišek is currently finishing his PhD study at the Comenius University in Slovakia. He received a master's degree in computer science from this university in 2004. He was the head of the problem committee for several international programming contests, including CEOI 2002 and most years of the Internet Problem Solving Contest (IPSC). In years 2006 to 2009 he serves as an elected member of the International Scientific Committee (ISC) of the International Olympiad in Informatics (IOI). His research interests range from theoretical computer science to education of mathematics, informatics, and algorithms.

Olympiads in Informatics, 2009, Vol. 3, 17–25

Taking Kids into Programming (Contests) with Scratch

Abdulrahman IDLBI
Syrian Olympiad in Informatics, Syrian Computer Society
e-mail: adlogi@gmail.com

Abstract. Since launching the Syrian Olympiad in Informatics (SOI) five years ago, encouraging children to participate in the contest has been a challenging task, a common problem in many places around the world. Students, and many educators as well, see programming as a tough subject to learn. In addition, the style of IOI tasks is generally considered unattractive.

We started overcoming those obstacles through using Scratch, a graphical programming language developed at the MIT Media Lab. Scratch allows kids to start learning programming concentrating on the concepts rather than the syntax, while providing them with the ability to construct diverse projects that are attractive and meaningful to them. Scratch also allows examining children's programming skills against interesting tasks, and gives them the opportunity to move more smoothly into learning a traditional language like C++ and other computer science topics.

Key words: introducing programming to children, Scratch contests, Syrian Olympiad in informatics.

1. Introduction

Through the past few years, studies have warned about the decreasing interest among students to study computing-related fields, as they consider them tough and somehow uninteresting. Similarly, people organizing activities related to computer science contests, with challenges in programming and algorithms, constantly complain about the weak motive among youth to participate in them, especially when compared to the willingness to participate in robotics or IT contests for example. This problem becomes more obvious as the targeted audience gets younger. Taking the previous two examples of more acceptable contests among youth may give us a clue to the potential reasons of the problem: (1) sole programming is not as fantastic as building robots, (2) and children are not exposed to it early in their daily life (nor most of the adults surrounding them) as they are to other IT topics. Having these two points in mind would lead us to a solution.

When the Syrian Olympiad in Informatics started five years ago, the contest was divided into several divisions depending on age. The oldest division had preparation requirements similar to those of the IOI, and the national team for the IOI was chosen from its participants; but as a division got younger it contained less algorithms and programming in favor of more IT tasks (tasks related to knowledge of OS and desktop applications). While children were more familiar with IT skills than programming skills, testing IT skills or promoting them did not lead to a better preparation for the IOI in the older

division, nor did it help in discovering and preparing potential young computer scientists. In addition, the SOI organizers sought only students who were already considered distinguished in school mathematics or IT. Not approaching a wider audience of children meant two things: first, there was no way to discover hidden talents in computer science; second, without more children practicing programming the general public was not be able to recognize its importance.

To overcome the popularity problem facing our contest we changed our strategy. We decided to aim at all children and spread a culture of programming among them, which would provide a better opportunity of selecting future computer scientists. This change needed a powerful tool to be placed in the hands of children, and it was *Scratch*, a graphical programming language developed at the MIT Media Lab. Scratch allows kids to start learning programming concentrating on the concepts rather than the syntax, while enabling them to work on diverse projects that are attractive and meaningful to them. While recently some researchers used Scratch to introduce programming to university students and as a gateway to an advanced language like Java (Malan and Leitner, 2007), we argue that Scratch can be used in a similar way with younger children, both to prepare them to learn a language like C++ and to be used in contests with tasks similar to IOI tasks. Children in the context of this paper are mainly those between 7 and 15 years old.

2. What is Wrong with Programming?

The awareness and understanding of parents and educators have a major role in making any extracurricular activity for children succeed, but they are not the most important factors. The most important one is how fun and attractive children find the topic of the activity, and how closely it relates to them.

When it comes to programming, the languages used in the IOI (and many other typical languages) look like Greek at a first glance, and for many students they remain like that for a long time. Even the common simple start with a "Hello World" program raises several questions that cannot be answered in the first few sessions of a programming course. After that not-so-interesting start, children have to concentrate on remembering syntax details, such as semicolons and parentheses, so the compiler does not get angry at them, instead of concentrating on learning the programming concepts (e.g., variables, conditions, loops, etc.) in addition to logic. It turns out that "students must become masters of syntax before solvers of problems" (Malan and Leitner, 2007).

Even more, when children come to a programming course, they come with broad expectations and questions like "When are we going to make our first game (or virus)?", and those who are patient enough to learn the basics of the language would soon get frustrated when they discover that they cannot do more than simple operations on meaningless data sets using a text-based interface.

To set it in a single sentence: "Computer programming has been introduced using programming languages that are difficult to use, with proposed activities that do not connect with young people's interests and in contexts where no one has enough expertise

to provide guidance" (Resnick *et al.*, 2003). These reasons make programming with the commonly-used text-based languages inappropriate for introduction to a wide audience of children, making it hard to discover potential young programmers early.

To solve this problem while preparing for SOI, we used *Scratch* to introduce programming through the training of different divisions, and as a part of the contests for the children under 15 years old.

3. Different Programming Experience with Scratch

Scratch is a new graphical programming language that makes it easy for children to create their own interactive stories, animations, games, and arts. Coding in Scratch is much easier than in traditional programming languages: to create a script, you simply snap together graphical blocks, much like LEGO bricks or puzzle pieces. Scratch is designed to help young people (ages 8 and up) to develop essential skills such as creative thinking, clear communication, systematic analysis, effective collaboration, iterative design, and continuous learning (Lifelong Kindergarten, MIT Media Lab, n.d.).

Scratch follows the principles of making a successful software tool for kids (Maloney *et al.*, 2004). Those principles include:

- making the value and possibilities of the tool clear from the beginning;
- respecting children interests;
- the ability to create complete meaningful projects that can be shown to others;
- supporting a wide range of different types of activities, giving the ability to aim kids with different backgrounds and interests;
- the ability to get started quickly and without external help;
- the ability to learn additional features over time, and use the tool in more complex ways.

For that, Scratch is described as offering a low floor (easy to get started), high ceiling (ability to create complex projects), and wide walls (support for a wide diversity of projects) (Lifelong Kindergarten, MIT Media Lab, n.d.).

With these principles in mind, Scratch was designed with core features that include (Resnick *et al.*, 2003):

- *Building-block programming*: Programming by snapping together graphical blocks that fit in only syntactically-correct ways. This approach eliminates syntax errors (which have proven to be a major obstacle for learning text-based programming languages), allowing youth to focus on the problems they want to solve, not the mechanics of programming.
- *Programmable manipulation of rich media*: Scratch programs manipulate images, animations, movies, and sound; which offers programming activities resonant with youth interests, providing them with an opportunity to start from their own comfort zone, but then reach out to learn new things.
- *Support for multiple languages*: The possibility of translating Scratch interface to many languages (Scratch is now available in more than 40 languages) and switching dynamically among them allows children to work and think with the language

most comfortable to them, and then to talk about the knowledge they are building more creatively, which develops a sense of mastership of the recently-gained knowledge. Through our work in SOI, we could decrease the minimum grade for accepting students in Scratch courses from the 4th grade to the 2nd grade after translating Scratch into Arabic. We could also find effective 3rd-grade Scratch programmers after the translation, compared to not having any below the 6th grade before the translation.

After getting it for free, students can start programming at once by dragging blocks from eight categories on the *blocks palette* and snapping them together (only if they are syntactically correct) on the *scripts area*. The result of running programs is shown immediately on the *stage* where sprites (programmable objects) interact with each other (Fig. 1). The available blocks support various programming concepts such as loops, conditions, Boolean expressions, variables and lists (arrays) in addition to parallel execution and events (Fig. 2).

Scratch is not the first programming language intended to be used for introducing programming, and is much inspired by the ideas behind Logo. It is also not the only one today with languages like NetLogo, StarLogo and Alice. However, Scratch seems to have several advantages over others which have for example a too restricted virtual world or a high learning curve (Malan and Leitner, 2007).

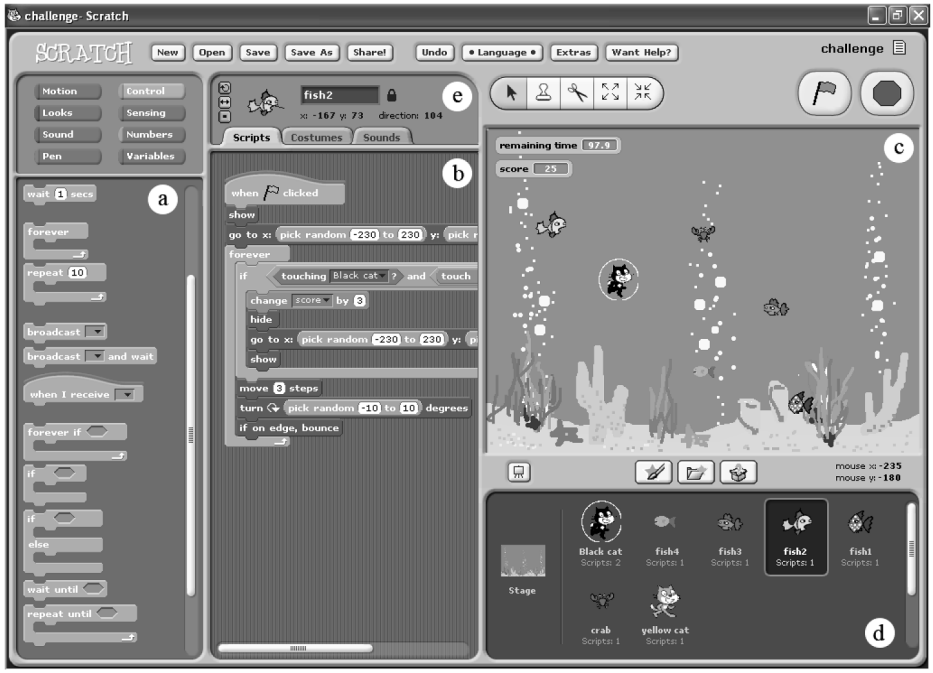


Fig. 1. Scratch interface (version 1.3.1) with the blocks palette (a), the scripts area (b), the stage (c), the sprites list – the objects to be programmed (d), and the current sprite's information (e). The shown project was one of the 1st division's tasks in SOI 2008.

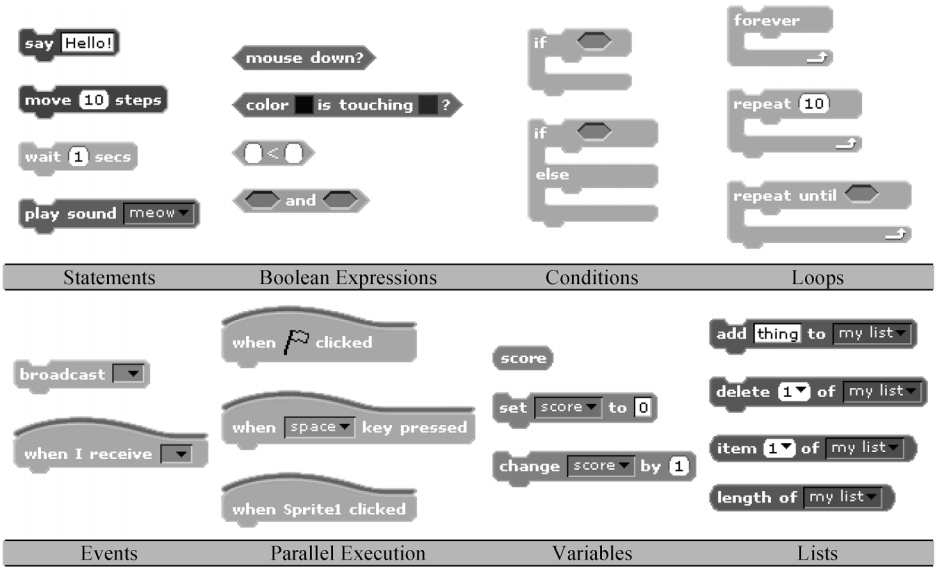


Fig. 2. Some of Scratch blocks showing supported programming concepts.

4. SOI Structure and the Utilization of Scratch

SOI consists today of three divisions depending on age: the 1st division for students under 12, the 2nd for students under 15, and the 3rd for students under 20. Scratch is used in SOI in two ways: to introduce programming before moving to C++ in all divisions, and as a part of the contest itself for the two younger divisions.

Students in the two younger divisions start their training by following a 10-session course which is open to all interested children under 15. In the first session, children learn how to move an object on the screen, to draw while moving, and to create simple loops. They do that while learning the concepts of what Seymour Papert calls “turtle geometry”, making use of children’s knowledge about their body and how they move, to draw basic figures and combine them together (e.g., drawing a square and a triangle to create a house). Through this exercise they obtain their first debugging experience (Papert, 1980). Children end this session with experiments on drawing more sophisticated geometric shapes, with some of them using nested loops (Fig. 3).

Next, children are exposed every one or two sessions to a new project. Each project has certain programming and thinking skills to be learned, and the instructor’s duty is to point out these skills when they are needed. The sessions are titled “Hunting the Parrot” or “Racing Game” instead of “Dealing with Events” or “Creating Variables”, allowing the children to learn serious issues through “hard fun”. That is providing learners with challenging activities which are deeply connected with their interests and passions (Papert, 1993). This method relieves students of feeling strangers to programming or its related science topics.

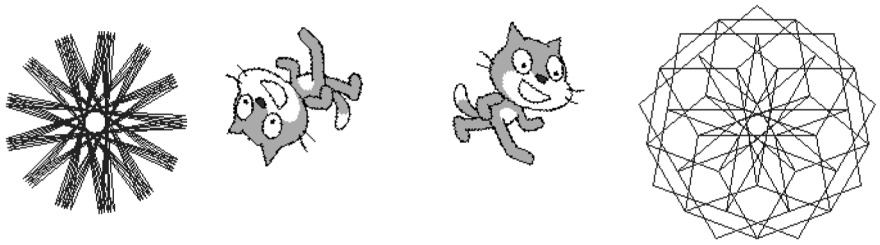


Fig. 3. Some of children creations by the end of their first session with Scratch.

Using Scratch at this stage gives children the potential to show their talents regarding computing. While instructors observe that some students are more interested in using Scratch as a design tool to create interactive media for example, others show interest in the programming process itself by using complicated and advanced programming structures and controls (e.g., using nested loops and conditions, familiarity with variables and using them in unexpected ways, etc.). By the end of the course, almost everyone has enjoyed working with Scratch, and the instructor can identify who enjoyed Scratch as a programming tool and nominate them to the next advanced course.

At the advanced course, children learn using more sophisticated techniques in Scratch, as searching and sorting lists, and make more extensive use of Boolean expressions. After that things get more formal with learning simplifying Boolean functions, expressing them as logical gates, and using truth tables. Then, programming with C++ is introduced depending on the child's previous knowledge of programming concepts using Scratch. While 1st division contestants are only required to comprehend simple programs and guess the outputs resulting from various inputs (with 15% of the total points of the contest), 2nd division contestants have additionally to complete IOI-style tasks (with all C++ tasks having 40% of the total points). Besides learning programming in C++, students learn more computer science skills and concepts such as estimating complexity and recursion, and are introduced to several basic data structures. In addition to testing those aspects of computer science implicitly through Scratch and C++ tasks, they are also theoretically tested through tasks similar to those used in the ACSL competitions (American Computer Science League, n.d.). The theoretical section takes 20% of the total points in each of the two younger divisions.

Scratch Tasks: In the contest, Scratch tasks have a considerable weight (65% of the total points for the 1st division, and 40% for the 2nd division). Contestants in both divisions face several Scratch tasks with various difficulties, and each is usually a game to be programmed. Contestants are provided with the stage and the sprites (the objects to be programmed) ready to be used so they do not waste time on painting the characters and objects of the game, and they are asked to add the behaviors (i.e., constructing the scripts) to accomplish specific missions. While a task description presents every detail about the required mission, students have also access to a working model of the mission as a Java applet, and by comparing it to their implementations they can make sure that they are on the right path.

Having a racing game for example as a task, kids are provided with the images of the car, the race route, and the obstacles; and with an explanation of the race rules: how the car behaves when driving on/outside the specified route or when it goes through obstacles, how it accelerates, how the score is calculated, and when the game ends. Contestants have to implement the scripts for doing that, with each part of the mission having a specific amount of points.

Submitted solutions are graded manually. Two graders check together each project and compare each partial behavior with the matching one from the working model. When the apparent behaviors are similar the corresponding amount of points is granted, otherwise, graders have to look for the scripts controlling the investigated behavior and estimate how far it was from attaining the desired results, and assign points according to their judgment.

Contestants in the 3rd division are prepared with IOI-requirements, and the national team is chosen from them. While they do not have Scratch tasks in their contest, they use Scratch through the preparation process which goes as follows: after selecting prospective students depending on their school records (especially in math) they are introduced to programming using Scratch for three or four sessions. At the beginning they learn fundamental control blocks (representing loops and conditions), and start then implementing a couple of projects. After being exposed to programming concepts in Scratch, they start learning C++ and other IOI-requirements. Here too, Scratch plays an important role in attracting teenagers to programming and helping instructors to distinguish prospective programmers through the way they use Scratch in.

5. Results: SOI, Scratch and the Community

As mentioned earlier, we decided to change to strategy in the SOI to aim at all children and give them the opportunity to get to know about programming in an interesting context, and Scratch was the right tool for that. Surveying students' opinions from various introductory courses showed that more than 90% enjoyed working with Scratch, though some said they felt board at some point when complicated concepts had to be explained by the instructor.

An interesting point was that about 60% of the surveyed students from both courses in the two younger divisions said they enjoyed more working with others, and they would prefer a contest where they solve tasks as a team rather than individuals. Most of the remaining 40% of the students were reported as male students.

As regards the general public, with the introduction of Scratch the national contest received more interest, with many schools asking us to train their students or to organize workshops for their instructors on preparing to the contest. Many university students were also attracted to the idea of teaching young children stuff they had not known about themselves before their university-level education.

The most important result was the change of SOI's contribution to the society. SOI is no longer a mere contest to recognize young students who are the most talented in

computer science. We think now about programming from a different perspective, an educational creative one: to help children develop themselves as creative thinkers.

When we talk to educators or parents we tell them that most students who come to SOI would not grow up to become professional programmers, but programming would still be important for everyone: it would allow them to express themselves more creatively and perfectly, help them to develop their logical thinking, and facilitate understanding the new technologies they are facing everywhere in their daily life. In other words, "the continual use of abstract thinking in programming can guide and discipline one's approach to problems in a way that has value well beyond the information technology-programming setting. In essence, programming becomes a laboratory for discussing and developing valuable life skills, as well as one element of the foundation for learning about other subjects" (National Research Council, 1999). This enhancement in role of the national contest would not have taken place without having a new tool that represents this philosophy, and this tool is Scratch.

Acknowledgements

I extend my thanks to Scratch team at the MIT Media Lab for their wonderful work. I am so grateful to my colleagues from the Computer & Automation Engineering Department at Damascus University who supported the adoption of Scratch in SOI. In particular, I would like to thank Beshr Al Nahas, Boushra Jbr, Waed Khwiess, Maya Taki and Kusay Tomeh for inspiring discussions and sharing results from their extensive experience with children and Scratch. I also thank my dear friend, Ahmad Baghdadi, for supporting me during this work and reviewing this paper.

References

American Computer Science League (n.d.). *Sample Problems*. `http://www.acsl.org/samples.htm`

Lifelong Kindergarten, MIT Media Lab. (n.d.). *About Scratch*. `http://info.scratch.mit.edu/About_Scratch`

Malan, D.J. and Leitner, H.H. (2007). Scratch for budding computer scientists. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. ACM, 223–227.

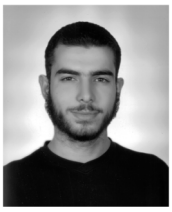
Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B. and Resnick, M. (2004). Scratch: A sneak preview. In *Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing*. IEEE Computer Society, 104–109.

National Research Council (1999). *Being Fluent with Information Technology.* National Academies Press, Washington, DC.

Papert, S. (1980). *Mindstorms*: *Children, Computers, and Powerful Ideas.* Basic Books, Inc. New York, NY.

Papert, S. (1993). *The Children's Machine: Rethinking School in the Age of the Computer.* Basic Books, Inc. New York, NY.

Resnick, M., Kafai, Y. and Maeda, J. (2003). *A Networked, Media-Rich Programming Environment to Enhance Technological Fluency at After-School Centers in Economically-Disadvantaged Communities.* Proposal to National Science Foundation.



A. Idlbi is a fresh graduate from the Computer & Automation Engineering Department at Damascus University, and has been a scientific coordinator of Syrian Olympiad in Informatics since 2006. He was also the deputy leader of the Syrian team in IOI 2007. After participating in IOI 2004 and 2005, he has worked on introducing programming to the youth. His interests include promoting usage of new technologies to provide children with better learning opportunities.

Olympiads in Informatics, 2009, Vol. 3, 26–37

Tasks and Training the Intermediate Age Students for Informatics Competitions

Emil KELEVEDJIEV
Institute of Mathematics and Informatics, Bulgarian Academy of Sciences
Akad. G. Bonchev str., block 8, 1113 Sofia, Bulgaria
e-mail: keleved@math.bas.bg

Zornitsa DZHENKOVA
Mathematical High School
2 Elin Pelin str., 5300 Gabrovo, Bulgaria
e-mail: zornica.dzhenkova@gmail.com

Abstract. The preparation of informatics competitions for an intermediate age group of school students that includes 14 and 15 years old children has become more important because of the possible introduction of new kinds of international competitions, like the recent establishment of the regional Balkan Youth Olympiad for students aged up to 15.5 years. Our paper presents the Bulgarian experience of training, where the main tools for teaching are tasks. The paper continues the previous work of the authors (Kelevedjiev and Dzhenkova, 2008a), where beginners group's tasks were arranged and classified according to some chosen set of keywords. The present study examines the tasks given at the Bulgarian national competitions for the intermediate age group during the last eight years. Some features of these tasks are discussed and compared to the tasks of the youngest age group.

Key words: tasks in competitive informatics, informatics for the intermediate age school students.

1. Introduction

In recent years (2001–2008), the Bulgarian national competitions in informatics for school students have developed as a system that includes three National Competitions (autumn, winter, and spring tournaments) and the National Olympiad with three rounds (Manev *et al.*, 2007). The students are divided into 5 age groups A, B, C, D, and E, which comprised 11–12, 9–10, 7–8, 6, and 4–5th school grades, respectively. In the Bulgarian schools, the mentioned grades correspond to 18–19, 16–17, 14–15, 13, and 11–12 years old children, respectively.

The classifications of tasks given at Bulgarian competitions for the age group E and D (youngest beginners) is presented in (Kelevedjiev and Dzhenkova, 2008a; Kelevedjiev and Dzhenkova, 2008b), where an attempt is made introducing key-words to indicate basic task features from 3 different points of view: (1) programming language elements; (2) control constructions; and (3) algorithms.

For the intermediate age group (group C) we chose as a classification principle one point of view only – which algorithmic approach is necessary to apply in order to solve the task.

2. Classification

Our research into the contents and solution methods for the tasks of the intermediate group given at the Bulgarian national competitions during the period 2001–2008 outlines that the following main algorithmic topics have been used:

a) *Counting and Combinatorics* including enumeration.

b) *Searching and Exhaustive search*. These topics include some improved searching methods as well as backtracking approaches.

c) *Geometry* with the predomination of problem on rectangular grids as well as tasks about sets of segments and rectangles with their sides parallel to coordinate's axes. Also presented are tasks involving plane and elementary geometry, and there are even tasks about solid geometry and tiling problems.

d) *Dynamic programming* with a typical example being the task about finding the longest common substring of two given strings.

e) *Graphs* with examples for tasks including finding connected components, shortest paths, matching problems, Euler cycles, cliques, etc.

f) *Digits* from a number, *arithmetic* and *number theory* including factorization, prime numbers, fractions, etc.

g) *Data structures* like stacks and queues.

h) *Others*, which refers to the tasks that are harder to classify. The reader may get some concepts of what such kinds of tasks look like by browsing the example tasks 3.8, 3.9 and 3.10, given below.

For some tasks, it is naturally to use two or more of the topics listed above for adequate classification. See also the table in the Appendix, where some topics are given together with subtopics.

3. Example Tasks

The following tasks are chosen to present several main topics and trends in the competitive informatics for the intermediate age group in the Bulgarian national competitions.

3.1. *Topic "Counting"* (*Task 2003, NOI-2, 2. Symmetric numbers*)

A given positive integer is call symmetric, if it is read in the same way from left to right and from right to left. For example: 474 and 2002. Write a program `Count` that on a given integers a, and b, computes how many symmetric integers exist in the interval $[a, b]$, $1 \leqslant a \leqslant b \leqslant 999999999$.

Example input: 9 23. Output: 3.

3.2. *Topic "Enumeration"* (*Task 2001, NOI-2, 1. Sequence*)

Consider the sequence: 1, 2, 3, ..., 9, 12, 13, ..., 19, 21, 23, ..., 98, 123, 124, ..., 987654321, where in an ascending order are arranged all positive integers that have no '0'-s in their decimal representation, and for which, all digits are distinct. Write a program **Num** that on given element x in the above sequence finds its position number k in the sequence, and vice versa, on given position number k, outputs the corresponding element in the sequence. The input file contains values of x and k. The output should contain the corresponding answers.

Example input: 21 11. Output: 18 13.

3.3. *Topic "Searching"* (*Task 2007, AT, 1. Numeric table*)

In each cell of a rectangular table of m rows and n columns, a positive integer is written. The largest value of these integers is less than 100. Three cells in the table are called neighbors, if every one of them has a common side with at least one of the others. Write a program `Maxi` that computes the largest sums which can be obtained by adding integers in three neighboring cells. On the first line in the standard input, values of m and n ($1 < m < 10$, $1 < n < 10$) are written. Each of the next m lines contains n integers to describe the corresponding table row.

Example input:

2 2
1 3
2 4

Output:

9

3.4. *Topic "Searching"* (*Task 2001, AT, 1. The best*)

Given are N ($5 \leqslant N \leqslant 20000$) positive integers, each less than 20 000, and a positive integer K ($3 \leqslant K \leqslant 100$, $K < N$). Write a program `Maxk` that finds the largest K integers among the given ones. The first two values in the input are N and K, followed by the given integers. The output should contain the largest found K integers in increasing order.

Example input: 8 4 133 74 12 38 29 56 19 31. Output: 38 56 74 133.

3.5. *Topic "Searching"* (*Task 2002, ST, 3. Interval*)

Given is a sequence of N integers $A_1, A_2, \ldots, A_N$ ($5 \leqslant N \leqslant 10000$, $-20000 \leqslant A_K \leqslant 20000$). We call an interval any subsequence of contiguous elements $A_L, A_{L+1}, \ldots, A_M$ ($1 \leqslant L \leqslant M \leqslant N$), such that for any A_K, $L \leqslant K \leqslant M$ it holds that the value of A_K is between the values of A_L and A_M. Write a program `Interval` that finds the longest interval in the given sequence. The input contains the value of N, followed by

the integers of the given sequence. The output should contain two integers L and M that determine the longest interval.

Example input: 7 2 –3 –1 2 6 4 0. Output: 2 5.

3.6. *Topic "Geometry on grid"* (*Task 2001, AT, 3. Ice piece*)

A piece of ice is depicted as a set of cells in a table of N rows and N columns ($8 \leqslant N \leqslant 200$). In the table, all cells in the first and the last rows and columns are empty. The piece of ice is starting to melt. In an hour, the ice melts in the cells that have at lest two empty neighbors (above, below, left, or right). The ice in the other cells remains unchanged. On the right-hand part of Fig. 1 it is shown what occurs after an hour, starting from the configuration depicted on the left-hand part. Write a program `Ice` that computes how many hours are needed for all pieces of ice to melt. The input contains the value of N, followed by N rows, each contained N characters '0' or '*', where '0' denotes an empty cell, and '*' denotes a cell with ice. The output should contain the number of hours needed for the total melting of the ice.

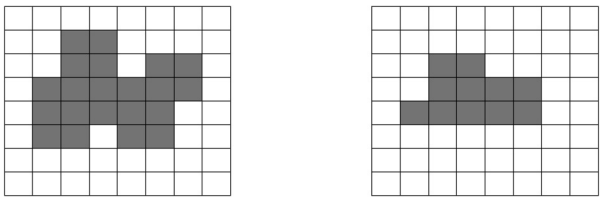


Fig. 1.

Example input:

```
8
00000000
00**0000
00**0**0
0******0
0*****00
0**0**00
00000000
00000000
```

Output:

```
4
```

3.7. *Topic "Dynamic programming"* (*Task 2001, AT, 2. Common substring*)

Given two strings A and B, each containing at least 3 and at most 250 decimal digits, we call the maximal common substring of A and B the longest string that is a substring in A and B.

Examples: For strings 112135349 and 66353, the maximal common substring is 353; the strings 112135349 and 67887 have no common substring. Write a program `Subs` that outputs the maximal common substring of two given strings.

3.8. *Topic "Others"* (*Task 2004, NOI-2, 2. Curious sequence*)

Consider the following integer sequence: 1, 11, 21, 1211, . . . , that starts with 1 and every element describes the previous one: as an example the third element is 21, because the forth element tells us one '2' and one '1' . Write a program `Selfdes` that on given input integers n and k ($0 < n < 10^9$, $0 < k < 20$) outputs the k-th element of the above sequence, where the first element of the sequence is n.

Example input: 2 4. Output: 3112.

3.9. *Topic "Others"* (*Task 2006, ST, 2. Matrix*)

Given is a matrix containing integers (between 1 and 10 000) with N rows and M columns ($2 \leqslant N \leqslant 1000$, $2 \leqslant M \leqslant 1000$). Two rows are called similar, if the first of them can be obtained by rearranging the elements of the second row. Write a program `Matrix` that outputs the maximal size of a set of matrix's rows such that no any two of them are similar. The input contains the values of N and M, followed by N lines, each containing M integers separated by spaces to describe the Nth row of the given matrix.

Example input:

```
5 4
10 1000 5 200
70 110 70 30
5 200 10 1000
4 6 11 45
70 70 30 110
```

Output:

```
3
```

3.10. *Topic "Others"* (*Task 2005, NOI-2, 2. Derivatives*)

For a given positive integer p, we denote by p' another positive integer, which is called first order derivative of p, so that the following rules hold:

1) $1' = 0$;

2) $p' = 1$, if p is prime;

3) $p' = (ab)' = a'b + ab'$, where a and b are divisors of p.

Applying the same rule to a derivative of the first order, we obtain the corresponding derivative of the second order, and analogously, we are able to compute a derivative of kth order using the $k - 1$ order. Write a program `Derive` that on given positive integers p and k ($p \leqslant 1000$, $k \leqslant 10$) computes the kth order derivative of p.

Example input: 16 3. Output: 176.

4. Trends

In Table 1, we present cumulative data about the number of tasks with assigned main algorithmic topics. The study is based on the text descriptions of the tasks (taken from (Bulgarian web portal site for competitions, 2009; Bulgarian web site for school competitions, 2009)) for the intermediate age group given at the Bulgarian national competitions in informatics during the period 2001–2008. The reader may refer to the Table 5 in the Appendix for the list of tasks.

We display diagrams to illustrate observed tendencies for monotonic or periodic trends in time appearance for the number of tasks from specific types (by means of algorithm topic involved) during the period 2001–2008 in the scene of the Bulgarian national competitions in informatics for the intermediate age group (Figs. 2–4).

As a measure of difficulty for a particular task we adopt the following 3 values:

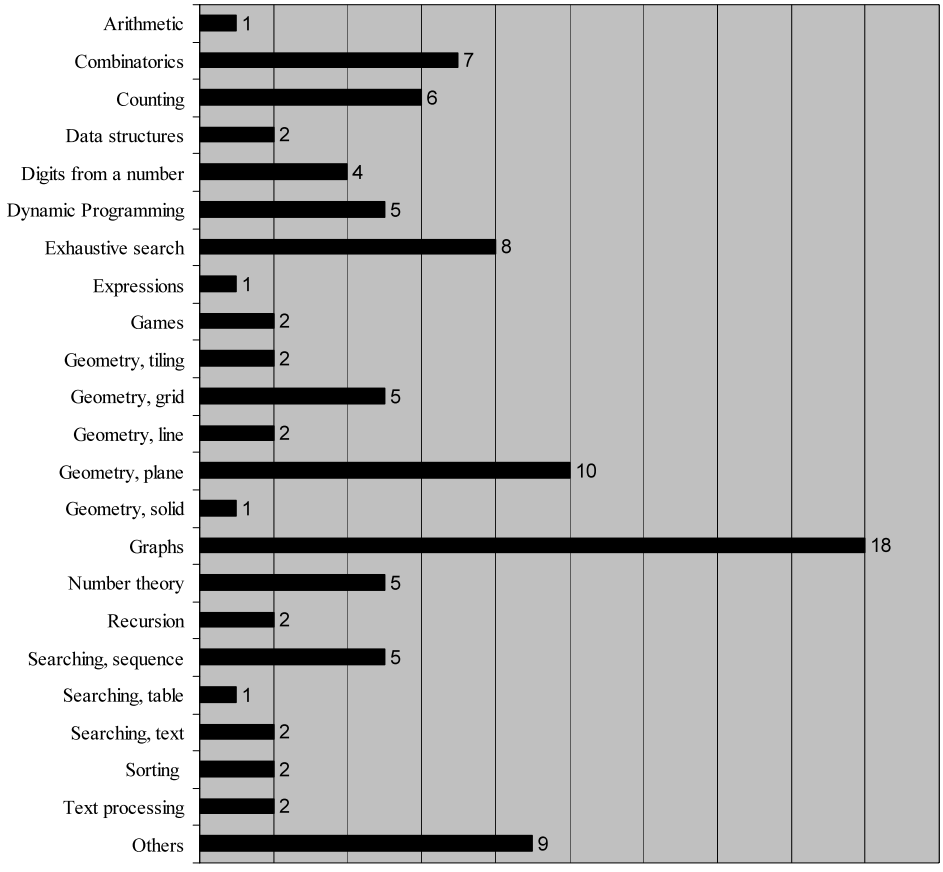
a) the percentage of students who solve the task by gaining the maximal score;

b) the percentage of students who do not solve the task at all;

c) the ratio of the above two values.

Table 1

Number of tasks with assigned main algorithmic topics



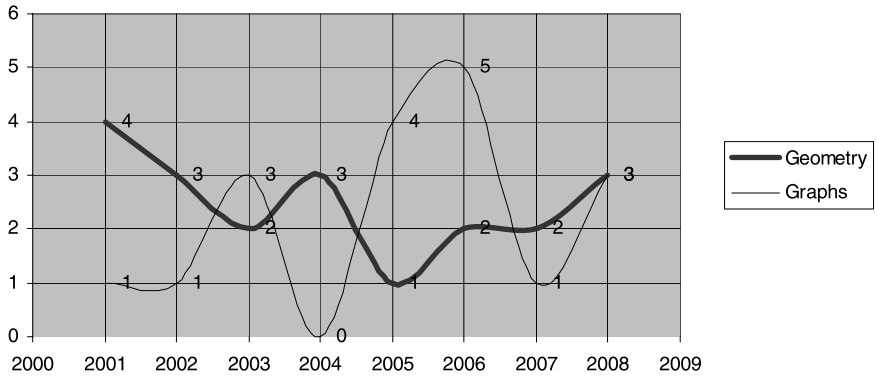


Fig. 2. Number of tasks belonging to the groups “Geometry” and “Graphs” during the years.

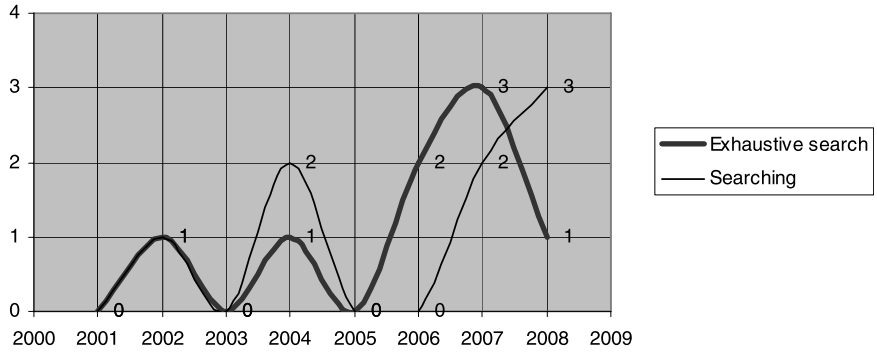


Fig. 3. Number of tasks belonging to the groups “Exhaustive search” and “Searching” during the years.

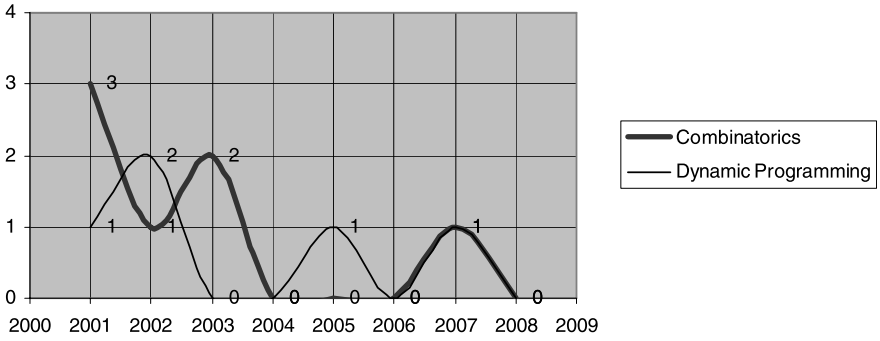


Fig. 4. Number of tasks belonging to the groups “Combinatorics” and “Dynamic programming” during the years.

Computing these values for the results of the tasks groups using the disposable data, we present Tables 2, 3 and 4.

5. Conclusions

Although the data presented in the above tables and graph samples are not statistically significant, they give us some ideas about the variety of themes. Assigning topics to each task is influenced by personal opinions, but there are some more or less steady principles for choosing them. In many cases the topic names are self-descriptive and publishing information about tasks together with these keywords is easily understandable and can help teachers in their training process with students, as well as help the authors of tasks for future competitions.

Comparing with the similar study of task classification for the beginner's group (Kelevedjiev and Dzhenkova, 2008a; Kelevedjiev and Dzhenkova, 2008b) we can observe here the underlying role of algorithms which is influenced by an increase of tasks difficulty for the intermediate group.

The value called "ratio" in the Tables 2, 3, and 4, gives a good estimation in our opinion for both measures: the difficulty of the tasks for students and the appropriateness of choosing the tasks by the organizers of the competition. As an example, the tasks from the "special" group, called "others", have the minimal value for "ratio".

Table 2

Geometry tasks: excellence and poor results

Task name	**Year**	**% excellence**	**% poor**	**ratio**
ice piece	2001	12.50	62.50	0.20
tiles	2002	0.00	32.56	0.00
dominoes	2003	2.17	60.87	0.04
movement	2003	39.29	32.14	1.22
kingdoms	2004	3.85	76.92	0.05
rectangles	2004	0.00	50.00	0.00
abc	2004	2.38	69.05	0.03
ruler	2005	40.54	40.54	1.00
rectangles	2006	12.35	33.33	0.37
darts	2006	0.00	87.50	0.00
jumps	2007	21.43	17.86	1.20
crossing	2007	3.51	91.23	0.04
segments	2008	29.03	41.94	0.69
move	2008	15.00	45.00	0.33
angles	2008	13.89	41.67	0.33
Average		**13.06**	**52.21**	**0.37**

Table 3
Graphs tasks: excellence and poor results

Task name	Year	% excellence	% poor	ratio
path	2002	4.35	52.17	0.08
teleporting	2003	4.00	84.00	0.05
expedition	2003	0.00	84.00	0.00
islands	2003	28.57	42.86	0.67
tree	2005	26.98	47.62	0.57
gnomes	2005	6.25	34.38	0.18
tour	2005	0.00	80.77	0.00
school	2005	23.33	53.33	0.44
marriage	2006	1.79	73.21	0.02
phones	2006	10.00	66.67	0.15
trade	2006	53.33	36.67	1.45
plate	2006	0.00	51.28	0.00
mate	2006	20.00	43.33	0.46
man	2007	7.14	51.79	0.14
pebbles	2008	10.53	47.37	0.22
friends	2008	11.11	44.44	0.25
triangles	2008	36.36	18.18	2.00
Average		**14.34**	**53.65**	**0.39**

Table 4
Excellence and poor results for groups of tasks

Tasks' type	% excellence	% poor	ratio
Searching	18.89	44.45	1.35
Dynamic programming	12.84	59.83	0.43
Graph	14.34	53.65	0.39
Geometry	13.06	52.21	0.37
Counting	5.51	42.06	0.25
Others	6.27	50.15	0.20
Total Average	**11.77**	**52.59**	**0.39**

Appendix

Table 5 presents all tasks given at the Bulgarian competitions for the intermediate age group during the years 2001–2008. In the column "Competition", the names of the autumn, winter and spring tournaments are abbreviated as AT, WT and ST, respectively, and the second and third rounds of the Bulgarian National Olympiads in Informatics are denoted by NOI-2, and NOI-3, respectively.

