On Mathematical Contents of Computer Science Contests*

S.O. Shilova[a], N.V. Shilov[b†]
[a)]Institute of Mathematics, Russian Academy of Science,
Novosibirsk, Russia shilov61@inbox.ru
[b)]Korean Advanced Institute of Science and Technology
Daejeon, Korea shilov@ropas.kaist.ac.kr

May 15, 2005

1 Introduction

Science Olympiads and contests brings spirit of competitiveness to Science education. They benefit best students, engage them with research. Simultaneously Science Olympiads and contests challenge faculty to enhance teaching so that regular student can enjoy Olympiad and contest problems.

Computer Science contests become very popular with undergraduate students in recent years. Maybe the Association for Computing Machinery International Collegiate Programming Contest (ACM ICPC) is the most popular world-wide. The initiative was born at early 1970-ies in US, then evolved to North America Computer Science competition, and was formally inaugurated in 1977 at the first World Final ACM ICPC. Overall number of participants of annual multi-level contests (at local, sub-regional, regional and final levels) is about several tens of thouthands from more than 1,300

*This work is supported by Brain Korea 21 Project, The school of information technology, KAIST in 0000.

†While on leave from A.P. Ershov Institute of Informatics Systems, Russian Academy of Science, Novosibirsk, Russia

universities in 68 countries. Seventy-eight teams took part in the 29th Annual World Finals of the ACM ICPC, April 3-7, 2005, in Shanghai, China. In home country Russia the reputation of the Contest is so high, that President Vladimir Putin awarded early this year the organizers of North-East European Regional ACM ICPC by National Educational Award [8].

Basics of the adopted format of ACM ICPC follows. Every contest team consists of three undergraduates (to say nothing of a computer;-). A team has to solve 8-10 'real-world' problems in five hours competition. Teammembers jointly rank the difficulty of the problems, design a formal models of them and an algorithm that solves the formalized problems, implement the algorithms, test the resulting programs, and submit the programs to jury. Jury adopts a program being correct if the program successfully exercises a number of preliminary designed test data suites. All problem statements are provided with samples of test data and range of admissible data values. But teams have no access to jury test suites. Each incorrect submission is fined by a penalty. At the end of the contest, teams are ranked by number of correct submissions, and (if several teams have the same numbers) by value of penalties for incorrect submissions. (Please refer to [7] to learn more about the ICPC).

Unfortunately, ACM ICPC and many other Computer Science contests become much more about programming than about Science. They become more similar to a technical sport than to Science Olympiad due to limited role of research and innovative component in these contests. It seems that

- art of problem formal modeling,
- cooking book of algorithms in heads,
- rapid typing skills in hands

are three corner-stones of a success in these contests. Of course, all listed skills are related to Computer Science proficiency. The art of modeling is especially important since it is about research skills, not about technical skills. This research component puts Computer Science Contests in line with science Olympiad like Mathematics and Physics Olympiads.

But research component in Computer Science is not limited by art of modeling. In particular, it includes formal mathematical proofs of model properties and program correctness. Moreover, sometimes without these proofs, a utility of a program that 'solves' a problem is very conventional in spite of successful and extensive program testing.

At this talk we present a number of particular problems that fit Mathematics Olympiad and Computer Science Contests format simultaneously. In the case of mathematics Olympiad, the problems are about existential proofs. In the case of Computer Science contests these problems are about algorithm design and implementation, but they can not be considered correct without formal proof. The proofs in these cases can be carried out in pure traditional mathematical style, or in Computer Science way, i.e. in a manner that is mathematically strict but Computer Science in nature. We do believe that these problems can help to overcome some alienation when gifted Computer Science students consider Mathematics being too pure, and talented Mathematics students consider Computer Science being too poor. We would like to hope that Mathematics faculty and students can extend this list by a number of Computer Science contest problems that require formal proofs for validation.

2 Sample Problems

Computer Science is a branch of science like mathematics, physics, or biology. This branch of science has its own objects of studies and methods of research. Of course, this particular branch can adopt ideas, approaches, and methods from other branches. The progress of quantum computing in the last decade is the most popular example of how another branch of science can benefit Computer Science. But Computer Science can fertilize (in return) other branches of science too. Maybe, the advance of bioinformatics is the most recent evidence of contribution of this kind.

