Olympiads in Informatics

Volume 7, 2013

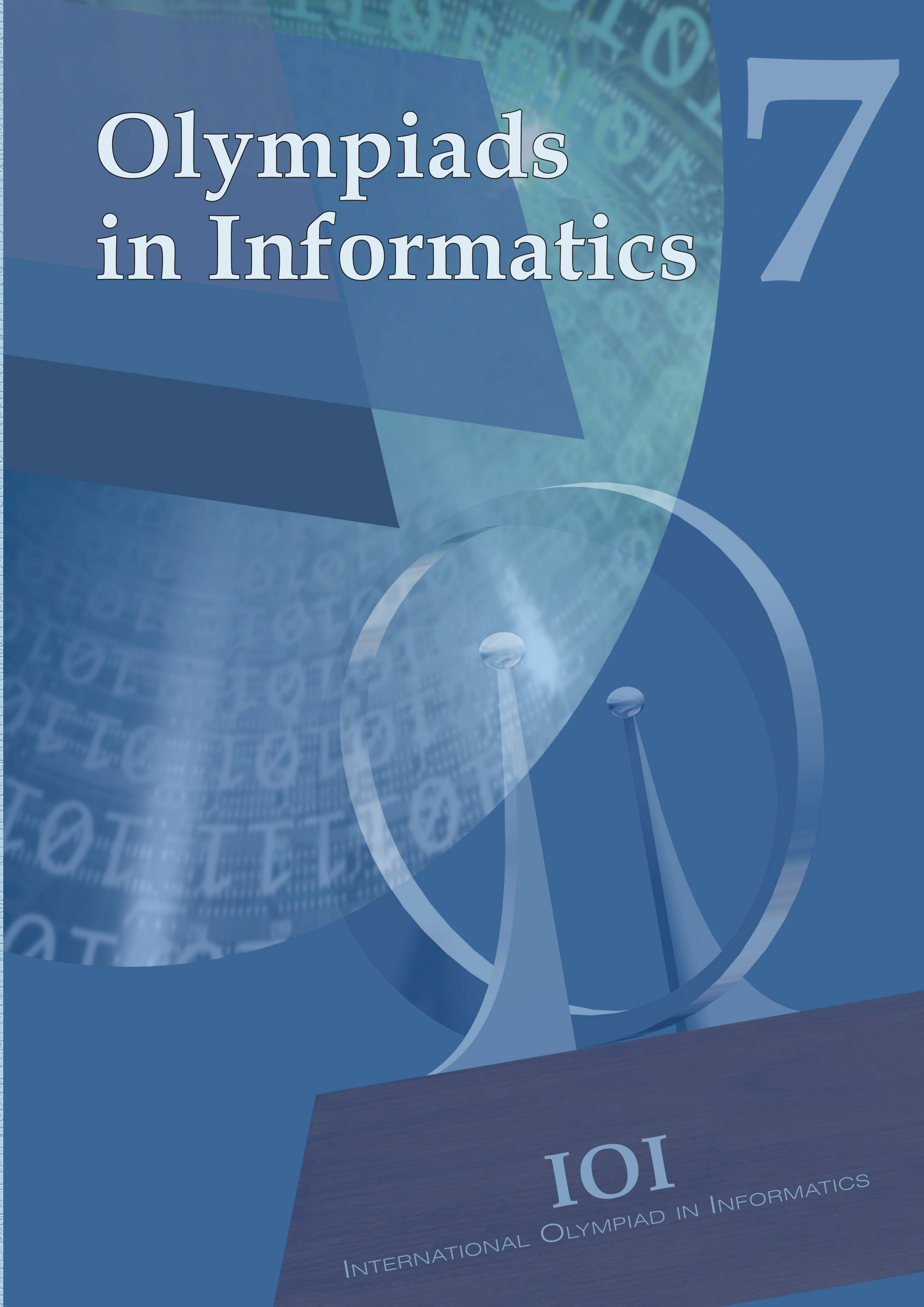
ISSN 1822-7732

9 771822 773007

Olympiads in Informatics Volume 7, 2013

Olympiads in Informatics

7



ISSN 1822-7732

INTERNATIONAL OLYMPIAD IN INFORMATICS
VILNIUS UNIVERSITY
INSTITUTE OF MATHEMATICS AND INFORMATICS

OLYMPIADS IN INFORMATICS

Volume 7 2013

Selected papers of
the International Conference joint with
the XXV International Olympiad in Informatics
Brisbane, Australia, July 6–13, 2013

OLYMPIADS IN INFORMATICS

ISSN 1822-7732

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

Publisher: Vilnius University Institute of Mathematics and Informatics
Akademijos 4, LT-08663 Vilnius, Lithuania

Olympiads in Informatics, 2013, Vol. 7, 1–2
© 2013 *Vilnius University*

Foreword

It passed unremarked in the introduction to the 6th volume of *Olympiads in Informatics* but that volume brought the number of full articles published by the conference to *101*. For an event held in partnership with the *International Olympiad in Informatics*, or the *IOI* as it is frequently called, a rather satisfactory co-incidence. So why was it not mentioned until highlighted in this volume's paper on the journal's first few years?

The number might, perhaps, have been left unmentioned because it is just a bit of fun. The type of numerology that might appear semi-seriously in the popular press, less-seriously mentioned in a volume of recreational mathematics; merely a co-incidence and one that, after all, requires the exclusion of some of the journal's contents (such as reviews). Making this deduction however ignores the fact that, for many of those who participate in both the conference and the olympiad, a lot of this is fun. These 101 papers all have serious content – and a huge amount of work goes in to running national and international olympiads – but there is pleasure in running these events, seeing students flourish and seeing colleagues. Even students have been known to have a good time.

It is certainly not because, as a community, we have no interest in numbers. Many of the papers that we have published derive some basic spark, some essential essence, from figures. We have looked at, and continue to look at, how we can go about measuring tasks; how should we grade and assign marks, in what ways is it meaningful to time evaluation, etc . . . Several papers have categorised and by extension counted tasks in those categories, before looking at student performance. If we removed combinatorics from the problems we set we would run very different types of contests.

Our community's interest in numbers goes beyond the technical aspects. We care about the number of students we bring in to our national events and we want to see that different groups are represented. The varied country reports that have been published since 2007 show the varied ways in which our national contests are run. We can use this knowledge to pick from success stories around the world and if, perhaps, a comparison with the numbers from elsewhere is occasionally humbling, it can drive us forward.

The real reason, of course, why the 101–IOI synchronicity was not considered in the 6th volume's editorial was that it was unknown to the editors. It took the observation from someone outside our usual community to mention the number. It has been an aim of the conference, since it was established, to become part of, and bring in, the wider pedagogical community. We have not had many papers authored from those outside the IOI but they are starting to appear. In this volume we report on two PhD thesis published in related areas. The fact that such research is taking place, both within these pages and outside, is delightful. Long may it continue.

As always thanks are due to all those who have assisted with the current volume – authors, reviewers and editors. A lot of work goes, not only to the writing of the papers, but to an extended period of review and correction and, in several cases, translation. Peer reviewing all of the papers takes a significant amount of time and work and special thanks should be given to those otherwise unsung reviewing heroes: Jonas Blonskis, Andrej Brodnik, Hugo Duenas, Michael Forišek, Gerald Futschek, Mathias Hiron, Rob Kolstad, Ville Leppänen, Krassimir Manev, Mārtiņš Opmanis, Rhein Prank, Miguel Revilla, Jūratė Skūpienė, Ahto Truu, Tom Verhoeff.

Last, but by no means least, particular thanks are due to the organisational committee for IOI'2013 in Australia without whose assistance we would be unable to hold the conference. Their assistance, during what is an already busy period, is gratefully received.

Editors

Olympiads in Informatics, 2013, Vol. 7, 3–13
© 2013 *Vilnius University*

Growing Algorithmic Thinking Through Interactive Problems to Encourage Learning Programming

Sébastien COMBÉFIS[1,3], Virginie VAN den SCHRIECK[2], Alexis NOOTENS[3]

[1]*Department of Computer Science Engineering, Université catholique de Louvain*
Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium
[2]*École Pratique des Hautes Études Commerciales (EPHEC)*
Avenue du Ciseau 15, 1348 Louvain-la-Neuve, Belgium
[3]*Computer Science and IT in Education ASBL, Belgium*
e-mail: sebastien.combefis@uclouvain.be, v.vandenschrieck@ephec.be, alexis.nootens@csited.be

Abstract. Attracting pupils from secondary schools (12–18 years old) to learn programming is not easy. It is especially the case in Belgium where there is no or very few programming and algorithm design courses in secondary schools. Another issue is that teachers who are in charge of computer science courses are afraid of teaching a matter they do not feel comfortable with, especially when they are not informatics teachers. This paper presents ILPADS, *interactive learning of programming and algorithm design skills*, an interactive website which aims at gradually growing algorithmic thinking skills to lead pupils towards the learning of the Python programming language. That website aims to serve as working material to support teachers for their computer science courses in secondary schools. Pupils can also use the website at home to continue learning on their own. The paper presents the interactive website and mainly focuses on the design of the ILPADS activities. Future work includes testing the website in real classrooms and evaluating it.

Key words: algorithmic thinking, learning programming, teaching, distance learning, interactive learning.

1. Introduction

Trying to attract more pupils to participate at the International Olympiad in Informatics (IOI) is not an easy task. It is especially the case in countries like Belgium where there are no or only a few computer science related courses in secondary schools (12–18 years old). Although there are pupils that are good at programming and designing algorithms, some of them may just ignore it and they consequently will not participate to the selection for the IOI (Combéfis and Leroy, 2011).

Moreover, there are no or few computer science teachers in the secondary schools of those countries where informatics is not part of the curriculum. That latter fact does not ease the task of introducing computer science and especially programming to the pupils. In such countries, it is even more difficult to develop and propose activities to be organised by the teachers in their classrooms, since they do not feel comfortable with the material to teach.

One possible way to propose activities to pupils is through online platforms. Various platforms, such as Pythia (Combéfis and Le Clément de Saint-Marcq, 2012), Putka (Urbančič and Trampuš, 2012) or France-IOI (Hiron and Février, 2012) do exist but are mainly focused on directly teaching programming. Most of the time, pupils and even teachers do not have any idea about what is behind the word programming and the notion of algorithm design. However, proposing online self-contained activities helps teachers to support taught courses and provides the possibility for pupils to continue learning at home.

Before teaching programming to pupils, it is important to develop their ability to think algorithmically. Computational thinking (Wing, 2006), also referred to as algorithmic thinking (Futschek, 2006), is a key ability that can be learned independently from programming, and that is maybe easier to introduce in secondary school. It has been defined as: *"the thought process involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent"* (Cuny *et al.*, 2010).

It is possible to teach and encourage algorithmic thinking without using a computer by proposing pen-and-paper exercises, as it is for example done in the frame of the Australian Informatics Competition (Burton, 2010) or for the Bebras contest (Futschek and Dagiene, 2009). The philosophy behind those concepts is that the learners do not need any additional background than the one they have while being in their schools. Pursuing those activities drives them towards developing algorithm design skills, but not programming skills.

This paper proposes learning activities supported by an interactive website whose goal is to develop algorithmic thinking amongst pupils. The activities are first designed to improve the problem solving skills of the learners. They also gradually drive the learners towards programming skills, as an embodiment of the reasoning they developed beforehand.

Section 2 draws up related work where activities are developed either online or in a pen-and-paper fashion, to teach algorithmic thinking to secondary school pupils without any prior knowledge in computer science. Section 3 presents ILPADS and focuses in particular on the design of activities, with a concrete example. It also motivates the design according to educational sciences theories. Section 4 discusses how the approach will be evaluated. Finally, the last section concludes the paper with some perspectives.

2. Related Work

This section presents related work on activities and tools that have been developed in order to develop algorithmic thinking by pupils from secondary schools.

Algorithmic thinking is somewhat different from natural thinking. Both consist of finding a solution to a problem, but whereas in everyday life problem solving is for humans, algorithmic thinking consists in finding a solution meant to be encoded in a computer. Several abilities are part of algorithmic thinking, including analysing the problem, finding basic actions that are adequate and constructing the algorithm with the basic actions (Futschek, 2006).

Futschek and Moschitz have been working on activities where learners can play algorithms, either virtually or by themselves (Futschek and Moschitz, 2010) or with tangible objects (Futschek and Moschitz, 2011). Their work focuses on the fact that the concepts of algorithmic thinking must be reduced to natural thinking for beginners. Learners are playing algorithms themselves, acting like intelligent processors than can execute algorithms. A model for learning by inventing algorithms proposed in Futschek and Moschitz (2010, 2011) proposes activities targeted at primary schools where pupils can manipulate tangible objects to discover the notion of algorithm. In those two approaches, the learners will not have to write an algorithm using a programming language, but will directly play with it. The activities proposed in this paper follow the same philosophy of thinking about algorithms by playing with them. But in this paper, the learners are driven up to programming their algorithms.

Other activities that have as a goal teaching algorithmic thinking to pupils are the ones proposed by CSUnplugged (Bell *et al.*, 2009). The proposed activities cover various subjects in computer science from numbers representation with binary numbers to cryptography. They are meant to be organised by a trainer with a group of pupils. In opposition to the activities proposed in this paper, the role of the trainer is essential for CSUnplugged activities.

Finally, as already introduced, another way to teach algorithmic thinking is through contests (Burton, 2010; Futschek and Dagiene, 2009). In those approaches, the pupils are first confronted to the problems on their own. Teachers are not obliged to go through the questions of the contest with the pupils afterwards.

3. The ILPADS Website

This section presents ILPADS, a website which aims at supporting the learning of algorithm design skills and programming, through interactive problems.

3.1. *General Presentation*

ILPADS is a website that proposes a set of learning activities. Each activity has, as a main goal, to develop the ability for the learners to solve an algorithmic problem. Activities are centred around concrete problems and are decomposed into three stages of increasing difficulty. At each stage, the learners get a new understanding of the problem and its solution.

Figure 1 shows the three stages of an ILPADS activity. They are not all mandatory and the learners can stop at any stage. However, progressing through the stages will lead the learner towards a solution in a programming language, which is the intent in a recruiting prospective for the selection for the IOI.

In the first stage, the learners are confronted to an interactive animation allowing them to play with an instance of the problem. It allows them to discover the algorithm and build

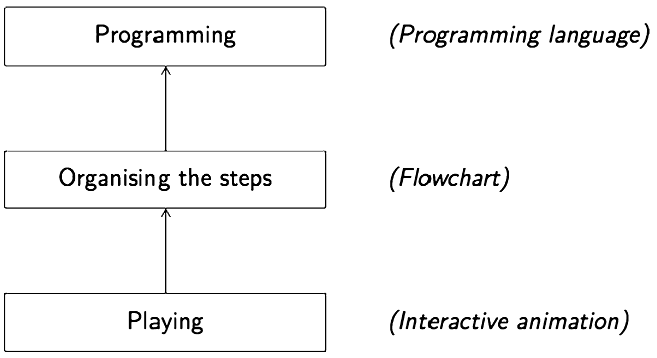


Fig. 1. The three stages composing an ILPADS activity. The stages drive the learner from the simple understanding of the algorithm to writing it in a programming language.

it in their mind. In the second stage, they have to concretise the algorithm they have in their mind. They do it with an executable flowchart that can be run on an instance of the problem. Finally, the last stage allows the learners to write a program representing their algorithm.

In order that decomposition to be possible, the chosen problem should not be too hard. It should indeed be ensured that the learners will get an algorithm idea during the first stage, that is, they have a preconceived, possibly wrong, idea in their mind. The first stage is therefore the crucial point in the design of ILPADS activities.

Each stage has a precise final objective in terms of what the learners will develop as skills. There is also a motivation for each step, related to the utility and usefulness of the trained skills.

1. Playing interactively on instances of the problem helps the learners understand the problem and guides them towards an algorithm. That stage is used as a mental gymnastics. After having played, the hypothesis is that it will be easier for the learners to solve new instances of the same problem.
2. The organisation of the algorithm found in the first stage into steps using flowcharts helps the learners take the algorithm out of their mind and concretise it. For the learners, it is the first step towards programming as they have to communicate their algorithm to the computer.
3. Finally, the learners will get to the writing of their algorithm using a programming language. That step is somewhat the holy grail as it will take the learners from a reasoning in their mind to a concretisation inside the computer.

3.2. *Activities Design*

This section presents how the different stages are implemented and what technologies are used to support them. The description is illustrated with an ILPADS activity example that is related to sorting algorithms.

3.2.1. *Playing Interactively*

In the first stage, the learners have to be able to play with the problem to be solved. Animations are important to provide an additional view of the problem. They help to understand algorithms better than a traditional textual or pictorial presentation (Rodger, 1996). ILPADS activities propose interactive animations that the learners can play with. The first stage is composed of a sequence of interactive animations that follow each other as in a comic strip (Biermann and Cole, 1999).

The learners are first completely free to play with the animation, and then some instructions are given to them in order to make an algorithm appear in their minds. As they are progressing in the story of the comics, their idea about an algorithm to solve the problem should be growing and becoming clearer in their mind.

Figure 2 shows the first interactive animation of the comics that is used for the sorting example. For that ILPADS activity, the learners are faced to a set of seven bottles, each with a different weight going from 1 to 7 ounces. The learners have to sort the bottles in increasing order of weight. To help them, they can use the provided machine which, once set with a reference weight, will output the lighter bottles on the left, the heavier ones on the right and the ones with the reference weight on the front.

The learners are completely free to play with that first interactive animation of the comics, to solve the problem, that is, sorting the bottles. The learners have to put the bottles in the right order on the result tray (not shown on the figure) and can check whether their solution is correct or not.

Since all the bottles have distinct weights, a solution to sort them is quite easily found. For example, the learners can successively set the machine with every value between 1 and 7, each time finding the next bottle to place on the result tray, which allows them to find the correct ordering.

The second box of the comics goes one step further. Now, the bottles do not necessarily have distinct weights. The weights are still between 1 and 7 ounces, but it may be the case that more than one bottle has the same weight.

Figure 3 shows a possible situation where the learners have set the machine with a reference weight of 3. Two bottles have that weight, two are lighter and three heavier.

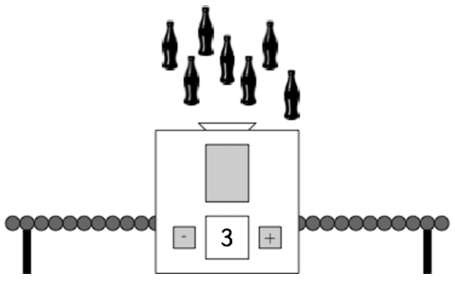


Fig. 2. The interactive animation of the sorting ILPADS activity. With that animation, the learners can ask the machine to group the bottles according to a reference value they chose. The goal for the learners is to sort all the bottles in increasing order of weight.

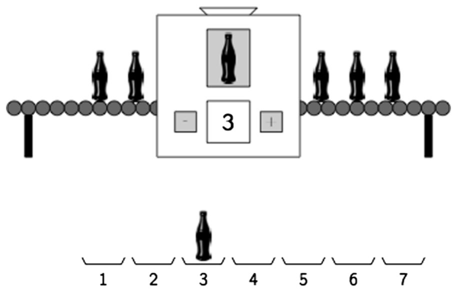


Fig. 3. The interaction animation of the comics for the sorting ILPADS activity. Contrary to the first interactive animation, there may now be bottles with the same weight. The learner cannot just rely on the position in the result tray to place the bottles.

One of the bottles weighing 3 ounce has already been placed on the result tray. The second bottle should be placed on position 4 since there are only two lighter bottles.

The idea with the second interaction animation of the comics is to drive the learners towards the selection sort algorithm. The learners should configure the machine with increasing values to fill the result tray from left to right. It is of course not the only possible solution, but the more convenient and easy to explain.

A comic strip always comes with additional textual information. Each box of the comics is enriched with textual information, coming as real-time feedback that can appear either when the learners check whether their solution is correct, or after any targeted action.

For example, for the sorting ILPADS activity in the interactive animation of the comics, the learners may get the following textual message when checking their answer:

> *"Congratulations, you succeeded in ordering the bottles in increasing order of weigh. The machine has done a total of 28 weight comparisons*."

That success message adds information about the comparisons that have been done by the provided machine. It serves as an indirect way to sensitise learners to performance issues. For the second interactive animation of the comics, the learners may be confronted to the following message when moving a bottle in the result tray:

> *"You are moving a bottle in the result tray, for the fourth time. Are you sure you cannot avoid it?"*

The goal of that message is to draw the attention of the learners on the fact that the way they are proceeding is not necessarily the best one. Another kind of textual information that may appear when the learners place a bottle on the result tray is:

> *"You placed a bottle weighing 3 ounce on position 2 but there are two lighter bottles. Are you sure it is a correct position?"*

Again, that message is used to lead the learners towards a correct solution. The added textual information is very important to support the learning and help the learners to find the algorithm to solve the problem. This is discussed in Section 3.3.