Table 5

Tasks given at the Bulgarian competitions for the intermediate age group during the years 2001–2008

No	Year	Competition	Problem No	Problem Name	Method
1	2001	AT	1	best	Searching, Sorting
2	2001	AT	2	common substring	Dynamic Programming
3	2001	AT	3	ice piece	Geometry, Grid
4	2001	NOI-2	1	num	Combinatorics, Enumerating
5	2001	NOI-2	2	line	Geometry, Plane
6	2001	NOI-2	3	maze	Graphs, Cycles
7	2001	ST	1	three	Geometry, Plane
8	2001	ST	2	war	Games
9	2001	WT	1	table	Combinatorics
10	2001	WT	2	ab	Digits
11	2001	WT	3	line	Geometry, Plane
12	2001	WT	4	sum	Combinatorics
13	2002	AT	1	path	Graphs
14	2002	AT	2	frequencies	Counting, Text
15	2002	AT	3	digit	Searching, Sequence
16	2002	NOI-2	1	table	Counting, Tables
17	2002	NOI-2	2	tiles	Geometry, Plane
18	2002	NOI-2	3	concert	Dynamic Programming
19	2002	ST	1	molecules	Combinatorics
20	2002	ST	2	bed	Dynamic Programming
21	2002	ST	3	Interval	Exhaustive search
22	2002	WT	1	primes	Number theory
23	2002	WT	2	para	Geometry, Solid
24	2002	WT	3	path	Geometry, Grids
25	2003	AT	1	lighting	Combinatorics
26	2003	AT	2	movement	Geometry, Grids
27	2003	AT	3	islands	Graphs, Grids
28	2003	NOI-2	1	Egypt fractions	Number theory
29	2003	NOI-2	2	count	Counting, Enumeration
30	2003	NOI-2	3	dominoes	Geometry, Tiling
31	2003	ST	1	expedition	Graphs, Matching
32	2003	ST	2	expressions	Expressions
33	2003	ST	3	teleporting	Graphs, Shortest path
34	2003	WT	1	shuffle	Combinatorics
35	2003	WT	2	tail	Data structures, queue
36	2003	WT	3	young	Sorting
37	2004	AT	1	supermarket	Data structures, queue
38	2004	AT	2	code	Searching, Text
39	2004	AT	3	temperature	Searching, Sequence
40	2004	NOI-2	1	enemy	Arithmetic, Fractions
41	2004	NOI-2	2	self-description	Others, Recursion
42	2004	NOI-2	3	abc	Geometry, Plane
43	2004	ST	1	grades	Others
44	2004	ST	2	squares	Exhaustive search
45	2004	ST	3	primes	Number theory
46	2004	WT	1	barcode	Others, strings
47	2004	WT	2	rectangles	Geometry, Plane
48	2004	WT	3	kingdoms	Geometry, Grid
49	2005	AT	1	hyperlinks	Text processing

To be continued

Table 5 (continued)

No	Year	Competition	Problem No	Problem Name	Method
50	2005	AT	2	school	Graphs, Connected components
51	2005	AT	3	numbers	Counting
52	2005	NOI-2	1	tree	Graphs
53	2005	NOI-2	2	derivative	Others, Computations
54	2005	NOI-2	3	simple	Counting
55	2005	ST	1	gnomes	Graphs, Paths
56	2005	ST	2	two	Games
57	2005	ST	3	row	Others, Sequences
58	2005	WT	1	ruler	Geometry, Line
59	2005	WT	2	run	Dynamic Programming
60	2005	WT	3	tour	Graphs, Cycles
61	2006	AT	1	right primes	Number theory
62	2006	AT	2	plate	Graphs, Euler cycles
63	2006	AT	3	darts	Geometry, Plane
64	2006	NOI-2	1	rectangles	Geometry, Plane
65	2006	NOI-2	2	bmax	Digits
66	2006	NOI-2	3	delmin	Number theory
67	2006	NOI-3	1	mate	Graphs
68	2006	NOI-3	2	trade	Graphs, Paths
69	2006	NOI-3	3	min	Others, Tables
70	2006	ST	1	socks	Others, Tables
71	2006	ST	2	matrix	Exhaustive search
72	2006	ST	3	marriage	Graphs, Matching
73	2006	WT	1	Morse code	Others, Strings
74	2006	WT	2	Puzzle	Exhaustive search
75	2006	WT	3	Phones	Graphs, Paths
76	2007	AT	1	numerical table	Searching, Table
77	2007	AT	2	puzzle	Exhaustive search
78	2007	AT	3	names	Counting
79	2007	NOI-2	1	similar	Others, Tables
80	2007	NOI-2	2	minimal	Exhaustive search
81	2007	NOI-2	3	inequality	Exhaustive search, Arithmetic
82	2007	ST	1	man	Graphs, Connected components
83	2007	ST	2	segments	Dynamic Programming
84	2007	ST	3	jumps	Geometry, Line
85	2007	WT	1	sequence	Searching, Sequence
86	2007	WT	2	crossing	Geometry, Plane
87	2007	WT	3	necklace	Digits
88	2008	AT	1	mirror	Searching, Sequence
89	2008	AT	2	pebbles	Graphs, Games
90	2008	AT	3	bank	Combinatorics, Coding
91	2008	NOI-2	1	friends	Graphs, Connected components
92	2008	NOI-2	2	angles	Geometry, Plane
93	2008	NOI-2	3	justify	Others, Text processing
94	2008	NOI-3	1	move	Geometry, Grid
95	2008	NOI-3	2	trans	Digits
96	2008	NOI-3	3	sword	Exhaustive search, Strings
97	2008	ST	1	disk	Others, Recursion
98	2008	ST	2	segments	Geometry, Tiling
99	2008	ST	3	subsequences	Searching, Sequence
100	2008	WT	1	future	Others, Long integers
101	2008	WT	2	triangles	Graphs, Clique
102	2008	WT	3	context	Searching, Text

References

Manev, K., Kelevedjiev, E. and Kapralov, S. (2007). Programming contests for school students in Bulgaria. *Olympiads in Informatics International Journal*, **1**, 112–123.

Kelevedjiev, E. and Dzhenkova, Z. (2008a). Tasks and training the youngest beginners for informatics competitions. *Olympiads in Informatics International Journal*, **2**, 75–89.

Kelevedjiev, E. and Dzhenkova, Z. (2008b). Competition's tasks for the youngest school students. In *Mathematics, Informatics and Education in Mathematics and Informatics*, *Spring Conference of the UBM*, Borovetz.

Bulgarian Web Portal Site for Competitions in Informatics. Retrieved 1 March 2009 from `http://infoman.musala.com`

Bulgarian Web Site for School Competitions in Informatics. Retrieved 1 March 2009 from `http://www.math.bas.bg/infos`

E. Kelevedjiev is a research fellow in the Institute of Mathematics and Informatics at the Bulgarian Academy of Sciences. His field of interests includes algorithms in computer science, operation research, digitization techniques, etc. He is a chairman of the Bulgarian National Committee for Olympiads in Informatics; leader or deputy leader of the Bulgarian teams for many IOI's and BOI's.

Z. Dzhenkova is a teacher in the Mathematical High School in Gabrovo, Bulgaria. She is coauthor of a manual for beginner's training in competitions and olympiads in informatics. Her field of scientific interests includes education in informatics and information technology; leader of school student teams and instructor in competitive informatics.

Olympiads in Informatics, 2009, Vol. 3, 38–59

Infrastructure for Contest Task Development

Rob KOLSTAD

USA Computing Olympiad
15235 Roller Coaster Road, Colorado Springs, CO 80921, USA
e-mail: kolstad@usaco.org

Abstract. The USA Computing Olympiad annually conducts six internet-based computer programming competitions, each including three to four algorithmic tasks in each of three divisions. Coupled with the training camp competitions, a typical annual USACO 'problem budget' approaches 75 new tasks at three distinct levels of difficulty.

In order to exploit a distributed coaching staff, USACO developers created and evolved the web-based 'probgate' problem-development system to speed production of acceptable quality programming contest tasks that are machine-gradable, well-accepted, and yield no or few complaints, regrades, or requests for clarification.

This paper describes each of the major modules and shows how they are used to simplify, speed up, and automate administration of contests regularly accessed by more than 1,000 students.

Key words: programming contests, automation, contest test data, contest data validation, contest preparation, contest automation, automatic grading.

1. Demographic Background

The USA Computing Olympiad is the USA's designated organization for training and selecting students to compete in the International Olympiad in Informatics.

In July, 2009, the United States had a population of just over 307 million people. The USA includes 9.8 million square kilometers (about half the size of Russia or South America; about twice the area of the European Union). The 3,500 mile (5,600 km) width of the contiguous 48 states complicates gathering students for training camps or on-site contests.

The USA has just over 29,500 high schools with a total of about 15M students; about 3.2M will graduate in 2009. Additionally, 0.3M high school students are 'home schooled' outside the traditional public and private school system.

Until recently, USA high schools included all levels of students and declined to utilize 'tracking' to segregate them by performance. The past decade has seen increased availability of 'magnet,' 'gifted and talented,' and 'charter' schools that offer focused or deeper programs for academically inclined students. Additionally, many high schools offer 'Advanced Placement,' 'Honors,' or International Baccalaureate coursework that enables students to self-select into more challenging curricula (thus segregating academically focused students from their potentially less-diligent peers).

All public secondary schools in the USA are governed by local (usually city-wide) school boards that are generally overseen by state-level school boards. The federal government uses the new No Child Left Behind legislation and a few other techniques to encourage a small number of national standards.

Of the USA's 15+ million eligible students for the IOI, about 150–175 (0.001%) participate in USACO contests (47% in bronze division, 36% in silver division, 16% in gold division).

2. Task Development Background

The manageable number of students coupled with the natural fit of computer programming contests to the internet enables the USACO to offer six annual online programming contests (plus a qualification round) to USA students. Each contest has three divisions; each division contains 3–4 tasks, usually 9–10 tasks total. Annually, these contests consume 55–60 tasks.

The selection process thus garners a large amount of data for each student, so much that the student's worst performance can be dropped from statistics (most folks have a bad day once in a while) while still yielding meaningful means, trends, and so on. The monthly contests also enable tracking of students' improvement over time (and correlation with their participation in the USACO training site).

The USACO invites 16 (or so) of the best students to participate in the USA Invitational Computing Olympiad, an on-site selection contest and training week for the IOI. This camp has evolved to include six contests (four shorter, two longer) of 3–4 tasks each, an additional $\sim$20 tasks.

In addition, the USACO administers bonus contests (e.g., over the New Year holiday break) and special training events that also consume programming contest tasks.

All in all, the USACO creates and then consumes 75 or more fairly high-quality tasks per year, all developed by a volunteer coaching staff of about a dozen coaches located around the world.

The original implementation of the USACO grader enabled automated administration and grading of tasks in a contest environment. Task development and deployment proceeded manually, including manual setup and installation of tasks into the contest system.

Manual setup is well-known to be error-prone, and the USACO was no exception. Bugs and other issues were exacerbated by the just-in-time development effort, which often resulted in tasks (including text, solutions, and test data) being created the night before a contest (especially at camp with its very high rate of task consumption).

This development methodology not only created tremendous pressure and stress on the coaches but resulted in errors in the task statements, occasional tasks that were not as solvable as believed, test data that failed to meet the task specifications, and contest environment integration errors. Coupled with the stress and schedule pressure, the coaches felt the overhead of task development and contest integration was high.

3. Task Development Paradigms and Related Work

The existence of online, web-available, automated grading systems coupled with the advent of ever-cheaper hardware and widely-available reliable free software for development has fostered development of many grading systems around the world. Evolution of such systems usually proceeds from a web page that enables compilation and execution to a 'sandbox' to contain the behavior of submitted programs (imagine the security implementations of letting anyone send source code to run on your site). The compile/run system acquires a database of users in order to differentiate whose tasks are getting which results. The major issue of scaling comes into play as the abilities of a single CPU are exceeded.

Then the real challenge emerges: continuous development of new tasks and administration of contests on a schedule. Once scaling solutions and dedicated staff (volunteers, usually) are identified, the second-order issues can be attacked. Some country's organizations (e.g., those of Poland's Diks *et al.* (2008), Canada's Gordon Cormack, and The Netherlands' Tom Verhoeff (2008)) have devoted tremendous attention to the challenge of using black-box testing to ascertain the quality of submitted solutions to contest tasks.

The USACO's focus has been on providing a large, growing, and widely-available set of contests to the largest possible audience. While judging the quality of task ideas and text is difficult and extremely subjective, the works by the previously-named authors provide some guidance for creation and judging of test data. By the criteria of high-quality black-box testing, USACO's test data does not measure up to the standards of the IOI (and probably those of Diks, Cormack, and others). However, problem setter Richard Peng's efforts through the 2007–8 and 2008–9 seasons have increased test data quality dramatically. The data still meet the simple objective criterion: Do the test data enable fair and defensible differentiation among the contest competitors, preferably throughout the range of skills in a given division? For USACO, mostly because of the task-weighted scoring technique (also seen in use at IOI2008 in Egypt), both the top-, mid-, and low-level participants (in the Silver and Gold divisions) are differentiated quite well across a 1,000 point scale.

Diks *et al.* (2008) shared their Task Preparation Process at 2008's Olympiads in Informatics conference. Using their methodology and manpower, their tasks end up with superior 'model' solutions, better test development (black-box testing), and dramatically more complete and in-depth analyses when compared to USACO's ongoing efforts. Their series of training booklets is further evidence of their development of insightful and thorough solution analyses.

Verhoeff (2008), also at 2008's Olympiads in Informatics conference, shared his proposal for the Peach Exchange Format. Compared to this paper's work, it has a much more formal set of roles (and their descriptions) for developers, potentially much better 'background' (often mathematical) information for task development, 'notes' for developers (see Verhoeff, 2008; p. 202), and a formal specification for a file structure layout for task exchange.

USACO's paradigms more strongly emphasize throughput in the task development area at the expense of detailed analyses and extremely thorough black-box testing.

4. Requirements

One advantage of conducting a dozen IOI-level and IOI-size (for half of them) programming competitions per year is the rapid accrual of realistic statistics on the kinds of mistakes that creep into contest tasks. After one particularly stressful camp during which Senior USACO Coach Greg Galperin created a huge task matrix on a whiteboard to chart task development throughout the week, it became clear that automated scripts could not only aid in task development and debugging but also in contest administration system debugging. The decision was made to create a web-based task development system (which would not only integrate the scripts but also provide a distributed user interface). In hindsight, this is the next logical step in developing a contest administration framework once a grading system with a secure 'sandbox' is stable.

Experience made clear the requirements for tasks, which include rules like these:

- Text must be clear and complete.
- Text must be easily editable for repairs.
- Text must be easy to export to both web and to paper – since online contests use one method and on-site contests use the other.
- It is desirable for text to have multiple coaches' ratings and comments – this helps determine which tasks belong on contests (vs. training or discard).
- Test data must be valid (i.e., conform to task's data explanation).
- It must be easy to add, remove, reorder test data.
- The system must ease creation of data validators (Verhoeff, 2000) – the data validator was one of the most effective schemes for eliminating complaints and regrades.
- It is desirable to create data validator mechanically – the ability to have a program read a task and deduce the input data format ensures a uniform, proper presentation of input requirements in addition to saving time.
- The system must support all styles of IOI tasks (simple answer, multiple answer, reactive, output-only, programmatic grader, etc.).
- The system must enable configuration of submission feedback (categories like: case is required to be correct for submission to proceed, report right/wrong for one or a set of tasks, or no feedback at all).
- The system must enable configuration of multiple-tests per case.
- Task must be solvable within the contest constraints.
- The system must provide automated evaluation of proposed solutions in contest environment (running in any other environment does not "prove" solvability).
- The system must provide comparison of answers from various solutions/solvers – this must be mechanical since answer sets can be large and similar.
- It must be easy to enter answer-graders and format-checkers.

Requirements for task development include:

– easy navigation,
– ability to create pool of tasks to use in various contests,
– ability to see status (including coach ratings) of each task,
– ability to see status of each contest,

- fast navigation for rating many tasks – fast, easy navigation encourages coaches to rate more tasks,
- convenient contest setup and parameter management,
- archiving scheme so tasks can smoothly move to training or other places – one-click archiving makes for easy reuse of tasks,
- collection of task analysis text,
- ease of set up and take down for contests, including automatic start/stop.

5. Chosen Task Development Paradigms

The task development paradigm centers on the task as a 'unit' with this (expandable) list of components:

- numerical problem ID,
- task author,
- task analysis status,
- task editing status,
- count and list of task solutions (and solvers) with solution-agreement status,
- data validator and its status,
- test data and its feedback type,
- task ratings,
- task algorithm,
- estimated time for 'ideal student' to solve,
- task abbreviation/short name,
- task title,
- task text.

The task's text includes these items (which are stored and manipulated as separate entities):

- full task name,
- short task name/abbreviation,
- assigned owner for task,
- presentation order,
- division,
- author,
- date,
- text body,
- input format (including broken-out start/stop line numbers),
- sample input,
- input explanation,
- output format (including broken-out line numbers),
- sample output,
- output explanation,
- the task's test data representation includes:

- each of the test cases,
- memory limit,
- default time limit,
- grader program for task output (optional),
- format checker for task output (optional),
- auxiliary filename and contents (e.g., for external dictionary),
- data validator program,
- scoring tables for aggregates (when score depends on more than one test case),
- per-test case time limit.

Notes for test data are consolidated with notes in the task discussion.

Additionally, the development system's paradigm includes the ability to set new task-development instances (for training tasks, university training/homework tasks, personal task development gateways, etc.).

6. Current Implementation

The system supplies several web pages to manipulate the task's components:

- the login page,
- the main status and navigation page,
- the text presentation, ratings, algorithm, solution time, ratings, and comments page,
- the problem solution page,
- the test data management page,
- the contest management page.

Subsections below detail each of these.

The task development infrastructure is part of an integrated training/contest administration system written almost entirely in the perl programming language (the 'sandbox' being the only exceptional program; C felt like a better tool for its implementation). About two dozen scripts (including administrative programs) total about 10,000 lines of perl for the contest development infrastructure. At the time of implementation, perl was one of only a few implementation tools mature enough to implement the system effectively. Today, one could probably use any of several languages and/or tools. The author is an expert perl programmer, so it was a natural choice at the time. The system runs on FreeBSD and Linux (with a migration to a pure Linux environment planned for the near future).

7. Login/Authentication

The current implementation of the task development system is accessed via a login page on the web; see Fig. 1. A database keeps track of both coaches and competitors' usernames and passwords (over 105K as of 1 May 2009).

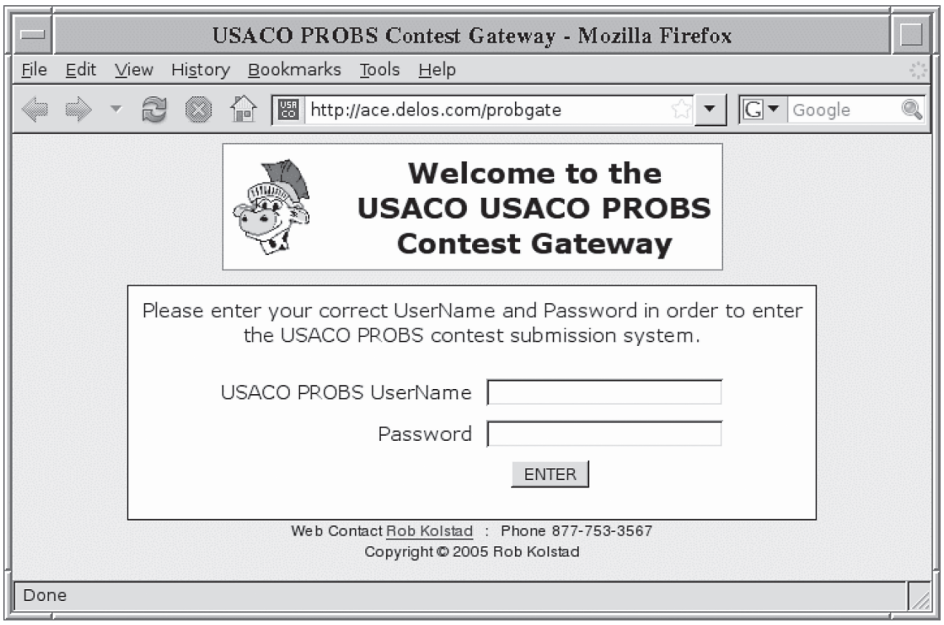


Fig. 1. Authentication page for problem editing gateway.

8. Status and Navigation

The main status and navigation page shows all the managed tasks in groups that have been assigned by the coaches. Typical groupings are for upcoming contests, unassigned tasks of a similar type, and unassigned tasks of a given division (e.g., gold, silver, or bronze).

Fig. 2 shows a small fraction of the top of the main status navigation page. The very top includes some submit buttons used for group naming and other functions:

- Submit new problem: plots a task input page that collects each of the sections mentioned above for a given task.
- Make new group: creates a new group with the name supplied in the input box.
- Rename group: Changes the name of the group whose checkbox has been clicked to the name supplied in the input box.
- Help: plots a page of explanatory text.
- Manage Contests: brings up the contest setup page.
- Replot this page: redisplays the page with all updates.
- Submit selector: activates a filter that displays only those groups whose name matches the regular expression entered in the Selector box (dramatically reduces traffic when working on a single contest).
- Enlarge and Shrink: Increases or decreases the typeface size.

The headings show up next followed by the first (of many) group headers. The group header rows include the header (and its checkbox) with the group name and some commands in addition to a single line status message previously entered by a contest director.

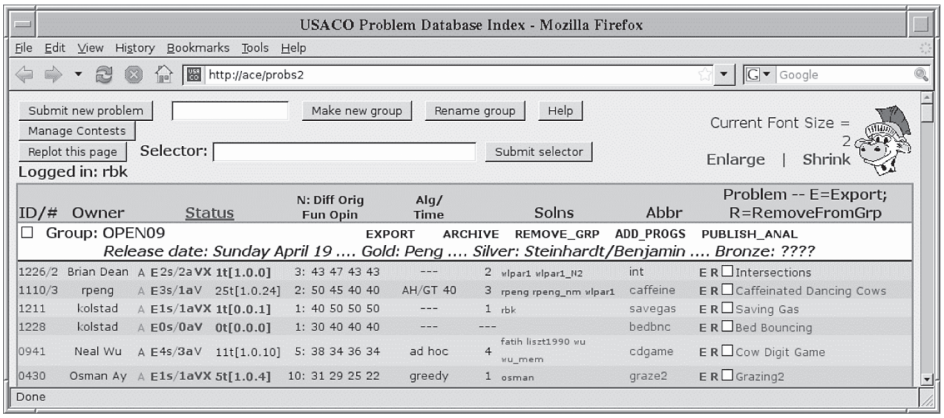


Fig. 2. Status and navigation page.

The per-group commands include:

- EXPORT – Copy this group's tasks to the contest whose name matches the group name.
- ARCHIVE – Copy this group's tasks to the training task development system and remove them from this page.
- REMOVE_GRP – Move the tasks in this group to the section below for unallocated tasks and delete the group name and its status message.
- ADD_PROBS – Move the tasks whose checkbox has been marked into this group.
- PUBLISH_ANALYSIS – Retrieve the analyses entered for this group's tasks and copy them to the web for display with task data.
- ADD_NOTE – Shown only for special users when the contest 'note' is empty; bring up note-text editing page.

The rest of the page shows the name and status for each of the tasks managed by this particular task management system (in this case, 188 tasks are summarized below). Each line contains a set of color-coded information about this task:

- Task Number: The numerical ID of the task. Clicking the number displays the text-editing page.
- Task Number/Order: The presentation order number of the task, if it exists, is displayed after the task.
- Owner: The next column shows the task's owner.
- Status: A: The color of the letter A indicates the existence of the analysis (red for missing, blue for present). Clicking the A brings up the analysis entry page.
- Status: E: The color of the letter E indicates whether the task has been final-edited (bold-blue for not-edited, black for edited).
- Status: Xs/Ya: X is the number of solutions submitted; Y is the number of solutions in agreement with the master solution. Bold-blue indicates insufficient solutions or agreements. Clicking this item brings up the solution submission and status page.

- Status: VX: Bold-blue V means the validator is not submitted; presence of the bold-blue X indicates the existence of test data that does not pass the validator's tests.
- Status: Xt[A.B.C]: X is the number of test cases that have been submitted. A is the number of tests required to be passed for submission to succeed; B is the number of tests whose results will be shown when a submission is made; C is the number of tests that will be run normally. Clicking this item brings up the test data submission page.
- N: Diff Orig Fun Opin: These five items represent respectively the number of ratings entered by coaches as their evaluation of the task: the average difficulty rating, the average originality rating, the average fun rating, and the average overall opinion. The integer averages are the product of ten and the mean of all the 1–5 rating entries.
- Alg/Time: These two items show the algorithm abbreviation (as entered on the problem display page by a knowledgeable coach) and the ideal student's solution time in minutes (also entered on the problem display page).
- Solns: The number represents the number of solution programs submitted; the subsequent login IDs are the names of the solutions submitted (or, sometimes, names assigned by the submitter).
- Abbr: The short problem name, its abbreviation.
- E: Clicking the E exports the single task to the contest administration system.
- R: Clicking the R removes the task from its group and moves it to the unassigned task list at the bottom of the page.
- Name: The long name (title) of the task.

Problem creation and contest configuration proceed in parallel. To set up a contest, a coach enters a canonical name (e.g., NOV09, DEC09, JAN10, etc.) and clicks **Make new group**. He then ticks the checkboxes of tasks to move to that group and clicks the **ADD_PROGS** tag on the group's header.

Coaches can read the task's text, enter ratings, enter solutions, enter test data, add a validator, and submit output format checkers and graders. Any replot of the status change shows all updates that have been entered.

When a problem is almost "ready" (i.e., it is final-edited, has at least two solutions that agree with the master solution, has a validator that passes all the input data, and has at least eight test cases), the background color around the "Status" fields changes to light green. When the analysis is entered (generally not required until the contest is almost over), the background shade changes to dark green.

A contest is fully ready when a dark green stripe runs completely through the "Status" column's fields.

9. Text, Comments, and Ratings

Fig. 3a shows the top part of a typical problem presentation page. It leads off with navigation links and then prominently displays the problem's name and owner (and also

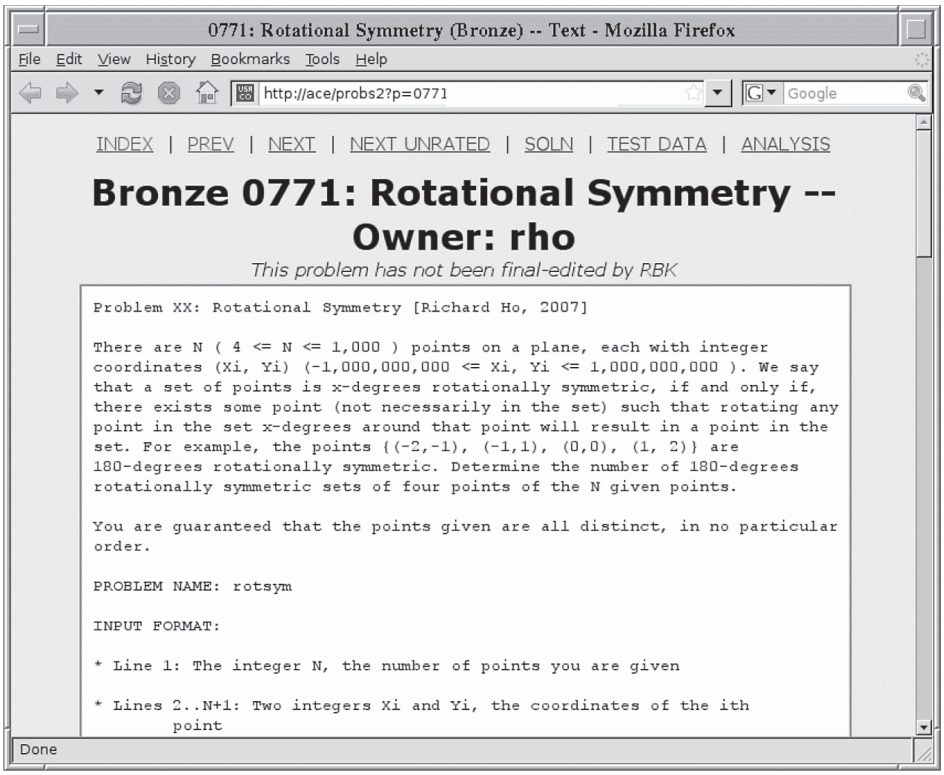


Fig. 3a. Top of text presentation page.

algorithm and solution time, if available). The editing status appears in italics just below the task name. The problem text itself then appears.

The problem text includes many components. The largest component is usually the problem statement itself which, by USACO convention, includes the limits on all the various input parameters.

Following the problem text, the problem's short name (abbreviation) appears with an easy-to-find capitalized heading.

Fig. 3b shows the input/output section of this task's display. The input format still uses the old-fashioned "Line 1, Line 2, ..." notation instead of the more modern "Next 3 lines..." notation. The starting line and optional ending line are entered for each "section" of input as part of the input process. These fields are used by the mechanical validator creation routine to create loops that check the input.

The "Input details" section is optional, as is "Output Details."

Fig. 3c shows the next set of information from the text page. It includes navigation buttons to:

- edit the problem's text, name, division, or input/output parameters,
- show recent changes to the text,
- navigate to the test data or solution pages,

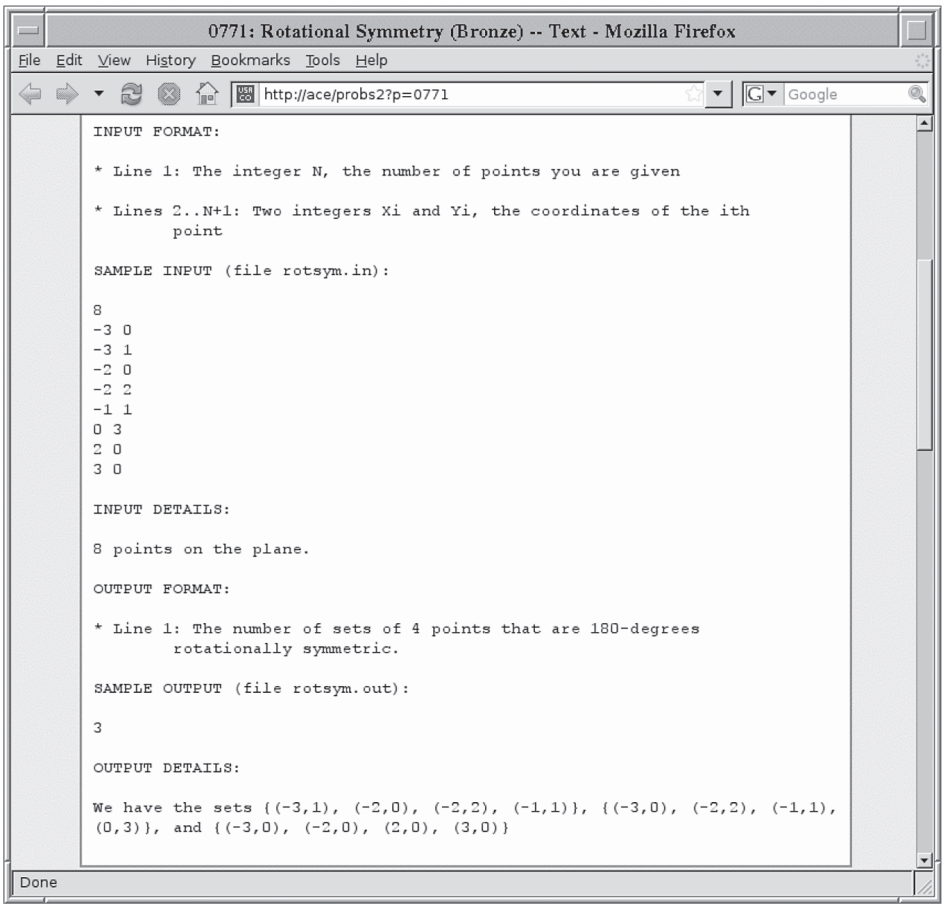


Fig. 3b. End of text presentation on text presentation page.

– remove the task altogether,
– remove all entered data except the task's text,
– mark the task's editing as approved (only available to special directors),
– a shortcut to submit a solution (presumably created while reading the text above).

Finally, Fig. 3d shows the rating and comment box along with the coaches' ratings. Some coaches are empowered to enter an estimated "Solution Time" and "Solution Algorithm" which then appear on the summary page. When a large pool of tasks is available, a coach can scan solution times and algorithms to create a contest very quickly.

10. Text Editing

The text editing page is mostly self-explanatory (see Fig. 4a) with fields to enter:

– the full task name,

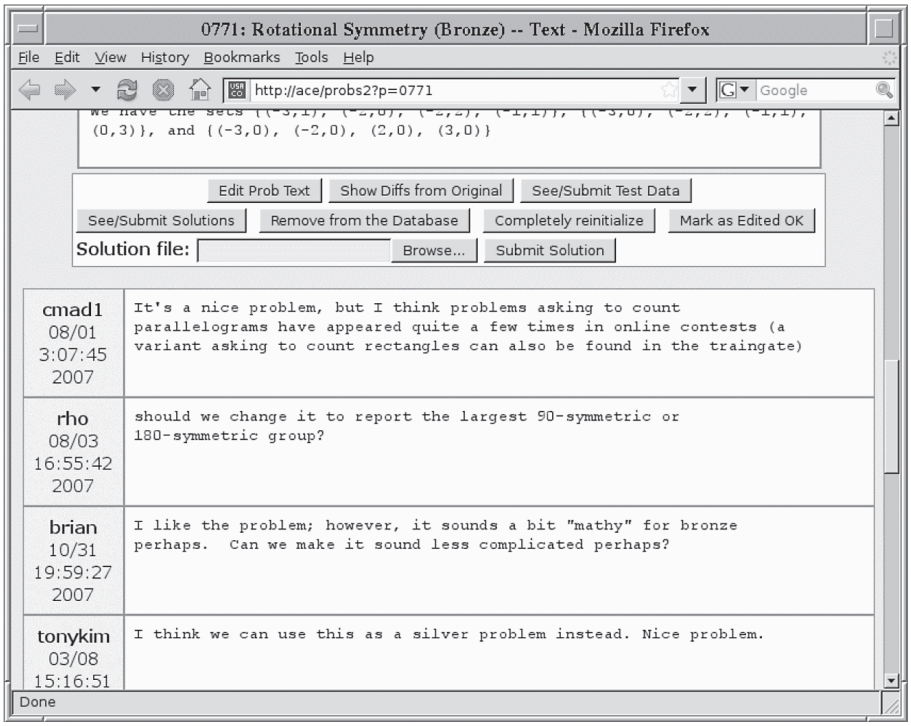


Fig. 3c. Text page navigation and comments.

- the short task name (abbreviation),
- the task's owner,
- the optional task presentation order,
- the task's division,
- the task's author,
- the task's date,
- when the task was used,
- the task's text.

The page's display then continues with the self-explanatory input and output specifications; see Fig. 4b. This figure has been edited to remove blank fields and white space. Terminology has evolved over time; the "Input Explanation" is displayed to competitors as "Input Details."

11. Solution Page

Coaches spend most of their time on the "Solutions Page" honing their solutions and ensuring the answers are correct. The page enables saving of multiple solutions for each coach so they can compare various algorithms and techniques.

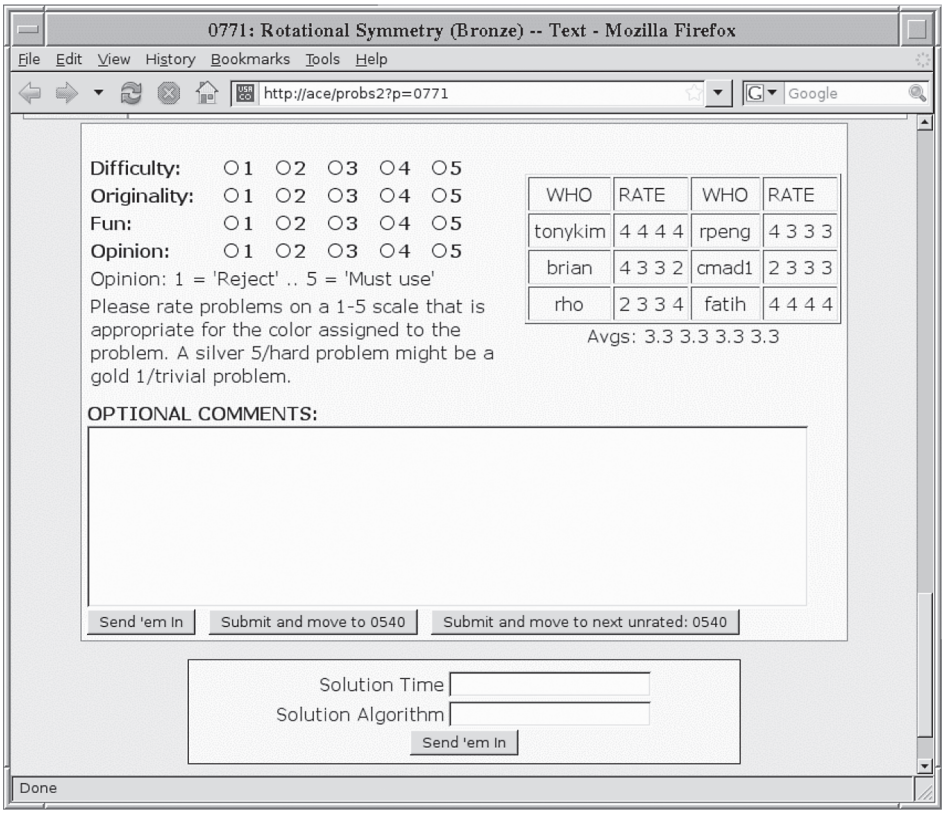


Fig. 3d. Ratings, comment box, and rating navigation.

Fig. 5 shows a typical solution page. The top line provides navigation. The table shows the status of all submitted solutions:

- Solver is the solution's name, most often that of the submitter.
- Size shows the length of the solution (or the number of cases in the official solution).
- Available displays which cases have been run (and, in the case of disagreeing answers, red denotes those that don't agree).
- Actions include deleting the solution, (re-)running it, copying it to the official solution (along with its answers), or 'X' to mark the available case solutions as invalid (in order to re-run them).

Auxiliary comparison buttons enable selected runs to be displayed either in full or as 'diff's. One can run or re-run all solutions.

The **Solution Agreement Table** displays smiling (or not-so-smiling) faces to display which programs are in agreement. The faces are clickable to show the differences in the output files. New, unrun test cases display as gray numbers in the 'Available' column; the Solution Agreement Table displays question marks until the solutions are run on the new data (generally with a single click to "Run all Solutions").

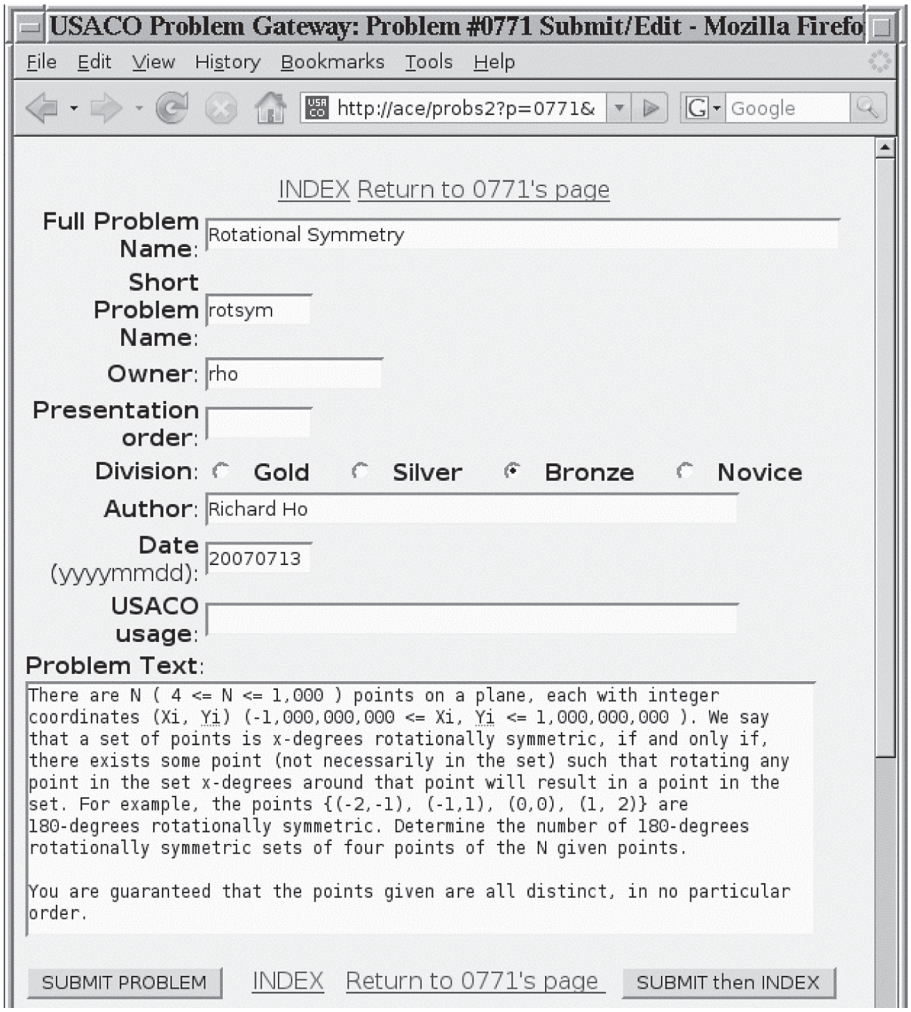


Fig. 4a. Text editing: parameters and body.

The **Best CPU Time per Case** table is self-explanatory and is used to ensure that tasks have plenty of "head room" for slightly inferior algorithms or implementations – or the effect of other programming languages (most notably, Java).

Finally, submission boxes enable sending in of new solutions and renaming existing solutions.

12. Test Data

The test data submission page handles not only the submission and display of test data but also (see Fig. 6a):

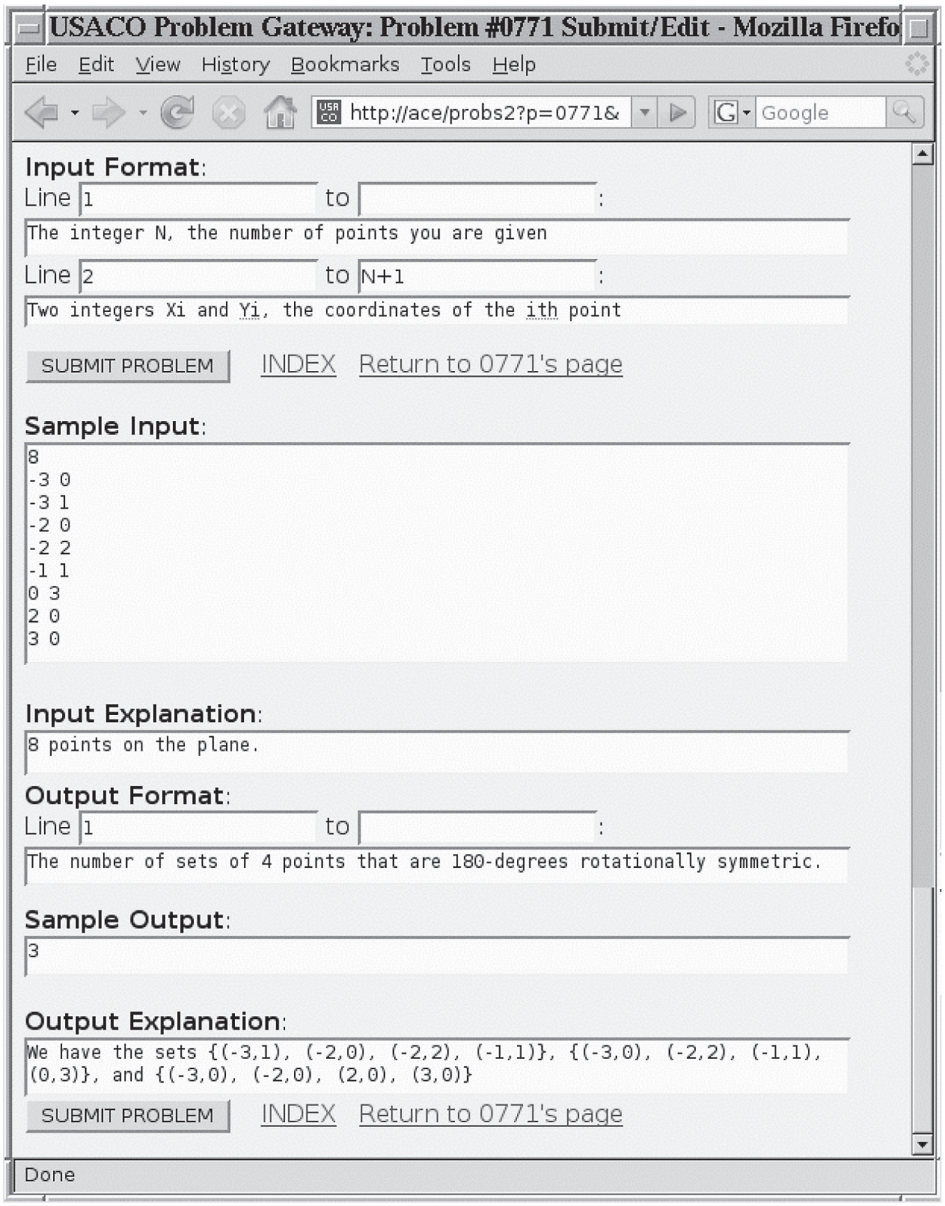


Fig. 4b. Input and output specifications.