At the same time, the common example of contribution of Mathematics to Computer Science is Theoretical Computer Science. It is possible to say that Theoretical Computer Science is the study of mathematical models of algorithms, data structures, programs, etc. with the use of mathematical logic, abstract algebra, topology, etc. This leads to advance of program and temporal logics, cryptography, domain theory, etc. In contrast, there are few examples when ideas, approaches and methods of Theoretical Computer Science contribute to advance in Mathematics. Nevertheless there is a number of Olympiad-level mathematical problems where Theoretical Computer Science can give new insights, if to interpret these problems in a Computer Science manner. In the new incarnation, mathematical problems become Computer Science contest problems, where working program is the primary

target but formal proof of algorithm correctness is absolutely inevitable (in contrast to modern Computer Science contests where djujement relies upon testing).

We have discussed already two sample problems of this kinds in [3, 2]. Please refer cited papers for full story, but let us sketch the formulations of these problems in the following format:

Math: Mathematical Olympiad problem first,

Comp: Computer Science contest problem next.

The first Mathematical Olympiad problem that can be solved with aid of Theoretical Computer Science and that can be transformed into Computer Science contest problem is the following Fake Coin Puzzle [3]:

Math: All valid coins have equal standard weight, a fake coin has another weight. Is it possible to identify a unique fake in a set of 14 coins, where one fixed coin is known to be standard, balancing them 3 times at most?

Comp: Write a program that inputs a number of coins M and balance limit N and outputs

- 'impossible', if it is impossible to identify a unique fake in set of M coins, where one fixed coin is known to be standard, balancing them N times;
- an executable program that implements an interactive scenario for fake coin identification otherwise.

(Assume that all test data range [1..150].)

We would like to remark now (in addition to discussion in the cited paper) that without formal proof of correctness of program's algorithm, the program can not be adopted to be correct due to the following reasons:

- test data ranges in [1..150] for testing a program in a feasible time;
- identification strategy for test data in this range can be developed by manual tuning a trichotomy;
- the trichotomy is not the right strategy in general case.

The next problem is Harbor Dispatcher Problem.

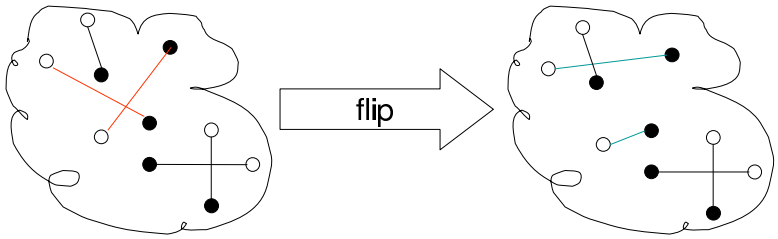
Math: There are N black and N white points on the plane. Every three of them are non-collinear. Prove that it is possible to connect black and white points in 1-1 manner by intervals without intersections.

Comp: There are N piers in a harbor and N vessels in its area. Positions of all piers are fixed. Initial positions of all vessels are known. Every vessel can move to every pier directly (i.e. by a straight line). Dispatcher has to assign an individual pier to every vessel. Every vessel must move to the assigned pier by the most direct route. To exclude collisions, intersections of these routes are prohibited. Write a program that inputs initial positions of vessels and piers and assigns individual pier for every vessel. (Assume that in test data N ranges in [1..100] and that all coordinates of positions range in [0..1000].)

Of course, a formal model for Computer Science Harbor Dispatcher Problem is a set of N black and N white points on the plane without collinear triples and we have to find a pair-wise intersection-free interval coupling (a 'good' coupling). Hence a straightforward mathematical approach to Harbor Dispatcher Problem consists in the following steps:

- proving existence of any good coupling;
- extracting algorithm from existential proof;
- implementing the algorithm.

In principle there are several opportunities how to prove existence of a good coupling in pure mathematical way. They lead to different algorithm design for good coupling construction. but there is also a pure Computer Science heuristic search approach to Harbor Dispatcher Problem (Fig. 1). The idea of the heuristic search for is trivial: start with some initial coupling and try to resolve local conflicts (i.e. eliminate local intersections) by flipping. Please refer [2] for proof of total correctness of this algorithm. Observe that without proof the heuristic search algorithm has a very small value. (Unfortunately, Computer Science contests do not take proofs in to account.) Let us remark also, that total correctness of the heuristic search algorithm is also a proof of existence of a good interval coupling.




```
VAR X : coupling;
X:=initial;
WHILE not good(X) DO X:=flip(X)
```