3.2.2. *Drawing Flowcharts*

Once the learners have played with the interactive animation and have gone through the different interactive animations of the comics, they are ready to get their algorithm out of their minds.

The idea of that second stage is to use the same animation as the one used in the previous stage. The difference is that learners will not be able to directly interact with the animation. The only way to control the animation is though a flowchart they have to design. It is essentially the same idea as the one of Scratch (Maloney *et al.*, 2004). It allows the user to build an algorithm visually by choosing and organising together blocks representing instructions or control structures.

Figure 4 shows a flowchart for a correct algorithm for the first interaction animation of the sorting ILPADS activity. There are two kinds of elements: diamond-shaped boxes are used to make a decision and rectangular boxes represent actions. Those two kinds of elements make it possible to represent all the basic operation of computer programs, that is, conditionals, loops and sequences of actions.

The actions can be parametrized with values that are directly related to the interactive animation, so that the learners can see directly the link with the animation. Moreover, it is important that the different actions available to design the flowchart are related to the ones the learners used during the first stage. In order to present to the learners the different possible actions, whenever the learners are clicking on the different elements of the interactive animation, a list of the possible actions is presented to the learners.

At any time, the learners can execute the flowchart and see directly the result on the animation. For the learners to succeed that stage, they have to design a flowchart that can solve any instance of the problem.

The goal of an ILPADS activity is not to directly teach learners how to use flowcharts, how to compose the different elements or link them together. However, provided that

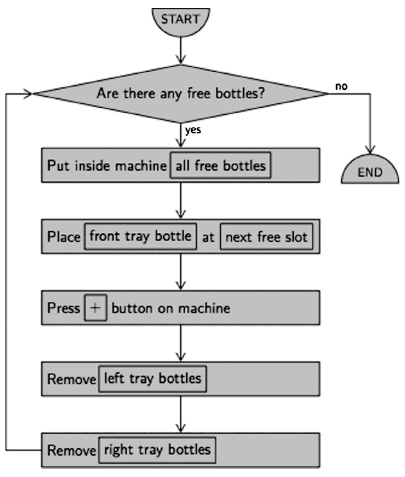


Fig. 4. Flowchart diagram that represents a correct algorithm for the first situation of the sorting ILPADS activity.

a simple flowchart example is given to the learners and since they are able to execute them and visually see the result of the execution, it could be able to teach learners to use flowcharts with a set of simple ILPADS activities based on very simple problems.

3.2.3. *Writing Down the Program*

The last stage consists of writing down the algorithm using a programming language. That stage should be made easier by the previous stage where the learners were forced to think carefully and structure their algorithm. During that stage, the learners will write their algorithm with the Python programming language.

For the learners, that final stage essentially consists in translating the flowchart designed in the previous stage into a Python program. To do so, Python functions corresponding to the actions from the flowchart are made available to the learners. The main goal is thus essentially to translate a flowchart to a computer program. Reaching that stage fulfils the main goal of the ILPADS website, which is to drive the learners from thinking in their minds up to writing their solution with a programming language.

That latter stage is supported by Pythia (Combéfis and Le Clément de Saint-Marcq, 2012) which is an online platform that can safely execute computer programs, test them and provide intelligent feedback about their correctness.

3.3. *Behind the Scene*

The interactive animations are developed using the HTML5 canvas element that is used to render 2D shapes (Smith, 2012) and Javascript that is used to animate the shapes and to interact with the user, in order to get an interactive animation.

The animations supporting the first stage are developed completely from scratch for each ILPADS activity. The Javascript code used to manage the interactive animation is decomposed in functions to have a direct mapping with the basic blocks provided to build the flowcharts. For the sorting ILPADS activity, such a function is the one executed when the user presses on the plus button (+) to increment the reference weight.

The flowcharts are also built with the canvas element of HTML5. The flowchart itself is represented as a linked structure whose nodes are related either to the functions defined for the animations of the first stage, or to functions that are especially defined for the flowcharts. An example of that latter kind of function is the one corresponding to the remove action of the sorting ILPADS activity.

Finally, as already mentioned in the previous section, the last stage is fully supported by Pythia. The programs written by the learners are checked thanks to the capabilities of the Pythia. In fact, the flowcharts designed during the second stage are checked the same way. Flowcharts can indeed be automatically translated into a Python program that is then checked by the Pythia.

3.4. *Supporting Learning and Ensuring Motivation*

The objective of the ILPADS activities is to help pupils learn new skills. Two main elements are at the heart of the design choice that was made for this work. The first one comes from active pedagogical learning methodologies supported by the *learning by doing* motto (Dewey, 1938). By placing the learners at the heart of the learning process, and by allowing them to play with an interactive animation, it increases their motivation and involvement.

The second important element that plays a role in the learning process is the feedbacks. Pupils get feedback through the interactive animations in the first stage and through the execution of the flowcharts in the second stage. For the third stage, feedbacks are integrated within those provided by the Pythia platform. Feedbacks are also important to lead the learners towards a correct algorithm which solves the problem.

Finally, the motivation of the learners is improved by ensuring a high level of self-efficacy (Bandura, 1977). Self-efficacy can be raised through four factors as identified by Bandura. First of all, success raises self-efficacy, and the ILPADS activities are driving learners towards success, in particular through the feedbacks. The second factor that enhances self-efficacy is the possibility for the learners to compare themselves with other learners in the same situation. That is possible, from the second stage where pupils have concretised their algorithms. The learners will get the possibility to visually compare their solutions, and for example to discuss together in class.

4. Evaluation Plan

The proposed activities have not been evaluated yet. Evaluation is important when designing new kind of activities. This section briefly presents the evaluation that is going to be done, which is clearly a future work.

The idea is to measure whether pupils that had previously trained with the ILPADS website are advantaged whenever confronted with a new instance of a given problem. The hypothesis is that pupils who played with the ILPADS website before do have a better structured and general algorithm in their minds than pupils who did not get that chance. The textual feedbacks that are brought for each interactive animation of the comics bring that advantage.

Experiments include confronting two groups of pupils with a set of problems. The pupils from the first group will use ILPADS and the pupils from the second one will just have a pen and a paper. Then, both groups will be confronted with new instances of the problem and their performance will be evaluated. Performance includes the rate of correct answers and the time used to solve the problems. The hypothesis is that ILPADS will help the pupils from the first group to structure the intuitive idea they have in their minds so as to be able to apply it to new instances of a problem they previously trained on.

Another important potential issue that may be raised is that by restricting what the learners can do with the flowcharts, it may restrict the learners' algorithmic thinking. The possible actions correspond to the one the learners can execute on the interactive animation in the first stage so that the only potential restriction may come from the conditions. Analysing the solutions proposed by the second group of pupils may help the designer of the activities to detect such missing conditions.

5. Conclusion and Perspectives

This paper proposes the ILPADS website that can be used to support teachers to propose computer science related courses in secondary schools. The goal of the website is to develop algorithmic thinking through problem solving that is supported by interaction animations. The learners are guided through three stages that develop their programming and algorithm design skills. Finally, at the end of an ILPADS activity, learners get to produce a solution to the problem as a computer program.

Providing online self-contained learning activities allows pupils to learn at home. This website can be used as an incentive to learn programming and, for example, to recruit more potential candidates for the IOI. It also helps teachers that are not comfortable with computer science and have to propose related activities in their schools.

Future work about ILPADS includes putting the components of the different stages altogether into one unique website. Also, developing an ILPADS activity takes a lot of time and new activities are already being designed. One future activity is about the "Guess Who?" game and another one is about the 15-puzzle (or Gem Puzzle) game. In addition to the design of new activities, work has to be done to ease the practical realisation of the two last stages that can be automated a lot, by developing an ILPADS activity creator. Last but not least, the approach has to be tested and evaluated as described above in the paper. More generally speaking, for a given activity, much attention has to be paid during the design so as not to restrict and limit the learners algorithmic abilities. Perspectives include a deeper study of how to assess whether a given ILPADS activity is well designed or not, that is, to evaluate its quality.

References

Bandura, A. (1977). Self-efficacy: toward a unifying theory of behavioral change. *Psychological Review*, 84(2), 191–215.

Bell, T., Alexander, J., Freeman, I., Grimley, M. (2009). Computer science unplugged: school students doing real computing without computers. *Journal of Applied Computing and Information*, 13(1), 20–29.

Biermann, H., Cole, R. (1999). *Comic Strips for Algorithm Visualization*. Technical report.

Burton, B. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.

Combéfis, S., Le Clément de Saint-Marcq, V. (2012). Teaching programming and algorithm design with pythia, a web-based learning platform. *Olympiads in Informatics*, 6, 31–43.

Combéfis, S., Leroy, D. (2011). Belgian olympiads in informatics: the story of launching a national contest. *Olympiads in Informatics*, 5, 131–139.

Cuny, J., Snyder, L., Wing, J. (2010). Demystifying computational thinking for non-computer scientists. Work in progress.

Dewey, J. (1938). *Experience and Education*. New York, The Macmillan Publishing Company.

Futschek, G. (2006). Algorithmic thinking: the key for understanding computer science. In: *Proceedings of the 2nd International Conference on Informatics in Secondary Schools: Evolution and Perspectives (ISSEP 2006)*, 159–168.

Futschek, G., Dagiene, V. (2009). A contest on informatics and computer fluency attracts school students to learn basic technology concepts. In: *Proceedings of the 9th World Conference on Computers in Education (WCCE 2009)*.

Futschek, G., Moschitz, J. (2010). Developing algorithmic thinking by inventing and playing algorithms. In: *Proceedings of the 2010 Constructionist Approaches to Creative Learning, Thinking and Education: Lessons for the 21st Century (Constructionism 2010)*.

Futschek, G., Moschitz, J. (2011). Learning algorithmic thinking with tangible objects eases transition to computer programming. In: *Proceedings of the 5th International Conference on Informatics in Schools: Situation, Evolution and Perspectives (ISSEP 2011)*, 155–164.

Hiron, M., Février, L. (2012). A self-paced learning platform to teach programming and algorithms. *Olympiads in Informatics*, 6, 69–85.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., Resnick, M. (2004). Scratch: a sneak preview. In: *Proceedings of the 2nd International Conference on Creating, Connecting, and Collaborating through Computing*. Kyoto, Japan, 104–109.

Milková, E. (2012). Development of algorithmic thinking and imagination: base of programming skills. In: *Proceedings of the 16th WSEAS International Conference on Computers*.

Rodger, S. (1996). Integrating animations in courses. In: *Proceedings of the 1st Conference on Integrating Technology into Computer Science Education ItiCSE 1996*, 72–74.

Smith, M. (2012). *HTML: The Markup Language (an HTML Language Reference)*, `http://www.w3.org/TR/html-markup/`

Urbančič, J., Trampuš, M. (2012). Putka – a web application in support of computer programming education. *Olympiads in Informatics*, 6, 205–211.

Wing, J. (2006). Computational thinking. *Communication of the ACM*, 49(3), 33–35.

S. Combéfis is a PhD Student at the Université catholique de Louvain in Belgium and works as a teaching assistant for the Computer Science Engineering Department. He is also following an advanced master in pedagogy in higher education. In 2010, he founded, with Damien Leroy, the Belgian Olympiads in Informatics (be-OI). He is now part of the coordinating committee that is in charge of managing everything which is related to the national contest. In 2012, he founded the CSITEd non-profit organisation which aims at promoting computer science in secondary schools and which is in charge of organising the Bebras contest in Belgium.

V. Van den Schrieck obtained her PhD in engineering in December 2010 from the Université catholique de Louvain. She is now professor in college and provides networking training. She has always been interested into computer science education and recently joined the volunteers who are working on projects for the CSITEd non-profit organisation with as a goal to promote computer science in secondary schools.

A. Nootens is studying computer science at Université catholique de Louvain. He is now a first year bachelor student. He has worked and is interested in web development. In particular he has a growing interest in HTML5 and related technologies such as the canvas. Being freshly graduated from the secondary school, he brings a fresh view about how computer science is perceived there and is interested in developing activities that may be used by secondary school teachers. He is also volunteering for the CSITEd non-profit organisation, and is in particular working on the ILPADS project.

Olympiads in Informatics, 2013, Vol. 7, 14–22
© 2013 *Vilnius University*

An Approach to Teaching Introductory-Level Computer Programming

Michael DOLINSKY
Department of Mathematics, Gomel State University "Fr. Skaryna"
Sovetskaya str., 104, Gomel, 246019, Republic of Belarus
e-mail: dolinsky@gsu.by

Abstract. This article describes a methodology, proposed by the author, for teaching beginners to computer programming from scratch. The methodology is dedicated to teaching groups of students with various levels of knowledge and motivation. The technical base of the methodology is the distance learning system "Distance Learning Belarus", briefly DL, created by the author.

Key words: programming, programming languages, improving classroom teaching; authoring tools and methods, interactive learning environments, intelligent tutoring systems.

1. Introduction

Teaching to program today is very hard and interesting task. This introduction is a brief sketch of the recent effort of researchers to make the teaching process more effective. The author thoroughly studied the publications of the journal "Computers & Education" because it is in accordance of the theme as well all the papers of the journal being open access. The main directions presented in the research are: interactivity of the classes; enhancing the learning performance using personalized diagnosis; designing an adaptive web-based learning system; analyzing effects of different types of feedback in a computer-based learning; different methods of examination; dynamic assessment; web-based learning; discussion forums; combining cooperative learning methods; computerized tool support for software engineering education; automatic evaluation; team work; computer simulations; developing computer skills of young children.

Interesting ideas are proposed in the articles devoted to introductory-level computer programming courses by Hawi (2010), Kordaki (2010), Mouraa and van Hattum-Janssenb (2011). These works also pointed the main problem of introductory-level classes: the various levels of motivation and skills of the students in one class.

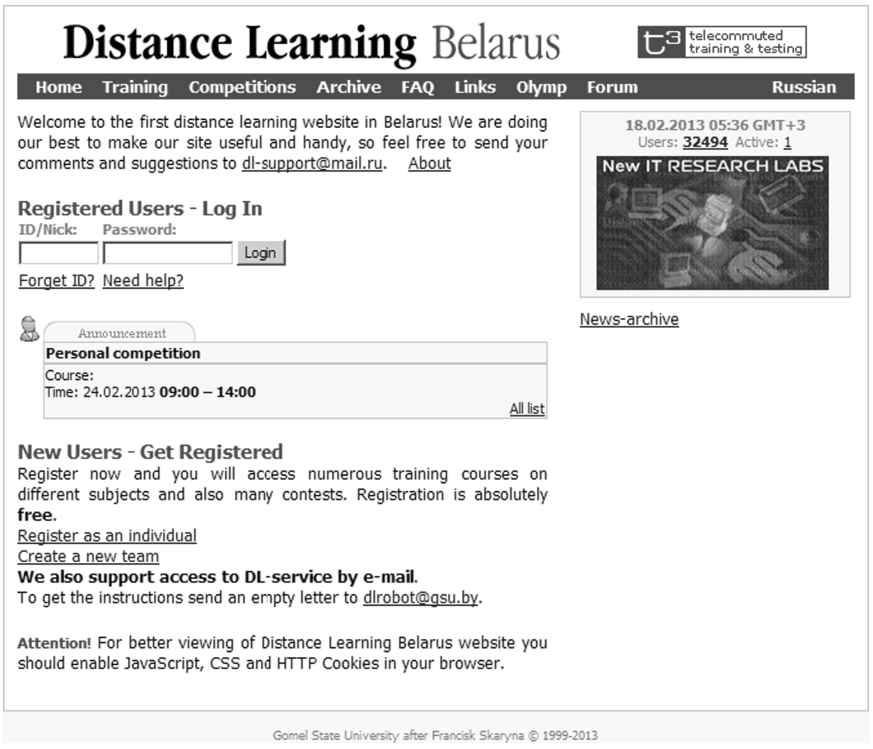
The author of this article has, for many years, taught introductory-level computer programming course at the Gomel State University "Fr. Skaryna"(Belarus). The paper summarizes his experience and methodology developed by him for teaching programming which could be interesting and useful for other teachers and researchers.

The technical base of the approach is the distance learning system DL available at the address dl.gsu.by; Section 2 presents this software tool. The site has Russian and English version, but essential part of the training materials is only in Russian. Section 3 gives a general overview of the training content. Sections 4 and 5 stress on the specificity of the theoretical and practical classes, respectively. Section 6 describes the marking scheme for students. Section 7 explains self-managed students work. Finally Section 8 contains conclusions.

2. Distant Learning with DL

Our DL is based on modern Internet technologies (Fig. 1). Hence it can be used for distance learning, decreasing the importance of geographical close positions between students and teachers or study materials, as well as excluding the necessity of participation of students and teachers in the study process at the same time. Moreover, the DL is used successfully to increase the quality of the education process in our own university.

The system distinguishes the following kinds of users: *viewer*, *student*, *tutor*, *teacher*, *author*, and *administrator*. The permissions of the users are increasing from viewer to administrator. Viewer is the only kind of DL user that could be unregistered. A viewer can only see the published results of any active or archived course. A student has access to the theoretical lessons and related tasks and can submit problem solutions to the system


Distance Learning Belarus
telecommuted training & testing
Home Training Competitions Archive FAQ Links Olymp Forum Russian

Welcome to the first distance learning website in Belarus! We are doing our best to make our site useful and handy, so feel free to send your comments and suggestions to dl-support@mail.ru. About

Registered Users - Log In
ID/Nick: Password: Login
Forget ID? Need help?

Announcement
Personal competition
Course:
Time: 24.02.2013 **09:00 – 14:00**
All list

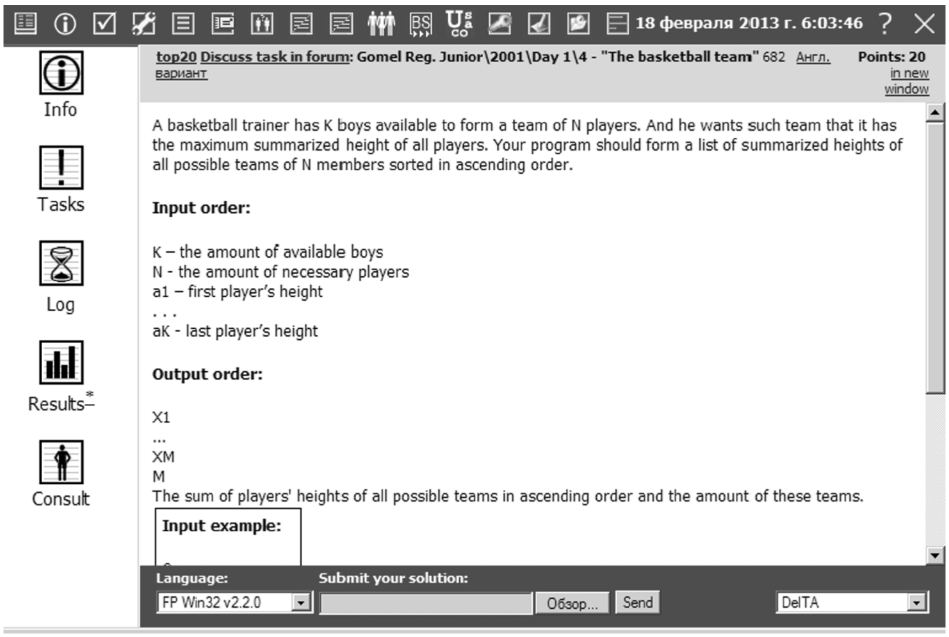
New Users - Get Registered
Register now and you will access numerous training courses on different subjects and also many contests. Registration is absolutely **free**.
Register as an individual
Create a new team
We also support access to DL-service by e-mail.
To get the instructions send an empty letter to dlrobot@gsu.by.

Attention! For better viewing of Distance Learning Belarus website you should enable JavaScript, CSS and HTTP Cookies in your browser.

18.02.2013 05:36 GMT+3
Users: 32494 Active: 1
New IT RESEARCH LABS
News-archive

Gomel State University after Francisk Skaryna © 1999-2013

Fig. 1. Start page of DL.



top20 Discuss task in forum: Gomel Reg. Junior\2001\Day 1\4 - "The basketball team" 682 Англ. вариант Points: 20 in new window

A basketball trainer has K boys available to form a team of N players. And he wants such team that it has the maximum summarized height of all players. Your program should form a list of summarized heights of all possible teams of N members sorted in ascending order.

Input order:

K – the amount of available boys
N - the amount of necessary players
a1 – first player's height
. . .
aK - last player's height

Output order:

X1
...
XM
M
The sum of players' heights of all possible teams in ascending order and the amount of these teams.

Input example:

Language: FP Win32 v2.2.0

DelTA

Fig. 2. Typical page for problem solving.