- memory limit,
- default time limit (optionally overridden on a case-by-case basis),
- output grader program uploading,
- format checker program uploading,
- auxiliary filename and actual file contents uploading,

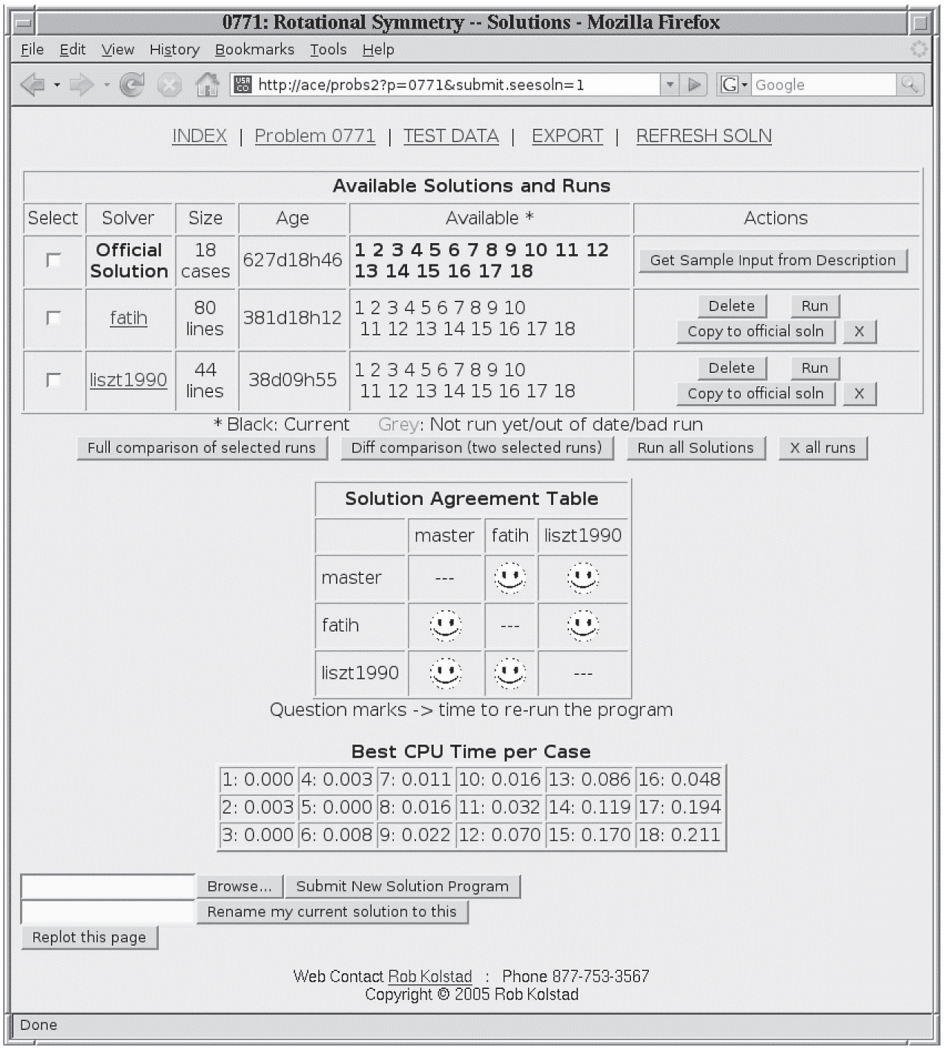


Fig. 5. Solution submission and display page.

- validator uploading.

The test data display includes:

- Test#: The case number of this test (not always sequential in early stages of test data development).
- Grade Type: Whether this test case is run upon submission (with or without a requirement that the test be passed for successful submission) or later for results.
- #Lines in and Lines out: size of the test case.
- CPU Lim: The override value for the maximum CPU time.
- Input and Output: summaries of the first lines of input and output for the test case.

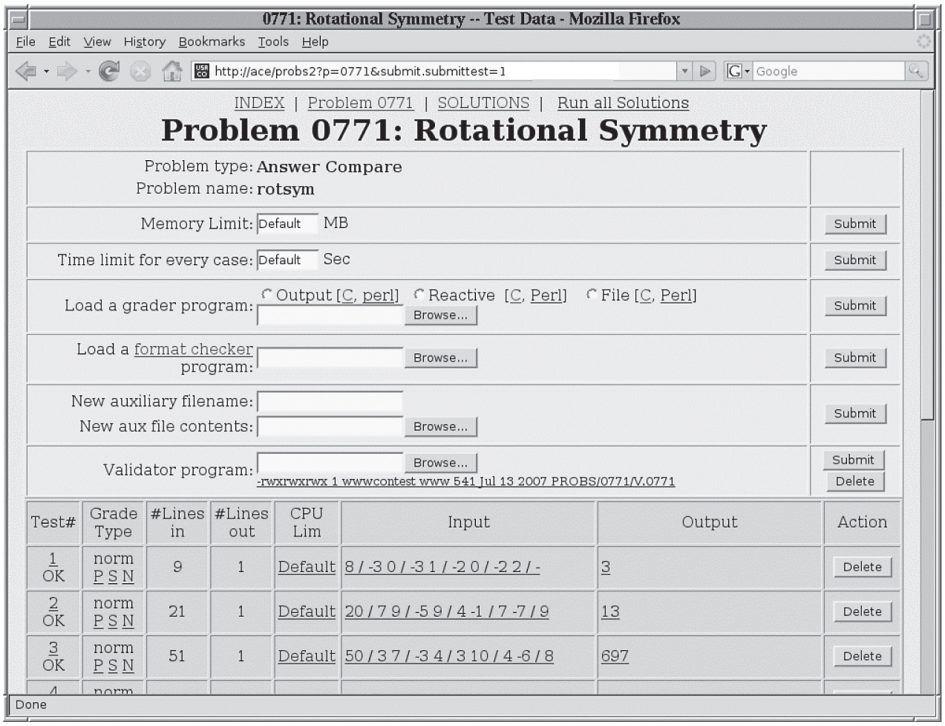


Fig. 6a. Test data programs and display.

- Action: A button to delete the test case.

The bottom half of the test data submission page (see Fig. 6b) includes:

- Delete ALL Test Files: The big gun.
- Re-Pack the case numbers: To make the case numbers be sequential (often used after test case deletion).
- Re-Run Validators: Does the obvious thing. This is handy when someone is editing the validator in the filesystem instead of via the web page.
- Set IOI dispositions: Obsolete functionality that marked every other case as "show result during submission."
- Create aggregates: Invokes functionality that enables combining of individual test cases into super-cases.
- Download all test cases: Creates any of four portable formats for moving all the test cases around (or editing) in a single file (which is reloadable below).
- Move case ...: Self-explanatory.

The bottom of the page is where the actual uploading or entry of test cases occurs. The fields are self-explanatory.

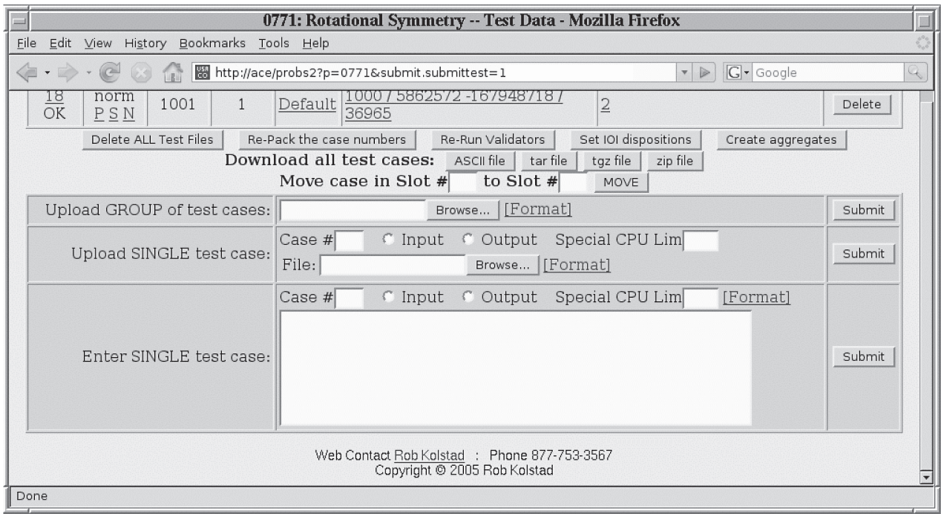


Fig. 6b. Test data submission.

13. Contest Management

The contest management page (see Fig. 7a) has a relatively boring presentation but is, of course, the center of administrative power. It uses the paradigm of cloning old contests to create new ones (modifying the ioiconfig.perl file from one contest to update it with the new contest's information). Each contest is stored in a contest directory with its configuration, problem texts, submissions, and so on. Functionality includes:

- Edit: displays the configuration editing page (see below).
- Clone: Creates a new contest with the name entered in the box near the top by cloning a previous contest.
- Enable logins/DISable Logins: Toggles the configuration to allow non-contest-directors to login or not.
- Activate/DEACTIVATE: Manipulates a table that shows which contests are available for logging in. Without available login, participants cannot even see information about the contest and cannot login.
- Enable Analysis Mode/Disable Analysis Mode: Toggle the configuration file variable that specifies whether the contest is analysis mode (vs. regular competition mode).

14. Contest Configuration

The contest configuration page enables input for the all the standard contest configuration parameters:

– contest name,

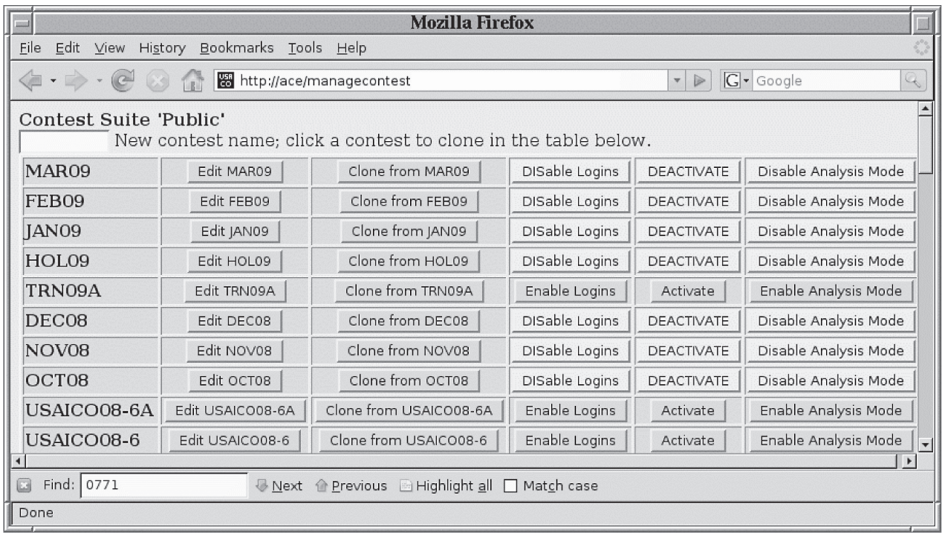


Fig. 7a. Contest management.

– brief description,
– contest duration (for each division),
– contest start and stop times,
– team, proctor, division, file submission, and analysis mode parameters,
– age break for legal participants,
– contest type and time zone.

15. Current Status

The system has been in production for half a decade with complete functionality as described in this paper. The system currently manages 998 tasks (most from the past; a few hundred for the future) and has 96 users who can edit ‘probgate’ (the contest problems). Another half dozen instances support other organizations. An additional 61 users (some of them the same as the previous users) are working in ‘traingate’ to create more training materials for USACO. During March of 2009, 21 different coaches performed 3,343 different actions (task reading, task submission, etc.) in the system.

Contests are ready on-time, with requests for translation sent out 3–4 days before the competition begins. A typical season of six contests now yields – for the entire year – perhaps half a dozen queries for clarification (usually from new competitors) and, in general, no more than a single grading protest.

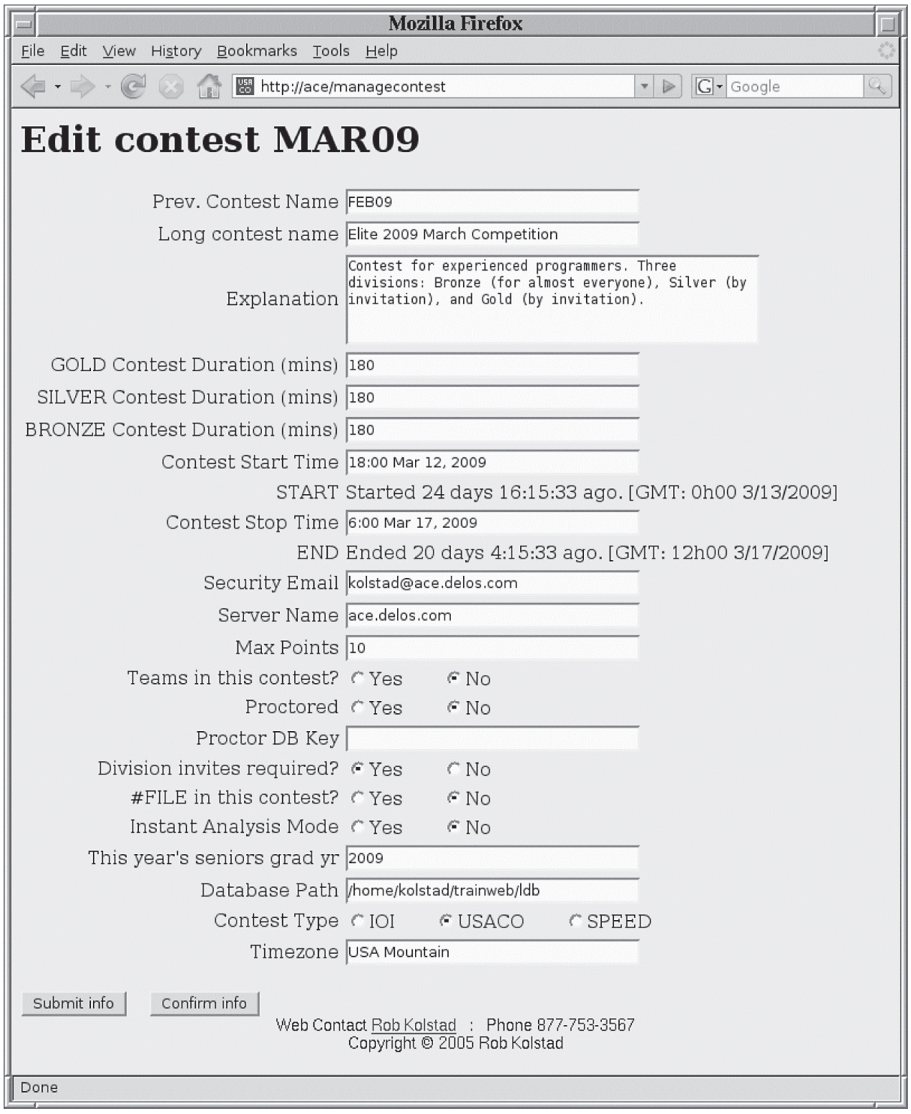


Fig. 7b. Contest configuration options.

16. Current Deficiencies

As systems do, the problem management system requires a facelift and internal update to bring it into the 21st century:

- Task timing (a function of the Linux kernel) is not reliable enough.
- The web display for contest tasks is too large; only filtering saves it from being too unwieldy to be used.
- Database navigation has efficiency, paradigm, and speed issues.

- The web displays are not as stylish or as functional as they need to be to be truly professional.
- The navigation among pages is inconsistent and poorly formatted.
- It is not yet possible to create one-off custom contests for trainees.
- It is not yet possible to create 'programming bees.'
- The older training pages are not integrated into this system and need to be.
- Logging is not always performed properly.
- The interface for 'confirmers' is not complete.
- The system should perform automatic tracking of 'solution times' for coaches who develop tasks.
- Processing the end of a contest still contains several manual steps; these should be integrated into the contest management subsystem.

17. Conclusion

While an IOI consumes six tasks (or perhaps 8–12, depending on one's point of view) in a year, the USACO consumes more than six dozen. The continuous development cycle has exposed many of the important factors in creating quality contests:

– data validation,
– multiple solvers,
– contest implementation without schedule pressure,
– automated contest configuration,
– running the USACO contest schedule would not be possible without this level of automation.

Our most important item for near-term future development is custom contests for training using old contest tasks.

The USACO system has proven successful for half a decade and complements the growing set of other systems available on the internet and contributes to the diversity of contests and environments that enable the field of competitive informatics to continue to grow and mature.

References

Diks, K., Kubica, M., Radoszewski, J., Stencel, K. (2008). A proposal for a task preparation process. *Olympiads in Informatics*, **2**, 64–74.

Verhoeff, T. (2000). *RobIn for IOI I/O*, unreleased draft, August.
`http://olympiads.win.tue.nl/ioi/twg/robin/Doc.txt`

Verhoeff, T. (2008). Programming task packages: peach exchange format. *Olympiads in Informatics*, **2**, 192–207.

R. Kolstad consults for the TAEUS corporation on intellectual property issues surrounding software and hardware. Rob is the head coach of the USA Computing Olympiad, and is also the head judge at the Pikes Peak Regional Science Fair. Rob earned his PhD in software from the University of Illinois at Urbana-Champaign after completing an MSEE at Notre Dame and undergraduate BASc degree from Southern Methodist University. Rob earned his Ph.D. in software from the University of Illinois at Urbana-Champaign after completing an MSEE at Notre Dame. His undergraduate BASc degree from Southern Methodist University was among the first computer science bachelor's degrees offered in the United States.

Olympiads in Informatics, 2009, Vol. 3, 60–66

Moe – Design of a Modular Grading System

Martin MAREŠ

Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
e-mail: mares@kam.mff.cuni.cz

Abstract. Programming contests often employ automatic grading of submitted solutions, but frequently in an ad-hoc way. This article describes our attempt at creating a modular and flexible grading system called *Moe*, which is not tied to the specifics of a single contest.

Key words: automatic grading, Moe, mo-eval.

1. Introduction

Many programming contests in the world, including the IOI, employ automatic grading of the contestants' solutions. This is accomplished by running them on batches of input data and testing correctness of the output. Time and memory limits are usually enforced during the process, which allows to take the efficiency of the algorithm into account.

During the last two decades, a multitude of such evaluation systems have been developed, but most of them are tied to the specifics of a single contest and they are usually neither publicly available nor well documented. This leads to waste of effort by re-implementing the same things over and over, and also to repeating mistakes somebody else has already made and understood.

This variability can be observed even within a single contest. Most of the IOI host countries have used grading systems developed for their national contests, often extensively modified to handle the different conditions of the IOI. Similarly, the regionals of the ACM ICPC differ in their contest environment between regions.

It was repeatedly suggested that a single unified contest system should be built and used at all the major contests. However, it has been argued that the differences between contests would make the system too complicated and cumbersome to maintain. Also, diversity and the ease of experimentation with new ideas are too valuable to lose.

In our previous paper (Mareš, 2007), we have proposed a modular grading system instead, that is a set of simple, yet flexible modules with well defined roles and interfaces. A contest organizer would then pick a subset of the modules and use them as building blocks of his contest environment together with other locally developed parts. This allows us to minimize the effort without sacrificing flexibility.

This paper is a report on the state of development of our modular system called *Moe*.

2. The Design of Moe

Moe is a successor of our previous system (MO-Eval), originally developed for the contests we organize, and subsequently generalized. Its primary target is Linux, but most modules should run on any POSIX-compliant operating system. The sole notable exception is the sandbox, which is intimately tied to the details of the OS and of the CPU architecture.

The source code of Moe and the (so far incomplete) documentation are available under the terms of the GNU General Public License from the website mentioned in the references.

2.1. *Available Modules*

Moe currently contains the following modules:

- *sandbox* – runs the contestant's solution in a controlled and secure environment, limiting its execution time, memory consumption and system calls. It is the most mature part of Moe, already in use at several contests (see below for the list of applications). The current version of the sandbox requires a recent Linux kernel on the i386 architecture. A port to the amd64 architecture is near completion, but it requires fixing several security issues in the kernel ptrace interface first, as noted by Evans (2009).
- *judges* – a set of utilities for comparing the solution's output with the correct answer at the given level of strictness, which can range from ignoring white-space characters to ignoring the order of lines or all tokens. The judges are built upon a library of functions for strict and fast parsing of text files, which can be easily used as a basis for custom judge programs. This part is also mature.
- *evaluator* (also known as *grader*) – this module controls the whole grading process. It calls the compilers, the sandbox and the judges as described by configuration files. It handles multiple types of tasks: e.g., batch, interactive, open-data. We have a reliable implementation in Bourne shell, but it is unnecessarily hard to maintain, so we plan to rewrite it in a higher-level language (most likely Perl or Python) in near future.
- *queue manager* – since the evaluation of tasks at a big contest must be performed in parallel, the queue manager can be used to maintain a queue of solutions and distribute them among graders running on multiple machines. Finished, but needs lots of polish to make it usable by a wider audience.
- *submitter* – handles submitting of solutions by contestants and passing them to the evaluation system. While the submitter has been designed for competitions which do not use a web-based interface, it can be also used as a clean interface between the web modules and the rest of the system. Working, but needs revision.

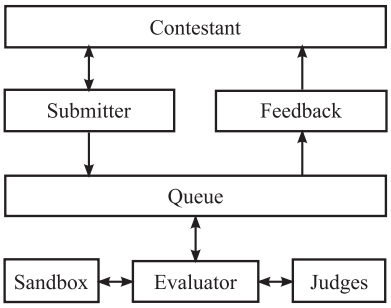


Fig. 1. Typical interconnection of modules.

- *test suite* – the correctness and security of most modules is critical to the success of the contest. We are building a test suite which tries to cover all known edge cases and attacks on system security. The test cases contain unit tests for various functions, regression tests for historic bugs, and security tests inspired by the analysis of possible attacks by Forišek (2006).

In near future, we plan to add several new modules:

- *feedback* – processing of evaluation results and generation of various reports (e.g., score tables, sheets with detailed feedback for contestants).
- *scoring* – multiple small modules for advanced scoring strategies, like gradual time limits (programs near the time limit get partial score) and approximation tasks (points are awarded depending on the precision of the result).
- *supervisor* – controlling several hundred computers at a big contest is no easy task. This module will maintain a queue of jobs (usually snippets of shell scripts or request to distribute files) and schedule their parallel execution at given machines, possibly using a tree-like topology to spread the load.

2.2. *Programming Languages*

Most parts of the system are free of assumptions on the programming languages used in the competition. The only dependencies are in the evaluator and in the sandbox. The evaluator needs to know how to compile and run the programs, which can be of course easily configured. The sandbox has to be set up to permit the system calls issued by the language's runtime libraries.

Pascal (compiled by either GPC or FPC), C and C++ (compiled by GCC) are supported since the first release of Moe and there are no known problems with them, except for the occasional slowness of Pascal I/O libraries.

C# compiled by Mono turned out to be more problematic, because its runtime libraries use various unusual system calls, which are forbidden in the default settings of the sandbox. Also, Mono spawns multiple threads even for trivial programs. We have patched the runtime environment to avoid threading and the most problematic system calls, while the rest is handled by a language-specific configuration of the sandbox.

We have added experimental support for Haskell recently, compiled by GHC.

Adding further compiled languages should be easy, as long as their runtime environment behaves in a sane way. Interpreted languages are also supported, but except for interpreting the C# byte code, they did not receive much testing yet.

2.3. *Interfaces*

To facilitate interconnection of the modules, each of them is written as a stand-alone program with basic parameters passed as its command-line arguments. In addition to this, we accompany every solution by a *status file*, which collects all information related to grading of the solution. All modules can contribute their bits: the submitter records which programming language has been used, the sandbox reports the execution profile (time, memory, number of system calls), the evaluator informs about the results of each test case, while the queue manager annotates on which grading machine has served the task and how long did the task wait in the queue.

We have resisted the industry-standard temptation of using XML as a one-size-fits-all format. Instead, the status file has the form of a simple structured text file, inspired by the Lisp S-expressions, but strictly divided to lines and typeless. This is very easy to generate and parse, especially in shell scripts which form the glue joining the evaluator modules. Moreover, it has the same expressive power as XML, so the files can be converted back and forth if an application wishes.

A typical example looks as in Fig. 2.

As the status files are a relatively recent addition to the Moe infrastructure, they are not yet fully supported by all modules, but they probably will be at the time of publication of this article.

```
task:pyramid                The name of the task [2pt]
lang:c                      Language of the solution
test(                       A section for a single test
    id:1                    Name of the test
    time:0.375              Run time in seconds
    mem:1355776             Memory consumption in bytes
    points:0                Points awarded
    status:RE               Status code
    message:Runtime error   Explanatory message
    exitcode:1              Program exit code
)
test(                       A section for another test
    id:2
    ...
)
...
```

Fig. 2. An example status file.

2.4. *Configuration*

We pay much attention to the configurability of the whole grader. Most aspects of the evaluation process are controlled by many configuration variables, whose values are gathered from several sources and these are stacked one onto another in a manner similar to the Cascading Style Sheets.

First of all, there is a top-level configuration file with global defaults. These can be overridden by per-task configuration files. The per-task configuration usually involves things like setting of time and memory limits, but it can modify any variable if needed. This way, we can extend the compiler options if the task requires a special library to be linked, or change the sandbox options to permit otherwise disallowed system calls. Finally, the settings can be modified for individual test cases.

A fragment of configuration can be also restricted to a specific programming language. This allows compilation commands, settings of the sandbox, or rules for interpretation of runtime errors to be defined differently for each language.

Moreover, the values of the settings are expanded before each use, which includes interpolation of references to other configuration variables. For example, this feature is commonly used to make the compilation command for each language refer to a variable with user-defined compiler switches, or to substitute time and memory limits to the list of sandbox options.

The configuration mechanism also serves as a core of our format of task packages. Essentially, the role of a task package can be played by an arbitrary directory, as long as it contains an appropriately named configuration file. Its variables then point to other files within the same directory, which contain the test cases, judges, model solutions, and other components of the task. As this file naming convention is usually fixed within a single competition, it is customary to use the configuration stacking to inherit most variables from a top-level configuration file and let each task care of its differences from the defaults only (see Fig. 3 for an example).

We are following the discussion on standardization of task packages initiated by Verhoeff (2008), but the Peach format proposed there is too restrictive for our use. We want to

```
IO_TYPE=file                          this is a standard batch task with file I/O
TESTS="1 2 3 4 5"                     names of test cases (files n.in, n.out)
POINTS_PER_TEST=1                     points awarded per test case
TIME_LIMIT=5                          time limit per test case in seconds
MEM_LIMIT=4096                        memory limit per test case in KB
TEST_4_TIME_LIMIT=10                  override for a specific test case
EXT_pas_MEM_LIMIT=8192                Pascal solutions get twice as much memory
OUTPUT_CHECK='$PDIR/judge             this task does not have unique output, so use
$TDIR/$TEST.in                        a problem-specific judge
$TDIR/$TEST.out
$TDIR/$TEST.ok'
```

Fig. 3. An example task configuration file.

keep the assumptions about task formats in Moe at minimum. It is quite possible, though, that a single format will be recommended as a default in the future, when some consensus is reached. Also, Moe's flexibility makes it quite easy to import tasks from other systems.

3. Applications

Our system is still under construction and many parts need lots of improvements. It is however lacking mostly in features and documentation, rarely in reliability. Its design and implementation have already proven itself at multiple occasions.

First of all, it serves as a basis of the competition environment at the Czech national olympiad in last six years. We also regularly use the same environment at the Czech–Polish–Slovak preparation camps whenever they are held in Czech republic. As the organizers of these camps enjoy experimentation with new types of tasks, we have used Moe modules in many previously unexpected ways. A nice example was awarding points by playing a tournament between all pairs of submitted solutions. (More such "hacks" are described in our previous paper.)

A new version of Moe, which pioneered the submitter, has been developed for the Central-European Olympiad in Informatics 2007.

Since 2007, Moe is used as the evaluation back-end of CodEx, which is an automated system for checking students' programming assignments at the Faculty of Mathematics and Physics of Charles University in Prague. To handle this load, we have created queue manager module. As Moe contributes only a small piece to a big puzzle here, we have introduced the status files to make data interchange easier. The CodEx version was also the first to support C# and Haskell.

Recently, the organizers of IOI 2009 have decided to use Moe's sandbox as a part of their contest environment.

4. Future Plans

We plan to continue the development of Moe in the forthcoming years. First of all, we wish to fill all the gaps, especially in the documentation, and make the use of status files systematic. We also want to rewrite the evaluator module and add the feedback, scoring, and supervisor modules as described in Section 2.1.

Further plans include extension of the submitter module to provide on-line feedback, which will make it directly usable in contests like the ACM ICPC. Also, we would like to support more operating systems, architectures and programming languages.

As most free-software projects, Moe is developed by volunteers. Any bug reports, suggestions for new features, patches to the code or any other contributions are heartily welcome. Also, if you use parts of Moe in your contest, please let us know, we are interested in your experience.

References

Evans, Ch. (2009). Linux syscall interception technologies partial bypass. Security advisory CESA-2009-001. Retrieved 27 February 2009 from:
http://scary.beasts.org/security/CESA-2009-001.html

Forišek, M. (2006). Security of programming contest systems. In *Informatics in Secondary Schools, Evolution and Perspectives*. Vilnius, Lithuania.

Mareš, M. (2007). Perspectives on grading systems. *Olympiads in Informatics*, **1**, 124–130.

Mareš, M. *et al.* (2009). *The Moe web site.*
http://www.ucw.cz/moe/

Verhoeff, T. (2008). Programming task packages: peach exchange format. *Olympiads in Informatics*, **2**, 192–207.

Martin Mareš is as an assistant professor at the Department of Applied Mathematics of Faculty of Mathematics and Physics of the Charles University in Prague, a researcher at the Institute for Theoretical Computer Science of the same faculty, organizer of several Czech programming contests, member of the IOI Scientific Committee and a Linux hacker.

Olympiads in Informatics, 2009, Vol. 3, 67–73

Using a Linux Security Module for Contest Security

Bruce MERRY
ARM Ltd
110 Fulbourn Road, Cambridge, CB1 9NJ, United Kingdom
e-mail: bmerry@gmail.com

Abstract. The goal of a programming contest grading system is to take unknown code and execute it on test data. Since the code is frequently buggy and potentially malicious, it is necessary to run the code in a restricted environment to prevent it from damaging the grading system, bypassing resource constraints, or stealing information in order to obtain a better score.

We present some background on methods to construct such a restricted environment. We then describe how the South African Computer Olympiad has used a Linux Security Module to implement a restricted environment, as well as the limitations of our solution.

Key words: linux security module, programming contest, sandboxing.

1. Introduction

The South African Computer Olympiad (SACO) is an annual programming contest, whose final round is modelled on the International Olympiad in Informatics (IOI). In particular, contestants submit solutions in source form, in a variety of languages, to an online submission system. The online submission system provides a variable amount of feedback immediately to the contestant (typically, the results of some sample case). Final scores are only made available after the contest, by running each solution on a variety of test cases.

While we are not aware of any contestants having attempted to cheat by submitting malicious solutions, we must nevertheless protect the integrity of the grading system by running them in a *sandbox*, or locked-down environment. In some ways, this is easier than sandboxing a general application, since solutions are only intended to have a limited interaction with the execution environment (read a file, do some computation and write a file), and so it is possible to use a more tightly locked environment than would be possible for running applications that had legitimate needs to access the display, keyboard, network and so on. However, we must also enforce the rules of the competition (such as execution time limits), so some additional work is required.

The following section lays out our requirements in more detail. Section 3 discusses a number of approaches to sandboxing. In Section 4 we describe which approach we chose and how we implemented it. Conclusions are presented in Section 5.

2. Requirements

2.1. *Security Requirements*

Usually, sandboxing is used to prevent applications from either damaging the host environment or leaking information about it (for example, by stealing passwords or credit-card numbers from a browser). For a programming contest, there are some other limitations that must be taken into account. Below is a list of some of the goals we aimed to meet:

1. Resources (particularly, CPU time and memory) must be restricted, to prevent a malicious (or more likely, incorrect) solution from blocking the whole system.
2. Resource constraints must be accurate. For example, if a program has a one-second time limit, it is acceptable for it to run for 1.5 seconds and terminate normally, as long as it is possible to determine that the time limit was in fact breached. It is also important that programs are not able to make it appear that they used fewer resources than they actually did.
3. Programs must be limited to a single thread. This prevents programs from taking advantage of multi-core systems to get more computation done in the time available, as well as simplifying a number of other implementation details.
4. Programs must not be permitted to spawn other processes. This prevents, for example, a problem involving mathematical computation from launching an external program like `bc` or `Octave` to do the computation.
5. Programs must not be able to communicate with the outside world (for example, through TCP/IP sockets). This prevents the solution from offloading processing onto a separate, possibly faster system, as well as ensuring secrecy of test data.
6. A single run of a program must not be able to communicate with any other run (for example, by leaving a file with precomputed primes in the filesystem, or using inter-process communication).

2.2. *Other Requirements*

While security is obviously the primary goal, some potential solutions may be deemed unusable for other reasons. Other requirements include

1. The setup time must be minimal. We do not have a large cluster of machines for evaluation, so throughput is a concern.
2. It must not significantly impact performance. This is so that CPU time limits are not affected by the security mechanism.
3. It must allow common operations by the standard libraries of the compilers used (GCC for C and C++, FreePascal, Python and Java). Some of these libraries do I/O using system calls that can also be used in ways that violate the security policies above, so it is unacceptable to block these system calls unconditionally.

3. Background

3.1. *System Call Interception*

In Linux and indeed most desktop/server operating systems, applications do not have direct access to devices such as ethernet ports or disk drives, and can only access memory that is allocated to them. In order to interact with devices or other applications, it is necessary to use *system calls*. A system call is similar to a function call, but transfers control to the operating system kernel which undertakes these actions on behalf of the user.

One approach to restricting the actions that an untrusted application can take is to intercept these system calls. Linux provides the *ptrace* system call, which allows one process to be notified about any system calls made by another. The controlling process can then override or suppress system calls that the untrusted application should not be allowed to make according to the security policy.

This interception process adds a small amount of overhead, because for each system call there is an additional context switch from the kernel to the process that makes the decisions and back again.

System call interception is also prone to security holes if not implemented correctly, mostly due to race conditions (Watson, 2008). This is because the values passed to the kernel can be modified by another thread between the time they are checked by the interceptor and the time the kernel sees them. However, this is less of a concern for us, since we do not need to support multi-threaded processes.

Another limitation of system calls is the sheer number of them: over 300 in current versions of Linux. Arguably, the system call interface is the wrong level of abstraction, because the same semantic operation (for example, extracting data from a file handle) can be achieved with many different system calls.

Examples of general-purpose, configurable system call interceptors include Systrace (Provos, 2003) and GSWTK (Fraser *et al.*, 1999).

3.2. *Linux Security Modules*

Linux provides an interface by which alternative security policies may be plugged into the kernel. Whenever the kernel is about to undertake some privileged action on behalf of the user, a hook in the current security module is used to determine whether it should be permitted. This is also used to implement the default security policy (for example, to allow the *root* user to access any file).

The interface for a security module is at a more appropriate level for our task: it deals with abstract actions, such as read from a file, rather than the specific system call used to achieve that action. It is also less vulnerable to the same race conditions as system call interceptors, because the security module accesses data within the kernel rather than userspace data that will later be copied into the kernel.

Unfortunately, the interface is frequently changed between kernel releases, and the documentation of the interface is often not updated to reflect the changes. This means

that a system based on the module cannot freely upgrade the kernel to take advantage of new features or security fixes to the kernel itself.

3.3. *Virtualisation*

Virtualisation allows one instance of an operating system (the guest) to run inside another (the host). When the untrusted program is run inside a guest operating system, it will be unable to access the resources of the host as it has no way to even address them (for normal uses of virtualisation, special configuration must be done to make host resources accessible from the guest).

This provides a high level of security, since rather than having to catch bad system calls, there is no way for a malicious application to even form a bad system call. And because any side effects of the program (such as leaving files in a temporary directory) are limited to the guest operating system, they can be wiped and a pristine guest operating system used for the next run.

The primary disadvantage of this approach is that the guest operating system would need to be booted for each run, adding significant overhead to the evaluation process.

4. Implementation

In the previous section, we listed three general approaches: system call interception, security module, and virtualisation. At the time we made the decision, we were not aware of the general-purpose system call interceptors, and writing one from scratch seemed a daunting task. Virtualisation was rejected because we wanted a light-weight setup that would not require us to maintain an operating system image separate from the primary operating system on the evaluation server. At the time, we were also only using a single machine for both the web front-end, compilation and evaluation, and we felt that booting a virtual operating system for each evaluation would be too expensive. We thus chose to use a Linux Security Module.

The Linux Security Module (LSM) framework provides a communication channel (`/proc/self/attr/exec`) for user processes to communicate with the security module. A wrapper program uses this channel to configure a restricted environment, then calls `exec` to launch the untrusted program. There are a number of commands that can be sent using this stream. There are some commands that are specific to Java (see 4.5); the remaining ones are listed in Table 1).

Once the process calls `exec`, the security restrictions come into place, and cannot be changed further. The `allow exec` command exists to allow further layers of wrappers around the actual program to execute (some interpreters, for example, are actually shell scripts which `exec` the “real” interpreter.

The majority of security module hooks are for more exotic functionality that a contest solution should have no need for, such as setting or querying scheduling policy, changing user, inter-process communication and so on. For each of these, we simply check whether we’ve flagged the task as restricted (which is quite easy, since the kernel task structure

Table 1
Commands that can be issued to the security module

Syntax	**Meaning**
`version` *major*`.`*minor*	Mark this process as restricted, and check for version mismatches
`allow threads` *n*	Set the number of threads the process may have at any time
`allow exec` *n*	Allow *n* calls to `exec`
`allow write`	Remove any extra restrictions on filesystem access
`allow write` *file*	Allow creation and write access to *file*

has a field available for the security module to store information), and if so, reject the call with the appropriate error code.

For calls related to creating or writing to files, we check both the global write enable, and the list of files marked as writable (for a contest problem, this would typically be just the output file). Note that there are multiple system calls to modify the data in a file (`write`, `fwritev`, `pwrite`, `truncate`, `ftruncate`, `sendfile`, ...), but they are all handled with the same security module hook – a distinct advantage over system call interception. In addition, the parameters to the hook use the kernel's internal representation of the filesystem, so there is no need to compare pathnames to determine whether are simply different ways to refer to the same file (due to symlinks, for example).

Initially we attempted to block all operations for which we did not explicitly see a need in a contest environment. However, we were surprised to discover the extent to which standard libraries depended on these calls for internal use. For example, we found that `glibc` preferred to use `mmap` for file I/O, and various files in `/etc` are consulted during library startup. In the end, we decided that it was easiest to rely on just the standard kernel security model for most read-only operations, and block or restrict only operations that had side-effects.

4.1. *Separate User Account*

Before the introduction of the kernel module, we used to have a weaker security system that merely executed programs under a different user ID, using `sudo` (Miller and Jepeway) to allow the user ID that runs the submission system to launch processes under this user ID. We decided to keep this model when adding the kernel module. While partly for defence-in-depth, this made it possible to allow the normal UNIX file access controls to govern read access without exposing the full set of test data, results etc. to submitted programs.

4.2. *Resource Limits*

The security module prevents multithreading and process execution, but CPU time and memory limits are enforced via the standard `setrlimit` system call. We use a wrapper program that forks, sets the limits and the security module settings, and finally launches

the program. It then waits for it to terminate and records the results (such as the exit code and execution time). `setrlimit` can only set a CPU time limit with one-second precision, so we round up the time limit for the purposes of `setrlimit` and check the actual time consumed after termination.

4.3. *Side Channels*

When the kernel module was initially introduced, jobs were run in parallel, starting as soon as they were submitted. We did not find a good way to prevent side channels between processes running concurrently. Although we are able to block the obvious routes such as sockets or files in a common area, there is a wealth of information available in `/proc` and our attempts to block access here led to instabilities in the kernel module itself. It is also known that shared caches between processors can be used to extract information by timing memory accesses, even if one of the processes does not intend to leak this information (Osvik *et al.*, 2006).

In our current system, jobs are queued and executed serially on each grading server, so side channels between running processes are not a concern. The most obvious side channel between processes that do not execute concurrently would be to leave a file in the filesystem. This is prevented by restricting the processes to write to only a specified list of files, all of which are purged after execution. There may still be side channels available (particularly related to uninitialised memory), but we believe that exploiting them reliably would be at least as much work as solving problems correctly in the first place.

4.4. *Alternate Root Directories*

In Linux (and other UNIX-like operating systems), it is possible to run a process in an environment where the root directory is actually only a subdirectory of the "real" root directory. This prevents the process from accessing any files other than those specifically placed in that subdirectory.

At present, we have not implemented this, largely due to the unwillingness to maintain separate copies of files in the alternative root filesystem. While we're not aware of any additional security we would gain from this, it would provide better defense-in-depth should any of the other security provisions fail (for example, should any of the contest test data become world-readable by accident).

4.5. *Java*

The Sun JVM (Java Virtual Machine) performs a lot of operations that would normally be blocked by the kernel module. While we initially tried to use the security module unchanged for Java, we found it impractical to let through the system calls that the JVM needed while simultaneously keeping the system secure for C and C++ programs. We have instead used the Java security manager to limit solutions to legal operations, and the command line option `-Xmx` to limit the maximum Java heap size.

The Java security manager uses a policy file which describes which privileged actions may be undertaken by which classes. The default configuration is quite permissive, allowing various operations not suitable for an olympiad (such as write access to any files, subject only to OS-level checks). We use the `-Djava.security.policy` command-line option to specify our own policy file. This custom policy limits all classes to just a list of explicitly allowed permissions – mostly querying of Java system properties, but also general read access, and write access to the output file. The file format permits a variable expansion syntax, so we are able to use a single file and provide the name of the output file on the command line for each evaluation.

5. Conclusions

When we started this project, we were under the impression that the Linux Security Module interface was a reasonably stable interface, suitable for third-party development of custom security modules. However, the interface changes with almost every kernel release, and maintainence has been more difficult than expected. Posts to the Linux Kernel Mailing List (Edge, 2007) suggest that in fact the interface is only intended for security modules maintained within the kernel tree, and as of Linux 2.6.24, it is no longer possible to build modules outside of the kernel tree. While it is a simple, low-overhead interface at a good abstraction level, we will have to consider whether other options will require less maintainence in the long term.

References

Edge, J. (2007). LSM: loadable or static? *Linux Weekly News*, 25 October.
`http://lwn.net/Articles/254982/`

Fraser, T., Badger, L. and Feldman, M. (1999). Hardening COTS software with generic software wrappers. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, May, 1999.
`http://www.isso.sparta.com/opensource/wrappers/`

Miller, T.C., and Jepeway, C. *Sudo Manual*.
`http://www.sudo.ws/sudo/man/sudo.html`

Provos, N. (2003). Improving host security with system call policies. In *12th USENIX Security Symposium*.
`http://www.citi.umich.edu/u/provos/systrace/`

Osvik, D.A., Shamir, A. and Tromer, E. (2006). Cache attacks and countermeasures: the case of AES. In *Proc. RSA Conference Cryptographers Track (CT-RSA) 2006*. Springer, 1–20.

Watson, R.N.M. (2008). Exploiting concurrency vulnerabilities in system call wrappers. In *WOOT'07 First USENIX Workshop on Offensive Technologies*.

B. Merry took part in the IOI from 1996 to 2001, winning two gold medals. Since then he has been involved in numerous programming contests, as well as South Africa's IOI training program. He obtained his PhD in computer science from the University of Cape Town and is now a senior software engineer at ARM.

Olympiads in Informatics, 2009, Vol. 3, 74–79

The Role of Reactive and Game Tasks in Competitions

Ilia NINKA
Department of Computer Science, Tirana University
Zog i Parë blvd., Tirana, Albania
e-mail: ilia.ninka@yahoo.com

Abstract. The need for more attractive tasks in teaching algorithms and in informatics competitions attracts many authors to reactive tasks as a powerful tool that created in student a desire for coping with hard tasks. In comparison with batch tasks, reactive tasks and especially the programming of games are very challenging, very comprehensive and fit perfectly with the story. In this paper an attempt to point out some attributes and priorities of reactive tasks in general, and game tasks in particular, versus batch ones is made.

Key words: batch task, reactive task, game programming, story, input data.

Teaching programming and algorithms in higher education is related with many difficulties both for the teacher and the students. The students hardly understand the need for complicated constructions and data structures related to the programming language, seeing that most of the problems presented by the teacher are elementary or may be solved by easier tools; spreadsheets, for example. The teacher is also faced with the difficulty for providing the students real world data for convincing them that it is necessary to write a program to develop the data and find the solution.

"Programming games will encourage students to learn more, and to apply what they learn to create new things, reaching the ultimate goal of education. Through the establishment of programming games as a core curriculum of Computer Science classes, students will learn algorithms faster and with a deeper understanding, and will want to do this because of the fun and accomplishment associated with the creation of a computer game" (Baibak and Agrawal, 2007).

During the process of creating algorithmic nature tasks informatics competitions there are some restrictions to be respected. We will consider them one by one from the view point of the topic of this paper.