Figure 1: Heuristic search algorithm and illustration how 'flip' works

3 Sylvester problem

A problem that we would like to address in this section is well-known problem of J.J. Sylvester (1814-1897) [5, 4]. The following story is citation from [4]:

> Here is the question that Sylvester originally raised in 1893 in a journal known as the Educational Times:
>
> > Prove that it is not possible to arrange any finite number of real points so that a right line through every two of them shall pass through a third, unless they all lie in the same right line.
>
> Sylvester used the term "right line" for straight line. Furthermore, two points which are the only two points on a line of a point/line configuration C have come to be known as ordinary lines. Today, we would state the result, in a way similar to the way Paul Erdös (1913-1996) did when he made the conjecture, about 40 years later:
>
> > If a finite set of points in the plane are not all on one line then there is a line through exactly two of the points.

(Let us emphasize that the problem was published in educational journal [6]. More than one hundred years later we discuss the problem at the 1st KAIST International Symposium on Enhancing University Mathematics Teaching.)

Below we reformulate P. Erdös variant of J. Sylvester problem as a Mathematical Olympiad problem and as a corresponding Computer Science contest problem:

Math: There is a finite set of points on the plane that are not collinear simultaneously. Prove that there exists a pair of points in the set that is not collinear with any other point in the set.

Comp: A Flatland is a perfectly flat country where every two cities are connected by a road that is an interval of a straight line. Apparently some roads can intersect under some angle, some can not meet at all, some can be a part of other roads, etc. For example, at a map of Flatland in Fig. 2 there are 5 cities (1, 2, 3, 4, and 5) and 10 roads ([1,2], [1,3], [2,3], [1,4], [1,5], [4,5], [2,4], [2,5], [3,4], and [3,5]). In this example roads [2,5] and [3,4] intersect, roads [2,3] and [4,5] do not meet, roads [1,4] and [4,5] continue each other, road [1,2] is a part of road [1,3]. A magistral is a road that is not a part of any other road. A magistral connects all cities along it. For example, road [1.3] in Fig. 2 is a magistral that connects cities 1, 2, and 3; road [3,5] is also a magistral that connects cities 3 and 5. It is known that there are at least 2 magistrals in Flatland. Write a program that inputs positions of cities in Flatland and outputs a magistral that connects perfectly two cities in Flatland. (Assume that in test data number of cities ranges in [1..100] and that all coordinates of city positions range in [0..1000].)

Of course, a formal model for Computer Science Flatland Problem is a set of points on the plane that are not collinear simultaneously. Hence a straightforward mathematical approach to the Computer Science problem consists in the following steps:

- solving Sylvester problem;
- extracting algorithm from existential proof;
- implementing the algorithm.

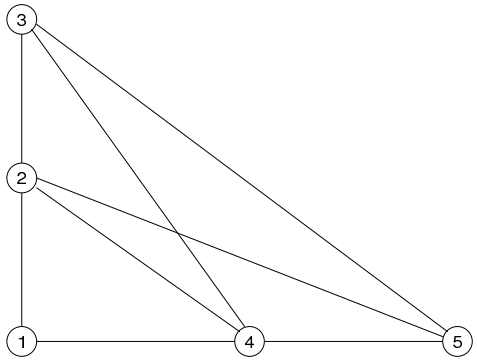


Figure 2: Sample map of Flatland

Please refer [5, 4] for mathematical solution of Sylvester problem. This mathematical solution leads immediately to the algorithm that searches for a pair of disjoint points $a, b \in F$ such that

$$\min_{c \in F \setminus L(a,b)} d(c, L(a,b)) \;=\; \min_{\substack{x \neq y,\, x,y \in F \\ z \in F \setminus L(x,y)}} d(z, L(x,y))$$

where

- F stays for the set of points (i.e. cities in Flatland),
- $L(x, y)$ stays for the straight line between disjoint points x and y (i.e. 'extended' roads between cities x and y),
- $d(z, L(x, y))$ stays for the distance between the line $L(x, y)$ and a point z outside this line.

We would not like to discuss further the above mathematical solution for Flatland problem. In contrast in the next section we develop a Computer Science solution for Flatland problem and exploit it for solving Sylvester problem.