(Fig. 2). Each submitted solution is automatically tested by DL. The results of testing are collected in a *course results table*. A student can see her/his own *test protocol*. A tutor is nominated by a course teacher and can do part of the teacher's work when it is assigned to them by the teacher. A teacher has access to all test protocols and solutions of their students as well as to create a group of their own students. The author's role is to create new courses, as well as to upload/change/replace theoretical materials and task sets of their courses. An administrator controls permissions and processes at DL.

DL allows usage of multimedia presentations not only of the theoretical materials but also of the task tests. A good example for these possibilities is the course ”English DEMO”. The tests for students in this course contain graphics, pictures, sound and video.

DL liberates the teacher from a considerable amount of work in organization and control of study process. In addition, DL automatically displays the current results of the study process, stimulating the students to work hard in order to ameliorate their published announced results.

Currently, DL is used for teaching students of mathematical department of Gomel State University in the following courses: ”Computers and programming” (for first-year students), ”Computer foundations” (for third-year students), and ”Foundation of computers” (for fourth-year students).

In addition, DL is actively used for studying informatics by secondary school students (from 1st to 11th degree) as well as for preparing them for competitions in informatics and programming. For this purpose the following training and competitive courses are recommended: ”ACM tasks”, ”Preparing for programming contests – Profy”, ”Preparing for programming contests – Beginner", and "Introduction in informatics".

Annually, the Gomel town round and the regional round of the national olympiad in informatics are using DL. In addition, from 1997 to 2006 Gomel Computer Science Week (`http://www.gsu.by/gcsw/`) was using DL to provide the set of contests: in Pascal/C programming for IBM PC; in solving chess problems; in solving mathematical problems; in digital system design; in assembler programming for microcontrollers Intel 8051/8086, Motorola 68HC05/08, Atmel AVR, Texas Instruments TMS370, Microchip PIC.

Now we have more than 32000 registered users from 80 countries.

3. Content of Training

Here and in the following sections we will concentrate on the introductory-level computer programming course that is studied by the first-year students in their first semester. The main difficulty in such course is the enormous difference in the students' level of knowledge and skills. Studying informatics in a Belarus high schools is oriented mainly to usage of computers, e.g., preparing documents with text and graphics editors, writing e-mails and surfing in Internet. Only a few students achieve deep knowledge in computer programming due to special lessons or individual work. The main studied programming languages are Pascal and C++.

The learning technology in the author's classes is based on weekly cycles. Each week begins with a theoretical lesson, then some practical lessons follow and an examination closes the week. The topics of the classes, arranged by months are given below:

September: Introduction to programming. Debugging. One-dimensional array (standard and non-standard algorithms).

October: Two-dimensional array. Geometry. Strings – elementary algorithms. Strings – standard functions.

November: Strings – user defined functions and procedures. Sorting. Queue.

December: Recursion. Recurrence relations.

Most of the students are using Pascal because it is easier for novices, but the experienced students that know C++ can use it to solve the tasks. The main objective of the course is not to teach the programming language syntax, but the fundamental knowledge about basic algorithms and program development, testing, and debugging. Special attention is paid to good structuring of programs that helps to understand easier what the program is doing and how is doing it.

In the mentioned above topics the following concepts are considered:

Introduction to programming: data types (`char, string, longint, real`), arithmetic operators ($+$, $-$, $*$, /, `DIV, MOD`).

Debugging: execution – line by line, to the cursor, to the end; open of watch window, add variable to watch windows, windows control.

One-dimensional array: sum, count, maximal, minimal, operators `for` and `while`.

Two-dimensional array: rows, columns, diagonals. Algorithms for walking a two-dimension array.

Geometry: coordinates of a point, distance between two points, one-dimensional array of the distances from point to set of points, two-dimensional array of distances between all pairs of points of a set.

Strings: standard procedures and functions (`copy, delete, insert, pos`, etc.), user defined procedures and functions.

Sorting: bubble sort, exchanging sort, counting sort.

Queue: problem of knights on a chessboard, filling fields.

Recursion, Recurrences: general knowledge and simple examples.

4. Theoretical Lessons

The quantity, as well as the quality, of the theoretical material for each topic is carefully selected. All necessary fundamental knowledge is presented in such way that it can be acquired by the good students (about 1/3 of the group) within 15–30 minutes. The remaining time of the lesson is used for working in small teams (2–3 students) on comprehending the new knowledge and using it for solving specially chosen tasks with ascending difficulty.

Theoretical classes are held in an auditorium, equipped with beamer, screen, mobile notebook for the lecturer, wireless access to the university network, as well as power supply for students laptops. Each student of the class has a computer in front of him. These computers can be used by students as an alternative to the big screen, where the lecturing material is projected. After the lecturing part of the theoretical lesson is finished, students can browse the theoretical materials uploaded in the system and post questions on the topic.

To stimulate students' cognitive activity, from the beginning of the lecture (to be more exact, from the break before the lecture) a team contest starts. The contest consists of set of task on the topic with ascending difficulty. The students can discuss tasks in the teams. They have to write programs that solve the problems and submit them for automatic evaluation. During the contest the teacher's screen displays the dynamically changing table of the contest's results. At the same time a special personal web-based learning page is open for weak teams that help them understand the theory and to succeed in solving some of the easier tasks of the contest. The tasks are chosen in such a way that no team may solve all tasks (otherwise new tasks are added) and at the same time each team can solve at least one problem (thanks to the help provided during the contest for solving easiest tasks).

5. Practical Lessons

There are three types of practical lessons: *learning lessons*, *common exams*, and *individual exams*. For learning lessons students can choose one of the three possible levels of decreasing difficulty: *individual tasks*, *learning tasks* or *preparing* for learning *tasks*. During the lessons of the last two types the system automatically provides to students problems

Table 1
Numbers of main/all tasks by topics

Theme	Number of main tasks	Number of all tasks
Introduction to programming	28	4126
One dimension array	44	704
Two dimension array	19	430
Geometry	26	160
Strings	139	1552
Sorting	12	124
Queue	18	147
Recurrences	8	23

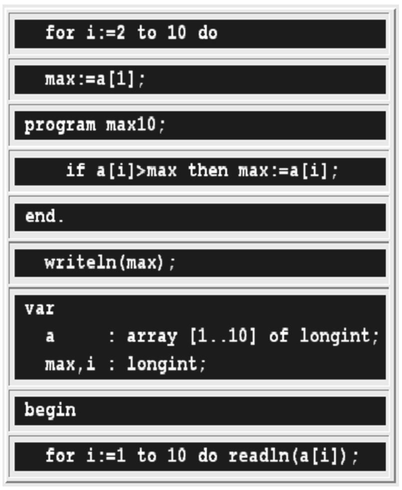
from a *problems tree* prepared for personalized instruction. "*Preparing for learning*" *tree* contains the easiest problems in order to provide to beginners possibility to start up. The tree is chosen instead of a sequence of tasks to provide learning more adaptive to a student's preparation level. Some students need more detailed explanations but for some a less detailed explanation. If a student makes a mistake solving exercises, the system automatically present them with the first one from the corresponding learning tree. Additionally there are the buttons "Don't know" and "I understood" given to students for their own navigation on the problems tree. All touched (by students) exercises get the color (green if it solved and red otherwise), so students as well as teacher (for any student) can analyze the learning path.

Because of the automatic delivery of problems to the students we succeed in achieving adaptive and individual teaching of students, independent of the teaching of the others. For each topic we have uploaded in the system small number of *main tasks* which are compulsory for solving by each student and large number of *additional tasks* (see Table 1) organized in tree structures. Trees of tasks are visualized when student cannot solve the main task or some of the additional tasks.

Additional tasks could be very different by form and content. The main task always demands student send the text of a program (in Pascal or C/C++) that solves the given task (Fig. 2). But the additional problems are dedicated to teach the student how to write the corresponding program if they cannot do it themself. The first type of such tasks demands the student submit an output for given task input. The goal of such an exercise is to provide the student with understanding what the program must do. Then the student has to do exercises for choosing right algorithm, constructing a program with given program lines (Fig. 3), typing the program with help from the system (Fig. 4), filling the gaps in a program (Fig. 5), etc.

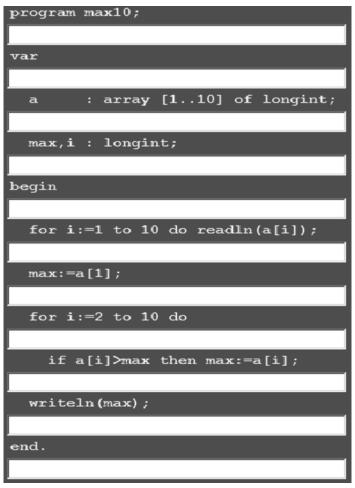
Note, that exercise in Fig. 4 gives to student essential help, highlighting by red color mistaken letters, and by green color right letters.

Every week one of the practical lessons is dedicated to a group exam. It includes 20–30 tasks (common for all students) with various difficulties. Because the problems are common, students have good opportunity to discuss their solutions after the class.



```
  for i:=2 to 10 do
  max:=a[1];
program max10;
    if a[i]>max then max:=a[i];
end.
  writeln(max);
var
  a     : array [1..10] of longint;
  max,i : longint;
begin
  for i:=1 to 10 do readln(a[i]);
```

Fig. 3. Constructing the program from lines.

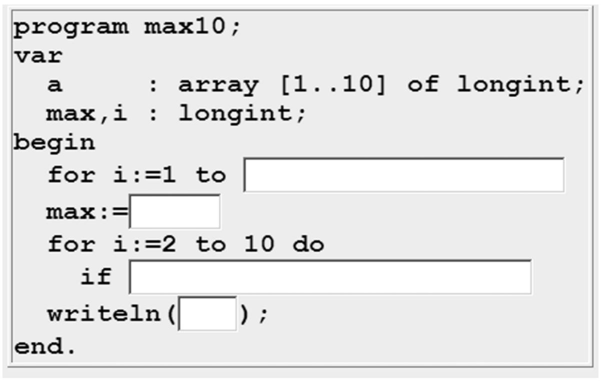


```
program max10;
var
  a     : array [1..10] of longint;
  max,i : longint;
begin
  for i:=1 to 10 do readln(a[i]);
  max:=a[1];
  for i:=2 to 10 do
    if a[i]>max then max:=a[i];
  writeln(max);
end.
```

Fig. 4. Typing the program with help.

Something more, for most difficult tasks special educational materials are prepared that become available after the exam.

To combat cheating during the exams by students (where they send solutions to colleagues or receive solutions from them) there are individual exams. Such exams includes 10 tasks on topics taught during the semester. Each student gets their own personal set of tasks, chosen from the problem bank randomly. Another important feature of individual exam – it cannot be done from student's notebook but only from a stationary computer of the university class and in special account that has read/write permissions only on a special empty folder in the local computer. Each student can repeat the individual exam until they gets the needed mark.



```
program max10;
var
  a     : array [1..10] of longint;
  max,i : longint;
begin
  for i:=1 to [          ]
  max:=[    ]
  for i:=2 to 10 do
    if [          ]
  writeln([   ]);
end.
```

Fig. 5. Filling the gaps.

6. Evaluation

Evaluation of the course is built to achieve the following objectives:

- to encourage students to attend each theoretical and practical class;
- to encourage students to work hard each minute of each class;
- to make each student's mark as objective as possible.

The final score is the minimum from the score of individual exam and the average score earned during the semester. The average score is accumulated from marks from theoretic examinations, practical examinations, solving learning problems, solving individual problems, bonuses and attending classes. Bonuses are assigned by teachers to encourage students' conscientious learning and cognitive activity. For example, bonuses earned by a team during theoretical classes are proportional to the number of solved tasks. Skipping classes reduces the average score.

7. Self-Managed Students Work

All theoretical and practical classes are performed on the base of the web-based e-learning system dl.gsu.by. Beside the other functionality it gives students good opportunities for independent work. In particular, students can work off the skipped classes by solving individual problems, problems from learning or preparing for learning sets. Essential help in the independent students work is provided by the *forum*, where everybody can post their own questions and get answer from other students or the teacher. In addition, the solutions of all individual problems are described in the special topic in the forum. Links to these solution descriptions are systemized by the teacher, so it becomes an additional bridge for passing from solving relatively easy learning problems to solving more difficult individual problems.

8. Conclusion

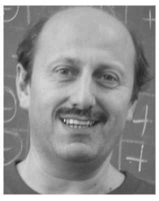
This paper describes the author's methodology of teaching introductory level programming course. The methodology is oriented to teaching groups of students with various levels of motivation and preparation. An excellent technical base of the course is the web-based teaching/learning system DL (dl.gsu.by), created with the supervision of the author. Applying the described teaching methodology provided essential shift up in the quality of teaching and especially in the teaching of the less prepared and motivated students. At the same time, all other students, including the most prepared and motivated, were also satisfied with such an approach for studying. Their opinion can be found in the forum of the site, where all students are encouraging to answer the questions "What you like (or dislike, or propose to change) in theoretical lessons, practical lessons and evaluation system?"

References

Hawi, N. (2010). Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. *Computers & Education*, 54(4), 1127–1136.

Kordaki, M. (2010). A drawing and multi-representational computer environment for beginners' learning of programming using C: design and pilot formative evaluation. *Computers & Education*, 54(1), 69–87.

Mouraa, I.C., van Hattum-Janssenb, N. (2011). Teaching a CS introductory course: an active approach. *Computers & Education*, 56(2), 475–483.



M. Dolinsky is a lecturer in Gomel State University "Fr. Skaryna" from 1993. Since 1999 he is leading developer of the educational site of the University `dl.gsu.by`. Since 1997 he is heading preparation of the scholars in Gomel to participate in programming contests and Olympiad in informatics. He was a deputy leader of the team of Belarus for IOI'2006, IOI'2007, IOI'2008 and IOI'2009. His PhD is devoted to the tools for digital system design. His current research is in teaching computer science and mathematics from early age.

Olympiads in Informatics, 2013, Vol. 7, 23–35
© 2013 *Vilnius University*

Pushing the Boundary of Programming Contests

Michal FORIŠEK
Comenius University, Bratislava, Slovakia
e-mail: forisek@dcs.fmph.uniba.sk

Abstract. In traditional programming contests the tasks almost always focus on design of efficient algorithms. In this paper, we discuss that this does not have to be the case in the future, and we give a significant number of concrete tasks that cover other areas of Computer Science. All of the tasks presented in this paper have actually been used in past programming contests.

Key words: task design, task types, sample tasks.

1. Overview

This section contains the overview of the paper's main topic: the set of tasks used in programming contests. We map the landscape of relevant programming contest, and discuss prior research related to the topic of this paper.

1.1. *Programming Contests Landscape*

In this paper we are dealing with programming competitions – competitions that involve writing computer programs. However, there are multiple kinds of programming competitions and we will only be interested in a particular subset of those: In terms of the terminology from Pohl (2011), in this paper we are interested only in *short-term* contests with clearly defined *tasks*.

(This leaves out competitions in which the contestants work on open-ended projects, such as the Imagine Cup or various types of robotics contests (Petronič, 2011). It also leaves out long-term competitions with clearly defined tasks, such as the Marathon matches track at TopCoder and contests like the CodeCup (Vegt, 2006) and ICPC Challenge where the contestants implement programs that play a game against each other.)

In other words, we shall consider short-term competitions that focus on creative thinking and problem solving. In these competitions, the contestants are given a particular set of problem statements, and the primary goal of the competition is to find solutions to the given problems. The computer programs produced by the contestants can then usually be seen as the "final proof" that a given problem has been solved.

Some of the largest worldwide competitions that fit into this group include:

– the ACM International Collegiate Programming Contest (ICPC),
– the International Olympiad in Informatics (IOI) along with the corresponding national olympiads for secondary school students,

– company-branded contests such as the Google CodeJam and the Facebook Hacker Cup,
– large portals that host regular contests, such as CodeForces, CodeChef and the Algorithm track at TopCoder.

Currently, all of these contests share a common characteristic: the competition problems focus on the design of efficient algorithms.

Most of these competitions acknowledge this focus openly. For instance, the Google CodeJam rules state that *The Google Code Jam Contest is a competition designed to engage programmers from around the world in algorithmic programming.* and the Facebook Hacker Cup official blurb contains the characteristic *In the Hacker Cup, programmers from around the world will be judged on accuracy and speed as they race to solve algorithmic problems to advance [. . .].*

The most popular track in TopCoder competitions is actually called *Algorithms* and competitions in this track are the traditional short algorithmic problem solving contests.

The ACM International Collegiate Programming Contest (ICPC) does not stress the algorithmic aspect and only focuses on programming in its official description: *The contest fosters creativity, teamwork, and innovation in building new software programs, and enables students to test their ability to perform under pressure. Quite simply, it is the oldest, largest, and most prestigious programming contest in the world.* Nevertheless, in the last decade the problems used at the ICPC World Finals are all of algorithmic nature.

The International Olympiad in Informatics (IOI) regulations only state the following objectives for the IOI:

– *To discover, encourage, bring together, challenge, and give recognition to young people who are exceptionally talented in the field of informatics.*
– *To foster friendly international relationships among computer scientists and informatics educators.*
– *To bring the discipline of informatics to the attention of young people.*
– *To promote the organisation of informatics competitions for students at schools for secondary education.*
– *To encourage countries to organise a future IOI in their country.*

One may note the complete absence of any mention of algorithms. On the other hand, each particular IOI competition has its own competition rules, and in the past years these rules always stated: *All of the tasks in IOI [year] are designed to be algorithmic in nature.* Almost without an exception, the IOI is using tasks that focus on algorithmic problem solving. See Verhoeff (2009) for more on IOI competition tasks.

As with all the rules, there have to be some exceptions. In this paper, we describe our effort to create some of those exceptions – to design programming competition tasks where the goal is other than just designing the most efficient algorithm. The tasks presented in the main sections of this paper come from two main sources:

1. The **Internet Problem Solving Contest (IPSC)**: an annual competition organized since 1999 by a group Comenius University faculty and students, including the author of this paper.

While primarily a programming / algorithmic problem solving contest, the IPSC strives to push the boundary of these competitions and include tasks that one would not expect in a regular programming contest.
IPSC is conducted online and open for everyone. In 2012, 1306 teams from 81 countries have registered for the contest.
2. The **Slovak Olympiad in Informatics (OI)** for secondary school students, and the related **Correspondence Seminar in Programming (KSP)** – a long-term Slovak national competition for secondary school students.

1.2. *"Algorithm Design" Does Not Equal "Problem Solving"*

The overwhelming preference of algorithmic problems is easily explained: Their nature makes them very suitable for short-term contests. It is easy to create "toy problems" with clearly defined statements that can be solved completely within the limited length of a contest.

One other benefit of using these algorithmic tasks in practical programming contests is the resulting scalability of these contests: the contestants' programs can be tested without the need of human interaction, which makes it possible to organize very large contests. (However, note that such evaluation is necessarily imprecise. There can always be false positives where an incorrect program is evaluated as correct. See Forišek (2006) for a discussion of some disadvantages of black-box testing in programming contests.)

There has been a substantial number of prior research publications dealing with various ways in which the spectrum of task topics for these programming contests can be broadened. Below we give a brief overview of these papers.

Computer Science without the Computer

As long as the contest is reasonably small, it is perfectly feasible to use theoretical tasks, have contestants write down their solutions on paper and have them checked manually. Some of the rounds of the Slovak OI (Forišek, 2007) work this way, and we believe that it is a good thing (Forišek, 2006). Pohl (2008) offers some more perspective on this topic and provides experience from the German OI (Bundeswettbewerb Informatik). Burton (2010) describes the pen-and-paper structure of the Australian Informatics Competition.

A lot of Computer Science can be introduced to the kids without actually using a computer – and starting at a surprisingly early age. The "Computer Science Unplugged" book of activities (Bell *et al.*, 1998) should be among the tools of any Computer Science educator. Some more activities in a similar spirit can be found in Forišek and Steinová (2010).

Kubica and Radoszewski (2010), van der Vegt (2012) and Ginat (2011, 2012) present a set of algorithmic tasks that are well suited for solving on paper, without programming.

New Task Types for Programming Contests

Kemkes *et al.* (2007) examine the use of open-ended tasks: clearly stated tasks without a known optimal solution, graded based on the quality of the solution produced by the contestant. Ribeiro and Guerreiro (2007) suggest tasks that use graphics, in particular a Graphical User Interface (and also comment on the possibility of adding visualization to some traditional algorithmic tasks from past IOIs). Truu and Ivanov (2008) propose a way in which testing-related tasks can be used, and discuss several examples. Opmanis (2009) gives a classification of task topics used in a Latvian competition in mathematics and informatics. Among others, these topics include data mining tasks, word problems, logic problems, and tasks on analysis of algorithms. Skupiene (2006) examines the possibility of including the programming style into the grading scheme. Kulczynski *et al.* (2011) give examples of tasks that are solvable using precomputation and/or visualization of the problem.