According to the IOI 2008 competition rules, the tasks could be of the following types:

- ***Batch tasks*:** Solutions comprise a single source file of a computer program which reads data from the standard input (stdin) and writes its answer to the standard output (stdout).
- ***Reactive tasks*:** Solutions comprise a single source file of a computer program that is compiled together with an "opponent" library provided by the organizers, and

interacts with it according to the specification given in the task description. Such solutions are not allowed to read anything from the standard input, or write anything to the standard output.

- ***Output-only tasks*:** Solutions comprise a set of "output" data files. The contestants submit a zip or tgz archive file containing some or all the output data files.

In this paper game tasks and reactive tasks are considered from a common point of view. Output-only tasks are not considered.

1. The Story

Each task in the IOI, traditionally, is described as a real life situation and in some cases with characters endowed with real names and real habits. Sometimes, due to the diversity of participants, these create undesired ethic or moral situations.

Due to this tradition it happens that after having developed a task the author invents a story to fit it as much as possible. Sometimes the invented story is successful but there are cases that the story does not fit the problem as expected.

The students at first have to throw off the story and to discover the real problem to be solved and programmed. Sometimes this is quite natural, when the story and the problem are in harmony, but it happens that this may not be so easy especially when the story does not fit well with the problem.

When the task is a game one there is no need for a story. The story and the problem are the same. For the student this is a quite clear situation; he loses no time in discovering the problem behind the story, but only has to think how to solve the situation. In such case the student feels motivated because this situation is similar with other game situations which they had to surpass since childhood and they have some prior experience in such situations even without to the necessity of using computers or programming.

2. The Size of Input Data

In almost all competitions the input data is a real problem in itself when very large files are to be constructed. The concern for such large files is related to the aim to estimate the efficiency of algorithm used by the student. Sometimes the enormous quantity of input data is far from being a natural description of a real world situation as the story pretends to give. In reactive tasks, especially in game tasks, the input data is not such a concern for the author.

Let see some of game tasks given in the IOI. In these tasks the input data are quite natural and fit perfectly with the story.

a) *Task **4*** (Long-list of tasks, IOI'1990, Minsk, Belarus). *Given integer number K. A strip of paper is divided into N cells ($K \leqslant N \leqslant 40$). Two players choose and cross out K empty adjacent cells one by one. The winner is the one who has made the last move.*

In this task the input is ***only two integers*** K, N, where $K \leqslant N \leqslant 40$.

b) *Task* ***RUBIK'S TOOLKIT*** (IOI'1992, Bonn, Germany). *Write a program that allows the user to repeatedly solve any of the given three sub-problems ... in any order. You may assume that the length of each input string is at most 35.*

We escaped formulation of the three sub-problems which is long enough, but in this task the input data are quite reasonable, only 35 characters!

c) *Task* ***LETTER GAME*** (IOI'1995, Eindhoven, the Netherlands) . . . **Input Data**. *The input file INPUT.TXT contains one line with a string of lowercase letters* (*from 'a' to 'z'*): *the letters collected. The string consists of at least 3 and at most 7 letters in arbitrary order. The "dictionary" file WORDS.TXT consists of at most 40,000 lines. At the end of this file is a line with a single period ('.'). Each of the other lines contains a string of at least 3 and at most 7 lowercase letters. The file WORDS.TXT is sorted alphabetically and contains no duplicates.*

In this task the input data to be faced, while relatively large, is a common dictionary that the students use in their daily work in school.

d) *Task* ***A GAME*** (IOI'1996, Veszprém, Hungary) ... **Input Data.** *The first line of file INPUT.TXT contains the size* N *of the initial board.* N *is even and* $2 \leqslant N \leqslant 100$. *The remaining* N *lines contain one number in each line, the contents of the initial board in left to right order. Each number is at most 200.*

In this task a common game board is supposed to have no more than 200 numbers.

e) *Task* ***MAGIC SQUARES*** (IOI'1996, Veszprém, Hungary) ... **Input Data.** *The file INPUT.TXT contains 8 positive integers in the first line, the description of the target configuration.*

f) *Task* ***THE GAME OF HEX*** (IOI'1997, Cape Town, South Africa) *Your program must not read from or write to any files. Your program must not receive keyboard input, and must not produce output on the screen. It will receive all its input from the functions in the hex library.*

As it is seen from these tasks the input data are a complement of the task itself. This may not be the case in some batch tasks. For the story's sake the authors sometimes go so far that reality is forgotten making the story sound very strange! Let us consider only one batch task:

g) *Task* ***SEEING THE BOUNDARY*** (IOI'2003, Kenosha, USA).

Now let us examine farmer's Don ***field***. It is $500\,\text{km} \times 500\,\text{km} = 250\,000\,\text{km}$ square! This is almost the surface of Italy! But what about the rocks! This looks not as a farm but as a stone depository with as many as 30 000 huge rocks! No machine could do any agricultural work in this field! But what about the farmer Don himself: he is frightened by the fact that he must be cautious not to touch the rock, not to stand ***within*** a rock, and not to stand on a rock!

3. Inventing Strategy

Being quite natural and endowed with a rich flavor of challenge, the reactive tasks and game tasks arouse the interest of the student for not only trying to win a game but to

discover the best algorithm that ensures the victory when they have to move first. Even when the contest is finished, students are more biased to discuss game tasks with the aim to discover what they missed doing during the contest. This was for example a case when in the transit area in the airport; while waiting for the flight the students continued the game and discovered a quite simple algorithm.

Nearly all games require seeing patterns, making plans, searching combinations, judging alternative moves, and learning from experience – all being skills which are also involved in many daily tasks. So, Ginsberg (1998) was right when declared: "More than just competing with people, game-playing machines complement human thinking by offering alternative methods to solving problems".

4. Not Only Competition

"Games are thus the most ancient and time-honored vehicle for education. They are the original educational technology, the natural one, having received the seal of approval of natural selection". Written by Chris Crawford, in his book *The Art of Computer Game Design*, this statement proves the importance of games in any aspect of education. Games have been used throughout time as an instrument of instruction for all different aspects of life. Puzzles to learn logic, mathematical games to enhance basic math skills, and even reading games to increase reading ability have all been used successfully to teach children the basic skills that they will need in life. "It logically follows, then, that using computer games is an effective way to teach computing skills, and utilizing course curriculums that teach how to program computer games would invariably teach the basic skills required to program anything" (Baibak and Agrawal, 2007).

Nowadays we are dealing with a reduction in students which are fond of algorithmic and programming. This reduction is reflected not only in the number of students interested learning algorithmics and programming, but also in the quality of the participants in these events. A quite different view is presented when the students have to program a game. They have some inner motivations to consider this game as a challenge making the efforts to find the best winning strategy.

Programming games will endow the students with some skills which will be very useful for their future activities. Nowadays the computer game market is in expansion and the students will be the future programmers and more. As Gordon Novak Jr. (see website) noticed: "Games are good vehicles for research because they are well formalized, small, and self-contained. They are therefore easily programmed. Games can be good models of competitive situations, so principles discovered in game-playing programs may be applicable to practical problems".

In the first IOI there was a game task, and the game tasks continue to be presented in the IOI tasks sets in a sporadic way. From the first IOI till now there have been 23 reactive and game tasks versus 97 batch ones. There are only two IOIs where two game tasks were presented – IOI'2001, Tampere, Finland and IOI'2006, Merida, Mexico. Perhaps there are two HSC leaders fond of game tasks – Jyrki Nummenmaa, and Cesar Cepeda – who must be followed by others.

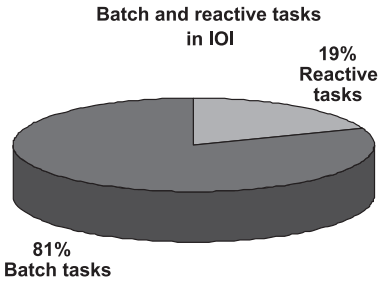


Fig. 1. Batch and reactive tasks in IOI.

According to the IOI 2008 Competition Rules for the reactive tasks the task statements should define among others:

- the interface specification of the "opponent" library,
- explanation of how to interact with the "opponent" library,
- instructions on how to compile their programs with provided "opponent" library.

These are the same characteristics as the game tasks where the player 1 (the contestant) plays against the player 2 (the opponent library).

Programming game tasks are very closely related with research activity. According Susane Epstein (1999): "There are two principal reasons to continue to do research on games ... First, human fascination with game playing is long-standing and pervasive. Anthropologists have catalogued popular games in almost every culture ... Games intrigue us because they address important cognitive functions ... The second reason ... is that some difficult games remain to be won, games that people play very well but computers do not. These games clarify what our current approach lacks. They set challenges for us to meet, and they promise ample rewards."

Considering the importance of programming games a Games Group has been formed in the University of Alberta which produces high-performance, real-time programs for strategic game-playing (University of Alberta GAMES Group, 2006). The group employs a variety of techniques from many areas of computer science, including artificial intelligence, parallel processing, and algorithm analysis.

At the Stanford University there is a research project by the Stanford Logic Group, part of the Stanford University Computer Science Department. Their AI Magazine article describes the General Game Playing concept and the AAAI GGP competition (AAAI General Game Playing Competition, 2008); a brief GGP Overview is also available. The GGP website contains information about the Logic Group's research in general game playing, and forms the central resource for General Game Playing Competitions, the first of which was held at AAAI '05 in Pittsburgh. The website also hosts a GGP Game Manager, allowing General Game Players to connect and play single or multi-player games online, in order to help them to prepare for future competitions.

5. Conclusions

The reactive tasks and especially game tasks at the IOI must be considered as very useful tool for making this event more attractive to the students. These kinds of tasks are very challenging, and students are very motivated to undertake their programming. These kinds of tasks are very close to real life situations, and the students do not spend too many efforts understanding or remembering them. The game tasks are in harmony with the story describing them and do not need too much input data. More attention must be paid to this kind of tasks at the IOI.

References

AAAI General Game Playing Competition (2008).
http://games.stanford.edu/competition/competition.html

Baibak, T. and Agrawal, R. (2007). *Programming Games to Learn Algorithms.* American Society for Engineering Education.

Ginsberg, M.L. (1998). Computers, games and the real world. *Scientific American*: *Exploring Intelligence* (special issue – Winter 1998). http://www.psych.utoronto.ca/users/reingold/courses/ai/cache/1198ginsberg.html

Gordon Novak Jr. Web-site. (n.d.)
http://www.cs.utexas.edu/∼novak/cs34312.html

Epstein, S.L. (1999). Game playing: The next moves. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*. Orlando, FL.

University of Alberta GAMES Group (2006).
http://www.cs.ualberta.ca/∼games/

I. Ninka is full professor in data structures and algorithms, artificial intelligence at Tirana University, Albania. He is head of Computer Science Department, a member of the Albanian National Commission for Olympiads in Informatics. Since 1993 and was head of the Commission (1999–2001). He was member of the organizing team of BOI (2001), president of the SC of Balkan OI'2001, leader of Albanian national team for IOI'1999–2008. He is author of more than 20 scientific papers, 6 university textbooks and 12 textbooks for secondary schools.

Olympiads in Informatics, 2009, Vol. 3, 80–100

Team Competition in Mathematics and Informatics "Ugāle" – Finding New Task Types

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

Abstract. Existing olympiads in mathematics and informatics are fixed form competitions for individuals with quite stable lists of task types. Outside the scope of these competitions falls a lot of interesting and challenging tasks like puzzles, games, logic tasks, and practical tasks outside the classroom. Team competitions offer a new dimension in a task solving process where successful collaboration between team members is one of basic requirements for achieving high results. This paper describes an annual (since 1996) Latvian team competition in mathematics and informatics for high-school students called "Ugāle". Classification of the main task types is given and representatives of these task groups are given. Suitability of different task types in different contests is discussed. The evolution of the form and content of the competition is described.

Key words: olympiads in informatics, team competition, classification of competition tasks, grading.

1. Introduction

Most of scientific olympiads are organized on an individual basis. The most popular of them in the field under investigation are the International Olympiad in Informatics (IOI, 2009) and the International Mathematical Olympiad (IMO, 2009). Both olympiads have supporting infrastructure at regional (i.e., Baltic) and national (i.e., Latvian) level (NMS, 2009; LIO, 2009). Individual competitions (different contests and olympiads) have a long lasting history in Latvia. Annual state and open olympiads in mathematics with more than 3000 participants are organized by prof. Agnis Andžāns (Ramāna and Andžāns, 2002). Latvian olympiads in informatics have been organized since 1987, and since 1992 Latvian team participates on IOI. Latvia was one of the three founders of the annual Baltic Olympiad in Informatics at 1995 (BOI, 2004; BOI, 2008).

There are several well-known team competitions for high-school students in informatics. The most popular are the IPSC in Slovakia (IPSC, 2009) and the Open Team Cup in Russia (Open Team Cup, 2009). In mathematics the most important team event in Latvia is the "Baltic Way" competition for secondary school students in mathematics (Baltic Way, 2008).

At the IOI and IMO tasks are quite frozen in their form. Despite the theoretical possibility to use interactive and open input tasks, for the last two years (at IOI'2007 and

IOI'2008) only the so-called batch tasks were offered. At the IMO the use of computers, calculators or other electronic devices is prohibited, therefore it is practically impossible to discuss "changes to the competition format". At the same time at the IOI there are no tasks which can be solved without computer.

It is quite clear that the mathematical and informatics competitions cover separate areas, leaving relatively big part of tasks uncovered. Some kinds of tasks do not fit into the IOI curricula (Verhoeff *et al.*, 2006), other tasks are deemed unsuitable for competition by general audience, for example, such kinds as data processing, numerical puzzles, logical games and cryptarithms. At usual informatics olympiads the so called "open ended problems" (Kemkes *et al.*, 2007) are not used. Different task types and their suitability for competitions as well as sources of inspiration are discussed by Burton and Hiron (2008), and Pankov and Orusulov (2007).

The idea of team events at competitions in informatics is discussed by Burton (2008).

2. Task types – Human Brain Versus Computer

If we look at tasks from the viewpoint of usability of a computer in their solving process, we can group these tasks in three main groups:

Gr1) Tasks which can be solved without usage of computer and where computer cannot give reasonable help to speed up the solution process. In the solution process pure mathematical skills are necessary and it is nearly impossible (at least for contestants) to use computer for essential help in the solution process. For example, the tasks "Prove that there is no largest prime number" or "Prove that among any five integers you can find three with sum divisible by 3" lie in this group.

Gr2) Tasks which can be solved only by use of computer. Besides native "write a program which solves a task for any input data" also a lot of "find all numbers with particular properties" tasks lie in this category (if exhaustive search is necessary and a relatively simple computer program can give the necessary results in reasonable amount of time).

Gr3) Tasks which may be solved either by help of a computer or without it.

The presence of a particular task at some competition assumes it belongs to one of these groups. It is obvious that tasks with completely unknown solution cannot be placed in any of these categories.

The so called "batch" tasks used at the IOI always belongs to Gr2. Moreover, the IOI rules prescribe usage of tasks with strictly specified programming languages and tools. Some years ago the so called "open input" tasks were included in the list of possible task kinds. As a rule, these tasks allow solving of some subtasks without computer or using different tools beyond the IOI toolset and therefore can be classified as Gr3 tasks. One of such tasks "Table" was suggested by the author and included in the programme of BOI'2003 (BOI, 2003), but the achieved results were lower than at the usual "batch" tasks – there were no contestants either at the on-site or offline competition who got full score.

The best result for several subtasks was achieved by different contestants. The last open input task included in the official programme of IOI was the task "Forbidden subgraph" at IOI'2006 (IOI, 2006). Output-only tasks are discussed by Vasiga *et al.* (2008).

At the IMO as well as at the World Sudoku Championships (WSC) and other competitions where human skills to solve particular tasks are examined, use of electronic devices is prohibited (IMO Regulations, 2009; Sudoku, 2009) and it may seem that here all tasks "by definition" lie in Gr1. This is not true at least for SUDOKU – a computer program able to solve classic SUDOKU puzzles is relatively simple (Brouwer, 2006). Thereby, allowing usage of computer will kill competition and make it completely uninteresting – hence banning electronic devices is the only possible solution. In other cases it is not always obvious which way of solution is more preferable. Many tasks in the World Puzzle Championships (WPC, 2009) are quite interesting to solve by computer programming or may be reformulated to become such.

There is also another pitfall – if usage of computers is allowed then there may be a tendency to look at all tasks as Gr2 tasks – not trying to think about problem more deeply without touching the keyboard. "When all you have is hammer, everything looks like nail". Several nice examples of such tasks are given by Ginat (2008). If you have calculator at hand it is quite hard to press yourself to make mental calculations. On the other hand, if there is no calculator and you are asked to provide some simple calculations, you may try to find some electronic device which helps you to make these simple calculations instead of using your own brain. You somehow get obsessed with the idea "give me calculator and I will show you how to get the result". All these thoughts can be transformed to a higher level if we replace calculator by computer.

3. History of "Ugāle" Competitions

Ugāle is town with 2677 inhabitants (2003) in the Western part of Latvia. In 1996 a teacher of Ugāle secondary school, Aivars Žogla, came up with idea of a joint competition in mathematics and informatics where together with classical mathematical problems a computer program for strategic game must also be written. The other main difference from classic olympiads was the idea of making this a team competition where up to three contestants solve tasks as one team.

Every team consists of three contestants and one (at the final round) or two (at the semi-finals) computers. During 5.5 hours teams must solve tasks (usually 8 to 10) given by a jury.

Competition format has changed during these years. The first two competitions (1996 and 1997) were organized as one-round competitions. Since 1998 a preliminary semi-final round is organized and the best teams together with a host team participate in the final round which usually takes place in May in the premises of Ugāle secondary school. The number of participating teams in the final round is fixed (12), but a number of teams in semi-finals vary year by year and in recent years was between 55 and 75 teams (see Fig. 1).

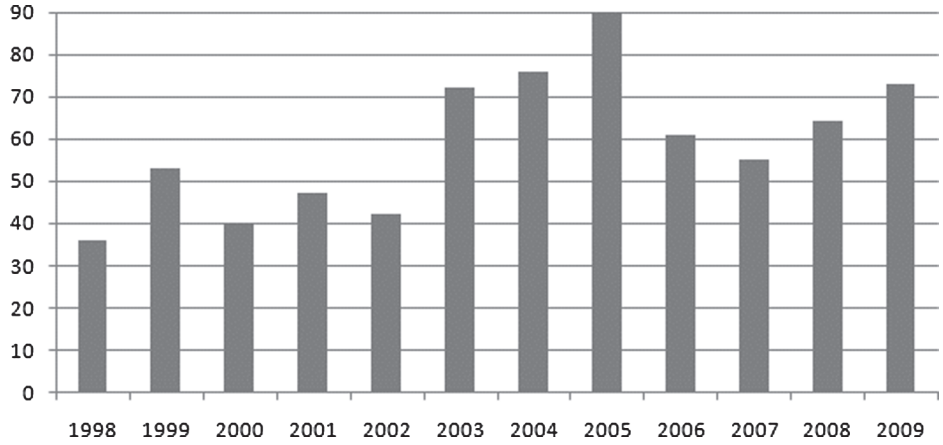


Fig. 1. Number of participants in semi-finals of Ugāle competition.

In the final round each team works in a separate classroom and there are no barriers for free communication between team members. However, they can also work as three individuals and during these years only some tasks forced cooperation of team members. During all the problem solving time, the captain of the team can come to the jury and ask questions concerning task descriptions and clarify technical details (like what must be submitted as solution, is it allowed to use some complementary thing not mentioned in task description, etc.).

4. Tasks and Grading

As in every scientific olympiad the main idea of "Ugāle" competitions is to give the opportunity to work on unusual tasks, still using basic knowledge obtained at school. However, the chosen competition format also influences the allowed kinds of tasks. Correctness of solutions at the final round must be checked by jury members during a limited amount of time (approx. two hours). Therefore, to lower the pressure on the jury, answers and solutions must be short in their form. It is quite hard to grade "classical" mathematical solutions like proofs in such a short amount of time and such tasks are not included in a competition's task set. Also traditional IOI batch tasks are not included since there are a lot of pure programming competitions where such tasks are used.

However, the mathematical and informatics part of the competition is not lost. Simply, during the task selection process, tasks are chosen so that the answer to every of them is simple and short in form. The basic principle from math task solving: "Find all solutions and prove that there are no other" is kept alive, clearly stating tasks where it is enough to find any one particular solution and assuming finding all possible answers if such statement is omitted.

Mathematical tasks are formulated so that the answer is simple number, some sequence of numbers or set of possible answers. Finding the correct answer involves all

steps necessary for a full solution and from that viewpoint are of the same level of difficulty (there is no "shortcut" to the correct answer like guessing).

The real challenge is to prepare a task which could be solved either by using pure mathematic skills (without computer) as well as by writing a correct computer program which gives the answer in few competition hours.

In contrast to informatics olympiads, there is no constraint that written program must run in particular time (usually seconds or even its parts) and space limits during program execution. There are also no limitations on the software used during competition. Contestants may use the usual IOI programming languages as well as packages like Mathematica, MS Office, etc. A list of software, suitable for solving mathematical tasks is given by Turskiene (2002).

All teams are in the same conditions and therefore open-ended tasks of kind "find as good as possible a solution" are also acceptable. Such tasks are graded in manner that the team with highest achieved result are awarded maximum points, and other teams are awarded points proportionally to their score.

As a rule the full solution of any task can get maximum 100 points. If there are subtasks, then it is clearly stated how many points are given for the solution of a particular subtask. In early competitions there were attempts to differentiate points given per task, but such praxis (as well as at the IOI) was not continued. Therefore, the determination of difficulty level of a particular task is one of the essential competition components.

All answers and results must be written by teams on a special form ("answers sheet") and as a rule this is the only source for grading (another source may be computer program or some result file in electronic form on some media).

5. Task Categories

The first competitions in 1996 and 1997 now can be considered as warm-up competitions. The first competition task set contained nine pure mathematics tasks offered by prof. A. Andžāns and one programmable game task. The second competition contained too large a number (19) of tasks and lot of effort was wasted, because it was impossible to solve a reasonable amount of tasks in the given time. In recent years at the final round 10 tasks are offered and the obtained results show that such a number is reasonable.

In total 244 tasks were used during years from 1996 till 2009 (at the moment of writing of this paper only the semi-finals of 2009 have finished). Classification of tasks was done by including every task in one or more categories.

It can be seen that some groups of tasks are quite stable, because every year (or even more – in every competition) at least one task from a particular group is included in the task set. Other groups are disappearing or appearing. For example, the currently existing format excludes tasks with mathematical proofs or argumentation, which was regular part of earlier competitions.

In the following chapters main task groups are described. The total number of tasks is given in brackets.

5.1. *Data Processing/Mining Tasks (24)*

One task of this kind has been included in every task set starting from 1997. The main idea of such tasks is to get some numerical results from given data (usually as one or two text files). These tasks assume the usage of spreadsheet processing systems (like Microsoft "Excel") or DBMS (like Microsoft "Access"), however solutions may be found with other tools (like programming in the languages from the IOI list). An example of such a task is "Football" from semi-final of Ugāle'2004:

> *In the text file futbols.txt there is information about all games of a football championship. During the championship two rounds of matches are played – each team plays two games with each other – one at home and second as visitor. For a win a team is awarded 3, and for loss – 0 points. If game ends in draw, each team gets 1 point. Each file row contains information about one game in the following format:*
>
> ***Host team name – Visitor's team name; Goals scored by host team; Goals scored by visitor's team; (further names of scoring players as well as minute of match when goal was scored are given. At the beginning all the host team and then all the visitor team players are listed).***
>
> *For example, one row can look like*:
>
> `Badgers - Monkeys; 1; 3; Apse; 88; Lipenbergs; 37;`
> `Priede; 8; Millersons; 55`
>
> *Your task is to find answers to the following questions*:
>
> 1. *How many teams participated in championship?*
> 2. *How many goals were scored by visitor teams?*
> 3. *How many goals were scored in total?*
> 4. *How many matches ended in draw?*
> 5. *How many goals were scored by Smits?*
> 6. *In which game was the maximum number of goals was scored?*
> 7. *How much goals were scored in the first halves of matches (till the 45th minute, inclusive)?*
> 8. *Who scored the maximum number of goals? How many? In which team he plays?*
> 9. *How many players scored just once?*
> 10. *How many different draw results were in all championship matches?*
> 11. *Which team won the championship and how many points did it obtained?*
> 12. *Which team got last place and how many points did it obtained?*

5.2. *Cryptarithms (15)*

A usual cryptarithm is a well-known type of puzzles. The solution is a correct arithmetical expression. In the task digits are substituted by letters or different symbols and it is known that equivalent digits are substituted by the same letter and different letters covers different digits. Several modifications of the classical format also are used.

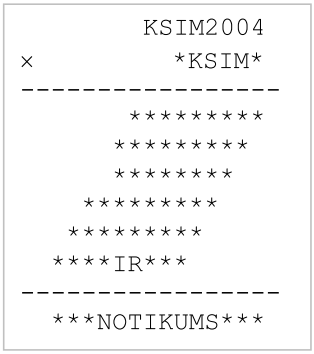


Fig. 2. Cryptarithm.

An example cryptarithm from final of Ugāle'2004 is given:

In the given cryptarithm (see Fig. 2) substitute letters and asterisks by digits so that:

- *equal letters will be substituted by equal digits*;
- *non-equal letters will be substituted by non-equal digits*;
- *there will be no number (even as the result of particular multiplication) with 0 as the first digit*.

Asterisks may be substituted by different digits.

Take into account that in the first multiplicator there are two digits "0", but in the final result – letter "O". It is enough to find one of the possible solutions.

5.3. *Word Problems (6)*

From historical viewpoint a small but quite interesting group is word problems. Famous mathematician Andrei Toom wrote: "Word problems are very valuable in teaching mathematics not only to master mathematics, but also for general development" (Toom, 2005). The author completely shares this viewpoint and as a representative formulation of task "Let's meets in jungle!" (Semi-final of Ugāle'2000) was given:

In the jungle there is a small lake which is permanently filled by springs under it. The full lake can be drained in one day by 3 tigers and 10 elephants, against 10 tigers and 3 elephants that can drain lake in two days or 8 tigers and one elephant that can drain lake in four days.

Calculate in how in many days the lake can be drained by:

a) *One elephant*;

b) *One tiger*;

c) *One elephant and one tiger drinking together?*

Remark: *"Drained" means that there is a moment when lake is completely empty. Of course, continually, water is coming from the springs.*

5.4. *Geometry (36)*

The next four sections describe task types that have become classic at mathematical olympiads. Algebra, Geometry, Combinatorics and Number theory are four "whales" of mathematical competitions (Andžāns and Ramāna, 2002). For example, the task set of the mathematical team competition "Baltic Way" contains exactly five tasks from each of these four groups. Despite their apparent solidness, A. Toom characterizes classical geometry (together as word problems) as the "Cinderella of American education" (Toom, 2005).

Geometrical tasks are especially suitable for development of abstract thinking and demand different knowledge from other branches of mathematics. Geometry has a lot of faces and every task set must offer at least one geometrical task. To give a bit wider insight, instead of one representative task, several task examples will be given.

In the first competitions, classic olympiad tasks were used, but in recent years they have slightly changed to an expected form of result which usually is some numerical value. Task "Two triangles" (semi-final of Ugāle'2008) is one of the representatives:

> *The intersection of two triangles is hexagon with inner angles (in this order): 87°, 141°, 105°, 137°, 104° and 146°. Calculate the angles of these triangles!*

If you remember the general remark concerning multiple solutions, this is exactly the case, because this task has more than one solution and in the task description there is nothing allowing taking for granted that it is enough to present just one of them. Looking for suitable tasks, the content of tasks from olympiads in mathematics are also is taken in account. Obviously, the task types at olympiads are not a solid matter – they changes over time (see Fig. 3).

Construction tasks have not been presented at the Latvian Olympiad in Mathematics for the last three decades. So it was good reason to use such tasks at Ugāle competition. Task "Elegant pentagon" (final of Ugāle'2004) is one of them:

> *Let's say that a convex pentagon is "elegant" if the following conditions are satisfied:*
>
> 1) *it can be inscribed in circle*,
> 2) *the length of all sides and radius of the circumscribed circle can be expressed in whole centimetres*,
> 3) *all sides and radius of the circumscribed circle are of different length.*
>
> *Let's say that a convex pentagon is "partly elegant" if only the first two conditions are satisfied. Your task is to construct either an elegant (100 points) or partly elegant (30 points) pentagon.*

In the middle of the 20th century in the schools of Latvia there were quite popular tasks concerning measuring distances in nature. At that time such tasks were included in secondary school curricula. Tasks of this type may be mention the measurement of height of a particular tree, width of a river, distance to a far object, etc. In the author's

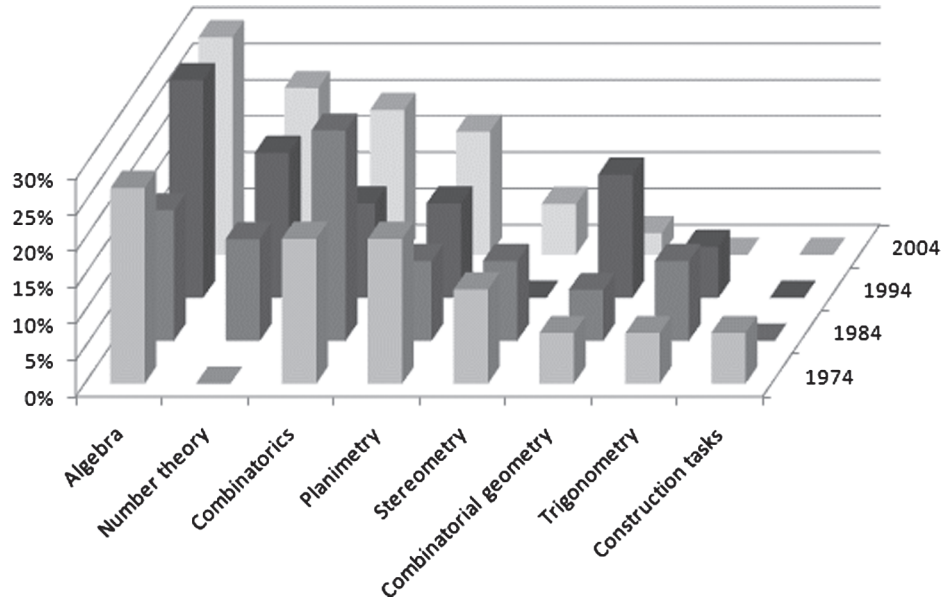


Fig. 3. Task types at Latvian Mathematical Olympiads (Bonka, 2004).

opinion tasks of this kind are very close to the nature of geometry (the word itself means measuring the earth) and are undeservedly forgotten. An attempt to check these skills in the contemporary youth was done by including the task "Distance in nature" in the final round of 2005:

> *You are given a rope with length equivalent to the width of the Ugāle secondary school front door).*
>
> *At both sides of school building there is one marked lamppost (see Fig. 4). Calculate the straight distance in ropes between these two lampposts without the destruction of the school building!*
>
> *Only those solutions where the difference between jury's and submitted solution will be less than doubled length of rope will be graded. If you wish to suggest the use of other tools beside rope, consult the jury in advance!*

This task became quite popular at this competition and teams showed good results – all teams got points and 4 out of 12 teams got a full score. A similar task was included in the task set two years later.

One more (and again completely different) geometrical task is the task "Pencil" at the final of Ugāle'2003:

"A pencil, the cross-section of which is a regular hexagon with side length equal to 0.5cm, was sharpened by a cone-shaped sharpener with the angle between the generatrix and the altitude equal to $\pi/8$ in a way that the length of the pencil didn't change. How much of the pencil (in cubic centimetres) was removed? Provide as an answer a decimal fractional number, the more precise the better. Maximum points will be awarded for correct 9 digits after the decimal point."

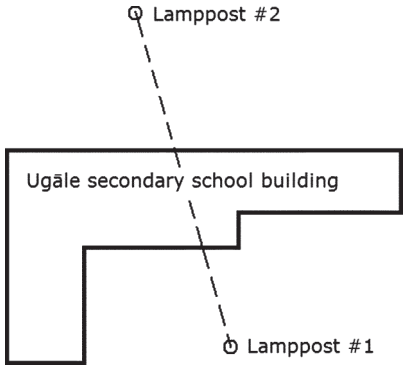


Fig. 4. Configuration of school building and lampposts.

Fig. 5. Sharpened hexagon pencil.

The small waves at the end of the sharpened part (see Fig. 5) make calculations non-trivial and the task – interesting. The number of decimal digits necessary for full points does not allow neglecting this area.

5.5. *Number Theory (31)*

These tasks are appropriate for Ugāle competitions due to their natural relationship between mathematics and programming – tasks can be solved in various ways, but the best results takes some combination of skills from both disciplines.

Two examples from this task group will be given.

The task *"Twenty-digit number"* (final of Ugāle'2001, idea from (Puzzles, 2000)):

> *Arrange two of each of the digits 0 to 9 to form a 20-digit number. Your number may not begin with a zero.*
>
> *You are then scored on your number as follows*:
>
> *For every n ($n > 1$) consecutive digits where the first digit is not 0 that form a square number, you score n points.*
>
> *For example, if your number was 98543676011023475928, you will score two points for 36 and three points for 676 – for a total of 5 points. You may not count 01 as a two-digit square.*
>
> *What is the maximum number of points you can score?*

This task can be easily turned into an open input task by specifying different sets of usable digits. However, the essence of the task remains the same – contestants are in the

same starting positions and still must compete for a better result where the possibility of getting the perfect result during the limited competition time is unclear.

The task "Ugāle Primes" (semi-final of 2005):

> *Let's name an n-digit prime $\overline{p_1p_2...p_n}$ as a Ugāle's prime if and only if all prefixes of this number $\overline{p_1}, \overline{p_1p_2}, \overline{p_1p_2p_3}, \ldots, \overline{p_1p_2...p_{n-1}}$ also are primes. For example, 71 and 311 are Ugāle primes (because 71, 7, 311, 31 and 3 are primes), but 27, 43 and 307 – are not (27, 4 and 30 are not primes). Find one, as big as possible Ugāle's prime!*

5.6. *Combinatorics (40)*

As well as the task from the previous section, combinatorial tasks are solvable in various ways. As an example task "APRICOT" (final of Ugāle'1998) is given:

> *A word consisting of n different letters is given. This word is written on a sheet of paper with squares so that every letter is written in its own square without empty squares in-between. The same word is written in the row below in the same manner just starting writing one square to the left. So this word is written on the next lines until it is written $n + 1$ time.*
>
> *In the Fig. 6 there is given an example for $n = 7$.*
>
> *Your task is to calculate the number of different ways the given word can be read on the sheet if you can start from the first square of any row and go to a neighbour square to the right or down and direction of reading may be changed no more than twice. One of the ways to read the word is denoted in the figure by darker squares (direction of reading is changed once).*
>
> *Solve this task for a) $n = 7$, b) $n = 36$, c) $n = 711$, d) $n = 1492$.*

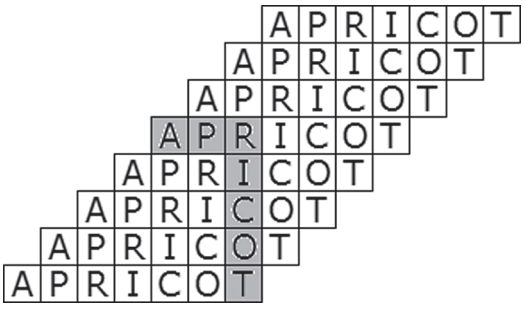


Fig. 6. Example for $n = 7$.

5.7. *Algebra (14)*

Classic tasks for contestants of mathematical olympiads. Task "System" (semi-final of Ugāle'2001):

Find a solution in integers:

$$\begin{cases} z^x = y^{2x}, \\ 2^z = 4^x, \\ x + y + z = 20. \end{cases}$$

5.8. *Logic (37)*

There are a lot of classic reasoning puzzles, where some clues are given and you must find out hidden consequences. One of the famous representatives of this kind is puzzle "Who Owns the Zebra?" (Zebra, 2009). But such tasks are not represented at usual olympiads. However, in some countries such tasks are used in theoretical or preliminary rounds of olympiads in informatics (Anido and Menderico, 2007).

At the end of the 90s a very interesting task "Self-referential aptitude test" by Propp (2009) was published and became a source of inspiration for several tasks.

Interestingly, solutions to such tasks can also be found by quite simple computer programs (however, these programs are quite unusual) and, therefore, such tasks also lie on the edge between mathematics and informatics.

Task "Concrete logic" (final of Ugāle'2002):

> *Answer the 20 questions below by "yes" or "no" so that all the answers do not contradict each other.*
>
> 1. *Are the answers to Questions 6 and 7 equivalent?*
> 2. *Is "no" the answer to Question 1?*
> 3. *Are the answers to Questions 4 and 20 different?*
> 4. *Are the answers to Questions 3 and 20 different?*
> 5. *Are this and the answer to Question 19 different?*
> 6. *Is "yes" the answer to Question 2?*
> 7. *Is "yes" the answer to Question 15?*
> 8. *Are the answers to Questions 11 and 19 equivalent?*
> 9. *Is "yes" the answer to Question 10?*
> 10. *Is "no" the answer to Question 13?*
> 11. *Is it true that Mr.Bērziņš doesn't like strawberries?*
> 12. *Is "yes" the answer to Question 16?*
> 13. *Is "yes" the answer to Question 12?*
> 14. *Are this and the answer to Question 11 equivalents?*
> 15. *Is it true that "no" is the answer to at least half of all the questions?*
> 16. *Is it true that "yes" is the answer to at least half of all the questions?*
> 17. *Are the answers to Questions 9 and 4 equivalent?*
> 18. *Is "yes" the answer to Question 7?*

19. *Is it true that the name of Mr.Bērziņš is Jānis?*
20. *Are the answers to Questions 3 and 4 different?*

5.9. *Analysis of Algorithms (10)*

Understanding programs written by other authors also is part of programmer's everyday job, but such tasks are not presented at competitions in "pure" form when fragment of code or pseudo code is given. Task is in finding something in or related to the given code: finding error, constructing the worst case counterexample, estimation of overall complexity, implementing the same algorithm in a more effective way. In some countries analysis of algorithms is part of the preparation work for the IOI (Forišek, 2007). An example of task where analysis of a given program must be provided is task *"Sorting"* (final of Ugāle'2003, author Aivars Žogla):

In the file SORT.PAS an algorithm is given that sorts the elements of number array `A[0..n-1]` *from position* `low` *till position* `high` *in non- decreasing order. By taking low* $= 0$ *and high* $= n - 1$*, the entire array will be sorted.*

Your task is to find an array containing each of the numbers from `1` *to* `20` *exactly once, for which sorting by calling procedure* `sort(A,0,19)`*, uses the maximum number of array element comparison operations* (*these rows are marked by* `{*}`). *For example, sorting the array* A={3,4,1,2,5}*, uses* 7 *comparison operations.*

```
{ =========== Start of SORT.PAS ===============}
const MAXN = 100;

type TArray = array[0..MAXN] of integer;

procedure swap(var A:TArray; i, j:integer);
var temp:integer;
begin
     temp := A[i]; A[i] := A[j]; A[j] := temp;
end;

procedure sort(var A:TArray; low, high: integer);
var i,j,middle:integer;
begin
     if high - low < 5 then
        for i := low + 1 to high do
        begin
            j := i;
            while j > low do
{*}               if A[j - 1] > A[j] then
                  begin
                     swap(A, j - 1, j);
                   j := j - 1;
                  end
               else j := low;
        end
```

```
     else
     begin
          middle := (low + high) div 2;
{*}       if A[middle] < A[low] then swap(A, low, middle);
{*}       if A[high] < A[low] then swap(A, low, high);
{*}       if A[high] < A[middle] then swap(A, middle, high);
        swap(A, middle, high - 1);
        i := low;
        j := high - 1;
        repeat
                repeat
                      i := i + 1;
{*}             until A[i] >= A[high - 1];
            repeat
                  j := j - 1;
{*}         until A[j] <= A[high - 1];
            if i < j then swap(A, i, j);
        until i >= j;
        swap(A, i, high - 1);
      sort(A, low, i - 1);
      sort(A, i + 1, high);
   end;
end;

begin
end.
{ =========== End of SORT.PAS ===============}
```

Another task of this kind is *"Function of functions"* (final of Ugāle'2005):

Function $f(x)$ is defined for all integers from 1 to 2000000000 and function values are positive integers. Function values are known for one hundred argument values (*Table 1*).

Your task is to write **as short as** *possible program in one of the programming languages Pascal, C or C++, which implements this function for all x values given in the table above. Do not worry about the values other than given, because your program will be tested only with the argument values given in the table.*

Function source code must be presented in one separate file and usage of other data files is prohibited. Program must read one x value from the standard input and the corresponding $f(x)$ value must be written to the screen and program must exit without waiting for additional user input (such as pressing a key). Program may use only the modules and libraries included in compiler's standard configuration.

If for any of the hundred given argument values the answer will be different from the value given in the table, the score for the task will be 0 points. Execution time for one particular test case must not exceed one second.

Programs with lower ***amount of source code in bytes*** *will get higher scores.*

Table 1

x	$f(x)$	x	$f(x)$	x	$f(x)$
1	2	3071	83	987654329	4312901
2	1	3073	439	987654331	9588877
3	2	3381	1127	987654791	31397
5	2	3383	199	987654793	31397
7	2	3385	677	987654803	31397
9	3	3403	83	987654809	83227
10	5	3419	263	987654821	31397
11	3	3493	499	987654857	141093551
13	3	173005	34601	987654861	329218287
15	5	173007	57669	987654865	197530973
16	8	173009	10177	987654881	57559
17	3	173021	409	987654883	31397
21	7	173027	2437	987654901	31397
22	11	173029	7523	987654971	48341
23	3	173049	57683	987688883	6059441
33	11	173051	1321	987688885	197537777
67	7	173419	4687	1999999001	285714143
77	11	173973	57991	1999999003	44711
105	35	173975	34795	1999999005	666666335
107	7	173983	9157	1999999009	32786869
109	7	173989	677	1999999015	399999803
111	37	9173003	1310429	1999999019	155171
115	23	9173005	1834601	1999999027	153846079
117	39	9173009	23581	1999999037	42553171
119	17	9173011	3023	1999999039	4640369
123	41	9174653	5639	1999999957	7782101
125	25	9174661	20899	1999999961	54054053
126	63	9174667	295957	1999999967	285714281
131	11	9174671	834061	1999999969	181818179
137	11	9174673	6323	1999999975	399999995
499	19	987654321	329218107	1999999981	285714283
3055	611	987654325	197530865	1999999997	37735849
3059	437	987654327	329218109	1999999999	64516129
3061	53				

Contestants were also supplied by table values in a separate text file.

The simplest approach would be trying to code the given values without any investigation. However, much better result could be obtained, by discovering some regularity.

Besides that, for choosing the right approach, a contestant's deep understanding of the possibilities of different languages and compilers was a great advantage. It is quite easy to understand the usability of such tasks in industry – in a world of microprocessors a requirement to fit in a given amount of memory is usual.

5.10. *Programming (25)*

One of the programming tasks different from the usual IOI tasks is task *"Smart program"* (final of Ugāle'1998):

> *Write a program in Pascal, C or BASIC, which*
> - *outputs on the monitor the number 1998 only once;*
> - *if the program code is modified by replacing one symbol by another in such a way that program still compiles and executes, it still only once outputs the number 1998.*

This task is quite tricky and needs an extremely deep knowledge of the chosen programming language. In some sense this task is designed for hackers not algorithmists. The essence of the task is different from the usual programming tasks (who care about your source code and tries to break it by changing it in any other competition?) and every single space character in code can be used against you.

Despite the short codes submitted, grading was done in a special way by the best possible jury members – three IOI medallists (Krists Boitmanis (silver on IOI'97, bronze on IOI'96), Renārs Gailis (silver on IOI'97) and Juris Kriķis (bronze on IOI'96)). Every submitted program was investigated by looking through carefully and trying to break the code. Because there were only 12 teams, this evaluation was successfully completed and only those programs where experts did not find any faults got full score. Technically possible, but hard to imagine would be the testing of this task without such a group of experts, because "grading" includes the possibility or impossibility of finding a counterexample (a symbol which can be changed to show that the second condition is not satisfied).