4 Solving Flatland Problem

We are asked to write a program that finds a magistral in Flatland that connects perfectly two cities. In other words, the program must output a

pair of cities ‘a’ and ‘b’ such that interval [a,b] is a magistral that does not contain any other city but ‘a’ and ‘b’. Thus it makes sense to introduce two data types CIT and MAG which values are all cities and all magistrals in Flatland respectively.

What are operations that can use arguments of these data types or return values of these data types? Two operations have been discussed already:

- ‘free way’ operation fw: MAG → BOOL returns TRUE if the magistral contains two cities at most, and FALSE otherwise;
- ‘magistral’ operation mg: CIT×CIT → MAG returns a magistral that contains both argument (cities).

It is also natural to assume that we know at least two cities in Flatland

- ‘initial cities’ constants ia, ib: $\_\!\!\rightarrow$ CIT are simply two fixed cities

so that we can start from these cities and a magistral that contains them.

What else can we use in solution of Flatland problem? Do we use everything that is provided by problem statement? - Of course, not: we have not utilized yet that there are at least two different magistrals in Flatland! How we can use this information? - For example, in the following manner:

- ‘side city’ operation sc: MAG → CIT returns a city that does not belong to the argument (magistral).

It also makes sense to introduce some other operations:

- ‘left’ and ‘right’ operations lf, rt: MAG → CIT; they return the utmost (opposite) cities of the argument (magistral).

For example, for the map depicted in Fig. 2 we (can) have:

- fw([2,3])=FALSE but fw([2,5])=TRUE;
- mg([1,2])=mg([1,3])=mg([2,3])= [1,3];
- ia can be city 1 and ib can be city 2;
- sc(mg([1,2]))=sc([1,3]) and it can be city 4 or city 5;
- lf(mg([1,2]))=lf([1,3]) and it can be city 1 or city 3;

```
VAR m: MAG;
VAR a,b: CIT;
m:=mg(ia,ib);
WHILE not fw(m)
DO
   a:=sc(m);
   b:=(lf(m) OR rt(m));
   m:=mg(a,b)
OD
```

Figure 3: A preliminary design for Flatland problem

- rt(mg([1,2]))=rt([1,3]) and it can be city 1, if lf([1,3])=3;
- rt(mg([1,2]))=rt([1,3]) and it can be city 3, if lf([1,3])=1.

A preliminary version of a heuristic algorithm for Flatland problem is represented in Fig. 3. In this design (_OR_) stays for a non-deterministic choice of an 'appropriate' end of a current magistral. A heuristic behind this design is very simple: let [x, y] be a magistral with utmost end cities x and y; for sure we have only cities x, y, and another city z outside this magistral; hence we can search for a new candidate for a 'free way' magistral among mg(x,z) and mg(y,z); hopefully, one of these two options leads to success.

5 Solution for Flatland Problem

Observe that if some computation of algorithm in Fig. 3 terminates, then the final value of variable 'm' is a magistral that contains perfectly two cities (due to loop condition 'not fw(m)'). Hence we should take care about algorithm termination. With this idea in mind, let us sketch in brief how Computer Science proves algorithm/program termination.

There exist a number of techniques for proving program termination. One of them was developed by laureate of the ACM Turing Award (1978) Robert W. Floyd. His method is based on mappings to well-founded sets. It can be briefly described as follows.

A well-founded set (WFS) is a partially ordered set $(D, \leq)$ without infinite decreasing sequences $d_1 > d_2 > ....$ Assume that F is a total mapping