Various related contests can serve as inspiration for new problem types in traditional programming contests. Here we would like to mention the First Spanish Parallel Programming Contest described by Almeida *et al.* (2012), and the multitude of interesting task types used in the Bebras (Beaver) contest Opmanis (2006).

Recently, Ragonis (2012) gave a fairly thorough classification of types of questions that may occur in Computer Science competitions (and Computer Science education, in general).

1.3. *Paper Structure*

The rest of this paper contains our original content: a selection of non-traditional tasks used in Slovak programming contests. The content is divided into multiple sections according to task type.

Due to the limited space, all task statements were shortened as much as possible. All original IPSC task statements are available online at `http://ipsc.ksp.sk/archive`. Whenever the statement summary in this paper seems inadequate, we recommend reading the full task statement *before* reading the corresponding solution section of this paper.

2. Areas Not Related to Efficient Algorithms

The tasks in this section are IPSC tasks that venture into other areas of Computer Science. Solutions to all of these tasks were submitted online and checked automatically.

2.1. *IPSC 2012: Invert the You-Know-What [Topic: Cryptography]*

Statement
You are given a password file: a collection of usernames and password MD5 hashes. Obtain some of the passwords.

Solution
Some of the most common MD5 hashes can easily be reverted simply by googling them/looking them up in a database. To obtain others, some additional analysis of the given data was necessary. Existing password crackers such as John the Ripper could also be used for partial credit.

2.2. *IPSC 2012: Keys and Locks [Topic: Cryptography]*

Statement Summary

The problem statement describes one actual way how master keys for physical locks can be made. The contestants are then asked to devise an attack on such a physical system, armed only with a small set of "blank" keys and an iron file. The attack is then played out by interacting with the grading system.

Solution

The attack was apparently an open secret among locksmiths before being discovered and published by Blaze (2003). The amount of resources needed is surprisingly small – the attacker can easily and efficiently learn the heights of the "teeth" of a master key, one "tooth" at a time.

2.3. *IPSC 2011: Lame Crypto [Topic: Cryptography]*

Statement

Alice and Bob are communicating using a particular cryptographic protocol (described formally in the actual statement). You get to eavesdrop on their messages and change some of them. Show that the particular protocol is faulty by correctly inserting a false message into the communication.

Solution

The contestants were given access to a webpage where they could intercept and possibly modify the packets sent between Alice and Bob. They had to discover a weakness in the cryptographic protocol we used, and then actually exploit it in "real time".

2.4. *IPSC 2011: Jedi Academy [Topic: Computer Graphics]*

Statement

Given a view of a 3D scene with many triangles, compute the color of a particular pixel.

Solution

This would have been a real pain to solve as a programming task. However, there are simpler ways of getting the result. The intended solution was to use OpenGL (or a similar library) to actually model the scene and have it displayed.

2.5. *IPSC 2008: Hidden Text [Topic: Computer Graphics]*

Statement

You are given a picture. Part of the picture has been blurred (using an algorithm very similar to the standard Gaussian blur available in most graphics editors). The blurred part originally contained some text. Recover it.

Solution
There were two main approaches that lead to a correct solution: First, there was the option to use the tools available in a graphics editor to repair enough of the blur to make the text readable. For the second approach the contestants had to realize that they know the font used in the image, so they can compute the blurred forms of individual letters and compare them to the blurred part of the image.

2.6. *IPSC 2008: Comparison Mysteries [Topic: Data Representation]*

Statement
Declare a numeric variable `x` and initialize it to a non-zero value. Your variable must then satisfy `x == -x`.

Declare three numeric variables `x`, `y`, and `z`. Initialize them to any values. Your variables must then satisfy `x == y`, `y == z`, but they must not satisfy `x == z`.

Solution
One of the valid solutions for the first subtask: `int x = -2147483648`. In two's complement representation the range of integers is not symmetric, and the largest negative value has no positive counterpart. Instead, it is its own negation. (Formally, 2^{31} and -2^{31} belong into the same remainder class modulo 2^{32}, and this class is represented by the integer -2^{31}.)

One of the valid solutions for the second subtask: `int x = 1234567890; float y = 1234567890; int z = 1234567891;`. This solution uses the fact that integers and floating-point numbers have incomparable ranges. In the comparisons `x == y` and `y == z` the int gets converted to a float, and the resulting rounding causes both comparisons to evaluate as true. On the other hand, the two ints clearly differ by 1.

3. Non-Traditional Computational Models

We love to challenge our students to write "programs" for various non-traditional computational models. Among other reasons, this eliminates much of the pre-existing knowledge such as fluency in a given programming language, and thus all contestants start in the same place. Additionally, these different computational models often require a different way of thinking.

Traditionally, each year the Slovak OI contains a graduated series of tasks in some such model. In the first round (a long-term round solved at home) they are given a short introductory text that defines the model and shows some small examples. In all rounds of the contest they are then asked to solve progressively more and more difficult tasks in the model.

Some notable models used in the past include: many-one and Turing reductions (2013), log-space programs (2012), Fractran (2011), a-transducers (2007), alternating machines (2005), reversible programs (2002), Wang tiles (2001), and many more.

Below we present just a few examples from this large batch of tasks.

3.1. *OI 2012: Log-Space: Permutation Cycles*

Statement
You are given n and a read-only array $A[1..n]$ that contains a permutation. Write a log-space program (i.e., a program that only uses $O(\log n)$ bits of memory) that counts the cycles of A.

Solution
Note that we cannot use the usual algorithm that uses graph traversal to count the cycles, because we cannot mark the elements of A as visited.

For each x, we walk along the cycle that contains x, and we count this cycle iff x is the smallest value it contains. In this way we are guaranteed to count each cycle exactly once.

(An alternate solution. For each length ℓ between 1 and n: For each x, verify whether the cycle that contains x has length ℓ. Let c be the count such elements. Then we just found c/ℓ cycles of length ℓ.)

3.2. *OI 2011: Fractran: Comparing Exponents*

Statement
A Fractran program is an ordered sequence of positive fractions. A Fractran computation is a sequence S of positive integers. The first integer is the input. Each of the following integers is computed using a simple rule: Let x be the current integer. We find the first fraction f in S such that xf is an integer. The value xf is the next element of the computation. The computation ends when no fraction produces an integer.

The input number n is guaranteed to be of the form 2^x3^y5. Write a Fractran program that terminates with the value 5 whenever $x = y$, or the value 7 whenever $x \neq y$.

Solution
One correct program:

$$\left(\frac{1}{6}, \frac{7}{10}, \frac{7}{15}, \frac{1}{2}, \frac{1}{3}\right)$$

Suppose that $x \neq y$. While both $x > 0$ and $y > 0$, the first fraction is used to decrease both by 1. If this brings us to the situation where $x > 0$ and $y = 0$, we can now use the fraction $7/10$. From this point on, the current value is not divisible by 5, but it is divisible by 7. Similarly, if $x = 0$ and $y > 0$, the first fraction that can now be used is $7/15$. Finally, we use the last two fractions to clear the remaining power of 2 or 3 in the current value, leaving only the 7.

And if $x = y$, we will use the first fraction x times, and then the algorithm terminates with the current value being 5.

3.3. *KSP 2013: Regular Expressions*

Statement (of one Subtask)
Write a regular expression that matches strings of `a`s such that their length is a composite number.

Solution
One such regular expression is “`^(aa+)\1+$`”. We take a sequence of at least two `a`s, and then use a back-reference to state that the rest of the input must consist entirely of a positive number of copies of the selected sequence. This regular expression can only be matched to a string of `a`s if its length has a non-trivial divisor.

3.4. *IPSC 2004: Gets and Puts*

Statement (of one Subtask).
Given is a simple programming language in which all you have are 26 integer registers and a single potentially-infinite queue. Write a program that is a quine (i.e., writes its own source code to the output).

Solution
The standard technique based on the recursion theorem can be used. Imagine a program that can be run once a queue already contains a sequence of integers. This program will do the following steps:

1. Insert an endmarker into the queue.
2. While not at the endmarker: take a number from the queue, **print** the command that inserts the number into the queue, insert the number back into the queue.
3. Throw away the endmarker. The queue is now in its original state.
4. While there is something in the queue: take a number, interpret it as an ASCII value, **print** the corresponding character.

Let this program be P. Let Q be the sequence of ASCII values of characters of P. The solution is a program that first inserts all elements of Q into the queue and then runs P.

4. Algorithm/Code Analysis

In this section we give several examples of tasks where the contestants are asked to analyze an algorithm, or its particular implementation. In addition to the traditional testing problem (find a bug in this program), there are many other variations possible: understanding what a program computes, improving its time complexity, ...

Included in this section is one example of a long-term task.

4.1. *IPSC 1999: Coins*

Statement

We have a set of coins. We want to pay the sum s precisely. We have two different programs. Each of them claims that it can decide whether it is possible to pay s or not. Find a counterexample for each of the programs.

Solution

The first program given to the contestants was the implementation of a simple greedy algorithm: always use the largest coin that does not exceed the remaining sum. There are very simple counterexamples, for example $s = 6$ and we have the following coins: $(5, 2, 2, 2)$.

The second program contained a slightly smarter algorithm. For each coin value v: Let x be the number of coins with this value I have. For each i between 1 and x: Pay i coins worth v, then use the first greedy algorithm to pay the rest. If no pair (v, i) works, return that it is impossible to pay s.

Probably the simplest counterexample combines two layers of the previous algorithm: let $s = 66$ and let the coins be $(50, 20, 20, 20, 5, 2, 2, 2)$.

4.2. *IPSC 2003: begin 4 7 add*

This task actually belongs both into this and into the previous section, because it deals with PostScript. PostScript is often considered to be just a language used to describe the layout of a page . . . but in fact it is a Turing-complete stack-based programming language.

Statement

Given are two PostScript documents that produce the answers you have to submit. Obtain those answers.

Solution

The easier document just iterated a permutation of letters. It was sufficient to find the period, edit the file to use a smaller number of iterations, and open it in a PostScript viewer.

The more challenging document contained a Markov source that generated random words that were irrelevant. But at the same time the code computed the sum of all primes between 1 and 200,000, inclusive, which was the answer.

4.3. *IPSC 2011: BFS Killer*

Statement

Given is a particular implementation of breadth-first search (BFS) on a rectangular grid of cells, some of which contain obstacles. Find a particular input that causes the BFS to have a very long queue.

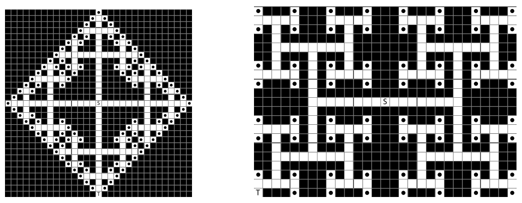


Fig. 1. Two mazes that require a large queue.

Solution
This task addresses a very common misconception[1] that the BFS, when executed on an $n \times n$ grid, will always use $O(n)$ additional memory. This is not the case. Below are two mazes that require a large queue – all cells marked by dots will be in the queue at the same time. For the maze on the right the maximum queue size is $\Theta(n^2)$.

4.4. *KSP 2007: Ordinary Sorting Algorithms*

Note: This is a large long-term task with many subtasks that can be used as separate task. The task involves both theoretical and practical (implementational) subtasks.

Statement
Three kids came up with weird algorithms to sort an array of length n.

Annie: I would just pick a random pair of elements and compare them. Whenever the one on the right is smaller, I would swap them. I would do this in a loop, and after every n comparisons I would check whether the array is already sorted. If it is, I would terminate the algorithm.

Billy: I would use recursion. An array of length 1 is already sorted. If I have an array of a longer length, I would make three recursive calls: I would sort the first two thirds of the array (rounded up if necessary), then the last two thirds of the array, and finally the first two thirds of the array again.

Cecilia: I would take $n-1$ pieces of paper. On the i-th piece I would write the numbers i and $i+1$. I would then proceed in rounds. In each round, I would shuffle the papers into a random order. Then, for each paper, I would read the two numbers on it, compare the elements on those indices, and swap them if necessary. At the end of each round I would check whether the array is sorted.

Implement the three algorithms. Create plots of their runtime as a function of n. Use those plots to guess their time complexity. Find and prove some lower bounds on their worst-case time complexity, – i.e., find inputs that are hard for that particular algorithm, and compute how many steps it would take for those inputs. Prove that algorithm B actually sorts. For extra credit, prove good upper bounds on their time complexity.

[1] A historical remark: a few years ago a task related to BFS was proposed to be used at the IOI. The author made a similar mistake in his intended solution, and the task had to be rejected.

Solutions Sketch
The implementations contain several challenges: In algorithm A, some contestants incorrectly implemented the step where a pair of elements is chosen uniformly at random. Some contestants were unable to implement the recursion in B. In C, incorrect and/or slow implementations are common for randomly shuffling the array of "papers".

The expected time complexity of algorithm A is $O(n^2 \log n)$. This is related to the Coupon Collector problem. E.g., the array $(2, 1, 4, 3, \ldots, 2k, 2k-1)$ contains $n/2$ independent inversions, and we have to hit each of them at least once in order to sort the array.

The time complexity of algorithm B is $\Theta(n^{\log_{3/2} 3})$, which is $O(n^{2.71})$. This can be shown by solving the recurrence $T(n) = 3T(2n/3) + O(1)$. The fact that B actually sorts is left as an exercise. (Hint: color the largest $n/3$ elements red and follow them during the sorting.)

A clear worst case for algorithm C is the reverse of a sorted array, which needs $\Omega(n^2)$ swaps of adjacent elements to become sorted. Algorithm C randomly chooses one from a particular class of oblivious sorting algorithms. Using the 0-1 principle it can be shown that any such algorithm works in $O(n^2)$.

5. Lateral Thinking

Finally, we present three "lateral thinking" tasks. Part of the design of these tasks is to encourage the contestants to "think out of the box", as the saying goes.

5.1. *IPSC 2005: Alpha Centauri Tennis*

Statement Summary
A fairly long and detailed problem statement describes the scoring of a fictional sport – the "Alpha Centauri Tennis". This is essentially the same as traditional tennis, only without tie-breaks, and it is generalized to n players. The essence of the rules remains the same: whoever scores enough points wins a game, whoever wins enough games wins a set, and whoever wins enough sets wins the match.

A match transcript is a string that records, for each ball played in the match, the player who scored the point.

The contestant is given a lot of match transcripts that are *guaranteed to be valid*. The task is to process them and determine the winner of each match.

Solution
As shown by our survey after the contest, a significant portion of teams implemented the full set of rules, computed the score of each match, and used that to return the winner. Needless to say, this approach was painful, error-prone, and slow.

Not to mention its complete uselessness. There is a much simpler solution: the winner of the match has to be the player who won the very last point in the match. Thus it was sufficient to output the last character of each match transcript.

5.2. *IPSC 2007: Know Your Crypto*

Statement (of the Hard Subtask)

We have to apologize for the hard problem, it is probably impossible to solve. This task was prepared at the last possible moment, and we didn't have any idea what to do with the hard input. So finally we decided that we will just randomly change all the letters of the plain text. This is the program we used:

(A listing of an incredibly simple C program followed. The program executed `srand(time(NULL));` and then shifted each letter of the input by a randomly selected amount.)

Solution

The problem intentionally misleads the contestants into thinking that the task is unsolvable, or that the solution lies in somehow reversing the pseudorandom `rand()` function using the fact that it is not perfectly random.

The correct clue is hidden in the words "at the last possible moment". This gives only a small set of values `time(NULL)` could have returned, and the contestant can try them all by brute force, and select the one that produces a readable plaintext.

(As an interesting note, similar poor choices of random seeds have been documented in practice, e.g., in Casino de Montréal in 1994.)

5.3. *IPSC 2012: (Blank)*

Statement

Solution

The statement above is printed correctly. It had precisely zero characters (a clear world record!).

Still, the task was solvable – by using the error messages from the grader. For your first submission, you were most likely to receive the following error message: "`Wrong answer: Not a~sequence of positive integers`". Obviously, your second submission would contain such a sequence. The easy subproblem was solvable by simply following the instructions, for the hard subproblem some additional insight was needed.

Acknowledgements. The author is grateful to all his colleagues and students who collaborated on preparing the competition tasks described in this paper. Task authors for individual IPSC tasks can be found on the IPSC website; all tasks are licensed under the Creative Commons Attribution-ShareAlike 3.0 license.

References

Almeida, F., Pérez, V.B., Cuenca, J., Férnandez-Pascual, R., García-Mateos, G., Gimenéz, D., Guillén, J., Benito, J.A.P., Requena, M.-E., Ranilla, J. (2012). An experience on the organization of the first spanish parallel programming contest. *Olympiads in Informatics*, 6, 133–147.

Bell, T., Fellows, M.R., Witten, I. (1998). *Computer Science Unplugged … Off-Line Activities and Games for all Ages*.

Blaze, M. (2003). Cryptology and physical security: rights amplification in master-keyed mechanical locks. *IEEE Security and Privacy*.

Burton, B.A. (2010). Encouraging algorithmic thinking withouty a computer. *Olympiads in Informatics*, 4, 3–14.

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

Forišek, M. (2007). Slovak IOI 2007 team selection and preparation. *Olympiads in Informatics*, 1, 57–65, 2007.

Forišek, M., Winczer, M. (2006). Non-formal activities as scaffolding to informatics achievement. In: Dagiene, V., Mittermeir, R. (Eds.), *Information Technologies at School*, 529–534.

Forišek, M., Steinová, M. (2010). Didactic games for teaching information theory. In: *Teaching Fundamentals Concepts of Informatics (Proceedings of ISSEP 2010)*.

Ginat, D. (2011). Algorithmic problem solving and novel associations. *Olympiads in Informatics*, 5, 3–11.

Ginat, D. (2012). Insight tasks for examining student illuminations. *Olympiads in Informatics*, 6, 44–52.

Kemkes, G., Cormack, G., Munro, I., Vasiga, T. (2007). New task types at the canadian computing competition. *Olympiads in Informatics*, 1, 79–89.

Kubica, M., Radoszewski, J. (2010). Algorithms without programming. *Olympiads in Informatics*, 4, 52–66.

Kulczyński, T., Łącki, J., Radoszewski, J. (2011). Stimulating students' creativity with tasks solved using precomputation and visualization. *Olympiads in Informatics*, 5, 71–81.

Opmanis, M. (2009). Team competition in mathematics and informatics "ugāle" – finding new task types. *Olympiads in Informatics*, 3, 80–100.

Opmanis, M., Dagienė, V., Truu, A. (2006). Task types at "Beaver" contests. In: Dagiene, V., Mittermeir, R. (Eds.), *Information Technologies at School*, 509–519.

P. Petroviṅć. Ten years of creative robotics contests. In: *Proceedings of ISSEP 2011*.

Pohl, W. (2008). Manual grading in an informatics contest. *Olympiads in Informatics*, 2, 122–130.

Pohl, W. (2011). Computer science contests for secondary school students, approaches to classification. *Informatics in Education*, 5, 125–132.

Ragonis, N. (2012). Type of questions – the case of computer science. *Olympiads in Informatics*, 6, 115–132.

Ribeiro, P., Guerreiro, P. (2007). Increasing the appeal of programming contests with tasks involving graphical user interfaces and computer graphics. *Olympiads in Informatics*, 1, 149–164.

Skūpienė, J. (2006). Programming style – part of grading scheme in informatics olympiads: Lithuanian experience. In: Dagiene, V., Mittermeir, R. (Eds.), *Information Technologies at School*, 545–552.

Truu, A., Ivanov, H. (2008). On using testing-related tasks in the IOI. *Olympiads in Informatics*, 2, 171–180.

van der Vegt, W. (2006). The CodeCup, an annual game programming competition. In: W. Pohl Ed., *Perspectives on Computer Science Competitions for (High School) Students*.

van der Vegt, W. (2012). Theoretical tasks on algorithms; two small examples. *Olympiads in Informatics*, 6, 212–217.

Verhoeff, T. (2009). 20 Years of IOI competition tasks. *Olympiads in Informatics*, 3, 149–166.

M. Forišek is an assistant professor at the Comenius University in Slovakia. Since 2006 he serves as an elected member of the International Scientific Committee (ISC) of the IOI. He is also the head organizer of the Internet Problem Solving Contest (IPSC). His research interests include theoretical computer science (hard problems, computability, complexity) and computer science education.

Olympiads in Informatics, 2013, Vol. 7, 36–41
© 2013 *Vilnius University*

Expecting the Unexpected

Steven HALIM
School of Computing, National University of Singapore
Computing 1, 13 Computing Drive, 117417 Singapore
e-mail: dcssh@nus.edu.sg

Abstract. In recent IOIs, the IOI community has observed creative tasks such as *'Language'*/*'Maze'*/*'Saveit'* (IOI Tasks, 2010), *'Parrots'* (IOI Tasks, 2011), and *'Odometer'*/*'Supper'* (IOI Tasks, 2012) that caught many contestants and their coaches off-guard. This is because it is very hard to train our students to be 100% ready for such *unexpected* tasks. Reading the IOI 2013 call for tasks requirements (IOI Call for Tasks, 2013) points us towards more of IOI tasks of such nature. In this paper, we share how the Singapore IOI team expects the unexpected.