One of programs in PASCAL which twelve years after the actual competition *seems to be* a correct solution:

```
var sk1,sk2:string;
Begin
sk1:='1998';
sk2:='1998';
if sk1<>sk2 then writeln('1998') else writeln(sk1);
End.
```

5.11. *Dominoes (12)*

The author has noticed that there is quite a big gap between the contestants' generation and the author's one in the sense of basic knowledge in the field of logical or board games. It is nearly impossible to say which rules of logical games are known to the general audience and which must be explained in task formulations. It is not easy to say whether all contestants are familiar with the rules of chess or checkers or not. In this situation classical dominoes are widely used in the competition every year and a set of dominoes is necessary attribute for every team.

Dominoes are quite simple and at the same time they possess a quite high combinatorial power. The following task "Area of dominoes" is the only one which was included

in a task set twice – at the final of Ugāle'2005 it was not solved and was included in the task set of the next year's semi-finals where it was solved by only one team out of 61:

> *All 28 pieces of the usual dominoes set must be placed in the area shown in Fig. 7 so, that:*
>
> - *every piece covers exactly two squares*,
> - *if two pieces share common edge, then in both halves the number of points must be the same*,
> - *in all four horizontal rows (indicated by arrows)the total number of points must be the same.*
>
> *It is enough to show one such distribution.*

Task "Who can find more?" (final of Ugāle'1997):

> *Fill a rectangle consisting of* 8×7 *squares with the numbers from 0 to 6 (in each square there must be one number) so that this rectangle can be covered by pieces of one set of dominoes (every square is covered by half of one dominoe, all squares are covered) in* ***as many ways as possible***.

After the competition this task was published on the website of the Latvian Olympiad in Informatics (More, 2009) and eight years after competition the solution (see Fig. 8) with **793648** different coverage was found by former IOI medallist Jānis Sermuliņš (gold on IOI'97 and IOI'99, bronze on IOI'98):

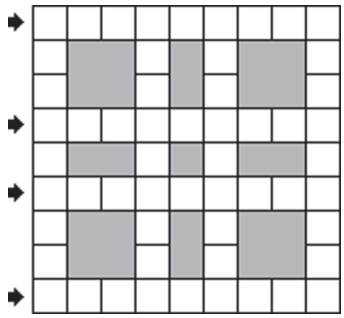


Fig. 7. Area of dominoes.

```
1 3 1 2 2 2 3 4
4 1 1 1 2 3 4 2
4 4 0 1 0 2 6 4
4 0 0 0 6 0 2 6
5 4 0 5 0 3 6 1
1 5 5 5 3 6 5 6
5 2 5 3 3 3 6 6
```

Fig. 8. Best known solution of the task "Who can find more?"

This task allows a lot of possible modifications.

- The same task but coverage must be unique.
- The same task but coverage must be not unique.
- Reverse task: Some distribution of numbers can be given and the task is to count the number of different coverages by the domino set.

It is quite clear that the original formulation and the reverse task needs to be solved by a computer program.

The two simple modifications can also be solved without computer. However, a computer program, if available, can give great help in checking generated solution candidates.

Dominoes can also be used in other ways. Let's say that some positive integer n *is representable* by dominoes if it is possible to make a row of dominoes from one set and the sequence of points in the dominoe halves in this row corresponds to digits in the number n (one leading empty halve is allowed if number of digits in n is odd). A row of such domino pieces will be named a "representation of n".

So, a representation of 1205 is  , and a representation of 105 –  .

It is clear that it is not possible to represent numbers with digits greater than 6 as well as numbers 453354 and 606, because the corresponding row of pieces cannot be built from one set of dominoes.

Using the above definition one can also define the simultaneous representation of several numbers – if all numbers can be represented by pieces of one set (there are no pieces necessary for representation of several numbers). For example, simultaneously representable are the numbers 122461, 33440001 and 224611.

Contrary, the numbers 134 and 3310 are not representable simultaneously, because both representations need the piece  .

These definitions were necessary to formulate the task "Number and its power" (final of Ugāle'1997):

> *Find an integer n which is simultaneously representable together with some of its powers (n^2, n^3, ...) and the number of domino pieces used in these two representations is as big as possible.*

After the competition this task also was published on the website and six years after competition four solutions with 26 used domino pieces (Domino, 2009) were found by the former participant of IOI'98 and IOI'99 Dmitrijs Rutko. One of these solutions is as follows: 51531210022106635 and its square 2655465606342463301645403311023225. However, it is not proved that 26 pieces is the absolute maximum.

5.12. *Other Types of Tasks*

Tasks come into fashion and go out of it. For example, SUDOKU tasks were quite popular for several years. As it is mentioned above, the simple SUDOKU task is not a hard task for a good programmer. However, the SUDOKU theme can be used in tasks, exploiting popularity of these tasks which allows us to expect that the rules are well-known and teams will be encouraged to solve such tasks.

Task "Reverse SUDOKU" (final of Ugāle'2006):

> *There is given solution of the classic* 9×9 *SUDOKU (at competition one particular SUDOKU solution was given – M.O.). Clear as many squares as you can so that remaining digits (which looks like classic SUDOKU problem) still give this unique solution.*
>
> *Grading: Only solutions with fewer than 51 digit in the table will be graded. Solutions with fewer digits in the table will score more points. If the SUDOKU solution will be not unique, points will not be given.*

There is no well-known algorithm for this task solving. At the same time it is known that theoretical limit is about 17 remaining digits (Sudoku, 2009).

Of course, the categories described in previous sections do not cover all tasks used in Ugāle competitions. Some of them are very specific and there is not an obvious category where to include such tasks. Several tasks with special software provided are not included due to the length of description and many technical details. A similar task type is described by Ribeiro and Guerreiro (2007). Among tasks not presented in this paper are black box testing tasks, tasks where some erroneous proof is given and the error must be found, tasks tightly coupled with physics and where cryptography must be mentioned.

Till now "Ugāle" competitions are competitions for Latvian students, so the main website and all available materials are in Latvian only (Ugāle, 2009). However, English speaking students can try to solve task set of the final of Ugāle'2003 (unofficial translation of tasks is done by I. Stepanovs) (Ugāle, 2003).

6. Conclusions

During its 14 years the idea of the Ugāle competition has shown its vitality. Definitely there is space for competitions with tasks different from the well-known olympiads. A lot of creative tasks was tested in such an experimental environment as Ugāle competition is. Unusual tasks with unusual grading schemas are challenging for contestants as well as for authors of tasks. The work of jury is not simple, because every new task type needs careful investigation and different solutions with or without a computer, as well as grading schemas, must be checked carefully.

Acknowledgements. I would like to thank prof. Kārlis Podnieks for his valuable comments.

References

Andžāns, A. and Ramāna, L. (2002). Latvian–Icelandic project LAIMA. In A. Andžāns and H. Meissner (Eds.), *Creativity in Mathematics Education and the Education of the Gifted Students*: *Proceedings of the International Conference*. Riga, University of Latvia, pp. 13–14.

Anido, R.O. and Menderico, R.M. (2007). Brazilian olympiad in informatics. *Olympiads in Informatics*, **1**, 12.

Baltic Way (2008). *Baltic Way Mathematical Team Competition 2008*. Gdansk, Poland.
`http://www.balticway08.math.univ.gda.pl/?p=history`

BOI (2003). Task "Table". *9th Baltic Olympiad in Informatics*, Tartu, 2003.
http://www.ut.ee/boi/xxx/TABLE.eng.pdf

BOI (2004). *The 10th Baltic Olympiad in Informatics*, Ventspils, 2004. http://www.boi2004.lv/

BOI (2008). *The 14th Baltic Olympiad in Informatics*, Gdyna, 2008. http://b08.oi.edu.pl/

Bonka, D. (2004). IKT ietekme uz matemātikas padziļinātas izglītības sistēmu Latvijā (Influence of ICT on Advanced Math Education System in Latvia). In *Starptautiskās konferences LatSTE'2004 rakstu krājums*, Rīga, LU, p. 89.

Brouwer, A.E. (2006). Sudoku puzzles and how to solve them. *Nieuw Archief voor Wiskunde*, **5/7**(4), 258–259.
http://www.math.leidenuniv.nl/∼naw/serie5/deel07/dec2006/brouwer.pdf

Burton, B.A. (2008). Breaking the routine: events to complement informatics olympiad training. *Olympiads in Informatics*, **2**, 11–12.

Burton, B.A. and Hiron, M. (2008). Creating informatics olympiad tasks: exploring the black art. *Olympiads in Informatics*, **2**, 26–33.

Domino (2009). *The best known known solution of task "Domino numbers"*.
http://vip.latnet.lv/lio/Neatdz/dom2lab.htm

Forišek, M. (2007). Slovak IOI 2007 team selection and preparation. *Olympiads in Informatics*, **1**, 59.

Ginat, D. (2008). The unfortunate novice theme of direct transformation. *Informatics in Education*, **7**(2), 173–180.

IMO (2009). *International Mathematical Olympiad*.
http://www.imo-official.org/

IMO Regulations (2009). *Regulations for an International Mathematical Olympiad*, Item D4.
http://olympiads.win.tue.nl/imo/imoregul.html

IOI (2006). *IOI'2006 Task "Forbidden subgraph"*. http://www.ioinformatics.org/locations/ioi06/contest/day1/forbidden/forbidden.pdf

IOI (2009). *International Olympiad in Informatics*.
http://www.ioinformatics.org

IPSC (2009). *Internet Problem Solving Contest*. http://ipsc.ksp.sk/

Kemkes, G., Cormack, G., Munro, I. and Vasiga, T. (2007). New task types at the Canadian computing competition. *Olympiads in Informatics*, **1**, 81.

LIO (2009). *Latvian Olympiads in Informatics* (in Latvian). http://www.lio.lv

More (2009). *The best known solution of task "Who can find more?"*.
http://vip.latnet.lv/lio/Neatdz/domlab.htm

NMS (2009). A. Liepas Neklātienes matemātikas skola (A. Liepas Correspondence mathematical school, in Latvian).
http://nms.lu.lv/

Open Team Cup (2009). *Командный Открытый Кубок по программированию* (Open Team Cup in programming). http://shade.msu.ru/∼ejudge/

Propp, J. (2009). *Self-Referential Aptitude Test*.
http://www.cs.berkeley.edu/∼lorch/personal/self-ref.html

Pankov, P.S. and Orusulov, T.R. (2007). Tasks at Kyrgyzstani olympiads in informatics: Experience and proposals. *Olympiads in Informatics*, **1**, 132.

Puzzles (2000). *"20-Digit Challenge"*, newsgroup "rec.puzzles".
Posted April 13, 2000 at http://groups.google.com

Ramāna, L. and Andžāns, A. (2002). Advanced mathematical education in Latvia. In A. Andžāns and H. Meissner (Eds.), *Creativity in Mathematics Education and the Education of the Gifted Students*: *Proceedings of the International Conference*. University of Latvia, Riga, p. 6.

Ribeiro, P. and Guerreiro, P. (2007). Increasing the appeal of programming contests with tasks involving graphical user interfaces and computer graphics. *Olympiads in Informatics*, **1**, 149.

Sudoku (2009). *Rules 4th World SUDOKU Championship*.
http://www.szhk.sk/images/stories/dokumenty/pravidla_sutaze.doc

Toom, A. (2005). *Word Problems in Russia and America*, pp. 4–9.
http://www.de.ufpe.br/∼toom/travel/sweden05/WP.PDF

Turskiene, S. (2002). Computer technology and teaching mathematics in secondary schools. *Informatics in Education*, **1**, 150.

Ugāle (2003). *Tasks and Complementary Files of "Ugāle'2003" final.*
http://vip.latnet.lv/lio/lietas/u03_final_english.zip

Ugāle (2009). *Team Competition in Mathematics and Informatics "Ugāle"* (in Latvian).
http://www.uvsk.lv/p113.htm

Vasiga, T., Cormack, G. and Kemkes, G. (2008). What Do Olympiad Tasks Measure? *Olympiads in Informatics*, **2**, 184.

Verhoeff, T., Horvath, G., Diks, K. and Cormack, G. (2006). A proposal for an IOI Syllabus. *Teaching Mathematics and Computer Science*, **4**(1), 193–216.
http://ioinformatics.org/admin/isc/iscdocuments/tmcs-2006-i-verhoeff-horvath-diks-cormack.pdf

WPC (2009). *World Puzzle Championship*.
http://www.worldpuzzle.org/championships/index.html

Zebra (2009). *Task "Who Owns the Zebra?"*
http://orion.math.iastate.edu/burkardt/puzzles/zebra_puzzle.html

M. Opmanis is researcher at the Institute of Mathematics and Computer Science of University of Latvia. He is deputy team leader of Latvian IOI team since 1996 and was team leader of Latvian team at Baltic Olympiads in Informatics since 1995 till 2007. M. Opmanis was head of jury of Baltic Olympiad in Informatics at BOI'1996, 1999 and 2004.

Olympiads in Informatics, 2009, Vol. 3, 101–111

Representational Means for Tasks in Informatics

Pavel S. PANKOV
International University of Kyrgyzstan
A. Sydykov st., 252, Apt. 10, Bishkek, 720001 Kyrgyzstan
e-mail: pps50ster@gmail.com; pps50@rambler.ru

Kirill A. BARYSHNIKOV
Avalon World Group Inc.
United Arab Emirates, Sharjah, SAIF Zone, office A3-067
e-mail: kiryakg@gmail.com

Abstract. Either a task in informatics reflects any image in mind or not; references (hints) to any reality can take place in the task. This reality can relate to local circumstances, the host town, the host state or to sponsors of the informatics olympiad (at the same time, the task should be "culturally neutral"). By our experience of conducting informatics olympiads in Kyrgyzstan since 1985 and submitting tasks to preceding IOIs we classify such references, survey such tasks and propose some techniques to make tasks more interesting and original and to attract different sponsors. Alternative types of tasks are also discussed.

Key words: olympiads in informatics, tasks, reality, local circumstances.

1. Survey of Ways to Generate Tasks and the Aim of Paper

Some authors including Diks *et al.* (2007, 2008) have thoroughly examined the phases of the task preparation process in such a way that it may be applied not only to informatics but to other sciences as well. Regarding informatics itself, among other ideas and classifications, Burton and Hiron (2008) offered two opposite ways to create a good task: "To wrap an abstract task inside a good story" and "To look around and to take inspiration from real things". Let us denote these ways as $A \to R$, *from_algorithm_(or from_abstract)_to_reality* and $R \to A$, *from_reality_to_algorithm*. In (Pankov, 2008) we reviewed ways to generate task ideas based on actions in real and imaginary spaces and called such tasks *natural.* In other words, we suppose that the good task should create an image in the mind of the contestant and that is why we prefer the way $R \to A$. In our experience, even problems from abstract mathematical spaces (Weeks, 1985) can be presented in a *natural* way in programming tasks. What is more interesting – such *natural* presentations are accepted relatively easy by children, giving them the opportunity to get knowledge about complex objects from early childhood.

The aim of this paper is to illustrate the process of taking any real object and creating a task for an informatics olympiad involving this object.

The second section offers classifications of objects, their presentations and ways to involve them in tasks, with some examples.

The third section proposes various types of tasks within these classifications including both generalizations of our tasks (Pankov *et al.*, 2000; Pankov *et al.*, 2003; Pankov and Oruskulov, 2007; Pankov, 2008), reminders of some known tasks (from our point of view) and some new types – "slow algorithms", graphical tasks and tasks of the "black box" type.

We shall not consider the full procedure of creating a task with all restrictions, tests etc. because it is not an aim of this paper and it was considered in details in (Diks *et al.*, 2007; Dicks *et al.*, 2008), Burton and Hiron, 2008) and other publications. We shall give either a brief description of "environment" (under the denotation "Task fragment") or a brief text of a task with a hope that it can be brought up to full-grown.

2. Classifications

2.1. *Classification of the References to Objects in the Text of Task*

In this section we propose to classify "representing" tasks by the explicitness of the definition of the real object:

– RE) explicit definition with the corresponding description of the task; we shall not mention such definitions;

– RI) implicit definition (in such a way that the contestant can guess about it; in national olympiads such guessing can also be involved into the task);

– RM) mixed definition: the name and some features of the object are defined explicitly and the other features are to be guessed from the description.

The second classification in this subsection is by the adequacy of the real object and its image. We shall estimate it by levels of adequacy from A1 (the lowest) to A5 (the highest). In the $A \to R$ case, the adequacy is usually lower.

2.2. *Classification of Objects*

This subsection shows various types of natural objects that can be used in informatics olympiad tasks.

O1) Host country of the olympiad.

O2) Host city of the olympiad.

O3) Host University of the olympiad.

O4) Sponsor of the olympiad.

O5) Local Sights.

O6) Local History, Customs, Events (but tasks at international olympiads should be "culturally neutral", so most tasks will be given at national olympiads).

O7) Location, Local Geography.

O8) National language.

O9) Animals including the current year's animal in the Oriental 12-year-cycle calendar: Mouse, Cow (2009), Tiger, Hare or Cat, Dragon, Snake, Horse, Sheep, Monkey, Hen, Dog, Pig. By tradition, each NOI in Kyrgyzstan has one task of this type.

O10) Numbers. By the tradition of some olympiads in informatics and mathematics, one set of tasks contains the number of the current year. Also, other numbers relating to jubilees or sponsors of the olympiad can be involved.

2.3. *Classification of Images of Objects*

This subsection classifies the images of the tasks objects:

I1) String constant (name, e-mail, URL-address etc.). This is the simplest way.

I2) Graphical presentation of string constant, mentioned in I1.

I3) Graphical image (flag, insignia, logo).

REMARK. If the graphical image is complex and consists of curved lines etc. then it is possible to simplify if for the purpose of a task.

Task fragment 1 (O1). The description of the Kyrgyzstan national flag (see Pankov and Oruskulov, 2007, Task 4): A solar disk with forty beams is placed in the center of the red rectangular background . . . (A4)

In graphical tasks with a small image of the flag we use the simplified version: a yellow circle on a red background (A1).

I4) Location (Address) of the object.

I5) Structure of the object. Classical but not so interesting

Task fragment 2 (O2, A1). A rectangular grid as a map of the streets of the host city (see Task fragment 23 below).

I6) Activity of the object (the most interesting but the most difficult case). The simplest example could be:

Task fragment 3 (including O9). Rules of the possible motion of "actors" (see (Pankov, 2008)).

3. Examples of Tasks

In addition to common types of tasks we offer the following types of tasks, mentioned briefly in (Pankov and Oruskulov, 2007):

T1) Acceleration of an algorithm. An algorithm is given explicitly, but works too slowly. Write a program yielding same results in acceptable time (traditionally, CPU time less than 1 second). At a national olympiad, if this algorithm is classified O1, O2, O6 or O8 then an additional question may be: What does such algorithm mean and where it can be used?

T2) Graphical tasks (with non-formal scoring, convenient for national olympiads). In our opinion, such tasks meet the Statute S1.7 of the IOI regulations "to bring the discipline

of Informatics to the attention of young people". In our experience, such tasks make olympiads more attractive for sponsors and reflect state and national features. Sponsors (usually from the IT industry) see their logos and attributes made more recognizable among young people.

T3) "Black Box" tasks. We refer to black box as a procedure with unknown content. Additional information is given as "genotype" ("inner") or "phenotype" ("external"). The task of the contestant is to write a procedure giving the same results; Its text can differ but results must coincide.

In the following examples we shall not describe restrictions. The final version of the task, with corresponding restrictions, can be derived from the advice in (Burton and Hiron, 2008), section 5 *"Improving the task"*.

REMARK. In our opinion, tasks themselves ought to contain text only; graphical images should be in examples only. We shall write most tasks briefly: "find . . ." besides of "write a program finding . . ." etc. Some tasks built by the means of the above techniques were published in (Pankov *et al.*, 2000; Pankov *et al.*, 2003; Pankov and Oruskulov, 2007; Pankov, 2008).

3.1. *Text Processing with String Constants*

3.1.1. *Tasks about "reading"*

Task 4 (O2, classical). A (long) word W of capital Latin letters is given. How many times can the word PLOVDIV be read from left to right in W (ignoring the other letters)?

Task 5 (O2, classical). Given a two-dimensional array of capital Latin letters P, L, O, V, D, I. How many times can the word PLOVDIV be read in the array, moving only right and down (without ignoring letters)?

Task 6 (O2, Generalization of Tasks 1 and 2). Given a graph (or a directed graph) with some capital Latin letter assigned to each of its vertices. How many paths (directed paths) in the graph carry the word PLOVDIV:

Task 6A: as a sub-word; *Task* 6B: as a sub-sequence; if a vertex can be passed several times by the path.

Task 7 (O2). Given a graph (or a directed graph) with some capital Latin letter assigned to each of its vertices. Find the minimal number of paths in the graph such that the concatenation of the carried paths words is the word PLOVDIV? A vertex can be passed several times.

Task 8 (O1 or O2). Given a graph (or a directed graph) with some capital Latin letter assigned to each of its vertices. Find the length of the shortest path that carries the word (*Task* 8A: PLOVDIV or *Task* 8B: BURGAS)?

REMARK. The principal difference between these words is that all letters in the word "BURGAS" are different, so any "standard" algorithm can be "wrapped" into this word, meanwhile the word "PLOVDIV" demands the modification of such an algorithm.

3.1.2. *Tasks about "transforming"*

Task 9 (O2). Given a (long) word W of capital Latin letters. How many sub-words at least should be deleted from W to get the word PLOVDIV?

EXAMPLE. Input: PLDODVDPLOVXDIV. Output: 2 ["PLDODVD" and "X" to be deleted].

Task 10 (O2). Given a graph (or a directed graph), with capital Latin letters assigned to some or all of its vertices. How many letters at least must be changed to obtain a path that carries the word PLOVDIV?

It is seen that there are many transformation possibilities (deleting, inserting, changing, gluing and their combinations).

There is no inherent difference between the two types of tasks mentioned above. A task on complex "reading" can be presented as a task on simple "transforming" but the range of input data must be different; thousands and even millions of characters are acceptable for "reading" tasks but dozens or hundreds for "transforming".

Task 11 (O8). Words in the Kyrgyz language, containing vowels A, E, I, O, U, and Y, can only have either consecutive same vowels or the following pairs of consecutive different vowels: AY, YA, EI, IE, OU, and UA. Given is a "word" W containing more than one vowel. At least how many vowels must be erased from W to obtain a new word, the sub-sequence of vowels of which contain only permited pairs of consecutive vowels?

EXAMPLE 1. Input: KYRGYZSTAN. Output: 0.

EXAMPLE 2. Input: TOOFEIGUZAEEWYQ. Output: 4 [OOEIUAEEY $\rightarrow$ OOUAY, and the correct "word" is TOOFGUZAWYQ].

REMARK. 1) Actually, there are eight vowels in Kyrgyz language (including OE and UE). 2) Other Turkic languages have similar rules.

3.2. *Graphical Images of String Constants*

Giving formal descriptions of letters as geometric objects is very difficult. For example, the letters L, O, V, I in their simplest geometrical forms (two segments, a circle, a vertical segment) are easy for description but letters P and D are not so easy. The abbreviation "IOI" itself is very convenient for geometrical presentations and transformations.

Task 12 (O3). Given are some points with integer coordinates. How many (at least) points must be added to them to obtain a configuration with symmetry (in their simplest form) of the type of the letter K (horizontal mirror symmetry); of the letter N (central symmetry); of the letter U (vertical mirror symmetry? (KNU is an abbreviation of Kyrgyz National University).

3.3. *Geometrical Images*

3.3.1. *Uncertainty of Restoring by Non-complete Information*

The main difficulty in such tasks is treating the boundaries of the domains.

Preface to the following Tasks 13, 14, 15: "The Bulgarian flag consists of white, green and red (from up to down) horizontal strips of equal size. The ratio of the horizontal size to the vertical one is 5:3".

Task 13 (O1, O9). In the night, a fire-fly can detect its exact position by means of the GPS navigation system and it knows that it is near a big Bulgarian flag. The fire-fly can switch on its lamp only a restricted number of times. Let all corners of the colored strips of the Flag have even integer coordinates and the fire-fly switches on its lamp at points with odd integer coordinates.

Given is a list of colored points as triples of two odd integer numbers and one of the four letters: W (white), G (green), R (red) and D (darkness, i.e., on / off of the Flag). At least two of W, G, R are presented. Find

Task 13A: the greatest possible size of the Flag; *Task* 13B: the boundaries of the center of the Flag.

3.3.2. *Composing*

Let a "block" be a rectangle of size 1×2 (vertical).

Task 14 (O1). A child has blocks of different colors: W – white, WG – half white and half green (one square of the block is white and the other is green), G – green, GR – half green and half red and R – red. Given are five non-negative integer numbers W, WG, G, GR and R.

Task 14A. Find the greatest possible size of a Bulgarian flag which the child can make using these blocks. [The given numbers are large].

Task 14B. At least how many additional blocks must the child make and paint in order to compose a Bulgarian flag? [Some of the given numbers are too small].

Task 15 (O1). Given ten non-negative integer numbers $W1$, $WG1$, $G1$, $GR1$, $R1$, $W2$, $WG2$, $G2$, $GR2$, $R2$ denoting the numbers of colored blocks kept by two children – the first and second child respectively. At least how many blocks of any color must be moved between children to give each of them the opportunity to compose their own Bulgarian flag (two flags could be of different sizes)?

Another version of Task 15:

Task 16 (O1). Given N triples of non-negative integer numbers $(W[k], G[k], R[k])$, denoting the numbers of colored squares of equal size kept by the kth child, $k = 1, 2, \ldots, N$. At least how many squares must be moved between children to give each of them the opportunity to compose their own Bulgarian flag? (Different flags could be of different sizes)?

Example of an implicit presentation of the Bulgarian flag.

Task 17 (O1, T3, RI). Given is a function $C(X, Y)$ that transforms the couples of integers $(X, Y), 1 \leqslant X \leqslant 1000, 1 \leqslant Y \leqslant 1000$ to letters. Its text is invisible to the contestant:

```
{ C=''D'';
  if X > 20 then
  { if 100 > X then
    { if Y > 500 then {if 520 > Y then C=''R''};
      if Y > 519 then {if 539 > Y then C=''G''};
      if Y > 539 then {if 560 > Y then C=''W''};
    };
   };
}.
```

Task 17A. "Inner" information: Variables C, X and Y are integer constants within the interval [1..1000], ">" signs are always between a letter and a number (or vice versa), fewer than 10 statements "if ... then ..." are used and arithmetical operations are not used in the text.

Task 17B. "External" information:

If $X1 < X2 < X3$, $X3 - X1 < 10$ and $C(X1, Y) = C(X3, Y)$ then $C(X2, Y) = C(X1, Y)$.

If $Y1 < Y2 < Y3$, $Y3 - Y1 < 10$ and $C(X, Y1) = C(X, Y3)$ then $C(X, Y2) = C(X, Y1)$.

(Continuation of both tasks) By means of calling and analyzing the function $C(X$,Y)

1) Write a function named $FC(X, Y)$ that gives the same results in a CPU time less than 0.001 second. Its size must be less than 500 bytes. (To prevent including a 1000×1000 array into the function).

2) What does the function $C(X, Y)$ mean?

REMARK. Such a task can be solved by calling the function several times and analyzing the results or by writing a corresponding program (or sequence of programs).

Example of implicit presentation of another object.

Task 18. DISTANCES (O1, O6, RI). Given a natural number N $(13 \leqslant N \leqslant 50)$, and two natural numbers L, M $(1 \leqslant L < M \leqslant 100, M \leqslant 5 * L)$. Your task is to choose N points with integer coordinates on a plane in such a way that the number of different distances between them, being not less than L and not greater than *M,* is as large as possible.

Input: A file with one line containing the integer numbers N, L, and M.

Output: A file with:

1) one line with the numbers N, L, M;

2) N lines with the list of chosen N points. Each line has to contain a label of the point (integer from 1 to N), its x-coordinate, and its y-coordinate (integers between 1 and 10000). All points must be different.

3) One line with the number of pairs of points being in distances within the interval $[L, M]$.

Scoring: if your output data is correct (the criteria of correctness must be here) then your score for one test case is

(1+ 9*(NumPairsInYourAnswer/NumPairsInBestAnswer) 2) rounded down.

REMARK. Despite of square root in the formula for the distance, this task operates on integer numbers, without rounding.

Task 19 (O1, T2, RI, for Kyrgyzstan). Given is the following algorithm which builds some image:

{ K *is integer;* U, V, X, Y *are real;* X: = 20; Y: = 0;
{ *for* K *from* (−3) *to* 4
{ U: = 0.7 * (X−Y); V: = 0.7 * (X+Y);
if K = 4 *then* { U: = 20; V: = 0}
else {*draw line*(3.4*K−1.7, 5−|K|)−(3.4*K, 8−|K|)−(3.4*K+1.7, 5−|K|)}
draw segment (X, Y)−(U, V); X: = U; Y: = V}
draw circle with center (0, 8) *and radius* 2;
}.

A) Choose a scale and show the image on the screen (save this file).

B) Correct the image according to its kind (save the second file).

C) Complete the image with one or two elements at your will according to its kind (save the third file).

Tests:

A) There should be a regular octagon, seven (mountain) peaks within it and a little circle on the middle of the highest peak.

B) The contestants were to guess, that the item A) contains elements of Kyrgyzstan State insignia. The circumscribed octagon must be replaced by a circle. The little circle denotes the sun (behind the mountains). Hence the bottom arch of this circle (within the middle peak) must be erased.

C) Possible elements of the insignia: beams of the sun; the inscription "KYRGYZ REPUBLIC" (in Cyrillic); hints on a surface of the lake (Issyk-Kul); a closer and lower mountain ridge; White Falcon; ears or cotton.

3.4. *Behavior of Animals*

Task fragment 20 (O9, I6). In Korea (host of IOI'2002) the naughtiness of the *cheong-gaeguri*, a small frog, is legendary. A frog always jumps through the paddy in a straight line, with every hop the same length. Different frogs can jump with different hop lengths and in different directions on the intersection points of a grid . . .

We hope that the following task yields new characteristic of a graph.

Task 21 (O9, I6). Given a connected graph of (narrow, sufficiently long) holes underground. One of its vertices is the entrance. A family of given number of mice (with the plan of the graph) is going to install themselves. But firstly they want to examine all the graph (beginning from and returning to the entrance) to discuss results. Find the minimal

time necessary for this purpose. The velocity of a mouse is one edge per minute. A mouse cannot turn back within an edge.

REMARK. The answer is not obvious even for a dendrite graph.

Task 22 (O9, O7, I6). Let a lake looks like an isosceles triangle; the basis of the triangle (northern coast) is 190 km and height (width of the lake) is 60 km. A village is located on northern coast of the lake, 20 km from the western corner. A horse runs with speed of 20 km/h and swims with speed of 10 km/h. Given a point on the coast of the lake, find the minimal time to reach the village from this point with an accuracy of 0.01 hour.

Using real numbers for tasks at informatics olympiads was considered in details in (Opmanis, 2006).

REMARK. This task, motion from south-western coast to a village across Issyk-Kul lake, reflects a historic fact. In commemoration of this feat, the village was named Toru-Aigyr (Bay Stallion).

3.5. *Structure*

We propose the following improvement of *Task fragment* 2.

Task fragment 23 (O2, A2). Let the streets in the virtual city of Plovdiv form a rectangular grid: X and Y are integers, $0 \leqslant X \leqslant N$, $-N \leqslant Y \leqslant N$, $Y \neq 0$. The line $Y = 0$ is the Maritsa river. It can be crossed by the seven bridges only, their x-coordinates are $B = \text{trunk}(N/8)$, $2 * B$, $3 * B$, $\ldots$, $7 * B$. Given the integer N, $8 \leqslant N \leqslant 1000 \ldots$

3.6. *Customs, History, Activity*

A brilliant example of the history with activity is the task "Fish" given at the practice session of IOI'2007.

Task 24. PILLAGERS (O6, I6, A5). Towns with some amounts of fish are situated on a long (straight) coastline. If a town ships F tons of fish to another town which is D km away then hungry pillagers descending from the mountains take $\min(F, D)$ tons. Each tourist needs 1 ton of fish. Given the positions of all towns and amounts of fish in all towns, find the largest integer Y such that each town can accommodate at least Y tourists.

In (Pankov, 2008) it was proposed the following task:

Task fragment 25 (O7, I6). See Fig. 1. *Regions that hosted finals of the NOI* (15 towns) in the paper (Dagiene and Skupiene, 2007).

Two friends with bicycles decided to make photos of these towns for the illustrated history of NOIs. The array of distances (in hours!) between some pairs of these 16 points is given. Write a program calculating the minimal number of days for such enterprise.

3.7. *Graphs*

The following technique transforms a string constant into an abstract graph.

Task 26 (O1, O2, O3, ...). Given is the constant word W, for example "PLOVDIV_IN_BULGARIA" (19 characters; the number and scope of the characters may be increased arbitrarily). Given two integers $1 \leqslant X < Y \leqslant 19$, find the necessary number of steps to reach $W[X]$ from $W[Y]$. At each step one may pass either to an adjacent character or to an identical character at other place in the word.

EXAMPLE. Input: $X = 4; Y = 13$. Output: 4 [$W[X]$ ="V"; W[Y]="U"; way: V-O-L-L-U].

3.8. *Numbers as an Aim*

The following two tasks are very simple but demonstrate "accelerating of a given algorithm" and the fitting of an object.

Task 27 (O10, A5). Given the following

Algorithm Year_Mult;
Integer I, J, K, N; *Boolean* Mult = *true*;
{ *output* ("Enter a natural number N, 1 <= N <= 2009"); *input* (N);
for I = N *to* 2009 { *for* J = N *to* 2009 { *for* K = N *to* 2009
{*if* I*J*K = 2009 *then* {Mult = false; *output*("Mult"; I, J, K)} } } };
if Mult *then output*("Mult_Nothing");
}.

Task 28 (O10, A2). Given the following

Algorithm Year_Add;
Integer I, J, K, N, Add = 0;
{ *output* ("Enter a natural number N, 1 <= N <= 2009"); *input* (N);
for I = N *to* 2009 { *for* J = N *to* 2009 { *for* K = N *to* 2009
{*if* I+J+K = 2009 *then* Add = Add+1} } };
output("Add = "; Add)
}.

(Both tasks:) Write programs implementing fast algorithms that calculate the same results in a CPU time of 1 second.

REMARK. Task 27 is an "apt" use of the number (A5) because its successful solution demands finding prime factors of 2009 itself. Task 28 is a "common" use of the number (A2) because 2009 can be changed to another large number.

4. Conclusion

This paper gives an overview and examples of the ideas that can be used to create competitive tasks for national and international olympiads in informatics.

The tasks built in such a way that would be interesting for young people and attractive for prospective sponsors. Also, such tasks with high level of adequacy (A4, A5) give less advantage to experienced participants because they would not be able to use known algorithms immediately. On the other hand, tasks of (O6) and (T1, T2, T3) are not used at IOIs. So, after conducting the Kygyzstani NOI (usually in March) we select candidates from schoolchildren who have demonstrated good results on set of tasks containing traditional types used at previous IOIs.

We also advise, before visiting any country, to learn more about its language, state symbols, history, geography, and customs.

References

Burton, B.A. and Hiron, M. (2008). Creating informatics olympiad tasks: exploring the black art. *Olympiads in Informatics*: *Tasks and Training*, **2**, 16–36.

Dagiene, V. and Skupiene, J. (2007). Contests in programming: quarter century of Lithuanian experience. *Olympiads in Informatics*: *Country Experiences and Developments*, **1**, 37–49.

Diks, K., Kubica, M. and Stencel, K. (2007). Polish Olympiads in Informatics: 14 years of experience. *Olympiads in Informatics*: *Country Experiences and Developments*, **1**, 50–56.

Diks, K., Kubica, M., Radoszewski, J. and Stencel, K. (2008). A proposal for a task preparation process. *Olympiads in Informatics*: *Tasks and Training*, **2**, 64–74.

Kemkes, G., Cormack, G., Munro, I. and Vasiga, T. (2007). New task types at the Canadian computing competition. *Olympiads in Informatics*: *Country Experiences and Developments*, **1**, 79–89.

Opmanis, M. (2006). Some ways to improve olympiads in iinformatics. *Informatics in Education*, **5**(1), 113–124.

Pankov, P.S., Oruskulov, T.R. and Miroshnichenko, G.G. (2000). *School Olympiads in Informatics* (*1985–2000 years*). Bishkek (in Kyrgyz & Russian).

Pankov, P.S., Oruskulov, T.R. and Miroshnichenko, G.G. (2003). *Olympiad Tasks In informatics, Devoted to Kirghiz Statehood, History of Kyrgyzstan and Great Silk Road.* Bishkek (in Kyrgyz & Russian).

Pankov, P., Acedanski, S. and Pawlewicz, J. (2005). Polish Flag. In *The 17th International Olympiad in Informatics* (*IOI'2005*). *Tasks and Solutions*. Nowy Sacz, 19–23.

Pankov, P.S. and Oruskulov, T.R. (2007). Tasks at Kyrgyzstani olympiads in informatics: experience and proposals. *Olympiads in Informatics*: *Country Experiences and Developments*, **1**, 131–140.

Pankov, P.S. (2008). Naturalness in tasks for olympiads in iinformatics. *Olympiads in Informatics*: *Tasks and Training*, **2**, 115–121.

Weeks, J.R. (1985). *The Shape of Space.* Marcel Dekker, Inc., New York.

P.S. Pankov (1950), doctor of physical-math. sciences, prof., corr. member of Kyrgyzstani National Academy of Sciences (KR NAS), is the chairman of jury of Bishkek City OIs since 1985, of Republican OIs since 1987, the leader of Kyrgyzstani teams at IOIs since 2002. Graduated from the Kyrgyz State University in 1969, is a main research worker of Institute of Theoretical and Applied Mathematics of KR NAS, a manager of chair of the International University of Kyrgyzstan.

K.A. Baryshnikov (1985), manager of the Avalon Worldgroup Inc (Sharjah, UAE). Participated in IOI'2002, in training the Kyrgyzstani teams for IOI'2003 and IOI'2004. Graduated from the Kyrgyz-Russian Slavic University in 2007.

Olympiads in Informatics, 2009, Vol. 3, 112–131

Baltic Olympiads in Informatics: Challenges for Training Together

Timo PORANEN[a] *, Valentina DAGIENĖ[b], Åsmund ELDHUSET[c], Heikki HYYRÖ[a], Marcin KUBICA[d], Antti LAAKSONEN[e], Mārtiņš OPMANIS[f], Wolfgang POHL[g], Jūratė SKŪPIENĖ[b], Pär SÖDERHJELM[h], Ahto TRUU[i]

[a] *Department of Computer Sciences, University of Tampere*
Kanslerinrinne 1 FIN-33014 Tampere, Finland
e-mail: {tp, helmu}@cs.uta.fi
[b] *Informatics Methodology Department, Institute of Mathematics and Informatics*
Akademijos str. 4, Vilnius, LT-08663 Lithuania
e-mail: {dagiene,jurate}@ktl.mii.lt
[c] *Department of Computer and Information Science*
Norwegian University of Science and Technology
e-mail: asmunde@stud.ntnu.no
[d] *Institute of Informatics, Warsaw University, Poland*
e-mail: kubica@mimuw.edu.pl
[e] *Department of Computer Science, University of Helsinki, PL 68, 00014*
e-mail: antti.laaksonen@mbnet.fi
[f] *Institute of Mathematics and Informatics of University of Latvia*
e-mail: martins.opmanis@lumii.lv
[g] *Bundeswettbewerb Informatik*
Ahrstr. 45 53175 Bonn, Germany
e-mail: pohl@bwinf.de
[h] *Department of Theoretical Chemistry Lund University*
P.O.B. 124, SE-22100 Lund, Sweden
e-mail: par.soderhjelm@teokem.lu.se
[i] *TÜ Teaduskool*
Tähe 4-143, 51010 Tartu, Estonia
e-mail: ahto.truu@ut.ee

Abstract. Baltic Olympiad in Informatics (BOI[1]) is an annual informatics competition established by the three Baltic countries Estonia, Latvia and Lithuania in 1995 for upper secondary school students. BOI was later expanded to include all countries located around the Baltic Sea. One of the main goals of the BOI is to bring gifted students together and let them gain experience from an international event before participating in the International Olympiad in Informatics. Another important goal is to bring together the team leaders from different countries and to share their experience by creating common tasks. All tasks are developed and discussed as well as translated

*Corresponding author

[1] BOI is also used as abbreviation for the Balkan Olympiad in Informatics. Within this article, BOI refers only to the Baltic Olympiad.

before the BOI event using online facilities. The paper reviews some parts of the history of the BOI and gives a short glance at the current state of informatics education in the BOI countries. The main attention is focused on the task preparation process as well as presentation and analysis of statistical data from the previous BOI. Finally some development ideas and discussion on the future of the BOI competition are presented.

Keywords: olympiads in informatics, programming competitions, training, task categories.

1. Introduction

One of the most important parts of teaching cognitive skills is the teaching of problem solving (Dagienė and Skūpienė, 2004). Computer programming is one of the modern ways to develop problem solving skills. It may be argued that competitions make teaching of programming more attractive (Verhoeff, 1997). Students that learn basics of programming, soon start to look for opportunities to demonstrate their skills, use distance learning tools like UVA Online Judge[2], share their interests and compare themselves with others. For such students, one of the most effective means to endorse their motivation are the competitions. Competitions allow the students to meet their like-minded peers from all over their home-country as well as from other countries and to build friendships. They may eagerly wait for the next competition, ready to show how their abilities have improved since the previous competitions.

In order to ensure better preparation for the International Olympiad in Informatics (IOI) and to strengthen regional relations, various regional olympiads are organised (e.g., African, Central European, Baltic and Balkan Olympiads). While national olympiads represent informatics teaching traditions of each country, regional olympiads are usually a mini-model of the IOI, allowing participants to experience what they will face in the IOI (Blonskis and Dagienė, 2006).

The main goals of the Baltic Olympiad in Informatics (BOI) are to bring together gifted students, help them to share their scientific and cultural experiences and to provide participating students with the experience of an international competition. Further important goals are to bring together the team leaders from different countries, allowing them to share their experience, e.g., by creating common tasks. Moreover, the BOI is integrated into the IOI team selection process of some of the participating countries, allowing them to take into account the students' results in a competition similar to the IOI. The competition gives the contestants an experience of an international event before they participate in the International Olympiad in Informatics. The participants of the BOI are upper secondary school students interested in the field of informatics and computer science from a maximum of nine countries (Denmark, Estonia, Finland, Germany, Latvia, Lithuania, Norway, Poland, and Sweden) around the Baltic Sea.

[2]Project "Integrating On-line Judge into effective e-learning" (UVa Online Judge) which has been funded with support from the European Commission under the grant number 135221-LLP-1-2007-1-ES-KA3-KA3MP helped in preparing this article.

In Section 2, we give a historical overview of the BOI. In Section 3, we explain how informatics is taught in the BOI countries. In Section 4, we describe how a BOI is organised, and in Section 5 we analyse tasks and solutions of the BOI 2007. The last section discusses our findings and gives ideas to improve organisation of regional contests in future.

2. History of the Baltic Olympiad in Informatics

The International Olympiad in Informatics (IOI) started in 1989. Soon first ideas about creating a regional contest for the Baltic countries emerged. In IOI 1991 in Greece, Håkan Strömberg (Nurmi, 2008) from Sweden proposed to organise a practice contest where all Scandinavian countries or at least Finland and Sweden could participate. However this plan was given up.

With the re-establishment of Estonia's, Latvia's and Lithuania's independence and their official recognition by the international community, teams from Estonia, Latvia and Lithuania were invited to participate in the Fourth International Olympiad in Informatics, which took place in Bonn (Germany) in 1992. At a glass of German beer, the delegation leaders of the three countries (Rein Prank and Indrek Jentson from Estonia, Māris Vītinš and Viesturs Vēzis from Latvia, Gintautas Grigas and Viktoras Dagys from Lithuania) had a discussion on how to help their students to prepare better for the International Olympiad. It was suggested that a common contest of the three countries could help in selecting the four strongest students from each country to participate in the IOI. The discussion about the Baltic Olympiads was continued at the two following IOIs in Mendoza (1993) and in Stockholm (1994).