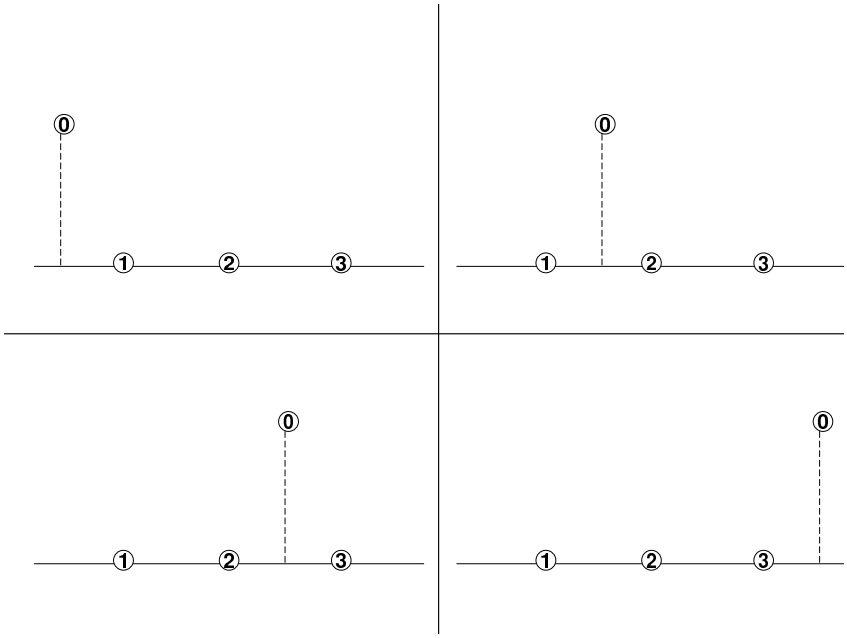


Figure 4: Relative positions between cities and magistral

from configurations of a program/algorithm into a well-founded set. If each iteration of every loop decreases the value of the function F, then it guarantees program termination. (Please refer to a comprehensive textbook [1] for details.)

The above description of method of R. Floyd gives us a clue how to determinize a non-deterministic assignment b:=(lf(m) OR rt(m)) in the preliminary design: the right choice must decrees some value...

Observe that WHILE-loop is executed iff the current magistral is not a free way, i.e. the magistral contains 3 cities at least. Let these cities be 1, 2, and 3. Hence there are 4 opportunities for relative positions of a side city 0 and 3 cities that belong to the magistral (Fig. 4). Observe again, that if to select city b in $\{1, 3\}$ so that city 2 lies in between the base of the perpendicular from side city 0 to the line $L(1, 3)$, then $d(2, L(0, b)) < d(0, L(1, 3))$ (Fig. 5). Hence it makes sense to use another operation instead of 'lf' and 'rt':

- 'best' operation bs: CIT×MAG → CIT returns the utmost city of the second argument (magistral) such that there exists another city in between this utmost city and the base of the perpendicular from the first argument (side city) to the line of the magistral.

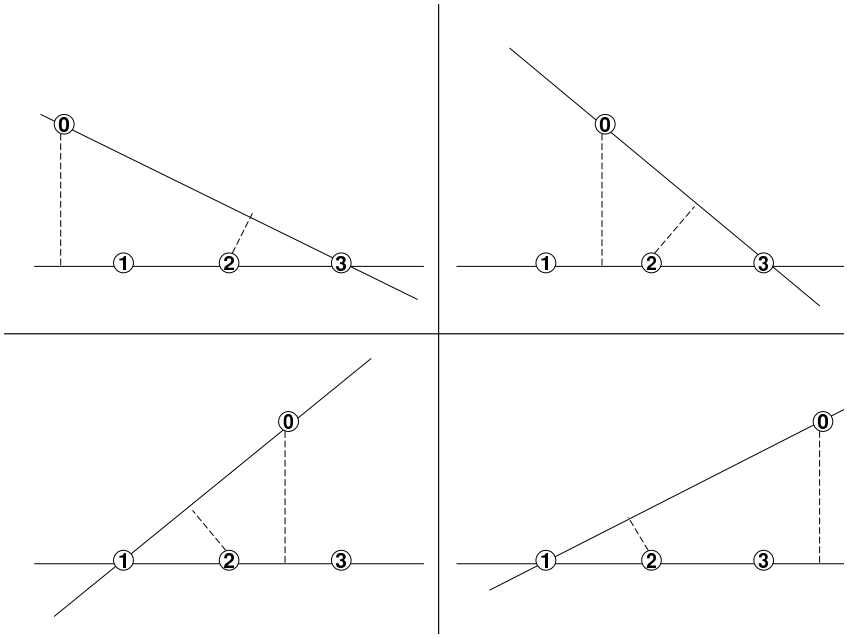


Figure 5: How to select next magistral and next side city

For example, in the Fig. 5, bs(0,[1,3])=3 in the first row and bs(0,[1,3])=1 in the second row. It also makes sense to define a new compatible operation instead of 'sc':

- 'next' city operation nx: CIT×MAG → CIT returns a city of the second argument (magistral) that is closest to the first argument (utmost city).

For example, in the Fig. 5, nx(1,L(1,3))=nx(3,L(1,3))=2 always.