Key words: IOI, creative task.

1. Introduction (IOI 2010)

The author is a regular member of the IOI community having attended the past three IOIs (2010 @ Waterloo, Canada; 2011 @ Pattaya, Thailand; and 2012 @ Sirmione–Montichiari, Italy). In the past three years, the IOI community has observed creative tasks that challenge the best high school students in informatics. In this paper, I seek to highlight some tasks which surprise both the students and the coaches *more* than the other – more traditional – tasks. In this paper, we define a *traditional task* as a task that asks students to write a program that reads input, performs some computation within the stipulated time and memory constraints, returns output, and is graded mainly based on the correctness of students' output against the judges' output.

On the night before IOI 2010 Day 1, the delegation leaders were presented with the task '*Language*' (task author: Gordon V. Cormack) – a problem on the field of *Information Retrieval* that has only been used once many years ago in IOI 1991 (IOI Tasks, 1991). In this task, students are given a sequence of Wikipedia excerpts, and asked to guess the language of each, in turn. After each guess, the students' program is given the correct answer, so that it may *learn* to make better guesses the longer it plays. The students' program is graded based on 'accuracy' – a performance metric that was *never*[1]. *used before* in IOI (Cormack, 2010). Upon seeing such task, many delegation leaders voiced their opinions on such task. Many lauded the novelty of the task but there were some who had reservations that some students would not be able to do well (or might skip the task altogether) as the task fell outside the IOI syllabus (Forišek, 2009). That night,

[1]This statement is based on IOI 2008–2012 tasks. The author has limited experience of the older IOI tasks.

this task was eventually voted to be used in the IOI 2010 Day 1 by the General Assembly (IOI GA Minutes, 2010). It turned out to be successful with most students attempted it. Using 'accuracy' as the performance metric yielded a good, smooth score distribution that showed which students were more creative that day.

Things became more 'exciting' when the IOI 2010 Day 2 problem set was presented to the delegation leaders as it contained task *'Maze'* (task author: Michal Forišek) and *'Saveit'* (task author: Mihai Pătraşcu). The task *'Maze'* is a variant of the '*longest path*' problem which is known to be NP-hard (Garey and Johnson, 1979) and thus has no polynomial solution. However, the test cases were fixed and made available to the students, making this a more approachable output-only task. Students were graded based on the *longest paths* that they can generate for each of the test cases. Students were free to use any computing tools that they have in their workstation to generate these longest paths. The task *'Saveit'* is a '*data compression*' task which illuminated the delegation leaders and the students as to why the organizers insisted on using the new judging system (RunC, 2010–2011) at that time. The novel part of task *'Saveit'* is this: students have to program *two* independent procedures that are inverses to each other: an encoder and a decoder. The communication format is not prescribed. The decoder *programmed by the student* must be able to decode the data from the encoder that is also *programmed by the student*. The grader runs these two procedures and gives higher scores to programs that use a fewer number of bits (which means better communication efficiency).

Based on the success of task *'Language'* on Day 1, there was no major objection for these two creative tasks (IOI GA Minutes, 2010). These two creative tasks obtained mixed results. The result of *'Saveit'* was okay, but too many students stuck at 50 points and only a few scored 25, 75, or 100 points. The result of *'Maze'* is again a smooth score distribution, which implies that scoring a few points in this task very important especially for students around medal boundaries that year.

IOI 2010 was closed on a high note: this scientific experiment was successful. With three unexpected tasks like that, the medalists were worthy winners. For the delegation leaders (and coaches), we all went home with a big homework: IOI training would never be the same again.

2. What We Did in 2011 (Preparation for IOI 2011)

For IOI 2011 preparation, I decided to train the Singapore students on *several* classic NP-hard problems (Garey and Johnson, 1979) that I had studied during my PhD days. We discussed on how to optimize the classic O($n^2 2^n$) DP solution for Traveling Salesman Problem (TSP) so that it can be used to deal with instances up to $n = 19$ or $n = 20$ (previously such a DP solution is already too slow for $n = 16$ or $n = 17$). We also discussed how to solve the smaller instances of the Quadratic Assignment Problem (QAP) (Halim *et al.*, 2007) and the Low Autocorrelation Binary Sequence (LABS) problem (Halim *et al.*, 2008) using only techniques that are allowed in the IOI syllabus (Forišek, 2009).

Obviously it is very hard (and near impossible) for the high school students to reach the author's 3–4 years of PhD work in just 5 contest hours and mainly only using the techniques in the IOI syllabus that have been exposed to them. Nevertheless, it was a fun and eye opening training. I found that some students (the eventual medalists in IOI 2011) managed to score highly, getting respectable solution quality for these NP-hard problems compared to the best results found by the author (using various local search techniques that are definitely *outside* the IOI syllabus).

The source of NP-hard problems for task inspiration may be plentiful (e.g., Garey and Johnson, 1979). However, setting up such training tasks is *very painful* as the trainer must properly define a *restricted subset* of the original NP-hard problem so that the training task becomes approachable for high school students. Otherwise, too many students will end up with near 0 score and thus become discouraged. We need to scale down the task so that a few students (the ones that have the potential to be selected in the team that year) will be able to get reasonably good solutions in 5 contest hours.

During the actual IOI 2011, the same judging system (RunC, 2010–2011) was still used. Therefore, some established countries in the field of informatics who had done their homework would have expected something unexpected.

I was betting on another NP-hard task in IOI 2011, but there was none. There was no major surprise on Day 1 and there were 'too many' potential gold medalists scoring 300 points that day – that is, full marks for all three tasks.

However, on IOI 2011 Day 2, another creative task *'Parrots'* (task author: Jittat Fakcharoenphol) was used. It is about '*Computer Networks*'. It has similar flavor to *'Saveit'* in the sense that it leverages upon the strength of RunC judging system. It uses the concepts of bit manipulation heavily (which I am grateful because I covered this in Singapore training program). Learning from the score distribution of *'Saveit'* in 2010, the Scientific Committee made another breakthrough by setting an 'interesting' scoring system for each subtask (not just 5 fixed scores: 0, 25, 50, 75, 100 as in *'Saveit'* but 4 fixed scores 17, 17, 18, 29, and *variable* scores up to 19 points for the last/hardest subtask in *'Parrots'* to differentiate the top students). The tasks selected in Day 2 and the scoring system managed to differentiate the top students. Only one student and the eventual winner managed to get the full 600 points over two contest days. This highlights the needs of such creative tasks to challenge today's best IOI students.

3. What We Did in 2012 (Preparation for IOI 2012)

After seeing two instances of 'data compression' tasks in *'Saveit'* (IOI Tasks, 2010) and *'Parrots'* (IOI Tasks, 2011), I decided that the Singapore team should do something about such a task type in our training program. Therefore, we upgraded our local grading server (Mu Judge, 2009–2013) to handle such a task type and I set a task titled *'Compress'* as one of the tasks for the Singapore IOI selection test 2012. It is about the field of '*Information Theory*' where students are asked to write a 'zip' program to compress the given set of text files *as small as possible* and then write another 'unzip' program to decompress their own compressed text file to get the original file *without any information loss*. In this task, the text files to be compressed are given so that the students can utilize whatever properties that they see in the given test files and use those properties for their benefit.

With this single task, I wanted to know which students are more comfortable in dealing with the code/decode tasks akin to *'Saveit'* and *'Parrot'* and which students know (or manage to 'reinvent') various data compression techniques during the 5-hours contest time, i.e., using Run Length Encoding, using the more efficient larger base to store integers, using the more space efficient adjacency list to store a graph data structure compared to adjacency matrix, using Huffman encoding (optional, as this is outside the IOI syllabus), and various other small tricks to further compress the given test files especially after looking at the properties of the text files.

In IOI 2012, my bet on this training type was correct. In the actual IOI 2012 Day 2, task *'Supper'* (task author: Richard Královič) was selected as one of the tasks. This task uses data compression as *part of* the overall solution and uses the same interface style as with *'Saveit'* and *'Parrots'* although the judging software was new: CMS (Maggiolo and Mascellani, 2012). It is therefore not surprising (and somewhat expected) that the Singapore IOI team managed to get reasonably high scores in that task[2] (see Table 1 – the data is not shown in detailed fashion to keep the spirit of IOI as an individual contest and not a country-based contest).

IOI 2012 Day 1 also had its surprises. The task *'Odometer'* (task authors: Michal Forišek, Giovanni Paolini, and Matteo Boscariol) redefined the meaning of *output only* task. In this task, the students do not submit the output of a certain input file after processing them locally, but the students submit 'codes' written using a meta-language to be graded by a special judging mechanism[3]. Each subtask is like a standalone problem on its own. This task can be a (very) *time consuming* task as students need to first understand the meta-language via some experiments before they can actually solve the subtasks. Good time management was compulsory in order to do well in all the 3 tasks given that day. There were 5 subtasks for *'Odometer'* but I can foresee that the Scientific Committee can easily increase the number of such subtasks to 6 or 7 to further increase the challenge of this task if they foresee the need to do so (it happened to be enough to differentiate the eventual winner and the runner up of IOI 2012). Tasks of this nature are frequently used in the Internet Problem Solving Contest (IPSC, 1999–2013). Fortunately I asked Singapore team members to join such a contest every year.

In terms of results, no country got 400 marks for these two creative tasks and task *'City'* (i.e., all 4 students from that country scored full 100 marks) whereas there exists at least one country with 400 marks for the other 3 tasks of IOI 2012 (i.e., task *'Rings'*, *'Scrivener'*, and *'Tournament'*). I foresee that with this trend of increasing skill level of the top students, the Scientific Committee of the future IOIs have no choice but to keep surprising these brilliant students (and their coaches) by using more and more creative tasks.

[2] Another potential reason is the *length* of the task description of *'Supper'*. Many students from non English-speaking countries have difficulties understanding the requirements of the task while this is not a problem for Singapore students. However, I believe future IOI task descriptions will be shorter than this task.

[3] CMS (Maggiolo and Mascellani, 2012) had to be tweaked to support this task.

Singapore's country rank on three recent creative tasks (by summing 4 students' scores). Data obtained from SnarkNews (2006–2013)

Task	Parrots (2011)	Odometer (2012)	Supper (2012)
Singapore's rough rank	Top 40 percentile out of 78 countries	Top 10 percentile out of 82 countries	Top 10 percentile out of 82 countries

4. Closing Remarks

I am thankful to the International and Host Scientific Committee as well as the various task contributors for their creative tasks. Though the more creative they are, the more difficult the IOI training will be, I feel that it should be the case as the IOI is the pinnacle of informatics competition among high school students.

The tasks selected by the Scientific Committee indirectly determine what will be included in the training program of various countries – especially the more established ones – in the following year. Although it is rather impossible to correctly predict what kind of tasks will appear in the future IOIs – as illustrated in Singapore case in 2011, we know that future tasks will be more to the creative side than the classic side, as shown in the recent IOI Call for Tasks (IOI Call for Tasks, 2013).

I believe that we have to train our students to be creative problem solvers by purposely exposing them to various non-classical tasks from diverse fields of Computer Science curriculum and then purposely use some of those tasks as part of the selection criteria. This way, the creative students are preferred over those who mastered many algorithms but not creative problem solvers. We hope that the selected creative students can perform as well as possible on these unexpected tasks during the actual IOI.

In Singapore, we do not have that many training tasks of this nature yet. The most obvious reason is due to the difficulty in preparing such creative tasks in the first place. During the actual IOI conference 2013, I am planning to release *some* of our 'creative' tasks to the delegation leaders to be used in the training programs in their countries.

References

Cormack, G.V. (2010). `http://web.archive.org/web/20100826074432/http://plg1.uwaterloo.ca/~gvcormac/ioi/ioi.html`

Forišek, M. (2009). `http://people.ksp.sk/~misof/ioi-syllabus/`

Garey, M.R., Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A Series of books in the mathematical sciences. In: Klee, V. (Ed.), W.H. Freeman & Co.

Halim, S., Yap Hock Chuan, R., Chuin, L.H. (2007). *An Integrated White+Black Box Approach for Designing and Tuning Stochastic Local Search. Constraint Programming*. Springer.

Halim, S., Roland, Chuan, R.Y.H., Halim, F. (2008). *Engineering Stochastic Local Search for the Low Autocorrelation Binary Sequence Problem. Constraint Programming*. Springer.

IOI Call for Tasks (2013). `http://www.ioi2013.org/competition/call-for-tasks`

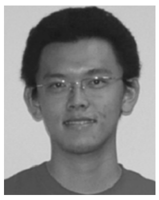
IOI GA Minutes (2010). `http://ioinformatics.org/admin/ga/ioi10/GA-IOI10.pdf`

IOI Tasks (1991). `http://olympiads.win.tue.nl/ioi/ioi91/tasks91.txt`

IOI Tasks (2010). `http://ioi2010.org/CompetitionTask.shtml`

IOI Tasks (2011). `http://www.ioi2011.or.th/tasks`

IOI Tasks (2012). `http://www.ioi2012.org/competition/tasks/`
IPSC Internet Problem Solving Contest (1999–2013). `http://ipsc.ksp.sk/`
Maggiolo, S., Mascellani, G. (2012). Introducing CMS: a contest management system. *Olympiads in Informatics*, 6, 86–99.
Mu Judge (2009–2013). `http://noi-1.comp.nus.edu.sg/mu/index.php`
RunC (2010–2011). `http://ioi2010.org/Environment.shtml`
SnarkNews (2006–2013). `http://ioi.snarknews.info`



S. Halim is a lecturer in SoC, NUS. He teaches several programming courses, ranging from basic programming methodology, intermediate data structures and algorithms, and up to the ‘Competitive Programming’ module. He is the coach of both NUS ACM ICPC teams and Singapore IOI team. So far, he and other trainers in NUS have successfully groomed several teams that advance to ACM ICPC World Finals four times as well as two gold, six silver, and seven bronze IOI medalists since year 2009.

Olympiads in Informatics, 2013, Vol. 7, 42–54

State Competitions in Informatics and the Supporting Online Learning and Contest Management System with Collaboration and Personalization Features MENDO

Mile JOVANOV, Bojan KOSTADINOV, Emil STANKOV,
Marija MIHOVA, Marjan GUSEV

Faculty of Computer Science and Engineering, University Ss. Cyril and Methodius
Rugjer Boshkovikj 16, Skopje, Macedonia
e-mail: mile.jovanov@gmail.com, bojan.kostadinov@gmail.com, emil.stankov@gmail.com, marija.mihova@finki.ukim.mk, marjan.gushev@finki.ukim.mk

Abstract. Macedonia has a tradition of organizing programming contests for high-school students as long as the tradition of IOI. In the last few years the organizer, the Computer Society of Macedonia (CSM), used the online learning and contest management system MENDO, firstly for the organization and carrying out of the national contests, and also for the promotion of the contests, programming and algorithmic thinking. CSM tries to provide a portal for the students that will offer complete learning materials and tools, and that will motivate them and allow them to learn programming and prepare for the competitions. In this paper we will present MENDO with its new features, and we will mention some new and original features for collaboration and personalization. At the end, we will present the great improvements in sense of the number of contestants and achievements of Macedonian students at the international competitions, showing that MENDO is a valuable asset for organization of competitions.

Key words: programming contests, collaboration, grading systems, automatic grading, contest management system.

1. Introduction

Competitions in informatics were introduced about forty years ago, with the idea of attracting talented young people to the science of computer programming. These competitions are usually synonyms for algorithmic programming contests (other types include architecture, design, development, specification, assembly, testing scenarios, etc).

Competitions in informatics have a long tradition in Macedonia. There were 23 national contest cycles till the end of 2012. After many competitions on national level, the best contestants represent themselves and Macedonia at the IOI – International Olympiad in Informatics, the BOI – Balkan Olympiad in Informatics (for high school students), and the JBOI – Junior Balkan Olympiad in Informatics (for primary school students).

Usually, these programming competitions require students to submit programs which are then run through a variety of test scenarios and judged accordingly. The difficulty, however, lays not so much in the programming but rather the design of the underlying algorithms (Burton, 2008). More often than not, these contests are based on automatic grading of the submitted solutions. This is accomplished by running them on batches of input data and testing correctness of the output. Time and space limits are usually enforced during the process, which allows to judge not only by the (approximation of) correctness of the solution, but also by its time and space complexity, as explained in Mares (2007).

Besides the automatic grading capability, a contemporary programming contest system requires some other functionalities that would facilitate the preparation of the students for competitions, like some kind of a communication page (or a forum), a page for sharing appropriate learning materials, an info page, etc.

In this paper we present the system that supports the Macedonian competitions in informatics, named MENDO, along with its improvements during the last three years of its usage. The paper is organized as follows. In Section 2 we describe the format of the informatics competitions in Macedonia, their organizer, and the challenges we are facing. In Section 3 we present the need for contest management systems, and we mention the necessary building modules of such a system. Section 4 presents the architecture of our system MENDO, with the improvements from its first deployment until today. In this section significant specific realizations of some modules are explained. Additionally, some user oriented features of MENDO are explained including some unique ones for collaboration and personalization. Further, in Section 5 we present some results from the VI Junior Balkan Olympiad in Informatics, held in 2012 – a contest organized using MENDO. Finally, in Section 6, we present results from the use of MENDO in the Macedonian competitions in sense of the quality of the organization, the pupils' interest for the competitions, and most significantly the accomplished results in international competitions.

2. Competitions in Informatics in Macedonia

Competitions in informatics have been held in Macedonia since 1990. There were 23 national contest cycles till the end of 2012. Every year, the contestants go through many levels of competition in order for the best to be selected: Qualifications, Regional Competition, State Competition, Macedonian National Olympiad, and Preparations for international contests (sometimes including additional competitions). The selected pupils represent themselves and Macedonia at the BOI/JBOI and at the IOI. The main organizer of the competitions in informatics is the Computer Society of Macedonia (CSM).

2.1. *Computer Society of Macedonia (CSM)*

CSM was formed on an initiative of a group of professors at the Institute of Informatics, Faculty of Natural Science and Mathematics, at the University of Ss. Cyril and Methodius in Skopje, in the year 2000. This organization continued with the activities previously performed by the former Mathematical and Computer Society of Macedonia (Jovanov, 2012).

CSM is one of the holders of the idea for affirmation of the informatics society in Macedonia. Members of this organization are computer science teachers in high schools and primary schools, information technologists, as well as professors and teaching assistants at the Faculties of the Universities in Macedonia.

Among the main goals of CSM are the following:

- introduction, popularization and promotion of informatics and information technology and its application;
- encouragement and introduction of informatics in all areas of the society, especially in the education;
- organization, implementation and participation in informatics competitions for pupils and students.

2.2. *Format of the Macedonian Informatics Competitions*

The first steps of informatics in high school education were made in the middle 1980s. A few years later, in 1990, the first State Competition in Informatics was held in Prilep. In 1993, CSM started to organize Regional Competitions in Informatics. The first Macedonian National Olympiad in Informatics was held in 1997. The competitions for primary school pupils began to be held in 2007. So far, CSM has organized 20 Regional Competitions, 23 State Competitions and 16 Macedonian National Olympiads for high school pupils, and 6 State Competitions for primary school pupils. (In Macedonia, Primary school consists of 9 grades, followed by obligatory Secondary school which can be gymnasium with 4 grades or vocational school with 3 or 4 grades).

At the beginning, in the first few competition years of the high school competitions, all the contestants were given only one set of programming tasks in each competition. Later, it was decided that it would be better to have the two groups of contestants solve sets of tasks with different degree of difficulty. So, the contestants were divided into two groups (named A and B), having a different set of tasks for each group (one set being more difficult than the other). After few years, a different naming convention for the contestant groups was introduced – from then on they were named ‘Easier group’ and ‘Harder group’.

The format of the competitions evolves every year, depending on many factors, such as the number of interested pupils, the inclusion of programming in the schools curricula etc. Presently, the competitions are organized for high school and primary school pupils, as follows. The contestants, depending on the level of acquired knowledge in the programming area, at the start of each contest cycle have to choose between:

- ***Beginners group***: primary school pupils or high school pupils with no experience, usually in their first year of high school education, not older than 15 years on first of January, in the actual year (eligible to participate at JBOI).
- ***Basic group***: first or second time contestants that consider themselves not to have enough experience to participate in international competitions.

- ***Advanced group***: contestants that consider themselves to have enough experience to solve complex algorithmic problems and to participate in international competitions (BOI, IOI) .

All the competitions are entirely conducted through a system called MENDO.