The First Baltic Olympiad in Informatics took place in Tartu (Estonia), 21–23 April 1995. Following the standards of the IOI, it was agreed that the competitors of the Baltic Olympiad in Informatics would have to solve six problems during two competition days. The delegation of each country would consist of eight participants and two team leaders. Why eight competitors? With a team of eight competitors, team leaders were left with both sufficient choice in selecting their IOI team and the option to include younger students, who would probably join the national team in a year or two, for training purposes. The delegation leaders from each country had to propose three problems in advance, discuss them through electronic mail and select the six competition tasks from all the offered problems; translate them into the local languages after final approval and bring the translated tasks to the olympiad in printed form.

The integrity of the goals of the olympiad and mutual trust of the delegation leaders made it possible to organise a relatively short (3–4 days) and inexpensive event. Naturally, a week-long IOI is a real festivity for the contestants and their delegation leaders, which remains in their memory for challenging problems, new friends and interesting excursions. The Baltic Olympiad in Informatics (BOI), however, can be distinguished from the IOI by cosy and neighbourly atmosphere.

The second BOI was organised in Riga (Latvia) in 1996. Due to financial problems, the number of competitors in each team was decreased to six. The third BOI took place

in Vilnius (Lithuania). While the first two olympiads were a good start for BOI, the first guest, Poland, was invited to the third olympiad in 1997. In 1998 the host country (Estonia) continued the new tradition of inviting neighbouring countries as guests and asked Finland and Sweden to join the BOI in Tartu. In 1999 Latvia invited Finland, Sweden, Poland and USA as guests. However, this was the time to reconsider the concept of member and guest countries. Sweden proposed to host BOI in 2000 and at this time the participating countries from around the Baltic Sea became member countries. Germany was the last Baltic see country to join the Baltic Olympiad in 2001 in Poland. Friendly relationships with Norway made her a permanent participant of BOI. The host countries still maintain the tradition of inviting guests to BOI (e.g., Israel was invited to BOI in 2005 and Switzerland participated in BOI in 2008). The current BOI member countries are Denmark, Estonia, Finland, Germany, Latvia, Lithuania, Norway, Poland, and Sweden.

The organisation of BOI has changed over the years. To keep the event manageable, the number of contestants per team was decreased to 6. The team leaders propose and discuss the tasks in advance, but now each country is asked to submit at least one task proposal (with 9 participating countries there is no more need for each country to come up with three proposals). Even though the tasks are decided in advance, the final formulation is approved during BOI. Modern contest and grading systems are used to manage the contest. The neighbourly help of countries with more experience of managing contests to host countries with less or no experience makes it possible to host well organised contests in all countries.

3. Informatics Education in the Baltic Sea Countries

In all the Baltic Sea countries, topics of informatics (software usage, programming, etc.) are taught in different ways and using various approaches. In Estonia, Finland, Norway, and Sweden software tool usage is integrated to other school subjects, but in Latvia, Lithuania, and Poland secondary schools have some elements of informatics education as separate subjects or modules. Table 1 summarises the current state of informatics education in the Baltic Sea countries' school systems. The first and second columns show the name of the country and its population, respectively. The last column gives an overview of the informatics education in the corresponding country.

In the following eight subsections we give a brief overview of informatics education and the BOI team selection in some of the participating countries

3.1. *Estonia*

In Estonia, three years of primary and six years of elementary school are mandatory for everyone. After that, pupils normally elect to attend either a gymnasium or a vocational school. A gymnasium means another three years of general education, with the prospect to go on to an institution of higher education upon graduation. A vocational school typically takes four years to give a profession in addition to a secondary education. In theory,

Table 1
Informatics education in primary and secondary schools in the Baltic Sea countries

Country	Pop. (mill.)	Informatics education
Denmark	5.4	Included in the subjects in high schools.
Estonia	1.3	Nominally integrated to other subjects. Many schools still choose to teach IT as a separate subject.
Finland	5.4	Integrated to other school subjects. Optional subject in some schools.
Germany	82.5	The situation of education in informatics varies widely between 16 states. In a few states, informatics is a mandatory subject in grades 6 or 8. More often, informatics is an optional subject in grades 9–10 and in most states it is an optional subject in grades 11–13.
Latvia	2.3	In grades 5–7 in primary schools, in secondary schools and gymnasiums informatics is a mandatory subject.
Lithuania	3.4	Information Technology is a mandatory subject in lower secondary levels starting from 5th grade to 10th. Informatics is one of the optional modules in grade 10 as well as in upper secondary levels, grades 11 and 12.
Norway	4.8	Integrated to other subjects. A mandatory technology course including some informatics is coming to the upper level. In some high schools, informatics is an optional subject.
Poland	38.1	Mandatory in all levels, starting from 4th grade.
Sweden	8.9	Integrated to other subjects. In secondary schools, informatics is a separate subject.

it is possible to enter a university after graduating from a vocational school, but in practice it is perhaps a bit more common to go to a vocational school for one and a half to two years to acquire a profession after graduating from a gymnasium and failing to get into a university.

Mastery of information technologies is recognised as an important skill in the national curriculum, but nonetheless informatics (or computer science) is not a separate subject. Instead, pupils are supposed to acquire the necessary skills in the process of using computers to learn other subjects. Schools are allowed, but not required, to offer informatics as a separate course and many schools do so. The curriculum only defines in general terms the ICT skills graduates of comprehensive schools should possess. These requirements describe the usage of computers as tools to create presentations, search for information and perform minimal statistical analysis (compute averages and create diagrams), but no topics usually associated with computer science. Also, no centrally approved text books, lecture plans, or other teaching materials are provided, and schools are expected to make their own decisions. Only a few schools offer computer science or programming classes either as part of a specialisation or in the form of an extracurricular activity.

A national olympiad in informatics has been organised (and supported by the Ministry of Education) since 1988 (with the exception of the year 1991). The participation in the preliminary rounds has ranged from 50 to 150 pupils, with the 30–40 best invited to the national finals. Out of those, 15–20 are further invited to training camps to select the teams for the international competitions. The camps take place on weekends.

Estonia has participated in the IOI since 1992, in the BOI since 1995, and a few times

in the Central European Olympiad in Informatics (CEOI) as a guest country. Since the beginning, the main goal of participating in the BOI has been to give an international experience to the members of the future IOI team. Also, the BOI serves as the last selection round to pick the four IOI team members out of the six or eight BOI team members and usually the IOI team is announced immediately after the BOI.

3.2. *Finland*

In Finland, pupils usually start a voluntary one-year long pre-school when they are six years old, and after that, they attend the compulsory primary schools. Comprehensive school takes a total of nine years and is divided into a lower level (grades 1–6) and an upper level (grades 7–9). After comprehensive school, the two main options to study further are vocational and upper secondary schools.

In the lower level, informatics education is integrated to other school subjects, like Finnish language and literature (information searching, writing with technical tools) and mathematics (logical thinking, combinatorics, etc.). Only in the upper level and in the upper secondary school it is possible in some schools to study information technology as an optional subject. A drawback of the Finnish informatics education system is that the quality and content of teaching depends highly on the particular school's resources and the teacher's own activity, skills and knowledge. There is no general syllabus.

The Finnish national informatics competition for school students (called "Datatähti" which literally means "Data Star") consists of two stages. The first is an open on-line stage, where the students have 2 weeks time and send their solutions by email. Roughly 15 best students are invited to the second and final stage, which is organised on-site. As an incentive to participate, the top 10 competitors are granted a free entry (i.e., without having to take an entry exam) to most universities in Finland. Based on the results, about 8–15 best students are selected to the IOI training. These students receive learning material and monthly programming tasks by email, and they also participate in a training camp, usually organised in late March or early April. Finland's BOI team is selected based on the results of the national competition as well as the level of achievements during training (both email tasks and the training camp). In 2008, 33 students participated in the first round of the national competition. This may seem low, but it is in fact the second highest number since the year 2002. The main reason for joining the BOI was to provide a good practice opportunity for Finland's IOI team.

3.3. *Germany*

In Germany, school education is supervised by the 16 federal states. Each state has its own rules and administration for the school system, so that, in fact, there are 16 different school systems in Germany. In the following, the more or less typical case will be described, if not stated otherwise.

Children enter school at the age of six. After four years of primary school, there are mainly three options for secondary school (that is the official term in Germany; in other

countries, "elementary" is used instead). One of them is the "Gymnasium" that takes eight years to prepare children for academic education. The other school types take six years to prepare for a vocational education. Vocational education is then done in a so-called "dual" way: practical aspects are learned within one company, more theoretical aspects in vocational schools.

For informatics education, the situation varies widely among the 16 states. In 2007, a bachelor thesis investigated the situation in detail (Weeger, 2007). In almost all states, a basic "IT education" of using IT systems is integrated into other subjects within secondary education. In most states, informatics is an optional subject, but in only two states, informatics is compulsory for all students (in three other states, individual schools may decide to make informatics compulsory for their students). The amount of teaching is about 1 hour per week.

"Bundeswettbewerb Informatik" (Federal Contest in Informatics; BWINF) was founded in 1980; since 1985, it is organised annually. According to the dimensions presented by Pohl (2006, 2007), BWINF is a long-time (homework) task contest, with mixed submission of executable programs (all programming languages are allowed, except assembler and machine language) and solution descriptions, and with manual grading of submissions. Tasks cover many aspects of informatics; there are no age or other divisions. Hence, BWINF differs a lot from olympiad-style short-time task contests, with automatic grading of source code submissions.

BWINF takes a year, its finals take place in autumn (September or October). In the following year, about 12 BWINF finalists enter the process of preparing and selecting the German IOI team, together with a few participants from another competition. After two training camps (each 2–3 days long), the number of IOI candidates is reduced to about half. Traditionally, the IOI team is then selected based on performance in a third training camp (5 days). Recently, performance at the BOI has more and more often determined the IOI team selection.

Germany has participated in the IOI since the beginning in 1989. Since 1997, a German delegation has taken part in the Central European Olympiad in Informatics (CEOI; in 2000 Germany became full CEOI member). Since 2001, Germany also sends a delegation to the BOI. Since the CEOI takes place in early summer, usually about a month or two before the IOI, Germany sends its IOI team to the CEOI in order to bridge the training gap between the BOI and the IOI.

3.4. *Latvia*

Basic education is almost the same as in Estonia. In the grades 5–7 there is a mandatory course of informatics. In grades 8–9 there is no separate subject but informatics is integrated in other subjects. Topics like text and image processing, spreadsheets, preparation of presentations, work with files and Internet are discussed.

In secondary schools and gymnasiums informatics is a mandatory subject. In addition to deeper investigation of already taught themes also databases and preparation of web pages are added. In last years there is also the possibility to teach separate subject "programming" which is chosen by specialised schools and gymnasiums. Secondary school

standard is closely integrated with the European Computer Dricing License (ECDL) and successful passing of school course gives possibility to obtain also the ECDL certificate.

Olympiads in informatics in Latvia are organised since 1986 and since 1988 they are supported and conducted by the Ministry of Education and Science. During the last few years, the national olympiad is organised in two groups (grades 8–10 and 11–12). There are three rounds: optional school round, regional round (approx. 100 contestants in each group) and final round (40–45 contestants in each group). After the final round there is a special selection round for participation in the BOI where the best 20 from both groups compete for a place in the BOI team – the best of the selection round are included in the team. The best four according to the BOI results are included in the IOI team. Latvia has participated in the IOI since 1992. It is one of the three co-founders of the BOI.

3.5. *Lithuania*

The Lithuanian school education mainly consists of three stages: elementary (grades 1–4), basic or lower secondary (grades 5–10) and upper secondary (grades 11–12). Full-time education is compulsory for all children from the age 6 or 7 to 16.

The teaching of informatics has a long tradition (Dagienė and Skūpienė, 2007) in Lithuanian schools; a rich experience in the field has been accumulated. The education programme of lower secondary schools, starting with the fifth grade, includes a separate course on IT, a part of which will be integrated with other subjects in future. A total of 68 hours in grades 5–6 are devoted to a course on IT. Thirty-four compulsory hours and 68 integrated hours for IT are suggested in the course designated for grades 7–8.

A course on IT in grades 9–10 is aimed at summarising and systematising students' knowledge as well as at purposeful usage of their skills, drawing attention to the right application of the technologies and their legitimacy. For those who wish to grasp the principles of computer work and its management, an optional module on algorithms shall be proposed (at the moment it is included in a compulsory IT course). For the course on IT in grades 9–10, 34 obligatory hours, 17 optional hours and 17 integrated hours are recommended.

An IT course for upper secondary grades 10–12 is being essentially revised. Several optional modules mostly oriented to the requirements for study courses in higher educational institutions are being developed. The content of IT is directed towards the trends of information technology usage and training in this field in other European countries. Developing algorithms and programming is one of the optional modules.

Lithuania was among the three Baltic countries to initiate the first BOI. The BOI team is selected among the 30 senior division finalists of the national competition. The main criteria are the scores of the finals, former achievements (BOI, IOI medals) and the age. As students of the three last grades (10 to 12) participate in the senior division, the students from younger grades have priority against older students if their scores in the finals are almost equal.

Only the participants of the BOI compete for the right to join the IOI team. In rare cases it happens that there is no fair way to choose exactly six contestants to the BOI. In

that case the extra contestants solve the BOI tasks in Lithuania at the same time as the BOI contestants thus taking part in competition to join the IOI team. IOI team is selected on the basis of the BOI results and (as they might also be approximately equal) taking into account former achievements, scores of national finals and the age.

The time gap between the national finals and the BOI typically is very small (sometimes less than a week) so the competitors have no training camps for the BOI. There is only one week-long training camp in summer before the IOI. Taking part in the BOI is highly important for the IOI contestants. The students know how to compete at home or in the training camp, but when they come to an international event, they have to adjust themselves mentally, sometimes failing to do so because of psychological reasons rather than because of difficult tasks, especially if the IOI is in a distant country. Adjustment process is not always easy, and the BOI with small and cosy community, but at the same time international atmosphere serves great for that purpose.

3.6. *Norway*

In Norway, pupils enter school the year they turn 6 (after a voluntary one-year preschool). Comprehensive school is divided into a lower level (grades 1 to 7) and an upper level (grades 8 to 10). Afterwards, pupils may choose to enter either a four-year vocational school or a three-year theoretical high school (for preparing for university studies in science, economics or humanities). While high school is voluntary, almost everyone enters one of the two types. Unfortunately, informatics is an underrepresented subject. In comprehensive school, one gets superficial introduction to the basic usage of computers through other courses; there is no course dedicated to computers. However, a new course entitled "Technology and design" has recently been introduced to upper-level comprehensive school, and this may offer new opportunities for exposing pupils to informatics. The theoretical high school has a mandatory course in "information management" which mostly consists of learning how to use Microsoft Office. Some high schools offer more advanced courses that include lightweight database usage with Microsoft Access and possibly macros/scripting with Visual Basic for Applications. However, this does not touch upon any theoretical aspects of computing.

The Norwegian Olympiad in Informatics (NIO) started in 2000/2001. It is formally hosted by the Norwegian University of Science and Technology (NTNU), but is run by a practically independent group of volunteers consisting mostly of students who are former IOI contestants. Due to the small number of people organising the NIO, difficulties with acquiring funding, and the lack of informatics education, there are problems reaching out to the high school pupils. Therefore, the number of participants has always been very low – typically between five and fifteen. Recently, the university has become more willing to sponsor NIO, so that the attendance is expected to increase.

There is only one qualification round. Four tasks of varying difficulty (one of them is very simple and one is at least at the BOI/IOI level) are published on web page, and are available there for a period of around three months, during which anyone who is interested may solve the tasks. They then submit solutions consisting of source code as well as code

documentation, algorithm descriptions and proofs of correctness. Points are awarded for each of these categories. Up to the 30 best participants are invited to the onsite finals at NTNU. Due to the low number of contestants, there is no need to use the BOI as a second elimination round. So, the four best contestants from the finals are invited to both the BOI and the IOI. The team that is sent to the BOI is normally a subset of the IOI team – regrettably, the BOI often collides with the high school spring exams, causing some of the pupils to decline to participate. There is no formal training programme (again due to the limited capacity of the organisers), but the participants are urged to solve selected tasks from earlier BOIs and IOIs, and to participate in a week-long national computer science camp called CyberCamp (there is significant overlap between the NIO and CyberCamp, both on the participating and the organising side).

3.7. *Poland*

Children in Poland start their education at the age of 6 with a one-year pre-school course, which is followed by six years of primary school, three years of a gymnasium and 3 or 4 years of a secondary school. Education of, so called, informatics starts in the 4th grade and continues in the gymnasium and the secondary school. However, children rather learn how to use information technology and software tools. Real informatics, including programming, is taught in some secondary schools. As a result, most of the contestants of the Polish Olympiad in Informatics (POI) are autodidacts. On the other hand, a few leading secondary schools have very strong representation in the POI every year – all thanks to active and competent teachers of informatics. Many efforts to improve education of informatics in Poland focus on training of teachers.

The POI (Diks *et al.*, 2007) also tries to influence and improve the education of informatics in Poland. Obviously, the human resources to work directly with all the pupils or teachers are not sufficient. The educational activities are twofold. Firstly, to provide various educational materials: handbooks, task sets, open contest servers, etc. Secondly, to train as many top contestants and their teachers as possible. The biggest such event is a summer training camp for teachers of informatics and POI finalists (excluding last-year pupils, but including the Polish IOI team). Training of the IOI team includes also two regional international contests: the BOI and the Central European Olympiad in Informatics (CEOI).

Qualification of the POI contestants for all the international contests (IOI, BOI and CEOI) is based on the results of the final stage of the POI. When choosing the BOI team, two goals are combined: training of the IOI team and training of promising future top contestants. Therefore, the Polish BOI team consists of top six contestants, excluding the last-year pupils. The BOI results are not used to qualify to the IOI team. Hence, the atmosphere during the contest is less stressful and more friendly and joyful. It seems that the BOI is the most amicable and least formal among all the international competitions in which Poland participates.

3.8. *Sweden*

Sweden has nine years of compulsory school, followed by three years of secondary school (gymnasium). In the compulsory school, informatics is integrated in other subjects, but in secondary school it is taught as a separate subject (30 minutes/week; not mandatory but always offered), with the focus being on using typical office software and the Internet. Many secondary schools offer programming as an optional subject. These courses exist on three levels (up to 2 hours/week) and typically include the fundamentals of a programming language, basic algorithms and special directions such as web programming.

The national programming olympiad has been organised since 1990, with the number of participants increasing to around 300 in the end of the 90s but decreasing to around 120 in the mid-00s. Also the top layer was significantly narrowed, probably reflecting the end of the generation growing up with computers that one had to program to do anything interesting. In the last years, the number of participants has increased again to around 200.

The qualification round is organised at those secondary schools that have interested teachers (currently around 60 schools). During the last three years, students from other schools have had the possibility to qualify through an online contest with a separate set of tasks. However, a major problem is to reach interested students that are not enrolled in programming courses at school. The national final with 30–40 participants is also held at the schools. The tasks are significantly easier than at the international level, and only one or two usually require knowledge of non-trivial algorithms. Although the dominating languages are C++ and Java, any language is allowed and there have been finalists writing in, e.g., Visual Basic, Python, Perl, Ruby and Haskell.

Sweden has never organised any training camp, and therefore the BOI serves an important purpose: it is usually the first time the students meet other persons with similar capabilities, and it is also the spark that ignites their interest for algorithms and motivates them to practice for the IOI. Nevertheless, the team for the IOI is selected already at the national competition; the two extra students in the BOI are selected among the finalists that are not at their last year.

4. Organisation of a BOI Contest

4.1. *Contest Organisation, Schedule and Other Activities*

The typical schedule of the BOI is a bit compressed compared to the IOI. The arrival of teams, opening ceremony and practice session are all scheduled to the first day. Thanks to the regional nature of the event, the travel distances are relatively short and most teams manage to arrive within the planned half-day slot. The combined arrival and opening day is immediately followed by two competition days with some leisure activities (sports, picnic, zoo, etc.) scheduled to fit between the competition rounds and jury meetings. The fourth day is a relaxing one (for the guests, at least) with an excursion, closing ceremony and a party. The last day is scheduled for departure.

Hosts may choose to extend the BOI with an additional excursion day. On the other hand, sometimes even the half-day excursion is skipped and then the awarding and closing ceremony is scheduled already before lunchtime. This way the teams can catch an overnight trip. Having a party in the evening of the closing day is more common, though.

BOI does not have any official regulations or syllabus for the tasks: all related regulations are agreed together with the leaders. In the IOI, a discussion to get an official syllabus has been started (Verhoeff *et al.*, 2006).

Grading systems are mainly developed and maintained by the host country, although there have been some exceptions. During BOI 2000 in Sweden, Polish grading system was used, and in BOI 2005 in Finland, Germany's grading system was used. Also, Lithuania has used Korea's (host of IOI 2002) grading system.

4.2. *Task Selection Process*

Task selection process used in the BOI is quite informal when compared to more formal selection process used in the IOI (Burton and Hiron, 2008; Diks *et al.*, 2008). Tasks are discussed in advance using e-mail. Typically, the call for tasks is sent out in February. About one month is given to prepare the task proposals. It is expected that each country comes up with one proposal; however, it happens that not every country prepares its proposal, while the organising country sometimes has several proposals. This is natural as the organisers usually have a strong scientific team to manage the contest and to work on the tasks. The proposals come in draft version (i.e., the wording may need improvement, constraints need to be fixed) together with some kind of solution suggestion.

The tasks are discussed on-line for two or three weeks. Team leaders ask and discuss various task related questions, point out if they had similar tasks in their national contests or training. In case of similar tasks the discussion goes on to what degree the tasks and solution are similar, can the task still be used in the contest or may be just as warm-up task. Moreover, the assignment of tasks to the two competition days needs to be determined. Typically, team leaders take into account the following principles: (1) the second day should be slightly harder than the first day, and solutions might be slightly longer; (2) similar tasks should not appear on the same day. After some discussion, votes are cast and the task sets are chosen. Then the host scientific committee works on the final versions.

The situation is different with test data. Some organisers prefer to prepare the tests by themselves; some expect the task authors to work on the tests. After task descriptions are finalised, the team leaders translate them in advance before BOI. Just as in the IOI, there are six tasks to solve during two competition days in the BOI. The tasks in the BOI also resemble the tasks in the IOI, although tasks other than traditional input-output tasks have rarely been seen in the BOI.

During BOI the team leaders discuss the tasks once again before presenting final versions of translations. Often the discussions are very short and last less than half an hour. Sometimes ambiguities are discovered, the wording needs to be changed and it takes longer till the final English versions and final translations are prepared.

4.3. *Task Analysis of BOI Contests*

We have analysed all the BOI tasks in the years 1995–2008 and classified them into five categories:

- Combinatorial search tasks where it is possible to go through all reasonable solutions (possibly with some optimisations) and choose the optimal solution.
- Dynamic programming tasks where the problem can be divided into independent sub-problems.
- Graph theory tasks where the problem can be transformed into a graph and solved by a graph algorithm.
- Mathematical tasks which include the tasks concerning arithmetic, geometry, number theory and probability.
- Ad hoc (creative, inventive) tasks which require an original nontraditional solution method or algorithm, and cannot be classified into the above categories.

Of course, it is not always clear which of these categories is the most suitable for a given task. For instance, some tasks can be seen both as combinatorial search tasks and as graph theory tasks. Similarly, almost all tasks include some ad hoc elements. However, this classification, though unavoidably incomplete, sheds some light on the distribution of the problem types in the history of the BOI.

Table 2 shows the number of tasks in different categories in the years 1995–2008. We have divided the years into three groups (1995–1999, 2000–2004 and 2005–2008) to examine long-range changes in the problem types.

In the years 1995–1999, many tasks falling into combinatorial search and ad hoc categories demanded careful implementation of straightforward algorithms. For example, one had to simulate an algorithm given in the task definition, evaluate an arithmetic expression or sort a table according to specific criteria.

In the years 2000–2008, graph theory and dynamic programming tasks dominated and most of the tasks required special knowledge of algorithms. This is an important change in the history of the BOI: in the first years, one could achieve full points from some tasks with basic programming skills, which is nowadays seldom the case.

Table 2
Task classification of the BOI tasks 1995–2008

	1995–1999	2000–2004	2005–2008	Total
Combinatorial search	9	3	2	14
Dynamic programming	3	6	9	18
Graph theory	5	9	5	19
Mathematics	3	5	2	10
Ad hoc	10	7	6	23

4.4. *Competition Results at BOIs and of BOI Countries*

Over the years, the BOI competition has developed. The very first Baltic Olympiads had the goal to help the teams of the three Baltic countries to prepare for the IOI as well as to aid in selecting IOI teams. Therefore the organisers of the first BOIs wanted to keep the high value of the medals and the medals were awarded based on the contestants' scores, not taking into account the number of the participants. The contestants strived to beat at least two members of their team (which would be a ticket to IOI) rather than get a medal. Since the BOI 2000, the IOI conventions for distributing medals (half of the contestants are awarded) were adopted. Since BOI 2001 in Poland, also the maximum score (100 per task, 600 overall) is in accordance with IOI. In all BOI contests, there have been six tasks (three tasks in both days). In the competitions from 1995 to 1999, the maximum score was 200. Maximum score per day was 100, and the maximum score for a single task varied between 20 and 50 according to team leaders' estimations on how hard it is to solve the task. In the BOI 2000, the maximum score was 300.

Table 3 lists the BOI competitions from 1995 to 2008, giving the number of participating countries and contestants, and showing how many and at what score boundaries medals were awarded. When considering contests starting from 2001, the gold medal limit has varied between 315 and 495, and the bronze medal limit has varied between 132 and 255.

As complementary information, Table 4 shows the performance of the BOI countries at the BOIs and, in comparison, at the IOIs. The most successful BOI countries so far have been Poland with 20, Lithuania with 6 and Estonia and Finland with 4 gold medals. Notice also that the USA team got two silver medals in 1999 and the Israel team got one

Table 3
The BOI contests 1995–2008

Year	Location	Countries	Contestants	Medals (G/S/B)	Medal Boundaries (G/S/B)
2008	Gdynia, Poland	10	59	4/9/13	364/205/134
2007	Güstrow, Germany	9	55	4/10/14	315/228/134
2006	Heinola, Finland	9	53	4/8/14	440/365/255
2005	Pasvalys, Lithuania	8	46	4/7/12	495/400/215
2004	Ventspils, Latvia	8	48	5/8/11	362/267/132
2003	Tartu, Estonia	7	48	5/9/15	435/247/145
2002	Vilnius, Lithuania	8	52	4/8/14	400/241/140
2001	Sopot, Poland	8	49	4/8/12	420/250/190
2000	Haninge, Sweden	7	38	2/6/8	264/222/132
1999	Riga, Latvia	7	44	1/3/4	199/157/144
1998	Tartu, Estonia	5	40	2/2/5	152/129/101
1997	Vilnius, Lithuania	4	36	1/2/3	152/127/104
1996	Riga, Latvia	3	20	1/1/1	171/144/114
1995	Tartu, Estonia	3	28	1/3/8	184/154/119

Table 4
Statistics on the BOI countries and medal distributions

Country	Joined BOI	Joined IOI	Students in 1st national round 2008	BOI medals (G/S/B)	IOI medals (G/S/B)
Denmark	2000	1992	Unknown	0/0/5	3/5/13
Estonia	1995	1992	53	4/6/21	5/15/21
Finland	1998	1992	33	4/8/17	5/17/21
Germany	2001	1989	1106	3/13/10	10/22/26
Latvia	1995	1992	200	2/13/26	4/16/28
Lithuania	1995	1992	3307	6/10/21	2/18/29
Norway	2003	2001[3]	0	0/1/1	0/1/3
Poland	1997	1989	949	20/24/13	26/22/21
Sweden	1998	1990	160	2/4/13	11/17/17

[3] Norway also attended in 1990 on a private initiative by a professor.

silver and one bronze medal in 2005. When considering IOI competitions, Poland has received 26, Germany 10 and Estonia and Finland 5 gold medals.

5. Analysis of Tasks and the Solutions of BOI 2007

In order to recognise what kind of basic algorithms and problem solving techniques are required in BOI contests, we analysed tasks and solution submissions of BOI 2007.

5.1. *Tasks*

In the BOI 2007 the following tasks (Battré, 2007) were used: Escape, Sorting and Sound on Day1, and Fence, Points and Sequence on Day 2. The maximum score for one task was 100.

In the task Escape, a group of prisoners is trying to escape from a prison. The only way out goes through a canyon. In the canyon there are soldiers standing on fixed positions. The range of view of soldiers is also fixed. The contestant should write a program that determines whether prisoners can pass the canyon unnoticed. If this is not possible, then the contestant should determine the minimum number of soldiers that have to be eliminated to pass the canyon safely. Partial scores can be achieved with a standard tree traversal algorithm (depth-first-search or breadth-first-search) and to get full points, the contestant should apply a maximum flow algorithm to find a minimum cut of the corresponding graph.

Task Sorting required the contestant to sort a list of players and their scores in decreasing order, using only an operation which moves a player from position i to position j without changing the relative order of the other players. The cost of such an operation is $i+j$. The contestant should find a sequence of sorting moves that minimise the total cost.

The optimal solution requires a dynamic programming algorithm; partial scores can be achieved with a brute force algorithm.

In Sound the contestant should analyse a number sequence representing air pressure to find the maximum length of a subsequence where the difference between the lowest and the highest value is less than or equal to a given parameter. Algorithms based on scanning the sequence more than once will give only partial points; the optimal solution requires scanning the input sequence only once and maintaining suffixes and prefixes of some subsequences. The method can be classified as a dynamic programming approach since the contestant should store values of certain partial solutions and use them to get an overall solution.

The name of the fourth task was Fence. The input is a list of rectangular areas (buildings of an estate) on a plane. One of the buildings is the main mansion of the estate. The goal of the task is to design an algorithm that finds minimum length of a fence for the main mansion such that the fence does not overlap any other rectangles. The problem can be modelled as a graph and can be solved using Dijkstra's algorithm for shortest paths.

The fifth task, Points, was a combinatorial problem where the contestant should find out how many possible ways there are to connect the given $3 * N$ points on a grid to form a polygon. The number of possible configurations should be counted modulo 1,000,000,000. To solve the problem the contestant should invent a correct recurrence equation.

In the last task, Sequence, input is a number sequence which should be manipulated using only the operation *reduce* (i) which replaces two consecutive elements i and $i + 1$ of the sequence with a single element which is maximum of these two elements. The cost of the operation is the maximum of these elements. The goal is to reduce the length of the sequence to 1 with the minimal total cost. Basic dynamic programming and greedy algorithms yield only partial score; to get full score, a greedy algorithm with an auxiliary stack implementation is needed.

5.2. *Solutions*

55 contestants participated in the BOI 2007. 39 of them used C++, 10 used C and 5 used Pascal in their solutions. One contestant used both C and C++. On the first day 464 and on the second day 335 submissions to the grading system were counted.

Table 5 lists, for each task, its type and solution method, statistical data on the number of submissions, scoring and length of the solutions. Second row lists the number of contestants who submitted a solution for the task. Task Sorting got solutions from only 28 contestants and the other tasks received 35–51 solutions, so Sorting can be seen as the hardest task of the contest. Tasks Sound and Sequence got 51 solutions each. The average score for Sorting was only 4.73 points, and for tasks Sound and Sequence contestants got on average 51.8 and 50.91 points, respectively. Average total score was 153.47 (bronze medal limit was 134).

The fourth row of Table 5 contains the average number of submissions for each task of those contestants who submitted at least one solution for the task. In the parenthesis is

Table 5
Solution statistics of the BOI 07 contest

Task	Escape	Sorting	Sound	Fence	Points	Sequence
Task and solution keywords	Graph, dfs-search maximum flow	Number sequence, dynamic programming	Number sequence, dynamic programming	Graph, shortest paths	Mathematical, combinatorics	Mathematical, dynamic programming, greedy
Solutions	44	28	51	40	35	51
Ave. score	22.58	4.73	51.8	4.09	19.36	50.91
Ave. submission	2.05(6)	1.09(5)	2.35(7)	1.87(9)	1.62(7)	2.36(15)
Ave. lines	138.75	99.5	75.29	150.78	195.37	60.86
Ave. words	346.7	267.39	208.59	478.05	1460.31	149.61

the maximal number of submissions from one contestant for the task. We noted that seven submissions were done for the wrong task, and six times a correct file was submitted after the wrong submission, but in one case the last submission remained incorrect.

Rows five and six contain average number of code lines and average number of words (counted using Unix *wc*-command). We took into account only the last submitted file. Contestants scored on average most points from tasks Sound and Sequence and also the length of the solutions for these two tasks were the shortest, on average 75.29 and 60.86 lines and 267.39 and 149.61 words, respectively. Longest solutions were submitted for the task Points (average length 195.37 lines and 1460 words), although there was one extreme case included: one contestant submitted a solution with 1074 lines and 32217 words and the other solutions were roughly ten times shorter. The contestant tried to enumerate in code a high number of distinct solution cases for this task.

There were also very short solutions for some tasks. If the length of the solution was less than ten lines, we noted that the algorithm usually gave only constant solutions (0 or -1) to all test cases.

6. Discussion

This article provides insights into the history of the BOI and into the inner workings of its competition. Although the event aims to be similar to the IOI, it is different in many important aspects. The main difference, probably, is the co-operative task development and selection process. This process ensures that the leaders of all participating delegations are well acquainted with the tasks, so that they will be able to precisely understand the results their contestants achieve in the competition. Thus, the BOI results provide a sound foundation for selecting the IOI teams as well as for improving or adapting further training activities.

It is subject to discussion whether the IOI task development and selection process could be influenced by the BOI spirit. Success at the IOI is very important to many delegations, so that it is necessary to keep tasks secret. Keeping tasks secret to contestants but

opening them to team leaders would require a completely different organisation scheme for the IOI – similar to that used for the IMO (International Mathematical Olympiad). For a smaller number of countries whose delegation leaders trust each other not to communicate tasks to contestants in advance, the BOI model proves to be very successful in organising a shared-load competition every year and integrating countries with different levels of experience in contest organisation.

Our positive experiences in organising regional contests encourage us to recommend similar practices for other countries in the IOI community.

References

Battré, D. (Ed.) (2007). *BOI 2007 tasks and Solutions*. Available at `http://www.boi2007.de/tasks/book.pdf`

Blonskis, J. and Dagienė, V. (2006). Evolution of informatics exams and challenge for learning programming. In *Lecture Notes in Computer Science*: *Informatics Education – The Bridge Between Using and Understanding computers*: *International Conference in Informatics in Secondary Schools – Evolution and Perspectives*, Vol. 4226, pp. 220–229.

Bulotaitė, J., Diks, K., Opmanis, M. and Prank, R. (1997). *Baltic Olympiads in Informatics*. Institute of Mathematics and Informatics, Vilnius.

Burton, B.A. and Hiron, M. (2008). Creating informatics Olympiad tasks: Exploring the black art. *Olympiads in Informatics*, **2**, 16–36.

Dagiene, V. and Skūpienė, J. (2004). Learning by competitions: Olympiads in informatics as a tool for training high grade skills in programming. In T. Boyle, P. Oriogun and A. Pakstas (Eds.), *Proceedings of 2nd International Conference Information Technology*: *Research and Education*, London, pp. 79–83.

Dagiene, V. and Skūpienė, J. (2007). Contests in programming: quarter century of Lithuanian experience. *Olympiads in Informatics*, **1**, 37–49.

Diks, K., Kubica, M. and Stencel, K. (2007). Polish olympiads in informatics – 14 years of experience. *Olympiads in Informatics*, **1**, 50–56.

Diks, K., Kubica, M., Radoszewski, J. and Stencel, K. (2008). A proposal for a task preparation process. *Olympiads in Informatics*, **2**, 64–74.

IMO. International Mathematical Olympiad (2009). `http://www.imo-official.org/`

Nurmi, O. (2008). Personal communication.

Pohl, W. (2006). Computer science competitions for secondary school students: approaches to classification. *Informatics in Education*, **5**(1), 125–132.

Pohl, W. (2007). Computer science contests in Germany. *Olympiad in Informatics*, **1**, 141–148.

Verhoeff, T., Horváth, G., Diks, K. and Cormack, G. (2006). A proposal for an IOI Syllabus. *Teaching Mathematics and Computation*, **4**(1), 193–216.

Verhoeff, T., (1997). The role of competitions in education, future world: educating for the 21st century. In *A Conference and Exhibition at IOI'97*.

Weeger, M. (2007). Synopse zum Informatikunterricht in Deutschland. Bachelor Thesis, TU Dresden. Available at `http://dil.inf.tu dresden.de/schule/Weeger/output.inf.tu-dresden.de/homepages/index9c3a.html`

T. Poranen is a university lecturer working at the University of Tampere, Department of Computer Sciences. He received his PhD degree in 2004 and since then he has been teaching software project related courses. His research interests vary from topological graph theory to software development. He was a deputy team leader of Finland's BOI 2007, IOI 2007 and IOI 2008 delegations.

V. Dagienė is professor working at the Institute of Mathematics and Informatics and Vilnius University. She has published over 100 scientific papers and many methodical works, written more than 60 textbooks in informatics and IT for secondary education. She has been chair of Lithuanian Olympiads in Informatics for many years, established the International Contests on Informatics and Computer Fluency "Beaver". She is vice-chair of the Technical Committee of IFIP for Education (TC3), member of the European Logo Scientific Committee, an elected member of the IOI International Committee (2006-2009). She is the Editor-in-Chief of the international journal "Informatics in Education".

Å. Eldhuset is about to complete his master's degree in computer science at the Norwegian University of Science and Technology, where he also is employed as a teaching assistant. He gives lectures in algorithms, programming and discrete mathematics. He won a bronze medal in IOI 2003 and joined the Norwegian Olympiad in Informatics as a co-organiser upon entering the university; he was deputy team leader and team leader for Norway's IOI delegations in 2005 and 2006, respectively.

H. Hyyrö received PhD in computer science in 2003 and is currently an assistant professor at the Department of Computer Sciences, University of Tampere, Finland. He has been responsible for organizing the algorithmic (i.e., programming) part of the Finnish national informatics competition for high-school students in the years 2007 - 2009. During this time he has also been the leader of Finland's IOI delegations.

M. Kubica, PhD in computer science, assistant professor at Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, Warsaw University, scientific secretary of Polish Olympiad in Informatics, IOI-ISC member and former chairman of Scientific Committees of BOI 2008 in Gdynia, IOI 2005 in Nowy Sacz, CEOI 2004 in Rzeszow and BOI 2001 in Sopot, Poland. His research interests focus on combinatorial and text algorithms.

A. Laaksonen studies computer science at the University of Helsinki. He has been a contestant in BOI and IOI several times: now he takes part in training Finnish teams for these competitions.

M. Opmanis is researcher at the Institute of Mathematics and Computer Science of University of Latvia. He is deputy team leader of Latvian IOI team since 1996 and was team leader of Latvian team at Baltic olympiads in informatics since 1995 till 2007 and on 2009. M.Opmanis was head of jury of Baltic Olympiad in Informatics at BOI 1996, 1999 and 2004.

W. Pohl was educated in Computer Science, and received a PhD in 1997 from the University of Essen, Germany. For many years, he investigated the use of artificial intelligence techniques for the improvement of interaction between humans and machines. In 1999, he changed position and perspective by becoming executive director of the German Federal Contest in Computer Science. Among his responsibilities is to coach the German IOI team and lead the German IOI delegation. Now, his interest lies in improving computer science contests, establishing new ones, and work on diverse other projects, everything in order to popularise computer science among youth. Hence, he coordinates the German participation in the international contest "Bebras". From 2003 to 2006, he was elected member of the IOI International Committee, and briefly held the position of executive director of IOI in 2006.

J. Skūpienė is a younger research fellow in the Informatics Methodology Department in the Institute of Mathematics and Informatics. She has published about 10 scientific papers. She is a member of the Scientific Committee of National Olympiads in Informatics since 1994 and a team leader in IOI since 1996. For a few years she was director of studies of Young Programmers' School, since 2004 she has been a coordinator of informatics section in the National Academy of Students. She is author/co-author of four books on algorithms and algorithmic problems.

P. Söderhjelm holds a PhD in theoretical chemistry concerning how to calculate accurate interaction energies in biological systems. He has a strong interest in problem solving, algorithms, and school issues, and has been involved in the Swedish national programming olympiad since 1999 after having participated twice in IOI. He was the coordinator of BOI 2009.

A. Truu is a software architect with GuardTime AS. He has been involved in programming competitions since 1988, first as a contestant and later as a member of the jury of the Estonian Olympiad in Informatics as well as a team leader to the Baltic, Central European and International olympiads and the coach of Tartu University's team to the ACM ICPC.

Olympiads in Informatics, 2009, Vol. 3, 132–143

Improving the Automatic Evaluation of Problem Solutions in Programming Contests

Pedro RIBEIRO

Departamento de Ciência de Computadores, Faculdade de Ciências, Universidade do Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
e-mail: pribeiro@dcc.fc.up.pt

Pedro GUERREIRO

Universidade do Algarve
8005-139 Faro, Portugal
e-mail: pjguerreiro@ualg.pt

Abstract. Automatically evaluating source program files is a crucial part of programming contests. The evaluation aims at discriminating programs according to their correctness and efficiency. Given the performance of today's computers, in order to be able to distinguish the complexity of solutions, it is often necessary to use very large data sets. This is awkward, because it is against the nature of the stated problem and puts an unintended burden on the input operations. Besides, by advertizing a limit for the size of the input, the problem description gives away information with which the contestants may guess the algorithmic complexity that their solutions must attain. It would be more realistic to omit that information and let the contestants discover the limits by analyzing the problem, using a scientific approach. The complexity of the solution can then be estimated automatically by measuring the execution time of the function that solves the problem in incremental test cases, and plotting it against the size of the input. By calling the function multiple times and taking the overall time, we may use only data files the size of which is related to the nature of the problem being solved.

Key words: programming contests, computer science education, automatic evaluation, asymptotic complexity, IOI, International Olympiads in Informatics.

1. Introduction

Serious programming contests, such as the International Olympiads in Informatics, share the need for an efficient and fair way of evaluating and distinguishing the solutions proposed by the contestants. Nowadays, this is typically done using an automatic method by which the submitted code is compiled and run against a set of pre-defined test cases. This black-box approach is very practical but has several drawbacks, some of which have been identified in previous work (Cormack, 2006; Forisek, 2006; Verhoeff, 2006), and others that are discussed in this paper. In any case, for the participants, the main challenge of programming contests is to develop correct and efficient algorithms for the problems that

are presented, and, therefore, the evaluation procedure must be capable of reliably ascertaining correctness and gauging efficiency.

Using the black-box approach, a program will be deemed correct, or, more appropriately, a program will be *accepted*, if it passes all the tests. If these tests are also set up to be sensitive to efficiency, it may happen that a correct program, i.e., a program that given unlimited time would computed correct outputs for the given inputs, will not be accepted.

The ACM ICPC contest takes the crude, yet practical, all or nothing approach: a program is accepted if and only if it passes all the tests and the tests are designed to exclude solutions with a complexity beyond a certain degree. The IOI, on the other hand, sets up its tests so that correct solutions on high complexity (and low efficiency) may get some points, but necessarily less points than more efficient solutions.

In the former case, a certain level of complexity is required; in the latter, complexity has to be discriminated. In both cases, some tweaking with the tests is necessary, using official solutions provided by the judges. Typically, all test cases must run within a certain amount of time, and that amount of time is known to contestants. Thus, the toughest inputs must be designed so that the judge's solution runs in, say, half the allotted time. The setup gets more elaborate if different languages are allowed in the competition, in order to accommodate the intrinsic computational overhead of each of them. For example, a perfectly fine solution written in Prolog might not measure up in efficiency with an algorithmically equivalent one written in C. Sadly, this extra setup often deters organizers from adding new languages to the set of accepted languages in a contest.