The final design of a heuristic algorithm for Flatland problem is represented in Fig. 6. Of course, the final design is not a 'formal' refinement of the preliminary one (Fig. 3). But we would like to hope that readers are convinced by our informal development that lead us from Fig. 3 to Fig. 6. The last thing we have to do in this section is to prove that the algorithm in Fig. 6 solves Flatland Problem.

Proposition 1 *If there are 2 magistrals at least in Flatland then the algorithm in Fig. 6 always terminates with a magistral that connects perfectly 2 cities as a final value of the variable 'm'.*

Proof: Assume that Flatland has 2 magistrals at least.

```
VAR m: MAG;
VAR a,b,c: CIT;
m:=mg(ia,ib);
a:=sc(m);
WHILE not fw(m)
DO
   b:=bs(a,m);
   c:=nx(b,m);
   m:=mg(a,b);
   a:=c
OD
```

Figure 6: Final design for Flatland problem

Let D be

$$\{d(z, L(x,y)) \ : \ [x,y] \text{ is a magistral, and } z \text{ is a city outside } [x,y]\}$$

where (as in the above)

- $L(x,y)$ is the straight line between points x and y,
- $d(z, L(x,y))$ is the distance between $L(x,y)$ and a point z.

Observe that D is a finite (since set of cities is finite) subset of $\mathbf{R}^+ = \{r \in R : r \geq 0\}$. Let $\leq$ be the standard linear order on real numbers $\mathbf{R}$. Then $(D, \leq)$ is well-founded set.

Let F: CIT$\times$MAG $\to D$ be a mapping

$$\lambda[x,y] \in \text{MAG}.\ \lambda z \in \text{CIT}.\ d(z, L(x,y)).$$

Then every iteration of WHILE-loop in Fig. 6 decreases value of F as follows from definition of operations 'bs' and 'nx' and Fig. 5.

Hence in accordance with method of R. Floyd, the algorithm in Fig. 6 always terminates. A final value of the variable 'm' must be a magistral that connects perfectly 2 cities, since the condition of WHILE-loop is 'not fw(m)', i.e. loop must iterate while the magistral connects 3 or more cities. ■

6 Conclusion

We have designed an algorithm for Flatland problem on base of some heuristics about useful data types and R. Floyd method for algorithm/program termination. Then we proved algorithm correctness and termination formally in terms of method of R. Floyd. Thus we can claim that our solution of Flatland problem is absolutely Computer Science in nature and mathematically rigorous at the same time.

Let us remark that without proof the algorithm can not be adopted as a correct, basically because it is just a local heuristic search. Observe also, that formal proof of algorithm correctness implies solution for mathematical problem of J. Sylvester (in formulation of P. Erdös). Thus our study is another evidence of opportunities to apply Theory of Computer Science to mathematical problem solving (at Olympiad level in particular).

Unfortunately, popular Computer Science contests do not take proofs in to account and hence joy, art, and science of mathematical algorithm development and validation remains out of their scope. But we would like to hope that Mathematics and Computer Science faculty can work together toward better attitude to Mathematics Contents of Computer Science Contests. We do believe that it enhance Mathematics and Computer Science university teaching.

References

[1] Apt K.R., Olderog E.R. Verification of Sequential and Concurrent Programs, Second Edition. Graduate Texts in Computer Science, Springer-Verlag, 1997.

[2] Shilova S.O., Shilov N.V. Etude on theme of Dijkstra. ACM SIGACT News, v.35, n.3, 2004, p.102-108

[3] Shilov N.V., Yi K. How to find a coin: propositional program logics made easy. In Current Trends in Theoretical Computer Science, World Scientific, v. 2, 2004, p. 181-214

[4] Malkevitch J. A Discrete Geometrical Gem. AMS Feature Column Monthly Essays on Mathematical Topic. July-August, 2003, at URL: `http://www.ams.org/featurecolumn/archive/index.html`.

[5] Pach J., Agarwal P. Combinatorial Geometry. Wiley-Interscience 1995

[6] Sylvester, J. J. Educational Times, 46, No. 383, 156, March 1, 1893.

[7] Assosiation for Computing Machinery – URL `http://acmicpc.org` ACM International Collegiate Programming Contest – URL `http://icpc.baylor.edu/`.

[8] Decree On 2003 Annual National Educational Awards of President of Russian Federation (In Russian). January 26, 2005, at URL `http://www.kremlin.ru/text/docs/2005/01/83056.shtml`.