2.3. *Challenges in the Organization of the Competitions*

The organizer of the informatics competitions in Macedonia, the Computer Society of Macedonia, is a non-government, non-profit organization. Hence, the organization of the competitions is based solely on sponsorships from companies, educational institutions (like the Faculty of Computer Science and Engineering in Skopje), and sporadically from donations based on application in some calls for projects. Having in mind the low finances, CSM has to find as cost-effective as possible way for the following:

- ***Engaging pupils in the competitions*** – We have to spread the information among the pupils and keep them informed. We have concluded that the best way to do that is to build and maintain a 'community' of competitors, and let them collaborate throughout the whole year. Of course, we also use some more traditional methods, e.g., contacting and informing the schools, contacting the computer science teachers, but there are obstacles in that regard too.
- ***Motivating teachers and school authorities*** – The most important thing (on this issue) is to motivate the teachers to inform and mentor the pupils. With years of our experience, we have found that the computer science teachers, mainly, do not want to spend extra time for tutoring gifted pupils and the greatest reason for this is that they are not familiar enough with the curriculum of the programming competitions. The best solution consists of two things: (1) Provide online materials, training and q/a support for the pupils, in every stage of their training for the competitions, and (2) Inform the teachers for this convenience and put additional pressure directly through the pupils. The school authorities can also be an obstacle for the participation of the pupils, mostly with ridiculous reason in mind like not spending money for the trip of the pupils to the contest venue. On this issue, we can only hope on the pressure of the pupils and the teachers on them.
- ***Keeping the participants informed, and 'in condition'*** – We have to publically present the information for the next steps in the competitions (dates, rules, procedures, results, etc.) in order to directly inform the pupils, as we have concluded that this is the most reliable way to spread information (opposite to informing through the teachers and schools). The greatest challenge of all is to motivate pupils to constantly prepare (work on solving problems) for the competitions, with public sets of tasks that can be solved at any time.

During the last couple of years, some events and movements in the Macedonian society and worldwide have worked in our benefit, thus providing us with better conditions for fulfilling our goals. For example, we had a dramatic improvement in the area of internet penetration, and nowadays we have similar conditions for internet access as most EU countries. Also, the government project "Computer for every pupil" allowed easy access to internet content from every classroom.

3. Contest Management and Grading Systems

A contemporary programming contest system, besides the automatic grading capability also requires other functionalities that would facilitate the preparation of students for competitions. That is why it should be referred as contest management and grading system.

At the International Olympiads in Informatics, the use of a grading system is an absolute necessity. By comparison of the data presented in Manev (2009) with current observations from IOI competitions we can conclude that there was drastic improvement in the time-consumption of the grading process at different IOI competitions. The time for grading was reduced from couple of hours to only minutes, and at very recent IOI's, with the use of tokens, to almost zero time.

Historically, in Macedonia for example, a few separated systems (evaluation program, contest website, forum page called "Communication window", etc.) supported the organization of the competitions. However, a lot has changed since the first competitions – the computer equipment has improved, new information technologies have been introduced, and at the same time, the technique and technology of development and judging competition tasks has improved.

The best solution, from a practical point of view, certainly is to integrate all the required functionalities mentioned above in a single automated system for complete contest management. This system would provide all the necessary contents for the students to successfully prepare and participate in programming contests, as well as all the necessary capabilities for the organization of programming contests (including submission of tasks and automatic grading).

3.1. *Present Contest Management Systems*

There are few existing systems for contest management and grading. Examples include Mooshak, Moe, CMS, DOMjudge and PC^2. Today, however, we can talk about self managed systems that completely administer contests (including gathering and grading of submissions, managing competitor's questions and clarifications, publishing results, providing statistics, etc.). Some systems even offer a continuous on-line training process.

Whatever the system offers, it should be designed such that it is simple, robust, secure and flexible. According to our observations and the observations from several other authors, standard modules for almost all grading systems are:

- **Sandbox** – ensures that the execution of a submission will not harm the system or the host computer; enforces time, memory and network restrictions;
- **Grader** – does compilation of contestant solutions, management of the sandbox, comparison of user output to correct output, and grading of a submission;
- **Controller** – handles the communication with the judges and competitors, and executes database operations;
- **Auxiliary modules** – handle printing, backups, storage, etc.

4. Architecture and Other Features of MENDO

Here we describe the system called "MENDO", which is currently used by the Macedonian Computer Society in the organization of national informatics competitions. This system was introduced in Kostadinov (2010). MENDO was developed following the goal of integration of all previously used modules' functionalities in one compact environment: uploading of the competition tasks (organizers) and the solutions (contestants), evaluation and grading of the uploaded solutions, publishing results and communication. The acronym "MENDO" in the Macedonian language stands for "Macedonian Electronic Competitions and National Olympiads", and as a word it means "teddy bear".

4.1. *Architectural Design and Technical Features*

MENDO includes all the modules that are standard for contest management and grading systems. It works as a web application, so judges and competitors use browsers to communicate with the system.

The controller and grader are written in Java, and can be run on every popular operating system (like Windows and Linux). For database operations we use Hibernate and C3P0 thread management to connect to a MySQL database and execute queries. There are two sandbox implementations: one for Windows and one for Linux. Besides the fact that only a small number of grading systems operate on Microsoft Windows, we found the operating system to be very stable, reliable and easily controllable. More details about the implementation of the sandbox are presented in Kostadinov (2010).

In our implementation, the auxiliary modules are part of the web application and are also written in Java and run on an Apache Tomcat server. Besides the main system, MENDO also contains a public forum and a wiki.

An important feature of MENDO is that it uses cookies to support SSO – Single Sign On across multiple applications (the main system, the wiki, and the forum). All applications are running on the same server. Since all subsystems use Single Sign On and automatic language detection, there is no need to change the language or login every time you switch from one subsystem to the other, since this is automatically done by the underlying system.

Other MENDO specific technical features can be summarized as follows:

- MENDO controls the entire system of the Computer Society of Macedonia, by providing automatic backups for itself and the other applications, self-tests of the application, the server and the operating system;
- The system has multilingual support (currently Macedonian and English, but we are planning to add more languages in the near future);
- MENDO is managed by several administrators and moderators, each with his own privileges and responsibilities. Every moderator and administrator can add tasks, create competitions, generate reports for each task and competition, and initiate system backups;
- The system easily distributes load. Plugging more graders is easy, thanks to the modular architectural design of the system;

- There is a heavy use of AJAX to simplify user interface operations (during registration, training sessions, competitions, etc). We use the jQuery 'write less – do more' javascript library to implement most of the event handling and AJAX operations.

4.2. *Employment of MENDO*

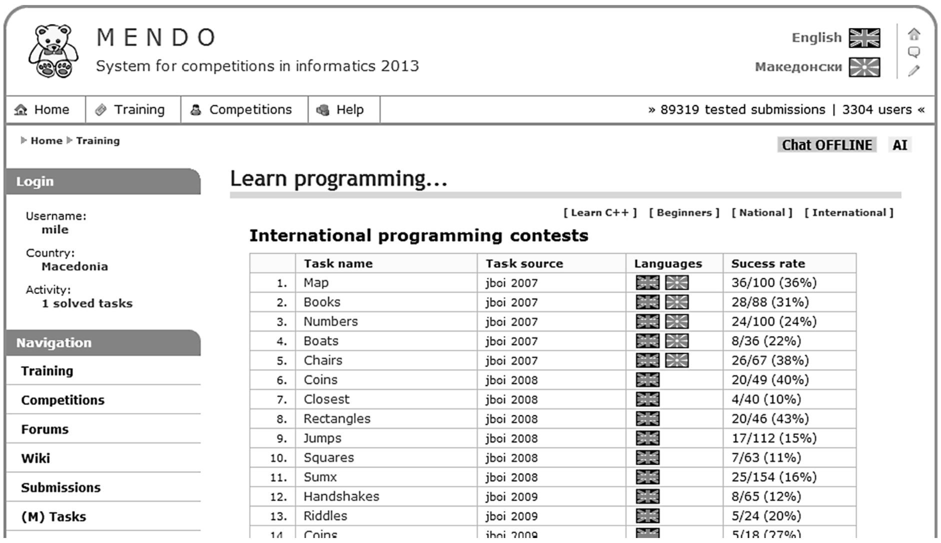
In practice, MENDO is used as:
- a training system (contains tasks from past contests, both national and international);
- a contest management system (for organizing official national competitions and open online tournaments);
- Macedonian algorithmic programming gateway, containing a news page, a lot of programming related materials (organized in a wiki), and a public forum.

4.3. *MENDO as a Training System*

Given the format of competitions, and especially the way of grading the solutions through test cases, the training system had to be able to provide the same kind of judging as real contest grading system. The main and simplest idea is to copy the real grading system, and use it in the training section.

The MENDO's training system *can be used 24/7* (a screenshot of the MENDO training section is shown in Fig. 1).

Every time a user logs in to MENDO's web-site, he can view all the tasks that are available for training, and he can submit a solution. After a solution has been submitted,


MENDO
System for competitions in informatics 2013
English
Македонски
Home
Training
Competitions
Help
» 89319 tested submissions | 3304 users «
Home Training
Chat OFFLINE
AI
Login
Navigation
Training
Competitions
Forums
Wiki
Submissions
(M) Tasks
[Learn C++] [Beginners] [National] [International]

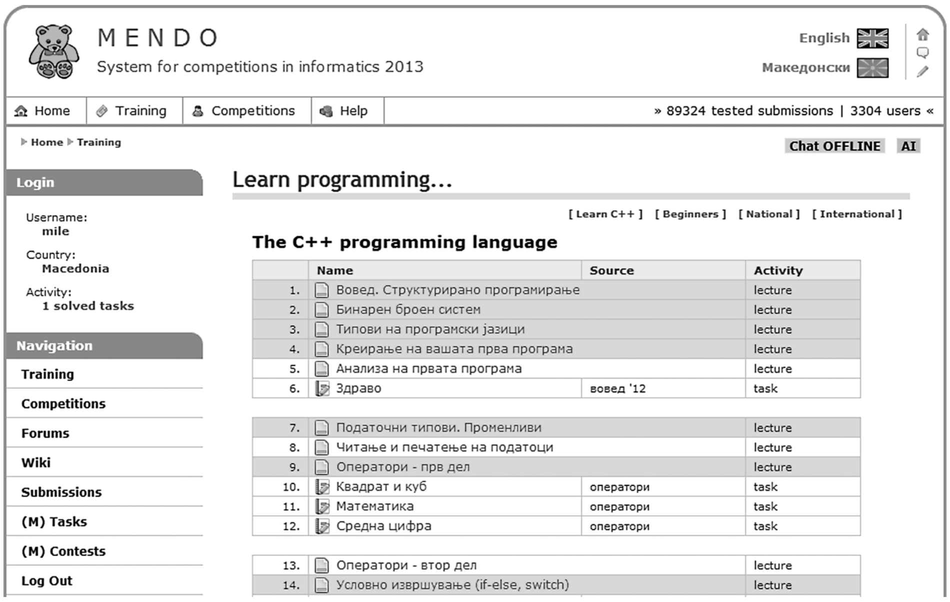
Learn programming...

Username: mile
Country: Macedonia
Activity: 1 solved tasks

International programming contests

	Task name	Task source	Languages	Sucess rate
1.	Map	jboi 2007	🇬🇧 🇬🇧	36/100 (36%)
2.	Books	jboi 2007	🇬🇧 🇬🇧	28/88 (31%)
3.	Numbers	jboi 2007	🇬🇧 🇬🇧	24/100 (24%)
4.	Boats	jboi 2007	🇬🇧 🇬🇧	8/36 (22%)
5.	Chairs	jboi 2007	🇬🇧 🇬🇧	26/67 (38%)
6.	Coins	jboi 2008	🇬🇧	20/49 (40%)
7.	Closest	jboi 2008	🇬🇧	4/40 (10%)
8.	Rectangles	jboi 2008	🇬🇧	20/46 (43%)
9.	Jumps	jboi 2008	🇬🇧	17/112 (15%)
10.	Squares	jboi 2008	🇬🇧	7/63 (11%)
11.	Sumx	jboi 2008	🇬🇧	25/154 (16%)
12.	Handshakes	jboi 2009	🇬🇧	8/65 (12%)
13.	Riddles	jboi 2009	🇬🇧	5/24 (20%)
14.	Coins	jboi 2009	🇬🇧	5/18 (27%)

Fig. 1. Screenshot of the MENDO training system – lists of tasks. Available online at `http://mendo.mk`



Username:
mile
Country:
Macedonia
Activity:
1 solved tasks

Learn programming...

The C++ programming language

	Name	Source	Activity
1.	Вовед. Структурирано програмирање		lecture
2.	Бинарен броен систем		lecture
3.	Типови на програмски јазици		lecture
4.	Креирање на вашата прва програма		lecture
5.	Анализа на првата програма		lecture
6.	Здраво	вовед '12	task
7.	Податочни типови. Променливи		lecture
8.	Читање и печатење на податоци		lecture
9.	Оператори - прв дел		lecture
10.	Квадрат и куб	оператори	task
11.	Математика	оператори	task
12.	Средна цифра	оператори	task
13.	Оператори - втор дел		lecture
14.	Условно извршување (if-else, switch)		lecture

Fig. 2. Screenshot of the MENDO training system – training lessons section.

the submission is added to a queue and judged as early as possible (the time slot is no longer than 1 minute, even during heavy-load competitions).

After a submission has been judged, the results of every test case are shown to the user in a form called detailed feedback. There is no limit to the number of submissions a user can make during a time period, but the system does support a couple of defensive mechanisms to prevent Denial of Service attacks.

MENDO also offers a special section consisting of organized materials that present an online step by step introduction to algorithmic problem solving and programming (with C++ as the programming language). This feature offers a number of lessons, combined with executable sample codes and proposed tasks connected to the information presented in the lesson in question (Fig. 2). Every user has a personal view of the lessons, showing her current progress.

4.4. *Specific MENDO Features*

As a system designed to support learning, MENDO has a few unique features that help students prepare for national and international competitions in informatics. We will present a couple of them: the ability to detect simple mistakes, downloading test cases and virtual contests.

Simple analysis of the source code – When a student submits a solution to a task (for example, a task from a previous national competition – available in training), the system automatically analyses the program output and the source code of the uploaded solution. If a simple mistake of not following specific rules is detected (printing additional data, using commands like system("pause"), writing to files instead of standard input/output, etc.), the system notifies the user that he has made a mistake that can be easily fixed and it provides to the user details about the mistake.

Downloading a test case in training mode – After a student submits a wrong solution to a task, he is presented with an option of potentially downloading one of the test cases that his solution does not correctly solve. Users are limited in the number of tests that they can download in a certain time period. We have found this to be a very positive change in the system and a lot of users are taking advantage of this feature to fix mistakes in their code instead of just giving up on solving a problem.

Virtual contest – The third option that we believe is very important (as a personalization feature) is the possibility to create virtual contests. Virtual contest is contest that is created on a user request and is based on the user's performance history in training section of the system (solving various tasks) and previous competitions. It is actually a simulation of a contest, seen only by the user that demanded it.

In the beginning, the system automatically determines the "best" tasks that should be included in the contest, from the set of tasks existing on the system. After the contest (the time period given to the contestant to solve the proposed tasks) the system automatically presents to the user a virtual scoreboard. The scoreboard consist names of other contestants, and the most important information on it is the ranking of the contestant that took the contest. The scoreboard is fictional but it is based on various system information from other actual competitions (the ones where the chosen tasks were included), and also the solvability of the tasks in the training section, etc.

Additional test case options – We have considered even more ideas with test cases. Some of them, explained in Jovanov (2011) are:

- ***Including hint(s) for a test case.*** This requires the problem maker to provide some kind of hint (or hints) for every test case for the program. This slows the process of task preparation, because it requires an additional effort. Nevertheless, when producing test cases, the problem maker almost always has in mind what part of the task (i.e., of the solution) is checked by each particular test case. The additional effort will be to put that in form of a hint, for the contestants that will use the task in the training section.
- ***Hint for a test case by a competitor*** – Following the previous idea and putting in force the full potential of the human – computer collaboration, the next step is to allow a user who has solved some particular test case of the problem, to provide a hint for that test case. There is more work to be done on determining (grading) the quality of the hint, depending on the user that provides it.
- ***Complete test case by a competitor*** – A competitor that has solved the problem could be allowed to provide a test case, that she believes is essential when grading the solutions. In this way, the system will gain better set of test cases for testing the subsequent submissions.

One other feature that is worth mentioning here is the ability of the system to produce reports and statistics. After the grading of each competition, the results are automatically published (this can be disabled, if necessary), and a report is generated. This report contains details and statistics for every competitor, every programming language, every task and even every test case that is part of that competition. The numbers can be presented in different formats.

5. MENDO's Employment in JBOI 2012

As a contest management system, we have used MENDO in the 2010, 2011, 2012 and 2013 contest cycles (more than 35 competitions, including both online and onsite rounds), and the system has proved to be quite stable, fast and robust. MENDO has also been used as the official contest system at the XVIII Balkan Olympiad in Informatics held in Petrovac Montenegro (2010), and at the VI Junior Balkan Olympiad in Informatics held in Ohrid, Macedonia (2012).

The members of the Scientific Committee of JBOI 2012 were familiarized with all the features that MENDO offers, and having them in mind they prepared valuable sets of tasks for each of the competition days (For example, the sets included tasks with detailed feedback). These sets produced very good distribution of scores in both competition days and overall, as presented in Fig. 3.

Although the system used was not the only and main reason for the presented results, it certainly had a significant impact on them.

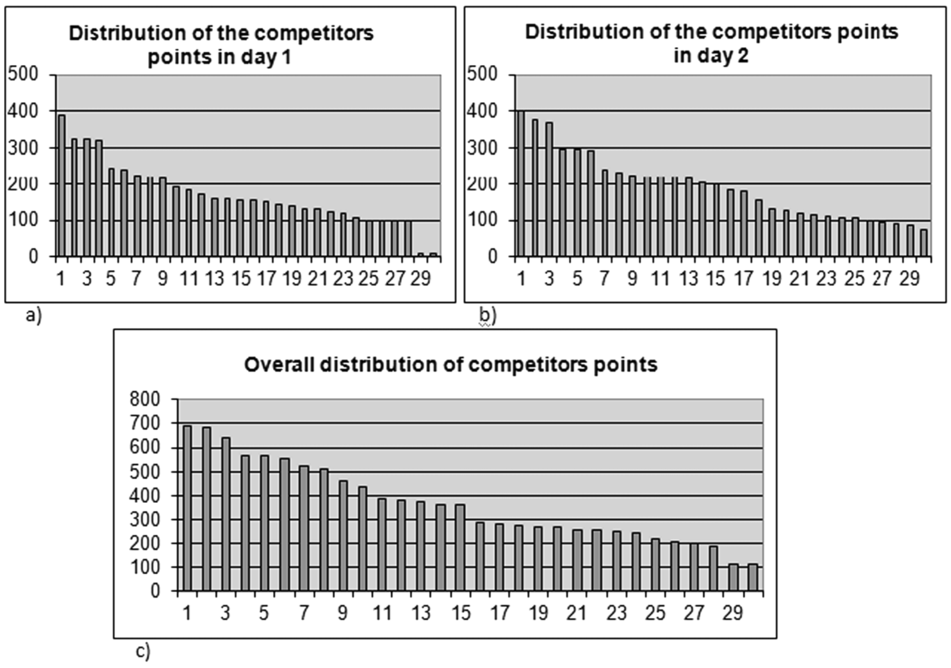


Fig. 3. Distribution of points scored by the competitors at the VI Junior Balkan Olympiad in Informatics 2012: (a) distribution in the first competition day (MIN = 10, MAX = 390, AVG = 166 out of 400 points), (b) distribution in the second competition day (MIN = 75, MAX = 400, AVG = 191,5 out of 400 points), (c) overall distribution (MIN = 115, MAX = 693, AVG = 357,6 out of 800 points).

6. Macedonian Competitions and Results of Macedonian Teams at International Contests since the Introduction of MENDO

We have introduced MENDO as a competition system in the 21st cycle of the Macedonian competitions in informatics (year 2010). Table 1 shows the number of participating contestants throughout the last five national contest cycles.

As can be seen from the first row of the table, the number of contestants in the Regional Competition (which is the most realistic indicator of the pupils' interest in the competitions since this is the starting point of the contest cycle where all the pupils that applied can participate), after the years of stagnation till 2010, grows rapidly from year to year. This means that one of the goals of CSM has been accomplished with the help of MENDO.

As we mentioned earlier, the best pupils at national competitions enter into the teams that represent Macedonia at IOI, BOI and JBOI. Macedonia has participated at almost each of the 24 IOI contests, 20 BOI contests, and 6 JBOI contests. The Macedonian pupils have won 19 medals in total at the 20 BOIs held so far. 9 of the medals have been won in the last 3 years. Moreover, pupils have won 10 medals in total at the 24 IOIs held so far, with 5 of them in the last 3 years. At the 6 JBOIs held till now, we have won 10 medals, 9 of which have been in the last 3 years. In Fig. 4 we can see the progress at the international competitions since the introduction of MENDO.