Therefore, we should look for more flexible ways of distinguishing the complexity of solutions. Additionally, we should consider the possibility of omitting from the problem description the information about the maximum size of the input data, in those cases where this limit is not inherent to the problem. Indeed, in most cases, the limit is rather arbitrarily chosen, only so that the available judge solution passes within a margin of safety. Besides, given the speed of modern computers, those limits sometimes are unreasonably large. This disfigures the problem statement, discloses the complexity required for the solution and overemphasizes the runtime importance of reading the input data.

In this paper, we report on a technique we are experimenting, which consists of having the automatic judge run the submitted program on a set of test cases with linearly increasing size and plot the execution time versus the size of the input. The test cases are of a moderate size, and are run many times, to overcome the lack of accuracy of the system clock.

This paper is organized as follows. In Section 2, we give a detailed description of the current kind automatic evaluation used in programming contest, pointing what we consider to be the main drawbacks. In Section 3, we propose several ideas that could lead to an improved evaluation of submitted code, along the lines discussed above. In Section 4, we present some preliminary experimental results that seem to confirm that the ideas have some merit. A conclusion follows, in Section 5.

2. Current Automatic Evaluation

At present almost all programming contests share a very strict model of automatic evaluation. We will focus on the IOI, but the other major programming contests follow a very similar line of thought and our considerations are valid for them.

Current IOI contest regulations (IOI'2008 Contest Rules) allow three kinds of tasks:

- **Batch tasks:** the most traditional, where the program must read data from *standard input*, process it and then produce a result on the *standard output* within some time and memory constraints – contestants must submit the source code of the standalone program;
- **Reactive Tasks:** the program must interact with a provided library and all input output is made within the context of the library, in a way that the next input depends on the previous output – contestants must submit the source code of the program, to be linked with the library;
- **Output Only Tasks:** contestants are given a set of input files for the problem and must only deliver the set of corresponding output files – no program code is submitted. In theory, contestants could compute the output files by hand, but typically they must write programs to compute them. These programs, however, need only to handle the given files, may adjust to them, and have no time limits other than the duration of the contest, and no memory limits, other than the memory available in the contest computer.

Looking at Table 1, we can observe that in the past five editions of the International Olympiads of Informatics, batch tasks prevail largely, allotting for more than 80% of the tasks. The extreme case was IOI 2008, where all tasks were of this type. There are many reasons for this, the main one being that it appears to be difficult to design good problems of other types, but what remains is that batch tasks do constitute the core of IOI. In other programming contests, such as the ACM ICPC, batch tasks are the only used form.

Having acknowledged this state of affairs, we will now focus our attention on batch tasks. With these, in order to score automatically the programs submitted by the contestants, a set of pre-determined input files is prepared beforehand. These input files are aggregated in several test groups (some of which may contain a single file) and a predetermined number of points are allocated to each test group. The submitted code is then

Table 1
Types of tasks in the last five IOIs

IOI Edition	Batch Tasks	Reactive Tasks	Output Only Tasks
2008	6	0	0
2007	5	1	0
2006	4	0	2
2005	5	1	0
2004	5	0	1

compiled and run against each of those test groups, receiving for each one the allocated number of points when it solves it correctly, that is, when it produces a correct output within the specified time and memory constraints for all its constituent input files. The final score is just the sum of points obtained in each test group.

Therefore, the input files, their grouping and the assigned number of points are crucial to the results of the contest. They must be designed and then created very carefully. When doing so, we must take into consideration two main aspects:

- **Correctness**: the program should provide a correct answer in the sense that it should solve all instances of the problem;
- **Efficiency**: the program should be evaluated having in account its asymptotic time and memory complexity.

Evaluating the correction is tricky and impossible to do with perfection in this kind of black-box testing. As Dijkstra famously said *"Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence"* (Dijkstra, 1972). Indeed, we cannot check all possible ways of making an error, and even when we do find that the program is not able to solve a specific test case, the same score (zero) is awarded whether it is a mere implementation bug or a fundamental reasoning error in the algorithm itself. And how can we really compare two different incorrect implementations? How should we distinguish them in terms of score? These shortcomings regarding program correctness have already been analyzed by other authors (Cormack, 2006; Forisek, 2006; Verhoeff, 2006).

Regarding efficiency, typically the judges create a set of correct model solutions of different complexities and the tests are designed in such a way that the model solutions achieve the preplanned number of points. A considerable degree of manual tuning regarding the actual system used is normally needed and there is no guarantee that the submitted programs will score as expected, even if the algorithm matches the required complexity. In general, one cannot predict every possible approach and students always find surprising ways of solving the problem at hand. And when the submitted program goes in a different direction than the ones that modeled the input, the number of points may not be what the problem setter had in mind for that type of solution. For example, how can we compare the efficiency of two programs that solve the exact same set of test groups if the test groups of this set do not cover all the expected complexity range?

Another important factor is input-output. Besides constituting a distracting aspect by itself (Vasiga *et al.*, 2008), it may represent a considerable part of the execution time. This fact complicates the evaluation of the complexity of algorithm used by looking at the overall runtime only. In addition, the language and the type of input-output that is used matters. For example, *cin* and *cout* in C++ are considerably slower than *printf* and *scanf*.

One other important issue is the constant increase in computer performance. When this kind of automatic evaluation started, we were able to use relatively small and self-contained limits for the inputs. These small limits would allow testing even linear time complexities. This is however not true for today's computers. In order to force small asymptotic complexities one must use huge limits that detach the problems from reality. Although realistic statements are nice, we are forced to have things like a sailing ship

with 100,000 masts (Problem Sails, IOI 2007), just so naïve solutions can be rejected. More than that, the poor CPU clock resolution may make it unfeasible to distinguish between small complexities like `O(N)` or `O(N log N)`. The minimum time limit ever used was 0.3 seconds (Problem Training, IOI 2007). Constructing an input that would pass with an `O(N)` algorithm, but not with `O(N log N)` would need a really large `N`. To separate between `O(log N)` and `O(N)` we would need an even larger `N`. But this would mean that the program would probably spend even more time reading the input (linearly) than on the actual solving phase. Besides, the value `N` would probably be too big to fit in memory.

A final important point is that by advertizing in the problem description the maximum size of the data used in the evaluation, we really give a big hint to the contestants. We do avoid the need for dynamic memory allocation, and that is a good thing. However, experienced contestants can more or less easily infer the expected time complexity of the best algorithm for the problem from those limits. And they will immediately try to just create a solution that matches that, without any other concerns. This is in a sense a very different situation from the scientific way of approaching a problem in the spirit of making our best possible effort, without knowing beforehand what that best is (at least without thinking about the problem).

3. Proposed Improvements

As we have observed above, a great deal of effort is put on devising the test cases. Test cases are typically divided in two families: one formed by small test cases (most of them handcrafted) devised to test correction, and the other made up of increasingly larger test cases, usually generated by other programs, that try to evaluate performance. We will focus on ways to improve this second family of tests. Our proposal is to include one or more of the following ideas, which we will then describe in more detail:

1. **Developing a specific function** (together with other auxiliary functions, if convenient) as opposed to developing a full program.
2. **Abstraction of input-output**.
3. **Repeat the same function call** several times to increase the clock precision.
4. **Do not give hints to contestant** about the best expected asymptotic complexity.
5. **Estimate asymptotic complexity** by looking at time-spent behavior on several tests.

For supporting the discussion of these ideas, we will use a toy problem: finding the smallest difference between two numbers of a set of integers. Let's call this problem `SmallDiff`.

Our first proposal is that the task of the contestant in solving the problem might be to develop a specific function rather than a complete program. Actually, this is already done in the TopCoder contests and also in introductory programming courses (Ribeiro and Guerreiro, 2008). This allows the judges to have greater control on how to manage the submitted code and creates the opportunity for new ways of testing which

would be infeasible for full programs. In the context of the problem `SmallDiff`, instead of presenting the traditional detailed input and output specifications, the problem description would specify the signature of the function to be developed, for example `int FindSmallDiff(n,s)` which would receive a set `s` of `n` integers and would return the smallest difference found. This would not in any way make it more difficult for the student to program the solution. In fact, it would do the opposite, by allowing the contestants to concentrate on the core algorithm that solves the task and by minimizing the impact of some of the distractors described in (Vasiga *et al.*, 2008). Indeed, in pedagogical environments, students seem to be caught by these distractors, wasting their effort on subsidiary issues, and then only addressing the true problem when there is not enough time.

In particular, submitting functions instead of full programs would allow minimizing the information processing details of input-output, thus accomplishing the input-output abstraction that we propose. All languages would have a much more balanced and equal performance in this field. Besides, we could now measure the time that the program spent computing the solution, and not the time spent doing input and output. One could argue that by giving the data already in memory to the program, the contestant does not have to think about how to store it and what data structure to use, but we can always give it on the most simplest of forms (a simple chunk of values), unprepared for efficient processing, which would require the contestant to migrate the data to the desired data structure. And we could even provide access to the data in a way to the similar traditional one by providing functions that get the next integer, the next string or whatever other kind of data, from the basic data structure that was used for the function argument. Thus, by abstracting the data we minimize implementation details and language differences, and at the same time we are able assess the core of the solution, without almost any input-output overhead.

Another concept we advocate is that instead of having really large input sizes to test efficiency, we could use smaller ones, whose size is compatible with the nature of the problem. In reality, what deters one from doing that in the present situation is that the clock resolution is too poor and small input sizes will lead to "instant" responses, all with zero time spent and no way of distinguishing among them. If an arbitrary precision clock was available, we could concentrate more in the quality of the test cases and not have to resort to randomization techniques in order to produce huge tests that are virtually impossible to verify manually. This can be done by calling several times the function that solves the problem, with the same arguments or with different arguments of the same size. It is true that an `O(N^4)` algorithm will probably be instantaneous when we run it twice with `N` = 10. But if we run it a few hundred times, what will it happen? We gain the accuracy that we need precisely by running it multiple times! This is the same technique that we might use to measure the thickness of a sheet of paper: if a stack of 100 sheets measures 5 cm, we conclude that the thickness of each sheet is 0.1 mm, even if we do not have an instrument that reaches that accuracy. The main point is that since we have the solution isolated in a function, we can call the function as many times as we want and then just compute the average time spent inside each call. It is true that

one must be careful not to let memory persistence between successive function calls be exploited by dishonest contestants in order to give quicker responses. This may be dealt with, for example, by always using different equal-sized instances of the problem. In the context of problem `SmallDiff`, we could call `FindSmallDiff` in the evaluation script the required number of times, providing arguments that are hardwired in the script or generated randomly, if the overhead of the randomizer is acceptable. Notwithstanding, in other problems, there may be other opportunities for persistence, such as storing pre-calculated values in static memory, and serious consideration has to be given to this issue.

Another idea that we propose is not to give hints to the contestants about the intended program complexity on the problem description. Indeed, just by looking at the maximum input size and allowed runtime, contestants may perceive whether there is a polynomial solution to the problem. This must be one of the first techniques in which IOI competitors are coached, in preparation for the competition. One can get very precise in these matters, and really identify the specific complexity needed, for the problem at hand, for example `O(N^2)`. This has a great impact on the problem solving aspect and the student approaches the problem with a mindset different from the one he would bring, should he have no idea of the targeted complexity. For example: what strategy should a contestant adopt when he is given an optimization problem and he has no clues as to the size of the input? If he did not know in advance whether there is a polynomial solution, would he try to find one or settle with an exponential approach? In the *real world*, this is what happens. We have open problems that we do now know how to solve optimally. In some cases, the problem has polynomial solutions, in other case, it does not. Sometimes we create efficient solutions only to verify afterwards that they fail on some particular test cases. We never know if we are achieving the optimal solution, unless we prove it ourselves. This contrasts sharply with knowing in advance not only the needed complexity but also being satisfied with our solution, not because it is the best possible, but because it solves the particular instances that will be tested within the given constraints. In our opinion, we should favor the more open ended approach, making students think how to really solve the problem and not on how to write a program that passes the test cases of a given size. Note that in what concerns implementations, limits can still be given (thus avoiding the need for dynamic memory allocation), while making clear that those limits are not related to the unknown efficiency that is sought.

In the example problem, `SmallDiff`, consider that it is said that the maximum size of the set is 10. Since the number of problems instances that will be tested remains secret, the contestant cannot know beforehand if the brute-force `O(N^2)` approach would suffice even if it run instantaneously. Neither will he know if simply ordering the numbers in `O(N log N)` would suffice. He would have to think if an `O(N)` is possible. As a side note, one of the authors of this paper, himself a participant in several programming contests, was once exposed to an ACM IPCP type contest where no limits were given. While in some cases this complicated the implementation by requiring dynamic memory allocation (what we avoid, as discussed), the fact was that the contest was very interesting and refreshing and also more challenging, because it was impossible to discover beforehand whether the problem was NP-hard or if it had a greedy or dynamic programming solution.

With no hints on the expected best running time, we needed to analyze the problem more thoroughly and try to figure out whether there were more effective approaches than the ones we were taking so far.

Our final idea regards measuring the time efficiency of the submitted programs. Typically one creates the test cases in such a way that our model solutions of different complexities will have the intended number of points. However, we cannot predict all the different solutions contestants will create and sometimes programs with better complexities than others will score fewer points because of other factors (Forisek, 2006). Passing a specific test case does not also really advertise that a specific time complexity was achieved. Or rather, not passing does not imply that the solution is wrong or that the required complexity was not achieved. In reality, it just shows that that particular test case is solved within the constraints. Pen-and-paper evaluation could be a solution to really scoring the programs with regard to their actual complexity, but that is unfeasible in practice for a competition of the size of an IOI and also moves away from the required objectivity in the evaluation. What one could do is to try to automatically estimate the complexity of the submitted solution by augmenting the data and plotting the time spent computing the result from those data. A curve fitting analysis could then be done in order to estimate the complexity and possibly even extrapolate other time data points. We are aware that this is impossible to do for all possible problems but the fact remains that many contest problems (or program assignments) have solutions with simple complexities than can be approximated automatically. In any case, even a trivial (although imperfect) curve can provide more information than just checking which test cases the program passes, because it give us a real feel for the asymptotic behavior of the function being evaluated. The main point is that by globally analyzing the runtime behavior of the solution, we get more information than by just timing it on each separate test case. We may not be able to infer the exact worst case asymptotic complexity of a program, but we may discover that in practice it spends half the time on a test case with half the data, while another solution for the same problem spends only a quarter of the time and still another one solves the problem in constant time. This is meaningful information that can be used for grading the program.

4. Experimental Results

In order to perform a first evaluation of the practical application of the proposed improvements, we implemented a preliminary, experimental version. As a proof of concept we will be using a simple problem which was presented as one of the easy tasks in a Portuguese IOI training campus. Stripping the problem to its core, we are given an array of integers (positive or negative), and we want to compute the sequence of consecutive elements which has a maximum sum.

We did this example in C but it can easily be ported to other languages. Instead of a complete program, we require a function `int calculate(int n, int v[])` that returns the maximal sequence as defined before, given an array `v` of `n` integers. The

contestant may concentrate only on solving the algorithmic task. We will now focus on how to test programs efficiency, particularly in estimating asymptotic time complexity. We experimented by repeating the exact same function call several times until the elapsed runtime was measurable, at a human scale, say more than one second. Then, the expected time for a single run of the function call is equal to the total time spent divided by the number of times the function was called. With this in mind we went further ahead and calculated the time it takes for a linearly increasing size `n`. We can then look at the data and do statistical analysis in order to discover a good *curve fitting* that can explain the data obtained.

Putting this into practice, we experimented with random test data and we programmed three model solutions of different complexities:

- **Solution A** is a brute force approach with two nested cycles choosing all possible upper and lower limits of the intervals and then running another cycle for each of these pairs in order to calculate its corresponding sum – this approach is `O(N^3)`.
- **Solution B** is a more refined approach, again with two nested cycles for the sequence followed then by a `O(1)` calculation of the corresponding sum using pre-calculated partially sums, which in turn was made in `O(N)` – the global complexity of this approach is `O(N^2)`.
- **Solution C** is even more optimized and simply maintains the current candidate sum on an auxiliary variable, linearly going trough all the numbers and summing while the candidate sum is positive and therefore contributes to a possible best – this approach is `O(N)`.

We then run these solutions within our improved evaluation framework for a series of increasingly bigger `N` (we used {1,4,8,12,16, . . ., 64}) and measured the time the solution takes for that `N` compared to the time the same solution takes for `N = 1`. Fig. 1 is a plot of the results obtained. Note the very nice curves that already visually suggest the corresponding complexities.

We can now use several statistical approaches to discover which class of complexity the observed behavior of the solution best matches. For the sake of simplicity we will just use one simple measure to show how much information this data provides. We calculated the correlation coefficient between the obtained times and the typical and classical complexity functions. The results are illustrated in Table 2. In gray we can see the maximum

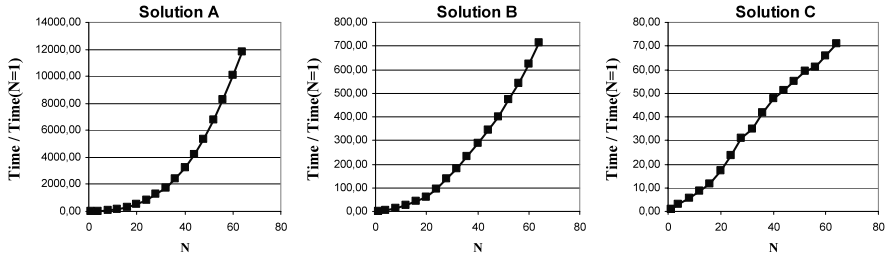


Fig. 1. Comparative time spent for running the three implement solutoins as `N` linearly grows.

Table 2
Correlation coefficient of the time spent by the solutions

Solution	Log N	N	NlogN	N^2	N^3	N^4	2^N	N!
A	0.6848	0.9264	0.9515	0.9912	**0.9993**	0.9869	0.6033	0.5722
B	0.7524	0.9666	0.9835	**0.9998**	0.9848	0.9564	0.5469	0.5183
C	0.8624	**0.9952**	0.9927	0.9586	0.8985	0.8417	0.4136	0.3906

coefficient obtained for each solution. This coefficient corresponds precisely to the class of time complexity to which the solutions belong.

Note than we can not only identify the most probable complexity, but we can also "approximate" the strength of our conviction in the result. Thus, we can use the identified complexity to automatically give more or less points. We did experiments with this method on the solutions submitted by our students and the results that we obtained confirmed our manual analysis of the code and were always strongly correlated to the respective program complexity.

It is out of the scope of this paper to give a more detailed mathematical analysis of the statistical significance or which kind of statistical measure should be used to better fit the data. What stands is that this preliminary yet elegant approach shows that this is indeed a path that can lead to meaningful results. The same approach could also be used when more variables are involved by extending the calculation to higher dimensions.

We are aware that the statistics do not prove that the programs have the corresponding asymptotical complexities. They only mean that, in practice, for the group of selected test cases, the runtime is somehow consistent and correlated with a certain function and therefore appears to grow following a pattern that we were able to identify. For example, when we detect a correlation to a linear function, we recognize explicitly that the program appears to take twice the runtime when the test case doubles the amount of data.

5. Conclusion

The IOI has been running for 20 years but the problems of the early 1990's are similar to those of the late 2000's. The structure of the event has not changed much either. Even the languages used have remained essentially the same. There has been the continuous increase in the speed of computers and on the memory available but, from the point of view of the competition, this is both a blessing and a curse, as we have discussed. In our opinion, the single most significant change occurred when manual evaluation was replaced by automatic evaluation. Automatic evaluation has been made ever more efficient, and lately the results come out minutes after the competition ends. It has also been made more accurate, when the system of points per test was replaced by the system of points by sets of tests targeted at certain complexity. Setting up such sets of tests is a delicate task that may be done in "serious" contests with a scientific committee devoted to it, but may be too time-consuming for the purpose of more informal contests, such as those carried out within programming courses, for pedagogical purposes.

The evaluation system that we envision simplifies the task of setting up a contest in several ways: first, it is not necessary to handle large data sets; second, the problem statements can be more natural, by avoiding the distraction of displaying huge limits for the size of data; third, the problems themselves become more interesting, since there is no clue on the intended complexity, giving contestants the illusion that they may target at something better that what the problem setters may already have achieved, when designing the problem; forth, the human judge does not have to program a solution for each level of complexity, in order to be able to design the test cases for that level of complexity; fifth, different languages can be added at will (provided the automatic judge and the operating system is capable of handling them, but these are other issues), and still be comparable, thus enriching the contest itself and opening it up to larger audiences. And, as a side-effect, by releasing the contestants from the drudgery of input-output, it allows them to concentrate on problem-solving proper. A further benefit is that the system can be used for pedagogical purposes from the early stages, even before students learn how to read and write data files (Ribeiro and Guerreiro, 2008).

More work is needed in order to obtain a robust system, but the preliminary results are promising and seem to indicate that this line of approach has some merit and deserves further consideration. If successful, it would significantly improve the methods of automatic evaluation used in programming contests.

References

Cormack, G. (2006). Random factors in IOI 2005 test case scoring. *Informatics in Education*, **5**, 5–14.

Dijkstra, E.W. (1972). The humble programmer, 1972 Turing award lecture. *Communications of the ACM*, **15**(10), 859–866.

Forisek, M. (2006). On the suitability of programming tasks for automated evaluation. *Informatics in Education*, **5**, 63–76.

IOI'2008 Competition Rules. `http://www.ioi2008.org`

Ribeiro, P. and Guerreiro, P. (2008). Early introduction of competitive programming. *Olympiads in Informatics*, **2**, 149–162.

Vasiga, T., Cormack, G. and Kemkes, G. (2008). What do olympiad tasks measure? *Olympiads in Informatics*, **2**, 181–191.

Verhoeff, T. (2006). The IOI is (not) a science olympiad. 2006. *Informatics in Education*, **5**, 147–159.

P. Ribeiro is currently a PhD student at Universidade do Porto, where he completed his computer science degree with top marks. He has been involved in programming contests since a very young age. From 1995 to 1998 he represented Portugal at IOI-level and from 1999 to 2003 he represented his university at ACM-IPC national and international contests. During those years he also helped to create new programing contests in Portugal. He now belongs to the Scientific Committee of several contests, including the National Olympiad in Informatics, actively contributing new problems. He is also co-responsible for the training campus of the Portuguese IOI contestants and since 2005 he has been deputy leader for the Portuguese team. His research interests, besides contests, are now focused on parallel algorithms for pattern mining in complex networks.

P. Guerreiro is a full professor of informatics at Universidade do Algarve, in Faro. He has been teaching programming to successive generations of students, using various languages and paradigms for over 30 years. He has been involved with IOI since 1993. He was director of the Southwestern Europe Regional Contest, within ACM-ICPC, International Collegiate Programming Contest (2006, 2007), and chief judge of the worldwide IEEExtreme Programming Competition in 2008 and again in 2009. He is the author of three popular books on programming, in Portuguese. His research interests are programming, programming languages, software engineering and e-learning. He currently spends too much time in tasks related to university administration.

Olympiads in Informatics, 2009, Vol. 3, 144–148

Using Subtasks

Willem van der VEGT
Dutch Olympiad in Informatics
Windesheim University for Applied Sciences
PO Box 10090, 8000 GB Zwolle, The Netherlands
e-mail: w.van.der.vegt@windesheim.nl

Abstract. In the Dutch Olympiad in Informatics subtasks are used to create a nice score distribution and to reward contestants for what they were able to solve. Using subtasks can provide a mechanism to distinguish between contestants at the International Olympiad in Informatics.

Key words: informatics olympiad, programming competition, task design.

1. Introduction

The Dutch Olympiad in Informatics (Dutch, 2009) takes three rounds. The first round tasks are published on our website; contestants can work on these tasks for several months, they are allowed to cooperate and use a language of their choice. The annual CodeCup (CodeCup, 2009) task is one of the tasks of this first round. Contestants with a result that is somewhat better then solving just one task are invited for a one day contest at a university. This second round is very selective; we want to identify the top ten students, but we also intend to offer a fair competition in which contestants can show what they are able to do in a few hours.

The top ten students get a trainings course for three days and usually they enter the USACO (USACO, 2009) to get more practice experience. The selection phase ends with a one day competition with three or four IOI-style tasks.

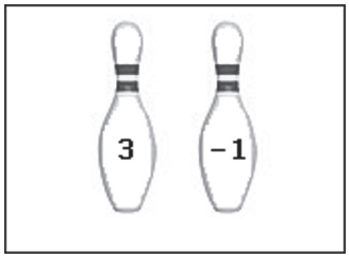
This paper addresses the way we organize our second round tasks. We make intensive use of subtasks, to get a nice score distribution and to allow all contestants to score at least some of the points. This has a lot of advantages, but of course there are also some flaws. We will discuss as examples the tasks "Bowling with numbers" and "Maximum height".

In the past IOI-competitions subtasks have been an item. In Athens (Athens, 1991) and Bonn (IOI'92 Problems, 1992) only one task was presented on each competition day. These tasks were manually graded and grades were given for different parts of the solution. In Eindhoven (The problems, 1995) and Vesprem (The IOI'96 Competition, 1996) grading was already automated; some of the three tasks of a competition day were split up in two subtasks. The subtasks have since disappeared; the introduction of the 50% rule was another way to distinguish between contestants. We think it is possible to use subtasks to realize the goals that are now served using different limits on test cases.

2. Task "Bowling with Numbers"

This was originally a task from the Canadian Computing Competition in 2007 (Canadian, 2007).

In short: A number of pins are placed in a row with equal distances. On each pin there is a number indicating the value of this pin. You get a bowling ball with a certain width and you are able to throw a few rounds. Maximize the total value of the pins you can drop. In the task description two players are mentioned: Alice plays perfect, Bob plays using a specified greedy strategy.



In Canada this task was used in two different sessions. In stage 1 of the competition it was a task with only positive numbers. But in stage 2 the problem was complicated by adding also negative values for the pins. In the Canadian classification system for olympiad tasks this second task was rated at the highest level.

In 2008 we used this task at the Dutch Olympiad in Informatics. We divided it into the following subtasks:

Subtask A:	Most valuable pin. Determine the maximum value available on all of the pins.
Subtask B:	Theoretical maximum. Find the maximum score you can earn if you were able to throw an infinitive number of times with a ball of width one.
Subtask C:	Implement the specified greedy solution
Subtask D:	Find the optimal solution.

None of the 19 contestants was able to find a program that produced an optimal solution for all test cases.

The score distribution was:

Table 1

Results for task "Bowling with numbers"

Subtask	Maximum score	Average score	#contestants with maximum score	#contestants with partial score
A	12	10.4	16	1
B	18	12.0	11	3
C	28	8.7	5	3
D	42	4.4	0	7
Total	100	35.5	0	17

3. Task "Maximum height"

In short: You get a graph with numbered places and connections. Every connection has a maximum heigth. You run a car service on this graph and you need to determine the maximum heigth of the vehicles that you can use, given certain constraints.

This task was submitted for IOI 2004. The task proposal had two subtasks. Since it was not used at IOI 2004, we used it in our national olympiad in 2005. This time it was split up in three different subtasks.

Subtask A: Cab service. What is the maximum height if you want to be able to use every road in the graph, i.e. find the minimal maximum height of all roads.

Subtask B: City service. For every city this service will bring you to all adjacent cities. If there is more then one immediate connection between two cities, you can ignore all but the one with the maximum height.

Subtask C: Overall service. All cities can be reached, but not necessarily by a short path. Now you will be looking for a spanning tree in which the minimal height of all connections is maximal.

Four contestants out of 42 were able to solve all subtasks.
The score distribution was:

Table 2
Results for task "Maximum height"

Subtask	Maximum score	Average score	#contestants with maximum score	#contestants with partial score
A	15	12.2	32	6
B	25	16.0	13	22
C	60	13.6	4	11
Total	100	41.8	2	37

4. Results and Other Experiences

In these two cases the use of subtasks worked rather well. We have got a nice score distribution, almost all contestants scored at least some points on these specific tasks. There is a large gap between the algorithmic complexity of finding a maximum number (subtasks A in both cases) and the optimal solution of an IOI-like task (as in the final subtasks).

There is a strong correlation between the order of the subtasks and the results of the participants. In these examples this has to do with the increasing difficulty of the subtasks. In an earlier olympiad however we used another task where four or five different algorithms were introduced to find a path in a given graph. On this occasion, one of the first subtasks was much more difficult then the other subtasks. Alas, almost no one tried to solve the easy subtasks, because the difficult one frightened them too much.

The system of subtasks also makes it easier to slip in a more theoretical subtask. Once we asked contestants to create a sample input file that could produce a specified output for a given algorithm. We only asked for this file, not for a program or a description how to find it. At the IOI we call this an output only task; we had just an output only subtask.

In this year's second round a game is played in which you can get stuck.

One of the subtasks is to output the minimal number of moves that is at least possible, whatever the initial position of the game, and however badly the game might be played. We even predicted the output for a specific case and asked the contestants to explain this strange output in a few words. In this case, they had to deliver a plain text file. Of course this file was examined and graded manually.

Sometimes subtasks are dependant. It is for instance not possible to solve subtask C without solving subtask B. In these cases it is fair to give some of the credits to those participants that are able to solve subtask B; they already found a part of the overall solution, but failed in the next step.

5. Subtasks at IOI

On several occasions we have spoken on new task types and competition ideas regarding the IOI. The idea of subtasks was not mentioned in the IOI-workshop 2006 (Report, 2006) though spreading the difficulty level was one of the topics discussed.

Using subtasks is one of the ways we can try to spread the difficulty level. The same was aimed by the former 50%-rule. This was intended to distinguish between correct programs and correct and efficient programs. The use of subtasks gives credit to those participants that solved a part of the problem. It also gives the possibility of incorporating small theoretical questions, like specific border cases, sample input or output. It will spread out the results, give more credit to the weak contestants but will still allow us to identify the very best and brightest.

References

Canadian Computing Competition (2007).
`http://cemc.math.uwaterloo.ca/contests/computing/2007/index.html`
CodeCup (2009). `http://www.codecup.nl`
Dutch Olympiade in Informatics (2009). `http://www.informaticaolympiade.nl`
Athens 3rd International Olympiad in Informatics (1991). Athens, Greece, May 19–25.
`http://ioinformatics.org/locations/ioi91/tasks91.txt`
IOI'92 Problems (1992). Bonn, Germany, July. IOI 1992 Bonn.
`http://ioinformatics.org/locations/ioi92/tasks92.txt`

The problems (1995). IOI 1995 Eindhoven.
http://ioinformatics.org/locations/ioi95/contest/index.shtml
The IOI'96 Competition (1996). Vesprem.
http://ioinformatics.org/locations/ioi96/contest/index.shtml
Report of the Competition Workshop (2006). Dagstuhl.
https://phreax.net/bwinf-competition-workshop/wiki/index.php/Main_Page
USACO (2009).
http://contest.usaco.org/ioigate

W. van der Vegt is teacher's trainer in mathematics and computer science at Windesheim University for Applied Sciences in Zwolle, the Netherlands. He is one of the organizers of the Dutch Olympiad in Informatics and he joined the International Olympiad in Informatics since 1992. He was involved in the IOI-workshops on tasks in Dagstuhl (2006) and Enschede (2008). Currently he is deputy team leader of the Netherlands.

Olympiads in Informatics, 2009, Vol. 3, 149–166

20 Years of IOI Competition Tasks

Tom VERHOEFF

Department of Mathematics and Computing Science, Eindhoven University of Technology
Den Dolech 2, 5612 AZ Eindhoven, The Netherlands
e-mail: t.verhoeff@tue.nl

Abstract. The competition tasks at the International Olympiad in Informatics have evolved over its 20-year history. We distinguish three periods in this evolution and highlight it from various viewpoints. The 101 competition tasks are presented in a table that summarizes their task type and difficulty level, and that classifies them according to concepts involved in their problem and solution domains.

Key words: computer science competition, International Olympiad in Informatics, competition tasks, algorithmics, history.

1. Introduction

The first International Olympiad in Informatics (IOI website, 2009) was held in 1989 in Pravetz, Bulgaria, now 20 years ago. The format of the event has evolved over the years, but its main principles have not changed. The high-level goal of the IOI is still that of promoting computer science (CS) among the youth, and of discovering and stimulating young talent in CS. The IOI is still offering a competition in algorithmic problem solving, where solutions must be implemented in one of a few programming languages (Verhoeff *et al.*, 2006).

Let me remind you of two reasons for restricting the competition to algorithmic programming problems, which nowadays are considered as belonging to a narrow subfield of CS. Back in 1989, CS was not a regular topic in the high school curriculum in most countries. Some school pupils did learn some CS – either by themselves, or through relatives or devoted teachers – mostly in the form of computer programming. This limited the options for a CS competition. Secondly, there were no widely accepted standards for the communication of algorithms. In a competition for high school pupils, who had no formal training in expressing algorithms, it was an obvious choice to require that they write their algorithms as computer programs, which are machine executable. That way, a lot of discussion about the intention and validity of an algorithm can be avoided. It also nicely circumvents the language barrier[1] in one direction, because programming languages are quite universal.

[1] An international contest for high school students cannot rely on a single language for presenting the contest tasks. It must also find a way for the contestants to communicate their work to the jury for evaluation.

It may be comforting to some and sobering to others (myself included) that not much has changed with respect to the educational context. CS is still not equally accepted among other topics in the high school curriculum across the globe, and the means for the communication of algorithms has not much improved. Compare this to the 'maturity' of mathematics. It would be unimaginable to drop math from the high school curriculum (though, no doubt, there would be supporters for that), and it has a universally accepted abstract notation for expressing mathematical objects, questions, and arguments.

What has changed substantially in the IOI since its inception is the difficulty level of the contest problems, usually called 'tasks', and the evaluation mechanism to determine a score for submitted work. The nature of the tasks has seen an increase, albeit small, in diversity.

This article will walk you through the 20-year history of IOI tasks, all of which can be found via (IOI website, 2009). It is not intended as a judgment of the past, though I will take the opportunity to raise some critical notes. I will highlight various aspects, which you can view from different viewpoints, such as

Task author who invents and formulates a task;
Contest director who integrates the whole task *set*;
Team leader who votes about the tasks and translates them;
Contestant who is challenged to solve the tasks;
Judge who prepares the evaluation of the submitted work;
Coach who trains contestants to obtain their best result;
Educator who may later use tasks for pedagogical purposes;
Academia and industry who are potential employers of contestants.

The 20-year history will be broken down into three periods, whose characteristics are summarized in Table 1.

Table 1
The major periods in the evolution of IOI tasks

	1st Lustrum[2] 1989 – 1993	**2nd Lustrum[2] 1994 – 1998**	**2nd Decennium 1999 – 2008**
Hours per day	4[3]	5	
Tasks per day	1[4]	3	
Scoring principle	Partly subjective	Based on test runs only	
Grading method	Manual	Automatic	
Present at grading	Evaluator, leader, and contestant		No-one
Grading platform	Contestant computers		Central Linux server[5]
Task types added	Batch	Interactive	Output-only
Preparation	Host SC only		ISC supervision[6]

[2] A lustrum is a five-year period; a decennium a ten-year period.
[3] Except 5 on IOI'92 and IOI'93
[4] Except 3 on Day One of IOI'93
[5] Except IOI'99 and IOI'00
[6] Except IOI'99

This period of 20 years involved 101 competition tasks. Appendix A provides a tabular overview, which is explained in Section 5. It is impossible to cover all of the tasks here in any detail. Having talked to many IOI stakeholders in the past, it became clear that putting together a top-10 of "best" IOI tasks is not a reasonable endeavor. I have picked one 'representative' task from each period, and I apologize for the necessarily subjective nature of my choices.

2. First Lustrum: 1989–1993

The first IOI featured a single task on a single competition day. The next three IOIs each had two competition days with a single task, while at IOI'93 there were three tasks on Day One and only one on Day Two. The input-output requirements were specified rather loosely. The first three years, I/O was through a console interface (keyboard and screen), and thereafter through disk files. Grading was done manually by an evaluator who operated the program on the contestant's computer.

The first IOI task ever, the problem selected from six candidates for competition at IOI 1989, was described as follows, quoted from the on-line version and (Kenderov and Maneva, 1989; p. 21). It had no title.

Problem 1. Given $2 * N$ boxes in line side by side ($N \leqslant 5$). Two adjacent boxes are empty, and the other boxes contain $N - 1$ symbols `"A"` and $N - 1$ symbols `"B"`.
Example for $N = 5$:

```
-----------------------------------------
| A | B | B | A |   |   | A | B | A | B |
-----------------------------------------
```

Exchanging rule: The content of any two adjacent non-empty boxes can be moved into the two empty ones, preserving their order.

Aim: Obtain a configuration where all `A`'s are placed to the left of all `B`'s, no matter where the empty boxes are.

Problem: Write a program that:

1. Models the exchanging of boxes, where the number of boxes and the initial state are to be input from the keyboard. Each exchange is input by the number (from 1 to $N - 1$) of the first of the two neighboring boxes which are to be exchanged with the empty ones. The program must find the state of the boxes after the exchange and display it.
2. Given an initial state finds at least one exchanging plan, which reaches the aim (if there is such a plan). A plan includes the initial state and the intermediate states for each step.
3. Finds the minimal exchanging plan which reaches the aim.

Results: Present at least one solution for the example mentioned above.

Note that the task is composed of several related *subtasks*, allowing contestants to earn *partial credit* for partial accomplishments. In the first lustrum, the subtasks are usually based on the processing phases for the 'full' task, such as reading the input data into a suitably defined data structure, carrying out a single state transformation, writing the resulting state in a suitable format, etc.

An IOI task is not complete without a plan for grading the work submitted by the contestants. Kenderov and Maneva (1989, pp. 40-41) present the following *grading scheme* for the first IOI problem.

Four test examples were prepared by the Jury so that to check the program behavior in various cases. Each test example determined the value of N and the initial state as a sequence of A's, B's and zeroes for the empty boxes.

TEST EXAMPLE 1. $N = 5$, `0 0 A B A B A B A B`

The solution had to be obtained in 4 steps.

TEST EXAMPLE 2. $N = 5$, `A B B A 0 0 A B A B`

The minimal number of steps had to be 3.

TEST EXAMPLE 3. $N = 3$, `0 0 A B A B`

A message for no existence of a solution was expected.

TEST EXAMPLE 4. $N = 4$, `0 A B A 0 B A B`

A message for incorrect input data was expected.

The Jury decided the maximum number of points to be 100, which should be distributed as follows:

Subproblem 1. Up to 10 points.

Subproblem 2. Up to 40 points:

- up to 15 points for finding at least one plan or up to 20 points for all the plans found out;
- up to 20 points for reporting the lack of solution;

Subproblem 3.

- up to 15 points for an attempt made for optimization;
- up to 40 points for complete optimization.

Other 10 points were planned to be given in addition – 5 points if some results had been obtained after executing the program and 5 points for good programming style, and original solution, etc. (at decision of the Jury).

To assess the difficulty level of the tasks, one should keep in mind that in the first lustrum we were still dealing with Apple II and MS-DOS computers (CPU clock rates up to approx. 25 MHz[7]), limiting programs to 640 KB RAM (without resorting to trickery for accessing extended memory). The allowed programming languages were: Pascal, Basic, Logo, and later also C and (non-standardized) C++; at IOI'91, FORTRAN was available as well.

The first IOI problem would still make a nice IOI task, but not for an entire (5 hour) competition day. In a high-school programming course, it can provide material for several lessons, even nowadays. It has a concise formulation and its solution involves some important CS concepts, in particular, graphs. Note that Edsger Dijkstra (1959) published his, now famous, shortest path algorithm for graphs, which can be used in this task, precisely 50 years ago.

[7] Unfortunately, it has proved impossible to trace hardware details for the first lustrum.

3. Second Lustrum: 1994–1998

IOI'94 offered three tasks of diverse difficulty on both competition days, thereby establishing a tradition for the years to come (but that may be overturned at IOI'09).

At IOI'95, a new kind of task was introduced, referred to as *reactive* or *interactive* tasks, as opposed to the 'classical' (single-)*batch*-style tasks. In a batch task, all input data is available at the beginning of the run and it does not depend on the program's behavior. In a reactive task, some output must be produced before new input becomes available. That input may depend on the preceding output. The program has a dialogue with a (programmed) environment, which may behave as an *adversary*. When all input is predetermined but not completely available at the start, we speak of an *online* programming problem (the opposite situation is called *offline*).

The first reactive IOI task was named *Wires and Switches* (IOI'95, Day Two) and described as follows.

Wires and Switches. In Fig. 1, a cable with three wires connects side A to side B. On side A, the three wires are labeled 1, 2, and 3. On side B, wires 1 and 3 are connected to switch 3, and wire 2 is connected to switch 1.

In general, the cable contains m wires ($1 \leqslant m \leqslant 90$), labeled 1 through m on side A, and there are m switches on side B, labeled 1 through m. Each wire is connected to exactly one of the switches. Each switch can be connected to zero or more wires.

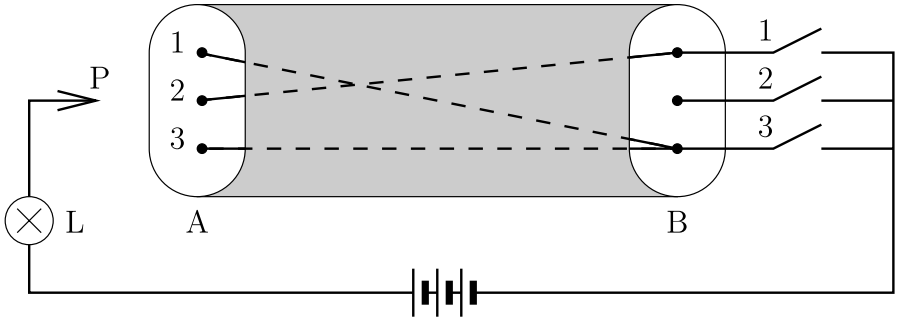


Fig. 1. Cable with three wires and three switches.

Measurements. Your program has to determine how the wires are connected to the switches by doing some measurements. Each switch can be made either conducting or non-conducting. Initially all switches are non-conducting. A wire can be tested on side A with probe P: Lamp L will light up if and only if the sensed wire is connected to a conducting switch.

Your program begins by reading one line with the number m from *standard input*. It then can give three kinds of commands by writing a line to *standard output*. Each command starts with a single uppercase letter: `T` (Test a wire), `C` (Change a switch), and `D` (Done). Command `T` is followed by a wire label, `C` by a switch label, and `D` by a list whose i-th element is the label of the switch to which wire i is connected.

After commands `T` and `C`, your program should read one line from *standard input*. Command `T` returns `Y` (Yes) when the wire's switch is conducting (the lamp lights up), otherwise it returns `N` (No). Command `C` returns `Y` if the new switch state is conducting, and `N` otherwise. The effect of command `C` is to change the state of the switch (if it was conducting then it will be non-conducting afterwards and vice versa); the result is returned just for feedback.

Your program may give commands `T` and `C` mixed in any order. Finally, it gives command `D` and terminates. Your program should give no more than nine hundred (900) commands in total.

Example. Fig. 2 presents an example conversation involving 8 commands relating to Fig. 1.

Standard Output	*Standard Input*
	3
C 3	Y
T 1	Y
T 2	N
T 3	Y
C 3	N
C 2	Y
T 2	N
D 3 1 3	

Fig. 2. Example conversation.

This task was devised by the author when moving into a new home, where the previous owners had lost the chart describing how light fixtures and wall outlets were connected to the switched fuse groups. A key idea for the solution is the binary search, another famous algorithm, which is often not fully appreciated (Feijen and van Gasteren, 1996; §12.3.3). The adversary used in evaluation attempted to minimize the amount of information conveyed at each query. For solutions and further details see (Verhoeff, 1995).

Starting with IOI'94, the grading process was automated so as to ensure that all submitted programs are given the same objective treatment. In preceding years, it had happened that a contestant program was allowed to continue running overnight in the hope that it would produce an answer. Setting an explicit limit on the run time also served to communicate the required efficiency level.