Starting with just 1 medal (in total at the 3 contests), we have reached a number of 9 medals in the last year (2012). Based on the opinions of the members of the organiza-

Table 1
Interest in the competitions in informatics in Macedonia
(expressed through the number of participating contestants)

Competition	Year				
	2009	2010	2011	2012	2013
Regional competition	51	55	118	209	290
State competition	44	45	68	95	118
Macedonian National Olympiad	22	23	19	21	21

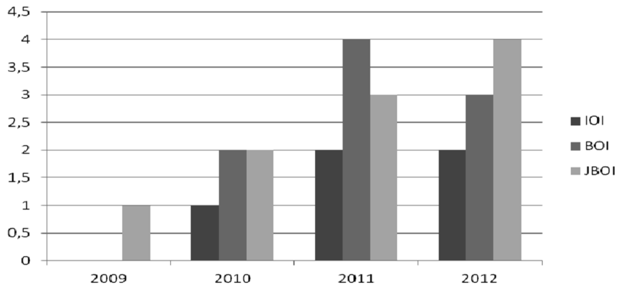


Fig. 4. Number of medals won at IOI, BOI and JBOI in the last 4 years.

tional and scientific committees, authors of the paper strongly believe that the MENDO system has played a major role in the presented progress.

7. Conclusion

In this paper we presented the format of the informatics competitions in Macedonia, the challenges of the organization and MENDO – the supporting system of the Macedonian competitions in informatics, with all the improvements during the last three years of its use. In the last sections we presented some results from a Balkan Olympiad in Informatics that used the MENDO system, as well as the results from its use in Macedonian competitions. The results are in favor of the system, in sense of the improvement of the quality of the contests through the number of additional features offered. Most importantly, MENDO plays an important role in the impressive improvement of the results of the Macedonian teams at international contests, as presented.

Using this system, or including the presented features in some similar system, can improve the quality of the contests and the whole process of training and elections of students for international competitions in most of the countries.

We constantly make enhancement to the system, and experiment with new ideas for improvements.

Acknowledgements. The research presented in this paper is partly supported by the Faculty of Computer Science and Engineering in Skopje.

References

XVIII Balkan Olympiad in Informatics Held in Petrovac. Montenegro (2010). `http://www.iccg.co.me/BOI/`

VI Junior Balkan Olympiad in Informatics Held in Ohrid. Macedonia (2012). `http://jboi.cs.org.mk`

Burton, B.A. (2008). Informatics olympiads: challenges in programming and algorithm design. In: Dobbie, G. and Mans, B. (Eds.) *Proceedings of Thirty-First Australasian Computer Science Conference (ACSC 2008)*, Wollongong, NSW, Australia. CRPIT, 74, 9–13.

Jovanov, M., Gushev, M. (2011). On-line feedback: a human-computer e-collaboration example. In: *Proceedings of the 8th Conference on Informatics and Information Technology, (CiiT 2011)*, Bitola, Macedonia, 187–189.

Jovanov, M., Stankov, E., Kostadinov, B., Ackovska, N. (2012). Progress of competitions in informatics: a success story. In: *Proceedings of the 9th Conference on Informatics and Information Technology (CiiT 2012)*, Bitola, Macedonia.

Kostadinov, B., Jovanov, M., Stankov, E. (2010). A new design of a system for contest management and grading in informatics competitions. In: *ICT Innovations Conference 2010, Web Proceedings*, 87–96.

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

Manev, K., Sredkov, M., Bogdanov, T. (2009). Grading systems for competitions in programming. *Thirty Eight Spring Conference of the Union of Bulgarian Mathematicians*.

M. Jovanov is a teaching and research assistant at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. He is the president of the Computer Society of Macedonia and has actively participated in the organization and realization of the Macedonian national competitions and olympiads in informatics since 2001. He has been a team leader for the Macedonian team since 2006. Currently, he is preparing his PhD thesis on the topic of online collaborative ontology building in e-learning environment.

B. Kostadinov is a postgraduate student and research associate at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. He is one of the organizers of the Macedonian national competitions in informatics. He has participated at IOI as a contestant and also as a team leader for the Macedonian team.

E. Stankov is a teaching and research assistant at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. He is a member of the Executive Board of the Computer Society of Macedonia and has actively participated in the organization and realization of the Macedonian national competitions and Olympiads in informatics since 2009. Currently he is finishing his master studies.

M. Mihova is an assistant professor at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. She is a member of the board of the Computer Society of Macedonia. Her research interest is in the field of applied mathematics, more specifically applied probability and statistics, with focus on mathematical models in reliability, especially reliability of multi-state systems.

M. Gusev is a professor at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. His research interest is in the field of parallel processing, computer networks, Internet technologies, e-business, mobile and wireless applications. He is an author and co-author of 18 books, 51 publications in reviewed international journals and 91 articles in reviewed international conference proceedings. He has participated in more than 86 conferences, workshops and seminars.

Olympiads in Informatics, 2013, Vol. 7, 55–60
© 2013 *Vilnius University*

Computer Maintenance via Batch Execution

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

Abstract. Organization of programming contests often involves maintenance of hundreds of computers. While parallel installation of operating systems is a well understood problem, further maintenance of installed systems is often cumbersome. We present a simple, yet powerful batch processing system, which makes this task easier.

Key words: cluster maintenance, batch processing.

1. Introduction

Large programming contests, like the International Olympiad in Informatics, ACM ICPC, or many regional competitions often involve hundreds of computers. Some machines are used directly by the contestants, while the others work behind the scenes as contest servers or worker nodes of a grading system.[1]

The whole infrastructure is usually set up for the duration of a single contest. Within a couple of days, the computers need to be installed and configured for their intended roles. During the contest, further administrative actions have to be performed: distribution of files related to contest tasks, fixing of minor bugs, pro-active checking for hardware problems, and so on.

Generally speaking, administration of the contest network is similar to that of a computer cluster. We will follow the terminology commonly used for clusters and call an individual computer a *node* of the system. Unlike a typical cluster, our nodes can act in different *roles,* so it will be useful to divide them to *groups* such that all nodes within a group are handled in the same way.

Parallel installation of software to multiple nodes, including the operating system, is a well understood task. The nodes can be booted via network (e.g., using PXELinux by Anvin *et al.* (2013)) and execute an automatic installation script. Then the software can be installed package by package, for example by a tool like FAI (Lange *et al.*, 2013). Alternatively, a full disk image can be copied. The latter approach is less flexible, but it

*Partially supported by the Center of Excellence – Institute for Theoretical Computer Science, Prague (project P202/12/G061 of the Grant Agency of Czech Republic).

[1]For a description of a typical contest system, please see Mareš (2009) or Maggiolo and Mascellani (2012).

can be significantly faster if a multicast-based or tree-based distribution protocol is used (see Knaff *et al.* (2012) or Mareš *et al.* (2009)).

Management of already installed nodes is a more complex problem. A part of its complexity lies in the ad-hoc nature of maintenance tasks. It is therefore hard to find a general framework where all such tasks fit. On the other hand, most tasks can be expressed as shell commands. There are multiple tools for running shell commands in parallel on a group of nodes. A typical example is ClusterSSH by Fergusson (2010): it opens one terminal window per node, the user then types commands in a master window and the keystrokes are broadcast to the other windows.

ClusterSSH is easy to use, but it has several drawbacks. First, it is hard to express tasks, which need to be applied conditionally, depending on the state of the node. More importantly, it fails when a node is inaccessible due to a temporary hardware or software problem. With an increasing number of nodes, the probability that there is a faulty node converges to 1. We need to save the failed commands and repeat them later, when the node recovers.

These problems have led to development of complex configuration engines – most notably GNU cfengine (Burgess, 1995) and Puppet (Puppet Labs, 2013). They are given a declarative description of the intended state of the cluster (which packages should be installed where, which user account should exist, etc.). The engine then periodically checks if the nodes conform to the description and attempts to fix any discrepancies.

This approach is robust with respect to software and hardware failures. It also makes repeated administrative tasks (e.g., installation of software packages) easy, as long as a parametrizable template can be written. On the other hand, one-of-a-kind tasks take too much effort to declare and the complexity of the engine makes failures unnecessarily hard to diagnose.

We have designed and implemented a new cluster management tool, built upon queues of batch jobs. It retains the straightforward simplicity of shell-based tools, but hopefully eliminates most of their drawbacks.

In the following sections, we describe the data model of our system. Then we discuss operations defined on the model and their reference implementation.

2. Data Model

The data model used by our system consists of multiple instances of two simple building blocks: jobs and queues.

2.1. *Jobs*

A *job* is an abstraction of an action, which can be executed on a node. Each job has a header and a body.

The *body* of the job is a script to be executed – by default, it is a shell script, but other interpreters, like Perl or Python, can be used.

The *header* is a collection of *job attributes,* written as arbitrary key/value pairs. The attributes can influence handling of the job. The basic attributes include:

- *ID* – a unique alphanumeric identifier of the job. If unspecified, the identifier is generated automatically, but jobs can be also given meaningful names.
- *Subject* – a one-line textual description of the job, intended to make lists of jobs comprehensible.

Execution of a job on a node consists of three phases:

- *Preparatory phase:* the body of the job is transferred to the node, possibly wrapped in a common *prolog* and *epilog* code (outside obvious use for initialization and cleanup, the prolog can also serve as a library of functions available to all jobs).
- *Running phase:* The script is run on the node, either within a terminal or non-interactively, depending on system configuration.
- *Cleanup phase:* The files related to the job are removed from the node.

Additionally, each job can request special handling in the preparation and cleanup phases. The attributes of the job can include shell commands to be run in these phases. The job can also specify its *attachments* – extra files uploaded to the node in the preparatory phase, available to the job during its execution in its current working directory, and removed in the cleanup phase.

Let us consider possible failure modes. A job can fail in the preparatory phase, in which case we just abort its execution (and optionally run the cleanup phase to remove all traces of the job). If a cleanup fails, the remnants of the job have to be removed manually (or by retrying the cleanup later). When running the job fails, we cannot be sure which part of the job was completed; it is even possible that the whole job was run, but the connection broke before the information about completion was transmitted. In all such cases, the execution of the job can be retried later. Because of that, all jobs are generally expected to be idempotent.

2.2. *Queues*

Scheduling of jobs on nodes is represented by *queues.* Each queue keeps its own set of jobs and known nodes. For each node, some of the jobs can be enqueued (scheduled for execution).

The jobs can be assigned to nodes individually, but it is more common to enqueue a job on a *group* of nodes. Groups are defined in the configuration of the system; each group contains a set of nodes and possibly includes other groups. When a job is enqueued on a group, the group is immediately expanded to its constituent nodes.

Later, a queued job is executed. The result of the execution is recorded in a *status file.* If the execution was successful, the job and its status file are moved to an archive of historic jobs. Otherwise, the job is kept in the queue and its status reported.

For each host, the queued jobs are naturally ordered by their identifiers and usually executed in this order. As the automatically generated IDs are based on timestamps, the jobs are by default executed from the oldest to the newest. However, applications requiring specific order can issue specific IDs.

Generally, all jobs in the queue can be executed simultaneously, but this is seldom welcome. The queue therefore specifies a *locking model,* which restricts the degree of concurrency. The default model forbids multiple parallel jobs on the same node, but parallelism between nodes is allowed.

3. Implementation

We have written a reference implementation of our queueing system. It was developed on Linux and it should run on all POSIX-compliant operating systems. It is available from `http://mj.ucw.cz/sw/bex/` and it can be distributed under the terms of the GNU General Public License.

The implementation is written in the Perl language in a spirit similar to the Git revision control system (Hamano and Torvalds, 2013). It consists of a core library, a driver program `bex`, which parses arguments and passes control to different subcommands, each handling one type of operations.

The whole program runs on one node, designated as the master of the cluster. It communicates with the other nodes via the Secure Shell protocol (SSH), leaving all the intricacies of authentication and encryption to it. No special software is required on the other nodes, except for basic system tools.

The following basic operations are supported:

- `bex qman` – manage queues: list available queues, or add a new one.
- `bex add` – add a new job and enqueue it on a set of nodes and/or groups. The job can be created either interactively in a text editor, or completely specified by command-line options. It is also possible to take a job which was already completed and re-queue it for repeated execution.
- `bex queue` – inspect a queue. Shows jobs in the queue, together with their status. The listing can be filtered to show only specific jobs, specific nodes or for example jobs whose execution failed. Additionally, jobs can be removed or moved to a different queue.
- `bex run` – run jobs sequentially. Takes a queue and runs all queued jobs one after another. Again, the set of jobs can be limited to a subset of nodes, or a specific job. Job locking rules are respected, so it is safe to run multiple instances of `bex run` simultaneously.
- `bex prun` – run jobs in parallel. Takes a queue and runs all queued jobs, as many at a time as allowed by the configuration. Locking rules are again respected (in fact, `bex run` is run internally to handle the low-level details). Current status of all nodes and jobs is shown on a status screen.

All jobs are allowed to be interactive, i.e., require a controlling terminal. When we run jobs sequentially, they inherit the terminal from which `bex run` was started, tunnelled via SSH. Parallel execution employs a terminal multiplexer to create a virtual terminal for every running job. An obvious choice is GNU Screen (Chowdhury *et al.*, 2013), but we were unable to create new virtual terminals on the background, which made interaction with the parallel execution almost impossible. We therefore use `tmux` (Marriott *et al.*, 2013) instead, whose design is less convoluted and which has much better remote control capabilities.

4. Conclusion

We have designed and implemented a new cluster management tool, based on parallel batch scheduling. The tool is very simple, but several years of experience with its use have confirmed that it is very effective. We have been using it to manage a network of about 60 computers at our department and also to manage contest systems at several programming competitions we organize.

The implementation should scale well to hundreds of nodes. The only problem we are aware of is the layout of the status screen in `bex prun`, which may become hard to read; this however does not impact running of the jobs and furthermore it should be easy to fix.

With an increasing number of nodes, SSH connections could become a bottleneck. Experimental results show that this is not an issue as long as the jobs are reasonably small. If huge attachments are needed, the performance can be limited not only by SSH, but also by the bandwidth of the network interface of the master node. Such cases can be worked around by using a clustered filesystem to distribute the files. A more systematic fix, which should not be hard to implement, could be to allow multiple instances of `bex prun` running on different master nodes and sharing a queue.

In the future, it could be useful to replace ID-based ordering of jobs by a full system of dependencies between jobs. This would allow parallel execution of independent jobs on one node, or operations like "requeue a job together with all dependent jobs". It would also lead to better handling of errors.

References

Anvin, H.P. *et al.* (2013). The Syslinux Project. `http://www.syslinux.org/`

Burgess, M. (1995). A site configuration engine. *USENIX Computing Systems*, 8(3). Available from `http://www.iu.hio.no/cfengine/`

Chowdhury, S.H. *et al.* (2013). *GNU Screen*. `http://www.gnu.org/software/screen/`

Ferguson, D. (2010). *Cluster SSH*. `http://clusterssh.sourceforge.net/`

Hamano, J., Torvalds, L. *et al.* (2013). *Git Revision Control System*. `http://www.git-scm.com/`

Knaff, A. *et al.* (2012). *Udpcast*. `http://www.udpcast.linux.lu/`

Lange, T. *et al.* (2013). *FAI – Fully Automatic Installation*. `http://fai-project.org/`

Maggiolo, S., Mascellani, G. (2012). Introducing cms: a contest management system. *Olympiads in Informatics*, 6, 86–99.

Mareš, M. (2009). Moe – design of a modular grading system. *Olympiads in Informatics*, 3, 60–66.

Mareš, M. *et al.* (2009). *The Shcp Tool, Part of the Sherlock Holmes Search Engine*. `http://www.ucw.cz/holmes/`

Marriott, N. *et al.* (2013). *TMUX*. `http://tmux.sourceforge.net/`

Puppet Labs (2013). *Puppet Open Source*. `https://puppetlabs.com/puppet/puppet-open-source/`

M. Mareš is an assistant professor at the Department of Applied Mathematics of Faculty of Mathematics and Physics of the Charles University in Prague, organizer of several Czech programming contests, member of the IOI Scientific Committee, and a Linux hacker.

Olympiads in Informatics, 2013, Vol. 7, 61–77
© 2013 *Vilnius University*

Algorithmic Results on a Novel Computational Problem

Minko MARKOV[1], Krassimir MANEV[2]
[1]*Department of Computing Systems, Faculty of Mathematics and Informatics*
"St. Kliment Ohridski" University of Sofia
5 J. Bourchier Blvd, P.O. Box 48, BG-1164 Sofia, Bulgaria
[2]*Department of Computer Science, New Bulgarian University*
21 Montevideo St., 1618 Sofia, Bulgaria
e-mail: minkom@fmi.uni-sofia.bg, kmanev@nbu.bg

Abstract. We propose a novel computational problem on undirected graphs, prove that it is $\mathcal{NP}$-complete, and design a linear-time algorithm for the special case when the graph is a cactus. The decision version of the problem is, given a graph and a train in it, the train being an edge sequence that forms a path with one vertex designated as the locomotive, can the train reach a certain target vertex, or not? The movement of the train consists of elementary, single-edge moves with the locomotive forward, the length of the train staying the same. Further, the movement of the train is restricted: the train cannot self-intersect.
Key words: train in a graph, $\mathcal{NP}$-completeness, algorithmic graph theory, cactus graphs.

1. Introduction

Pankov (2008) proposes the following problem, having already mentioned the concept of a train in a graph:

> **Task 2.** A graph is given. Firstly, the head H and the tail T of the train are in two neighbor vertices. Write a program finding one of the shortest ways to be passed by the train (moving forward only) in order to put its head to the primary position of T and its tail to the primary position of H.

Similar problems called PATH TRANSFERABILITY and PATH REVERSIBILITY are introduced and investigated by Torii (2008) though the results are not algorithmic. A vaguely similar concept under the name *snake* of length two or three is used for proving results on symmetric graphs by Tutte (1998).

We consider a modification of that problem, which is explained informally as follows. Imagine a network of train rails of unit length. The network can have arbitrary topology – it does not have to be planar and an arbitrary number of rails can meet at any junction. There is a train in the network, its locomotive being a point and its carriages being of unit length, each carriage being situated on a single rail. One endpoint of the train is the locomotive and the train can only move with the locomotive forward; unlike real trains, it cannot move backwards. Initially, the locomotive is situated over some junction x of the network. The train travels by discrete moves with the locomotive forward. Each elementary move has unit length, therefore after it the locomotive arrives at some junction y such that x and y are adjacent in the network, i.e., they are at the endpoints of some rail. Junction y must have been unoccupied by the train before the move commences. The length of the train does not change during its travel, so as the locomotive arrives at y, some junction previously occupied by the train becomes free. Given the network and some initial position of the train and some target junction, the question is, can the locomotive reach the target. The corresponding optimization problem is, what is the minimum number of elementary moves in order to get the locomotive to the target.

In the degenerate case when the train has zero carriages and consists of a single point – namely, the locomotive – the problem is equivalent to the SHORTEST PATH problem on unweighted undirected graphs. However, in general that is not the case. Without loss of generality we can assume the corresponding graph is connected and thus the only reason the locomotive cannot start moving towards the target on some shortest path is that the body of the train gets into its way. Superficially, that problem is a minimization one. However, if the body of the train blocks every path between the locomotive and the target, we must find a long enough path inside the network so that the whole train can be "stored" in it, thus freeing a path for the locomotive to go to the target. It is well-known the LONGEST PATH problem is $\mathcal{NP}$-complete on general graphs (Garey and Johnson, 1979) and that is a strong indication that our problem is intractable, too. Indeed, we prove its $\mathcal{NP}$-completeness by a reduction from HAMILTON PATH.

However, on some restricted graph classes the problem is solvable in linear time. There is an obvious trivial algorithm for trees. We propose a linear-time algorithm for the said problem on cactus graphs.

2. Background

2.1. *Graphs*

We consider undirected graphs without multiple edges or self loops. Let $G = (V, E)$ be a graph. To *delete* a vertex u from G means to transform G into $G' = (V \setminus u, E \setminus \{e \in E \mid u \in e\})$. We write $G' = G - u$. If the vertex set of a graph G is not named explicitly we denote it by $V(G)$. Likewise, $E(G)$ is the edge set. To *remove* an edge (u, v) from G means to transform G into $G' = (V, E \setminus \{(u, v)\})$. The result of the edge removal operation is denoted by $G - e$. For any vertex $u \in V(G)$, by $N(u)$ we denote the set of all vertices adjacent to u.

A path in G is a sequence $p = u_0, e_0, u_1, e_1 \ldots e_{n-1}, u_n$, for some $n \geqslant 1$, of alternating vertices $u_0, u_1 \ldots u_n$ and edges $e_0, e_1 \ldots e_{n-1}$ such that for $0 \leqslant i < n$, $e_i = (u_i, u_{i+1})$. u_0 and u_n are called *the endpoints of* p, and the remaining vertices are *the internal vertices of p. The length of* p, denoted by $|p|$, is n. The set of the vertices of p is denoted by $V(p)$.