On the programming language front, Logo was dropped as of IOI'95, and Basic disappeared at IOI'98, leaving only Pascal and C/C++.

Another break with the past was the decision to base the score of submitted programs *solely* on a set of automatic *test runs*[8] with carefully constructed secret input data. This was mainly motivated by practical considerations, but is still a controversial issue (Verhoeff, 2006). Although this minimizes the role of human evaluators at the IOI, it does

[8]The organizers use these systematically designed test runs to quantitatively measure program quality. They should not be confused with the – often ad hoc – test runs executed by contestants.

increase the burden of thoroughly preparing the test runs in advance, a job that is often underestimated, and it limits what aspects can be taken into account.

As a consequence of evaluating by automatic test runs only, subtasks had to be defined in a compatible way. Just reading and storing the input or determining some configuration without outputting it cannot be evaluated by test runs. In the second lustrum, subtasks were typically defined by requiring additional output for related but simpler computations, and crediting these separately. Another way to obtain a partial score was to solve a subset of the (secret) input cases.

4. Second Decennium: 1999–2008

The team leaders vote about acceptability of proposed tasks shortly before the actual competition, and then the selected tasks have to be translated. IOI tasks involve a lot of work to prepare. In particular, the choice of bounds and other constraints, and the ingredients for a fair evaluation require ample consideration. The gradual increase in the ability of the contestants and accompanying increase in the diversity and difficulty level of the tasks has only complicated this further. Hence, it is not easy to assess proposed tasks quickly, and thus the team leaders came to face an impossible duty.

At IOI'99 the *International Scientific Committee* (ISC) was established to supervise the preparations for and development of future IOI competitions. The ISC has more time to assess proposed tasks and help ensure their quality. It reports its findings to the General Assembly of all team leaders, who can then take this into account when deciding on tasks.

In the second decennium, another new type of task was introduced. They became known as *output-only* tasks, where the contestants do not submit their programs but only the output files for several given (i.e., non-secret) input files. Of course, creation of the output files requires algorithmic thinking, and in most cases also considerable amounts of programming. In fact, each input file could potentially be tackled by one or more dedicated programs (for instance, a program to analyze and classify the input and a program to handle a particular class of inputs, using the analysis results).

Evaluation of submissions for output-only tasks does not involve programming languages and program compilation and execution. On the other hand, since programs are not collected, there is also no trace of what algorithms the contestants have developed.

The first output-only task was *Double Crypt* at IOI'01. Here is the description of the more interesting output-only task *XOR* (IOI'02, Day One).

XOR. You are implementing an application for a mobile phone, which has a black-and-white screen. The x-coordinates of the screen start from the left and the y-coordinates from the top, as shown in the figures. For the application, you need various images, which are not all of the same size. Instead of storing the images, you want to create the images using the phone's graphics library. You may assume that at the start of drawing an image, all pixels of the screen are white. The only graphics operation in the phone's library is `XOR(L,R,T,B)`, which will reverse the pixel values in the rectangle with top left coordinate (`L`, `T`) and bottom right coordinate (`R`, `B`), where `L` stands for the left, `T` for the top, `R` for the right and `B` for the bottom coordinate. Note that in some other graphics libraries the order of the arguments is different.

As an example, consider the image in Fig. 3 (right). Applying `XOR(2,4,2,6)` to an all white image gives the image on the left. Applying `XOR(3,6,4,7)` to the left image gives the image in the middle, and applying `XOR(1,3,3,5)` to the middle image finally gives the image on the right.

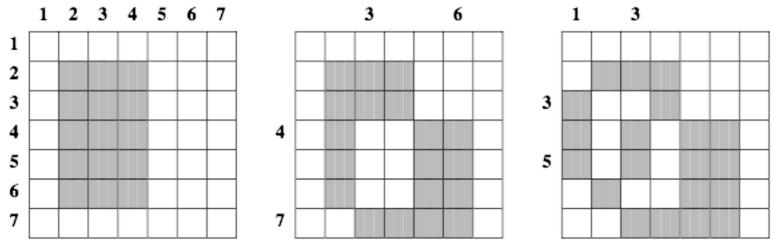


Fig. 3. Screen after each of three successive `XOR` operations.

Given a set of black-and-white pictures, your task is to generate each picture from an initially white screen using as few `XOR` calls as you can. You are given ten input files describing the images, and you are to submit files including the required `XOR` call parameters, not a program to create these files.

Output-only tasks were introduced to make certain classes of 'hard' problems acceptable at the IOI. Evaluation based on a limited set of test runs with secret inputs has the danger of biased results when the input space is 'convoluted'. In that situation, the test runs can explore only a – often very limited – subspace of allowed inputs, thereby possibly not covering corners where the contestant made mistakes or did exceptionally well (Forisek, 2006). An output-only task involves some specific non-secret inputs, so that the contestants know exactly which cases need to be handled.

XOR is such a 'hard' problem. In fact, the organizers did not know an efficient optimal algorithm. It is an *open-ended* task according to (Kemkes *et al.*, 2002). It does, however, have a nice approximation algorithm that is guaranteed to be no more than a factor 2 off the optimum. That is why this task used *relative scoring*, where contestant scores were based on how well they did compared to other contestants. Relative scoring was first featured in the memorable task *Toxic iShongololo* (IOI'97, Day One).

Subtasks as a means of offering an opportunity for partial credit came into disuse in the second decennium. Instead, partial credit could be obtained depending on program efficiency, by defining two or more *subclasses of inputs*, all of them requiring the same 'full' output. A typical way of classifying inputs is by their 'size'. 'Smaller' or 'simpler' inputs could be handled by less efficient, easier programs, whereas 'larger' or more 'complex' inputs required more sophisticated programs.

Test run clustering was introduced at IOI 2005 to reduce the opportunity for harvesting undeserved points by guessing and other forms of opportunistic programming that is not aimed at solving the actual computational task. The points for a cluster of test runs are awarded only if all test runs in the cluster are successful. More generally, the score for a cluster is defined as the minimum of the scores for the constituent test runs.

Another major change in this period (as of IOI'01) concerns the use of *centralized Linux servers* for grading. This allows better control over resources (time, memory, files,

network) used by submitted programs during the evaluation test runs. The contestants submit their work through a web interface to the contest support system, where evaluation takes place. Previously, test runs were executed on the contestant computers after the contest. Note that contestants would not necessarily be required to develop their programs under Linux. Since the central contest server and the contestants' development computers can differ (if only in configuration details), the contestants need a facility to do their own 'test runs' on the server, that is, have their program executed on the server with their own test input.

5. The 101 IOI Tasks

Appendix A presents an overview of the 101 competition tasks that appeared in the past 20 IOIs. There are many aspects one could want to summarize in such an overview, depending on one's background.

Task author: how much effort was needed for task creation; what alternatives, bounds, and other parameters were considered;
Contest director: how balanced was the task *set* as a whole;
Team leader: how much effort was needed for understanding, assessing the quality, and doing the translation;
Contestant: how much time was spent on understanding the task, on designing an abstract solution, on implementing it as an executable program;
Judge: how much effort was needed to prepare test data, checkers, and a summary of the design decisions behind the grading approach;
Coach: how easy were the results to understand and explain; what topics need attention in training;
Educator: how useful were the analysis and solutions; which parts could be used in school, covering what topics;
Academia and industry: to what extent did the tasks contribute to a better image of computer science.

Most of that information is hard to obtain or no longer available. Concerning the translation and comprehension effort, for example, one could measure the word, line, or page count of the task description. There is some variation in this length, ranging from half a page to three pages or more. But I find this metric too superficial. I have chosen to restrict myself to presenting information on

- task type,
- difficulty level (if data was available), and
- classification of technical features.

5.1. *Difficulty Levels*

Difficulty levels are distinguished on the basis of what percentage of contestants were able to 'fully' solve the task. We consider a submission scoring 90% or more as 'fully'

solving the task, i.e., modulo a 'small' mistake. The three main difficulty levels are: **easy** ($> 40\%$ 'fully' solved), **medium** (between 40% and 10%), and **hard** ($< 10\%$). The medium level is subdivided into three sublevels: medium-easy, medium-medium, and medium-hard; see Table 3.

This is admittedly a somewhat arbitrary definition of difficulty level and it is not an absolute measure, but relative to the actual population of contestants for a particular IOI. It is, however, objective and these same criteria have been used in various IOI questionnaires.

Unfortunately, scores per contestant per task are not available for all IOIs. The difficulty level of tasks for a particular IOI can be assessed together as a *set*, by considering the cut-off scores for the medals. Table 9 shows these scores relativized to the maximum score. Keep in mind that according to the IOI Regulations (no more than) half of contestants receive a medal, where the ratio of the number of bronze, silver, and gold medals is $3 : 2 : 1$. That is, approximately one half of all contestants receive no medal, one quarter a bronze medal, one sixth a silver, and one twelfth a gold medal. The lower the relative medal cut-off scores, the fewer points were needed to obtain medals, the harder the task set was for that population of contestants.

5.2. *Task Classification*

The classification concerns these three aspects:

1) the given context and input,
2) the computational task and output,
3) the (algorithmic) ingredients of a full-scoring solution.

The first two items together characterize the *problem domain*, and the third item the *solution domain*. In the overview tables, these three items are separated by semicolons. The terminology follows common practice; for instance, see (Skiena, 2008). It is worthwhile to consider integration into (Verhoeff, 2004) and (Verhoeff *et al.*, 2006). The classification is not entirely satisfactory; some tasks are hard to classify concisely.

Note that Kyryukhin and Okulov (2007) provides technical information (in Russian) on all competition tasks of the first 18 IOIs, including the task descriptions, analyses, solution guidance, classifications, and code snippets (both pseudo code and Pascal implementations). I have occasionally consulted this useful reference when making the classification in Appendix A, but my classification is not the same.

6. Conclusion

The past 20 IOIs involved 101 competition tasks, covering a wide range of CS topics, task types, and difficulty levels. I have summarized them in Tables 5 through 8. I recommend that these tables are verified, refined, and extended with further information, and are kept up to date. In particular, it would be interesting to refine the difficulty assessment, by separately measuring the difficulty of

- *comprehension*, for instance, in terms of number of concepts and definitions involved;
- *mathematical analysis*, for instance, by number and nature of key properties (lemmata) to be discovered;
- *algorithm design* (the focus of this article);
- *implementation* (mostly ignored in this article).

It would also be useful to have a cross-reference of the classification, which lists for each class all related tasks. An on-line data base comes to mind.

Many of the tasks are too hard to use 'as is' in regular CS courses for secondary education. The kind of algorithmics that nowadays plays a role at the IOI is too advanced for incorporation in the high-school curriculum. When looking at the difficulty level of the individual tasks and at the cut-off scores for medals, there is an obvious trend towards relatively harder problems, that is, towards problems and problem sets that are solved by a decreasing percentage of contestants. I do not think this is a good development.

Table 2 lists the five tasks that I have contributed to the IOI competitions.

While analyzing the past, it became painfully clear that the IOI community needs to do a better job at preserving its historic record. All task descriptions are available online[10], but other information is often lacking. In particular, it is desirable to know

- computing platforms and other constraints;
- results *per task* for *all* contestants (possibly anonymized);
- test data (preferably in digital form), checkers, and motivation;
- problem analyses, design options and decisions, exemplary[11] pseudo code and documented program texts[12].

In most cases it is not easy or plainly impossible to (re)evaluate your own attempt at a solution according to the rules of that IOI. Consult (Verhoeff, 2008) for further suggestions on this topic.

The next decennium will certainly see new developments. Computer hardware and programming languages evolve: object-oriented and component-based software on multi-

Table 2
IOI tasks contributed by the author

Task	**Year**	**Remarks**
Wires and Switches	IOI'95	Interactive, see (Verhoeff, 1995)
Median Strength	IOI'00	Interactive, see (Horváth and Verhoeff, 2002)
Double Crypt	IOI'01	Output only, uses AES[9]
Reverse	IOI'03	Output only, output is a 'program'
Mean Sequence	IOI'05	Easy, non-trivial

[9] Advanced Encryption Standard (established as a NIST standard in 2002).
[10] For IOI'90 it unclear on the (IOI website, 2009) which were the actual competition tasks.
[11] worthy of imitation
[12] Kyryukhin and Okulov (2007) come a long way in providing this information for the first 18 IOIs.

core networked processors with interactive multimedia user interfaces require multi-threading, distributed algorithms, and network protocols. The IOI does not have to follow these trends, but if the aim is to stimulate youthful talent, then it is advisable to investigate the possibility of attractive tasks involving newer technologies.

The programming languages that may be used in the IOI competition are now restricted to Pascal and C/C++. Java and Python have been around for more than ten years and are quite popular and accessible. It will be interesting to see how the IOI evolves on the language front, striking a balance between expressing algorithms elegantly and implementing them efficiently for actual execution.

Acknowledgments

The anonymous reviewers provided critical and helpful feedback on the first version of this paper. I would like to thank Gyula Horváth for assisting in the task classification and for collecting the score data for Table 9.

A Tabular Overview of Tasks and Task Sets

Tables 5 through 8 list all 101 IOI competition tasks of the past 20 years together with task type, difficulty level (where objectively assessable), and a classification. Table 3 explains the codes for task types and difficulty levels. Abbreviations for the classification are listed in Table 4. Table 9 shows the relative cut-off scores for medals, which can be used to assess the difficulty level of the task sets as a whole. For more detailed explanations, see Section 5.

Table 3

Type and difficulty codes used in Tables 5 through 8

Code	Task Type
B	Batch
I	Interactive
O	Output only
T	Theoretical

Code	Difficulty	'Fully' solved by[13]
E	Easy	40% – 100%
M	Medium-Easy	30% – 40%
M	Medium-Medium	20% – 30%
M	Medium-Hard	10% – 20%
H	Hard	0% – 10%

[13] Percentage of contestants scoring $\geqslant 90\%$ on the task.

Table 4

Classification abbreviations used in Tables 5 through 8

Code	Problem Features
Ari	Arithmetic
CG	Computational Geometry
Cnt	Counting
Comb	Combinatorial
Dist	Distance
DS	Data Structure
Enum	Enumerating
FSM	Finite State Machine
Gm	(Combinatorial) Game
Gr	Graph
Ham	Hamilton
i	implicit/implied
Mh	Manhattan
Mtch	Matching
Num	Number (integer)
Opt	Optimization
(un)Rank	(Un)Ranking
Rect	Rectangle
Sch	Scheduling
Srch	Searching
Seq	Sequence
Srt	Sorting

Code	Solution Features
Approx	Approximation
BB	Branch & Bound
BFS	Breadth First Search
BS	Binary Search
BT	Backtracking
DC	Divide & Conquer
DFS	Depth First Search
DP	Dynamic Programming
ES	Exhaustive Search
Exp	Exponential
Grdy	Greedy
MI	Mathematical Insight
MM	Meet in the Middle
MST	Min/Max Spanning Tree
(N)P	(Nondet.) Polynomial
Heu	Heuristics
Hash	Hashing
L(A)	Linear (Algebra)
Pc	Precomputation
Rec	Recursive
SP	Shortest Path
SL	Sweep/Scan Line

Table 5

Overview of competition tasks in first lustrum

1. IOI 1989, Pravetz, Bulgaria			
[Exchanging Boxes]	B	**M**	Seq; Srt, Opt; iGr, BFS
2. IOI 1990, Minsk, Belorussian Republic, SU			
[Sliding Puzzle]	B	**H**	2d Num Grid; Comb Opt; iGr, BT
[Watchmen]	B	**H**	Num Seq; Sch, Opt; ES
3. IOI 1991, Athens, Greece			
Square Problem	B	*E*	2d Grid; Ham cycle, Cnt; iGr, BT
S-Terms Problem	B	**H**	Grammar; DS, Enum, String rewriting; _
4. IOI 1992, Bonn, Germany			
Islands in the Sea	B		Nonogram (logic puzzle); Solve; BT
Climbing a Mountain	B		Constraints; Sch, Opt; BT
5. IOI 1993, Mendoza, Argentina			
[Necklace]	B		Color cycle; Cut Opt; L
[Company Shares]	B		Weighted Gr; Enum; all-pair SP
[Rectangles]	B		2d CG, Rect; Areas; z-Buffer, DFS
[Itinerary]	B		Gr; Cycle Opt; DP

Table 6
Overview of competition tasks in second lustrum

6. IOI 1994, Haninge, Sweden			
The Triangle	B		Num triangle; Opt; iGr, DP
The Castle	B		2d Grid; Cnt, Opt; iGr, DFS
The Primes	B		Magic digit square; Primes, Enum; Pc, BT
The Clocks	B		FSM, Sch, Opt; MI, LA
The Buses	B		Num Seq; Sch, Opt; BB
The Circle	B		Num cycle, Opt, Enum; BT
7. IOI 1995, Eindhoven, The Netherlands			
Packing Rectangles	B	*M*	2d CG, Rect; Area, Opt, Enum; ES
Shopping Offers	B	*M*	Constraints; Comb Opt; DP
Printing	T	M	Program analysis & modification
Letter Game	B	*E*	String list; Opt, Enum; ES
Street Race	B	*M*	Gr; Vertex Cnt, Enum; P
Wires and Switches	I	*M*	Find mapping; BS
8. IOI 1996, Veszprém, Hungary			
A Game	I	*E*	2p Gm; Sum Opt; MI+L or DP
Job Processing	B	**H**	Constraints; Sch, Opt; Grdy, MM
Network of Schools	B	**M**	Gr; Opt; DFS
Sorting a 3-valued Sequence	B	*E*	Num Seq; Srt, Opt; Srt, P
Longest Prefix	B	*M*	String list; Opt; Pc, P
Magic Squares	B	*M*	FSM; Event Seq; iGr, BFS
9. IOI 1997, Cape Town, South Africa			
Mars Explorer	B		2d Grid; Opt; iGr
Game of Hex	I		2p Gm; Moves; Heu
Toxic iShongololo	B		3d Grid; Opt; Heu
Map Labeling	B		2d CG; Rect placement; Heu
Character Recognition	B		2d Image; Approx Mtch; Heu
Stacking Containers	I		3d CG, unit cubes; Heu
10. IOI 1998, Setúbal, Portugal			
Contact	B		String; Cnt, Enum; Histogram, Srt
Starry Night	B		2d Image; Subshape Mtch; ES
Party Lamps	B		FSM; Enum; Linear Algebra, Brute Force
Picture	B		2d CG, Rect; Perimeter; SL
Camelot	B		2d Grid, 1p Gm; Opt; iGr, ES
Polygon	B		Number cycle, Ari; Opt, Enum; DP

Table 7
Overview of competition tasks in third lustrum

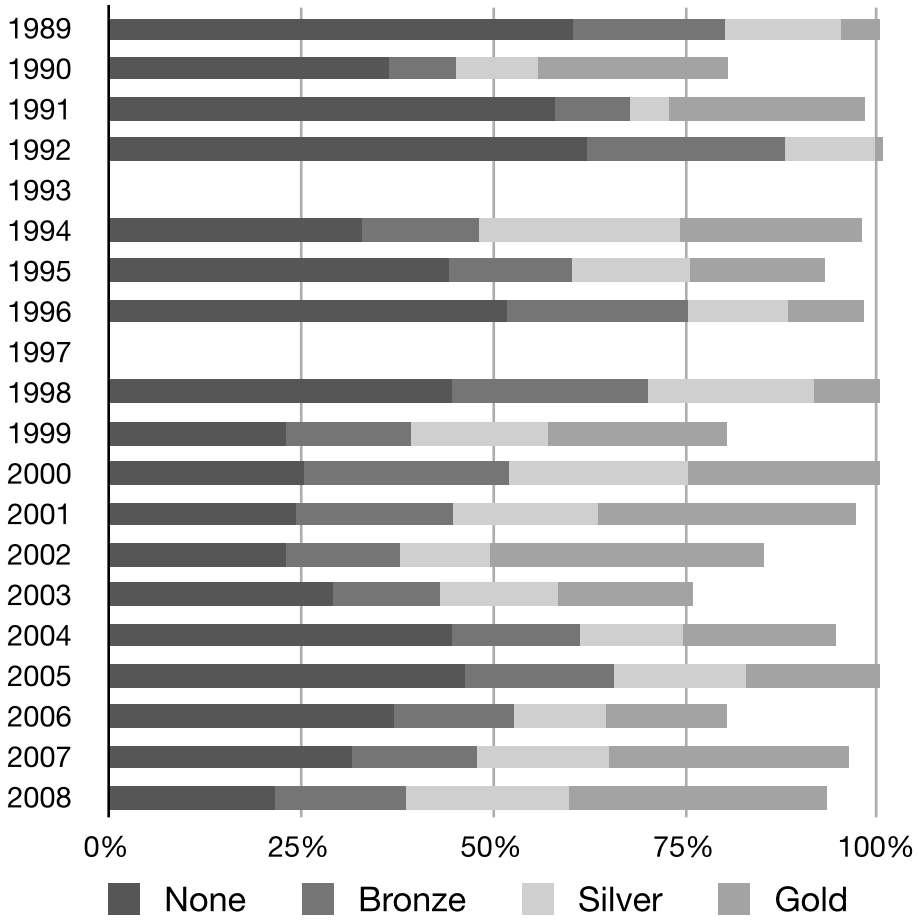
11. IOI 1999, Antalya-Belek, Turkey			
Little Shop of Flowers	B		Constraints; Comb Opt; DP
Hidden Codes	B		String list; Mtch, Opt; P
Underground City	I		2d Grid, implicit maze; Find loc; ES
Traffic Lights	B		Weighted Gr; Path Opt; iGr, SP
Flatten	B		1p Gm, Num Seq; Move Seq, Opt; iGr, LA, NP?
A Strip of Land	B		2d Num Grid; Rect, Area Opt; P
12. IOI 2000, Beijing, China			
Palindrome	B	M	String; Opt; DP
Car Parking	B	**M**	Num Seq; Srt, Opt; Srt, Grdy
Median Strength	I	**M**	Implicit Num Seq; Find median; Heap
Walls	B	M	Planar iGr; Opt; Dual Gr, all-pair SP
Post Office	B	M	1D CG; Comb Opt; DP
Building with Blocks	B	**H**	3d Grid; Set Packing Opt, Rot/Refl; BB
13. IOI 2001, Tampere, Finland			
Mobile Phones	I	**H**	2d Grid; Rect Cnt; Binary Tree
Ioiwari Game	I	M	2p Gm; Win; iGr, minimax
Twofive	B	**H**	String, Num, iSeq; (un)Rank; MI, Pc, DP
Score	I	**M**	2p Gm, Labeled Gr; Win; DFS, minimax
Double Crypt	O	M	Crypto func; Find keys; Hash, MM
Depot	B	**H**	2d Grid, Num Seq; Inverse, Enum; BT
14. IOI 2002, Yong-In, Korean Republic			
The Troublesome Frog	B	**M**	2d CG; Line, Opt; DP, Hash
Utopia Divided	B	**H**	2d CG, Num Seq; Sch; MI, DC, Srt
XOR	O	**H**	2d Grid; Rect, Opt; NP?, Approx
Batch Scheduling	B	**H**	Constraints; Sch; DP
Bus Terminals	B	**H**	2d CG, Mh Dist; Opt; P
Two Rods	I	**H**	2d CG; Rect, Find 2 segments; BS
15. IOI 2003, University of Wisconsin Parkside, U.S.A.			
Trail Maintenance	I	*M*	Gr; Subgraph Opt; MST
Comparing Code	B	**H**	String list; Mtch, Opt; P
Reverse	O	**H**	FSM; program; NP
Guess Which Cow	I	**H**	2d Grid; Query Opt; BFS, bit level
Amazing Robots	B	**H**	2d Grid, Sch; Solve maze; iGr, BFS
Seeing the Boundary	B	**H**	2d CG, Polygons; Visibility; polar SL

Table 8
Overview of competition tasks in fourth lustrum

16. IOI 2004, Athens, Greece			
Artemis	B	**H**	2d Bit Grid; Rect, Opt; Pc, DC, Srt
Hermes	B	M	2d CG; Dist Opt; DP
Polygon	O	**H**	2d CG; inverse Minkowski sum; NP
Empodia	B	M	Num Seq; Subsequence Enum; MI, L
Farmer	B	*M*	Num Seq; Comb Opt; DP
Phidias	B	*M*	2d CG, Rect; Cut Opt; DP
17. IOI 2005, Nowy Sącz, Poland			
Garden	B	**M**	2d Grid; Rect, Opt; Pc, SL
Mean Sequence	B	*E*	Num Seq; Cnt Num Seq; L
Mountain	B	**H**	Num Seq; Answer queries; Binary Tree
Birthday	B	**M**	Num Cycle; Comb Opt; L
Rectangle Game	I	*M*	Gm, Rect, Cut; Win; MI
Rivers	B	**M**	Num Tree; Comb Opt; DP
18. IOI 2006, Mérida, Yucatán, Mexico			
Forbidden Subgraph	O	**H**	Gr; Subgraph Opt; NP
Pyramid	B	**M**	2d Num Grid; Rect, Opt; Pc, Binary Tree
Mayan Writing	B	*E*	String; Cnt; L
A Black Box Game	IO	**H**	2d Grid; Find configuration; Rec, Exp
The Valley of Mexico	B	M	Gr; Planar Ham path; DP
Joining Points	B	**H**	2d CG, 2 Point sets; 1p Gm; Rec, DC
19. IOI 2007, Zagreb, Croatia			
Aliens	I	M	2d Grid; Find center; BS
Flood	B	**H**	2d Mh CG; Segment Enum; dual iGr, BFS
Sails	B	**H**	Num Seq; Comb Opt; Grdy, Diff Tree
Miners	B	*E*	String; Sch, Opt; DP
Pairs	B	**H**	1d/2d/3d Mh CG; Pair Cnt; Srt, SL
Trainings	B	**H**	Gr, Tree; Even Cycle, Opt; DP
20. IOI 2008, Cairo, Egypt			
Type Printer	B	*E*	String list; Sch, Opt; Trie, DFS
Islands	B	**H**	Weighted Gr; Path Opt; MI, Tree diameter
Fish	B	**H**	iGr; Comb, Cnt; MI, Srt, Binary Tree
Linear	B	*M*	String, iSeq; Comb, Rank; MI, Pc, DP
Teleporters	B	**H**	1d CG; Motion Opt; iGr, DFS/BFS, Grdy
Pyramid Base	B	**H**	2d CG; Rect, Area Opt; MI, BS, SL, DS

Table 9

Relative cut-off scores for medals (no data available for 1993, 1997)



References

Dijkstra, E.W.D. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, **1**, 269–271.

Feijen, W.H.J. and van Gasteren, A.J.M. (1996). Programming, proving, and calculation. In C.N. Dean and M.G. Hinchey (Eds.), *Teaching and Learning Formal Methods*, Academic Press, pp. 197–243. §12.3.3: `http://www.mathmeth.com/wf/files/wf2xx/wf214t.ps` (accessed May 2009)

Forisek, M. (2006) On the suitability of programming tasks for automated evaluation. *Informatics in Education*, **5**(1), 63–76.

Horváth, G. and Verhoeff, T. (2002). Finding the median under IOI conditions. *Informatics in Education*, **1**(1), 73–92.

International Olympiad in Informatics Website. `http://www.IOInformatics.org/` (accessed May 2009)

Kemkes, G., Cormack, G., Munro, I. and Vasiga, T. (2007). New task types at the Canadian computing competition. *Olympiads in Informatics*, **1**, 79–89.

Kenderov, P.S. and Maneva, M.N. (Eds.) (1989). In *Proceedings of the International Olympiad in Informatics*,

Pravetz, Bulgaria, May 16–19. Union of the Mathematicians in Bulgaria, Sofia.

Kiryukhin, V. and Okulov, S. (2007). *Methods of Problem Solving in Informatics: International Olympiads*. LBZ (BINOM. Knowledge Lab), Moscow (in Russian). `http://www.lbz.ru/` (accessed May 2009)

Skiena, S.S. (2008). *The Algorithm Design Manual* (2nd ed.). Springer-Verlag. `http://www.algorist.com/` (accessed May 2009)

Verhoeff, T. (1995). The lost group chart and related problems. In *Simplex Sigillum Veri, A Liber Amicorum for Prof. Dr. F.E.J. Kruseman Aretz*. Faculty of Mathematics and Computing Science, Eindhoven University of Technology. December 1995, pp. 308-313. `http://www.win.tue.nl/ wstomv/publications/kruseman.pdf` (accessed May 2009).

Verhoeff, T. (2004). *Concepts, Terminology, and Notations for IOI Competition Tasks*. `http://www.win.tue.nl/wstomv/publications/terminology.pdf` (accessed May 2009)

Verhoeff, T. (2006). The IOI is (not) a science olympiad. *Informatics in Education*, **5**(1), 147–159.

Verhoeff, T., Horváth, G., Diks, K. and Cormack, G. (2006). A proposal for an IOI Syllabus. *Teaching Mathematics and Computer Science*, **IV**(1), 193–216. `http://www.win.tue.nl/ wstomv/publications/ioi-syllabus-proposal.pdf` (accessed May 2009)

Verhoeff, T. (2008). Programming task packages: peach exchange format. *Olympiads in Informatics*, **2**, 192–207.

T. Verhoeff is assistant professor in computer science at Eindhoven University of Technology, where he works in the group Software Engineering & Technology. His research interests are support tools for verified software development and model driven engineering. He received the IOI Distinguished Service Award at IOI 2007 in Zagreb, Croatia, in particular for his role in setting up and maintaining a web archive of IOI-related material and facilities for communication in the IOI community, and in establishing, developing, chairing, and contributing to the IOI Scientific Committee from 1999 until 2007.

Olympiads in Informatics, 2009, Vol. 3, 167–173

jBOI – One More Possibility for Increasing the Number of Competitors in Informatics

Biserka YOVCHEVA
Konstantin Preslavsky University of Shumen
115 Universitetska str., 9700 Shumen, Bulgaria
e-mail: bissy_y@yahoo.com

Galina MOMCHEVA
Varna Free University 'Chernorizets Hrabar'
Chaika resort, 9007 Varna, Bulgaria
e-mail: gmomcheva@gmail.com

Petar PETROV
A&B Private School
13 Hristo Botev str., 9700 Shumen, Bulgaria
e-mail: peshoto_bg@yahoo.com

Abstract. This article aims to demonstrates a trend in raising the general number of competitors in informatics by creating and supporting competitions like the jBOI. It argues that participation in international competitions from an early age contributes to the participants' achieving sustainable high results in the future.

It discusses a number of organizational measures as well as sharing experiences in training students for similar competitions in Bulgaria. It also develops a general framework for competitions in informatics for pupils aged under 15.5 years of age.

Key words: competitions in informatics, programming, talents, IOI.

1. Introduction

The role of competitions in the process of learning computer sciences is quite obvious for the people involved in it. That's why companies like Microsoft and IBM annually initiate a lot of challenging competitions. Learning in competitive environments guarantees success in career.

A number of good ideas for promoting programming contests are suggested by Pohl and Polly who propose using programming contest problems with graded difficulty (Pohl and Polly, 2006).

Creating the jBOI is not a new idea because of the existence of such competitions like the Junior Balkan Olympiad in Mathematics (for students under 15.5 years old) or the International Mathematical Kangaroo. But as far as concerns to informatics it is quite

new. In our opinion after 3–4 years of preparation, students who begin their preparation at age of 11 or 12 years need a stimulus and a measuring ground to compare their progress. Moreover, a lot of students, who are new to this movement, can be attracted to participate; if they are selected and coached adequately they will succeed too.

2. Preparation and Organization of jBOI

So far the Balkan Olympiads in Informatics for Juniors (students under the age of 15.5) has taken place twice. Both times the competition was carried out due to the initiative of specialists from both sides. As of today, there is no organizational body managing the event, neither is there a set of fixed rules, nor a curriculum plan which the participants could use as a guide for their preparation.

Nevertheless both times the competition was carried out with an enormous degree of enthusiasm on the sides of the participants, as well as the organizers. There is also a notable increase in the interest towards the competition: 6 countries took part in the first jBOI 2007 in Serbia (`http://www.math.bas.bg/infos/`), 8 in the second jBOI 2008 in Bulgaria (`http://www.jboi2007.org`). The organizing of the third competition jBOI 2009 is left to the good will of the Greek organizers.

3. Preparation of the Participants in Bulgaria

3.1. *Preparation on General Basis*

The general preparation of young students in Bulgaria is performed on several levels:

- Local study groups

Local study groups are organized at schools and other centres, where regular classes are taught throughout the whole year and with conformity to an approved curriculum. This type of preparation is coordinated by the National commission for extracurricular work with the Union of Mathematicians in Bulgaria. Such study groups are active in a comparatively small number of towns in the country (see Table 1), as programming is not included as an obligatory subject for students of that age. To an extent, this offers a relative freedom to each centre in the selection of topics for its curriculum and in the selection of methods and approaches.

All the interviewed competitors highly rated the local private school as source of their preparation, and in dependence of the organization in the cities that private school is scholastic or non scholastic.

At the same time, all interviewed participants rely on study books, but only one of them rated them as the most important factor. What calls for attention is that students of the same study groups rate their sources of preparation in the same way, which is indicative of the importance of the organization of study groups and the methods of the teachers.

Table 1
Students (up to 15.5 years old) by grades that took part in competitions

Grade	2001 /2002	2002/ 2003	2003/ 2004	2004/ 2005	2005/ 2006	2006/ 2007	2007/ 2008	2008/ 2009
IV	26	35	31	2	12	16	12	6
V				31	19	42	33	42
VI				13	32	21	39	24
VII	43	46	42	17	21	20	14	23
VIII				15	34	33	24	18
Total	**69**	**81**	**73**	**78**	**118**	**132**	**122**	**113**
Towns	10	13	11	10	12	9	15	14

- National competitions in informatics

The role of national competitions is undeniable as coordinating agents and as powerful stimulus for better preparation. The practice of analyzing the tasks and their solutions after each competition, which was introduced in 2001, is an important medium for presenting the solution ideas of the authors of the tasks and for discussing different ideas for the level of preparedness of the participants. Even more so is the brochure with the tasks and the solutions, given by their authors, whose publication started in 2005.

- National training camps in competitive informatics

This form of training was introduced on 2006 and aims at intensive (one-week) preparation of the competitors with the best results throughout the year. A detailed description of these camps is provided in Manev *et al.* (2007). All interviewed students consider the preparation in these camps as very useful. Again, depending on their local study groups, they rate the camps as second or third in importance. Only one interviewees rated it lower than third place, and one as the most important.

- Online competitions

Participation in a number of online competitions and in various preparation websites is encouraged. Table 2 shows some of the most popular sources of this type.

A survey made among the students of 15.5 years of age shows that those with the highest ranking in online competitions are also the ones with the highest results in the national rank lists (`http://www.akla.org`) for their respective age. Furthermore, several national online competitions have been carried out. However, they are not yet a regular form for preparation. To the greatest majority of participant's registration in training websites is not done for the sake of registration. It is usually guided by the teacher working on their preparation locally or with the recommendation of the leaders of the national team. They help competitors in choosing the most suitable website and monitor their progress in their self-preparation. Such participations allow the respective mentor

Table 2
Online training competitions

	Training system				N of part
	N of probl.	hours	url	lectures	
TIMUS	10	5	http://acm.timus.ru/	no	7
TOPCODER	3	01:15	http://www.topcoder.com/tc	yes	5
USACO	3	3	http://train.usaco.org/usacogate	yes	7
COCI	6	3	http://evaluator.hsin.hr	no	1
UVA	10	5	http://acm.uva.es/contest	no	1
Z-TRENING	dif	dif	http://www.z-trening.com	no	1
ACM	10	5	http://acm.hit.edu.cn/index.php	no	1

of the student or of the entire team to monitor, control and encourage the individual preparation of each competitor. There are several major criteria for selecting the most suitable website:

1. The form of the online competitions. Ideally these should imitate the form of international student competitions. Regretfully, almost none do. Most online competitions imitate the student competitions organized by ACM. On the bright side, this form requires complete and accurate solutions to the given problems, which trains students to think about even the smallest details.
2. Maintaining an archive of competition problems, so competitors can work on a problem selected by them or by their teachers, and check their solution by comparing it to the original author solution with an 'online submit' option. An online testing system is a great advantage for any training website.
3. Providing lectures on basic topics and solution analyses of basic problems. It is especially useful if competitors working without a trainer can read an analysis of the author solution of a task they have worked on. Analyzing the solutions of other competitors is also useful (TopCoder).
4. Providing up-to-date rank lists of the registered participants. Comparing personal results with those of other participants' is an extremely important stimulus.

Three of the interviewees found online competitions the most important thing in their preparation. Only two students placed them at the bottom of their ranking lists. The majority of competitors cite the internet as one of the main sources for their preparation. Again their opinion is connected to the traditions in their local study groups, which is a further indication that organization at local level is a crucial factor for the future development of competitors. The analysis of the results of Bulgarian participants in internet competitions shows that those with high ranking in online competitions achieve high results in national competitions as well.

3.2. *Preparing the National Team after its Formation*

Selection rules for the national team for students of up to 15.5 years of age are the same as those for the national team for up to 20.5 year-olds. They are set by the National Commission with the Ministry of Education and Science and regulate selecting an extended national team, which – after a series of controls – is shortened to the final national team. The details of the procedure can be found in Manev *et al.* (2007). The preparation for the jBOI starts after the selection of the national team of up to 15.5 year-olds. Practice shows that this preparation is much more effective with the extended national team. Training the extended team achieves several important results:

- the general level of all noted competitors is maintained;
- younger students are greatly motivated if they work in a larger group of peers with similar abilities and preparation;
- strategically such training reduces the risk of losing shape on the evening of the competition;
- if necessity arises, a competitor from the final team can be replaced with another from the extended team, without the risk of lesser training.

Training the extended national team is carried out in several different forms:

- Control competitions

There are two types of control competitions – mock and official ones (carried out for the national team selection). Mock competitions are organized by the trainers of the team, while official ones are organized by the National Commission for the National Olympiads in Informatics with the Ministry of Education and Science. The leaders of the team aim to make both types of competitions as effective as possible. For this purpose after each competition they organize a discussion among the participants and the authors of the competition tasks (if possible). The goal of the discussion is to pinpoint the mistakes of each participant and work towards eliminating them. This is why the preparation of competition tasks, especially those for mock competitions, is extremely important. They are ranked as crucial and one of the most effective forms of training, by the greater part of the interviewees.

- Lectures at national training camps

At this stage lectures are not common and are only used if there are notable gaps in the knowledge of some of the participants. Such gaps may be due to incomplete training, differences in the curriculum, or differences in the organization and methodology of teaching in the different cities. Generally, a well-structured short revision of the school material is considered useful even for students without obvious gaps in knowledge. The gaps in question are discovered mainly by analyzing the results of control competitions. 50% of the interviewed competitors rank lectures (a combination of theory and practice) as the most efficient training method. On the other hand, 40% rank them as the least efficient method. The division here seems once again to be determined by the traditions of the local study group.

- Online training

This is realized through the various communication methods provided by the internet – chats, forums, e-mail. The goal here is to maintain the shape of competitors even when they are not in a period of intense training. Internet communication is also used for control competitions as well as for topic discussions. It is organized and led by the coaches of the team.

4. A Possible Framework for International Competitors for Juniors

We take the framework as a basic conceptual structure used to solve the complex problem of creating competitors in programming.

As a generalization of the situation and the obstacles in Bulgaria we could suggest the following framework for such an international competition in informatics for juniors.

Firstly, some documents or regulations have to be signed between participant countries in order to guarantee the existence of such new competitions. Otherwise some problems may occur during their organization.

There must be clear rules defined about the age of contestants participating in the contest.

Human resources play significant role in such a structure and could be divided into committees as follows: organizing committee, technical committee, scientific committee. It should not be considered a drawback if a person from a country takes part in several committees, if such participation is appropriate. It is helpful if coaches or even contestants have the possibility to communicate with committee members and make suggestions or express their opinions.

Some technical resources like installed compilers, environments, grading system (with rules) and computers are necessary too and the information connected to those issues could be updated and made available regularly.

Probably the two paragraphs above are familiar to contestants, team leaders and organizing committees in informatics competitions.

The next element from this framework is very important – the publicity. This includes not only publishing competition problems and their solutions (as usual), but developing a curriculum, preparing or selecting books, and recommending articles appropriate for the students or teachers involved. We know of an enormous number of books appropriate for the competitors but the different age of contestants leads to some problems concerning the mathematical concepts they are familiar with or the level of language (e.g., English) that they have.

As for the mentioned problems in preparation at a national level, we can summarize the necessity of on-line training system, training camps for students, seminars and workshops for teachers, [e]books and coaching system. The observations in our country show that coaching has different faces because it varies from teacher to team leader, through guardian and psychologist. Some basic issues that coaches have to face are: What to do?, How to do it?, Why do it? The forms of coaching also vary from Face-to-face to online

meeting. And the people engaged in coaching vary: from teachers, university lecturers to parents and classmates, present or ex-competitors. This is the person (coach) who helps contestants to develop their personal plan for individual action or development.And last but not least in this framework is the financial support: sponsorships (companies, foundations), fees, some forms of support by national or local authorities, government or ministry of education of the host country and international (e.g., European Union) funding programs (like Comenius – LLP for students from schools).

5. Conclusions

In conclusion the jBOI is one more possibility for increasing the number of competitors in informatics. jIOI could be one better possibility.

References

Manev, K., Kelevedjiev, E. and Kapralov, S. (2007). Programming contests for school students in Bulgaria. *Olympiads in Informatics*, **1**, 112–123.

Pohl, W. and Polly, T. (2006). Experience with graduated difficuty in programming contest problems. In *ISEEP 2006*, Vilnius.

`http://www.akla.org/akas1/lib/studentstandarts.html`

`http://www.math.bas.bg/infos/`. Portal for National Competitions in Informatics for Students.

`http://www.jboi2007.org`

`http://www.jboi2008.com/`

B. Yovcheva is a senior lecturer in informatics in The Konstantin Preslavski Universiti of Shumen and a director of A&B School of Shumen. She is team leader of the national team of informatics for 15.5 years of Bulgaria, 2009. She published over 20 scientific papers and many methodical works, wrote 8 textbooks in informatics and IT for secondary and high education. She is a member of Bulgarian National Committee for Olympiads in Informatics.

G. Momcheva is assistant professor in CS at Varna Free University, Bulgaria. She is a deputy team leader of the national team of informatics for 15.5 years of Bulgaria, 2009. She has 12 years experience as a teacher in informatics and ICT. She published over 15 papers and 6 textbooks for high school students. She is a member of Bulgarian National Committee for Olympiads in Informatics.

P. Petrov is young teacher in informatics in A&B School of Shumen. He is team leader of the regional team of informatics for 15.5 years of Shumen. He is an author/co-author of a book on algorithms and algorithmic problems.

Olympiads in Informatics

Volume 3 2009

M. FORIŠEK. Using item response theory to rate (not only) programmers 3

A. IDLBI. Taking kids into programming (contests) with Scratch 17

E. KELEVEDJIEV, Z. DZHENKOVA. Tasks and training the intermediate age students for informatics competitions 26

R. KOLSTAD. Infrastructure for contest task development 38

M. MAREŠ. Moe – design of a modular grading system 60

B. MERRY. Using a Linux security module for contest security 67

I. NINKA. The role of reactive and game tasks in competitions 74

M. OPMANIS. Team competition in mathematics and informatics "Ugāle" – finding new task types 80

P.S. PANKOV, K.A. BARYSHNIKOV. Representational means for tasks in informatics 101

T. PORANEN, V. DAGIENĖ, Å. ELDHUSET, H. HYYRÖ, M. KUBICA, A. LAAKSONEN, M. OPMANIS, W. POHL, J. SKŪPIENĖ, P. SÖDERHJELM, A. TRUU. Baltic olympiads in informatics: challenges for training together 112

P. RIBEIRO, P. GUERREIRO. Improving the automatic evaluation of problem solutions in programming contests 132

W. van der VEGT. Using subtasks 144

T. VERHOEFF. 20 years of IOI competition tasks 149

B. YOVCHEVA, G. MOMCHEVA, P. PETROV. jBOI – one more possibility for increasing the number of competitors in informatics 167

1822-7732(2009)3;1-C