If all vertices in a path are distinct, the path is *simple*. When we say *general path* we mean a path that is not necessarily simple. If p is a path such that the only repeating elements are $u_0 = u_n$ and $n \geqslant 4$, we say p is a *cycle*. *The length* of a path or a cycle z is the number of edges in it and is denoted by $|z|$. For any two vertices $u, v \in V(G)$, *the distance between* u *and* v, denoted by $\mathrm{dist}(u, v)$ is the length of a shortest path with endpoints u and v in G. If there is no path between u and v then we define that $\mathrm{dist}(u, v) = \infty$. For any vertex u and cycle s, the distance between u and s is the length of a shortest path connecting u to a vertex from s; we denote that by $\mathrm{dist}(u, s)$.

When we consider a path or cycle we actually think of the subgraph that has those vertices and edges, rather than some concrete description such as $u_1, e_1, u_2, e_2 \dots e_{n-1}, u_n$. A path and its description are conceptually different objects, and likewise with cycles. Any simple path of length $\geqslant 1$ has two different descriptions and any cycle of $n \geqslant 3$ vertices has $2n$ different descriptions. Let us think of cycles as subgraphs. *An arc* in a cycle s is any path p such that the vertices and edges of p form a contiguous sequence in some description of s.

Since we do not consider multi-graphs, we define paths and cycles by listing only the vertices and not the edges.

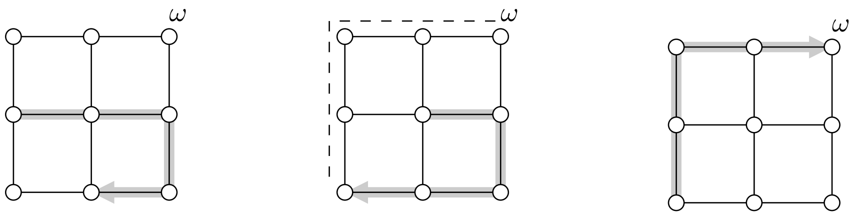
2.2. *Trains in Graphs*

A train in a graph G is a simple path $p = x_0, x_1 \dots x_q$ in G. There is a direction associated with the train: the leftmost vertex x_0 in its description is considered its first vertex and we call it, *the locomotive*. The vertices of G are partitioned into *occupied* and *free*, the former ones being precisely the vertices that belong to the train. An elementary move of the train is transforming it into another path $p' = x', x_0, x_1 \dots x_{q-1}$ where x' is a vertex such that $x' \in N(x_0)$ and x' is free before the move. The move is considered instantaneous. After it, vertex x_q is free and x' is occupied. *The target* is some vertex $\omega \in V(G)$ that may or may not be free initially. There is precisely one train in any graph we consider.

So far we have defined "train" as a temporary object being identified with its current position – after any move, the train is a different object. However, it is helpful to think of the train as permanent object whose current position is just one of its attributes.

The following two definitions are relative to some concrete position $x_0 \dots x_q$ of the train. For any $\beta \in V(G)$, *a* β*-trajectory* is any general path $p = x_0, x_1 \dots \beta$ such that the locomotive can reach β by moving along p. For any free vertex v, *a* v*-free path* is any simple path p with one endpoint the locomotive and the other endpoint v such that all its vertices at that moment, except for the locomotive, are free. A zero-length path in which the locomotive coincides with v is considered v-free, too. When we say simply *a free path* we mean an ω-free path; of course, ω must be a free vertex, unless it coincides with the locomotive.

The goal is to compute whether a ω-trajectory exists, in the decision version of the problem, or to compute an ω-trajectory of minimum length, in the optimization version. It is important to realize that initially there can be no free path and yet there can exist an ω-trajectory. If a free path exists initially the answer to the decision problem is trivially



(a) The initial position. The locomotive is where the arrow tip is. All paths between the locomotive and ω are blocked by the train itself.

(b) After the first move there is a free path – see the dashed line.

(c) The goal is achieved. Now the locomotive is at ω, having moved along the free path from Figure 1(b).

Fig. 1. There is an ω-trajectory although initially the train blocks all its possible paths to ω.

YES. Figure 1(a) depicts a graph and a train in it. The train is drawn with a thick gray line and the locomotive is where the arrow tip is. Initially, there is no free path. However, the train can get into the position shown on Fig. 1(b). Now there is a free path outlined by a dashed line. Clearly, the locomotive can reach the target along the free path (Fig. 1(c)).

Note that if the length of the train is zero, i.e., if the locomotive consists of a single vertex-locomotive, the problem becomes the SHORTEST PATH problem, which is a very well-studied problem. Therefore, we assume the train is always longer than zero.

2.3. *Cactus Graphs*

A cactus graph, shortly cactus, is a connected graph G in which every edge is in at most one cycle. The edges of G are partitioned into *tree edges* and *cycle edges*, the former being the ones that are in zero cycles. For any cycle s in G, *the constituents of s* are the connected components of G left after the removal of all edges of s. For each vertex $u \in s$, the u-constituent of s is the constituent that u belongs to.

Let v be any vertex of G. Let $G_1, \ldots, G_t$ be the connected components of $G - v$. Let $v_1, \ldots, v_t$ be new vertices. We are going to use each of them as a replacement of v in precisely one of $G_1, \ldots, G_t$. For each G_i, $1 \leqslant i \leqslant t$, let G'_i be G_i plus vertex v_i plus the one or two edges that used to connect v to vertices from G_i; now these edges connect v_i to those one or two vertices. Finally, rename v_i to v in all G'_i. The renaming does not mean we identify all v_i vertices. It means we get t connected graphs, each having a vertex with name v. We say that $G'_1, \ldots, G'_t$ are *the fragments of G relative to v*.

3. Intractability Results

Let us recall the formal definitions of two computational problems first: the newly introduced TRAIN IN A GRAPH in its decision version and a version of the the classical HAMILTONIAN PATH (Garey and Johnson, 1979, pp. 60).

Computational Problem TRAIN IN A GRAPH
Generic Instance: Undirected graph G, a train in it, a target vertex ω
Question: Is there an ω-trajectory in G? □

Computational Problem HAMILTONIAN PATH BETWEEN TWO POINTS
Generic Instance: Undirected graph G, vertices u and v in G
Question: Is there a Hamiltonian Path in G with endpoints u and v? □

For brevity we call the latter problem simply HAMILTONIAN PATH. It is $\mathcal{NP}$-complete just as the common version of the problem (Garey and Johnson, 1979, pp. 60).

Theorem 1. TRAIN IN A GRAPH *is $\mathcal{NP}$-hard.*

Proof. The notation "$\propto$" stands for "Karp-reduces to". We prove that HAMILTON PATH $\propto$ TRAIN IN A GRAPH. Assume G, u, v is an instance of HAMILTON PATH. Let $|V(G)|$ be n. Let $Z = \{x, z_0, z_1 \dots z_{n-1}, \omega\}$ be a vertex set such that $Z \cap V(G) = \emptyset$. Let $G' = (V \cup Z, E')$, where $E' = E \cup \{(z_0, z_1), (z_1, z_2) \dots (z_{n-3}, z_{n-2}), (z_{n-2}, v), (v, x), (x, u), (v, \omega)\}$. Let $x, v, z_{n-2}, z_{n-3} \dots z_1, z_0$ be a train in G' with x as the locomotive. The length of the train is $n - 1$. Let ω be the target in G'. We constructed an instance of TRAIN IN A GRAPH. Figure 2 illustrates our construction. The original graph G is drawn with solid lines while the added vertices are black and the added edges are drawn with dotted lines.

We claim there is a Hamilton path with endpoints u and v in G iff there is an ω-trajectory in G'. Assume there is a Hamilton path p in G with endpoints u and v. As $|V(G)|$ is n, $|p| = n - 1$. Of course, initially there is no free path in G' but let the train move along p until the locomotive is at some vertex $w \in N(v)$. Since the train's length

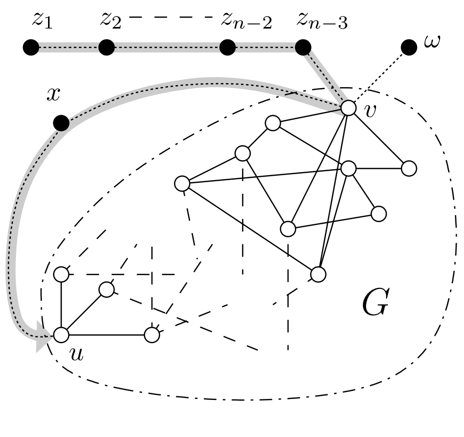


Fig. 2. The construction for the Karp reduction in Theorem 1. The original graph is G and the vertices u and v are in it. To G we add the vertices z_0, ..., z_{n-2}, x, and ω, plus the dotted edges, obtaining G'. In G' we construct the train indicated by the thick gray line.

is $n-1$, it is obvious at that moment the other endpoint of the train is at x and thus v is a free vertex. With two elementary moves the locomotive reaches ω.

Now assume there is a way for the train to reach ω, starting at the described initial position. Clearly, the ultimate move for the locomotive is from v to ω and the penultimate one, from some vertex $w' \in N(v)$ to v. In order the train to make that penultimate move, it must be the case that v is free. However, in order v to be free at that moment, at the initial moment there has to be a simple path p' of length $\geqslant n-2$ in G; furthermore, v cannot be in p'. Obviously p' is a Hamilton path between u and some vertex $w' \in N(v)$ in $G-v$. It follows immediately that p', v is a Hamilton path in G. □

Theorem 2. TRAIN IN A GRAPH *is in* $\mathcal{NP}$.

Proof. Unlike plenty of other $\mathcal{NP}$-completeness proofs, the fact that this problem is in $\mathcal{NP}$ is not obvious. A natural choice of certificate is an ω-trajectory. However, the trajectory is a general path so it is not immediately obvious its length is polynomial in the input size.

We prove that for every YES-instance of TRAIN IN A GRAPH, there exists an ω-trajectory such that no vertex appears more than twice in it. Assume we are given an YES-instance of TRAIN IN A GRAPH, the graph being called G and the train, $x_0, x_2, \ldots, x_q$ with the locomotive at x_0. Let $X = \{x_0, x_1 \ldots x_q\}$. If there exists a free path initially, the claim is obviously true. Assume there is no free path initially. However, since this is a YES-instance, there is an ω-trajectory p that the locomotive moves along. If all vertices of p are unique, we are done with the proof. Assume there are repeating vertices in p. Think of p as a string of vertex names with x_0 at the left end. Let the leftmost repeating vertex in that string be α. That vertex α is the first vertex that the locomotive "sees" for a second time as it moves along p.

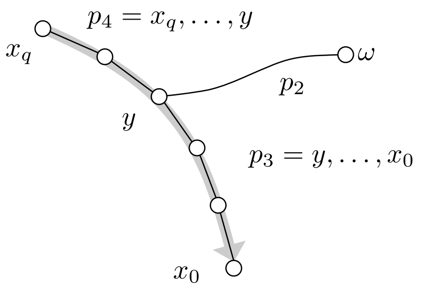
Now consider the moves of the train from its initial position until the locomotive hits α for the first time, moving along p. If at any moment during that sequence of moves there appears a free path, let the locomotive abandon the movement along p and instead follow the free path to ω. In this case there are no repeating vertices in the overall path the locomotive follows from its initial position to α.

Otherwise, consider the moment the locomotive gets to α for the first time. Clearly, the sub-path of p between the first and the second appearance of α inclusive forms a simple cycle s. Let the sub-path of p between x_0 and the first α exclusive be called p_1:

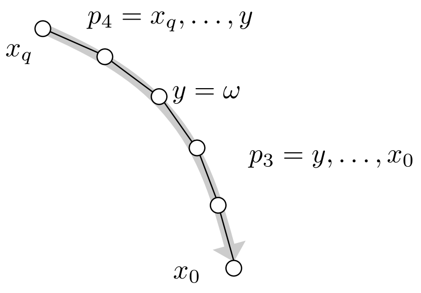
$$p = \underbrace{x_0 \ldots\ldots\ldots\ldots}_{p_1} \underbrace{\alpha \underbrace{\ldots\ldots\ldots\ldots}_{\text{no}\alpha\text{here}} \alpha}_{s} \ldots\ldots\ldots\ldots$$

Note that s and p_1 are vertex-disjoint because by construction α is the first repeated vertex during the movement of the train.

Relative to the initial moment, let y be the following vertex. If $\omega \notin X \setminus \{x_0\}$ then y is any vertex from X such that at the initial moment there is a path p_2 between ω and y that consists of free vertices, except for y. As G is connected, such an y exists. If



(a) ω is not train vertex. p_2 is a path connecting ω and the train.



(b) ω is a train vertex and p_2 is the single-vertex path ω.

Fig. 3. The two possible placements of ω and the initial train, the latter drawn with a thick gray line. p_3 is the subpath of the initial train between y and the locomotive. p_4 is the subpath of the initial train between y and x_q.

$\omega \in X \setminus \{x_0\}$ then $y = \omega$. Note that it makes no sense to consider the case $\omega = x_0$. The two said possibilities for y are shown on Fig. 3. Let p_3 be the subpath $y \ldots x_0$ of the the initial train. If ω is not a train vertex (Fig. 3(a)) then p_3 can be as small as the single vertex x_0. Otherwise (Fig. 3(b)), p_3 cannot coincide with x_0. Let p_4 is the subpath $x_q \ldots y$ of the initial train. Clearly, $V(p_3) \cup V(p_4) = X$.

To complete the proof, first assume $V(s) \cap (V(p_2) \cup X) = \emptyset$ (see Fig. 4(a)). Let the train move along p until the locomotive reaches α for the second time (Fig. 4(b)). At this moment, the path that consists of p_1, p_3, and p_2, in that order, is now free so the locomotive can reach ω along it.

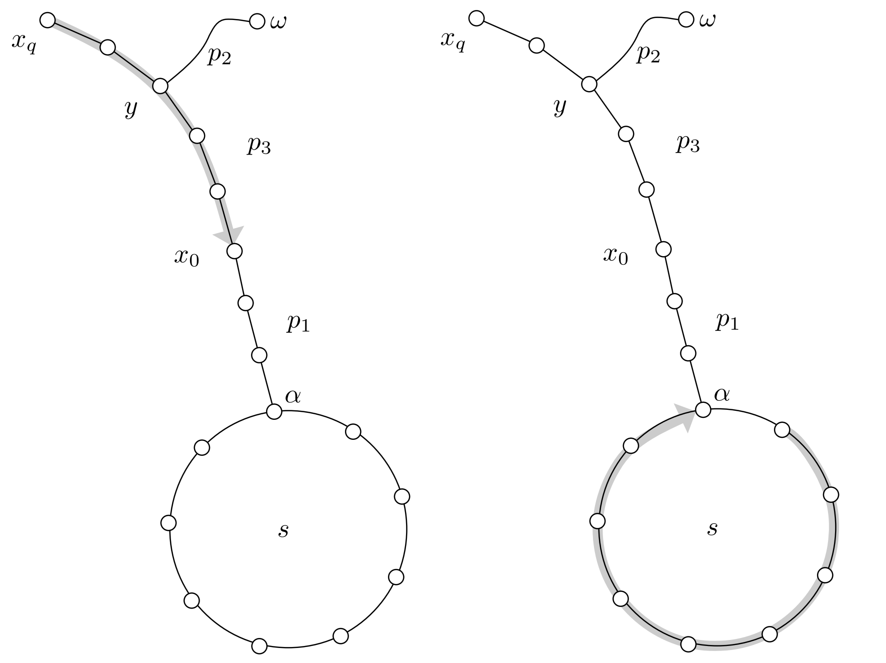
Now assume $V(s) \cap (V(p_2) \cup X) \neq \emptyset$. That overlap can be complicated as illustrated by Fig. 5(a). Whatever the overlap pattern is, clearly there exists a vertex $z \in V(s) \cap (V(p_2) \cup X)$ such that there are no vertices from s that are between z and ω *along the path union of* p_2*,* p_3 *and* p_4. On Fig. 5(a) the said overlap is only in p_3 and p_4 and vertex z is in p_4. The moment when the locomotive reaches z (see Fig. 5(b)) there is a free path. So the train abandons p and starts moving along that free path towards ω. □

COROLLARY 1. TRAIN IN A GRAPH is $\mathcal{NP}$-complete. □

4. A Linear-Time Algorithm on Cactus Graphs

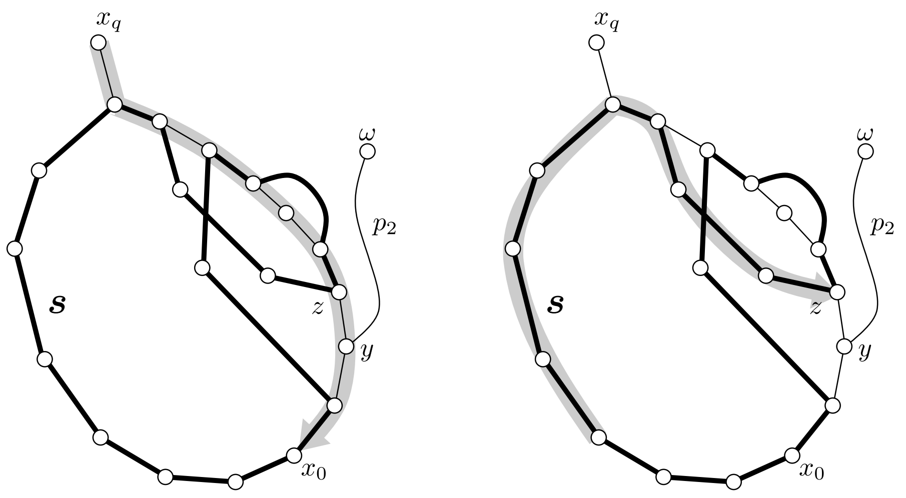
4.1. *The Precomputing and the Algorithm*

Although TRAIN IN A GRAPH is $\mathcal{NP}$-complete, it is still solvable by efficient algorithms on restricted graph classes. The solution on trees is trivial: root the tree at the vertex where initially the locomotive is and use any algorithm for tree traversal to find out if ω and the body of the train are in the same subtree relative to the root. If yes, the answer to the decision version is NO, else it is YES. In the latter case, the answer to the optimization problem is the length of the unique path between the root and ω.



(a) The initial position of the train. (b) The train has just made the U-turn.

Fig. 4. If s is disjoint with both p_2 and X, the train can use s to make a U-turn. After the U-turn there is an obvious free path.



(a) The initial position of the train.

(b) The locomotive, moving along s, reaches z. Now there is an obvious free path.

Fig. 5. s has common vertices with $V(p_2) \cup X$. s is drawn with thick black line.

Now assume the graph is a cactus graph. The cycles that are long enough so that the train can make a U-turn are of special importance. Informally speaking, the train can make a U-turn using cycle s iff it can enter s via some vertex $u \in s$, move only along s in either direction and exit the cycle via u. In order to accomplish that, the train's length must be smaller than $|s|$, otherwise u will not be free when the locomotive attempts to make the exit. It is clear that because of the nature of cacti, namely that cycles can have at most a vertex in common, any cycle is either long enough and the train can make the U-turn using it and no other cycles, or the cycle is useless with respect to making a U-turn. The cycle s cannot be of partial help for the U-turn: if the train exits s through a vertex that is not u, it has to leave s completely and make the U-turn in another cycle. It follows that those long enough cycles make the problem on cacti interesting. If there is none of them, the solution is completely analogous to the solution on trees.

The following algorithm solves the optimization version of TRAIN IN A GRAPH on cacti. Let $G = (V, E)$ be a cactus with vertex set $V = \{1, 2 \dots n\}$, a train $x_0, x_1 \dots x_q$, and target ω. *The long cycles* in the cactus are all cycles of length greater than q. Assume the locomotive is at x_0. Let PREPROCESSING be an algorithm that computes the following.

- For every edge $e \in E$ it computes whether e is a tree edge or a cycle edge.
- The array $D[1 \dots n]$ such that $\forall v \in V : \mathrm{dist}(x_0, v)$.
- In case that $q > 0$, a boolean value S set to FALSE iff ω, on the one hand, and $x_1 \dots x_q$, on the other hand, are in the same fragment of G relative to x_0.
- In case that $q > 0$ and (x_0, x_1) is a tree edge, assuming G' is the connected component of $G - (x_0, x_1)$ that contains x_0, the value

  $$\delta = \min \{2 \times \mathrm{dist}(x_0, c) + |c| \mid c \text{ is a long cycle in } G'\}.$$

  If there are no long cycles in G', $\delta = \infty$.
- In case that $q > 0$ and (x_0, x_1) is a cycle edge,
  - the cycle s that (x_0, x_1) is in,
  - $|s|$,
  - the function $\psi(\,)$

    $$\psi(u) = \min \{2 \times \mathrm{dist}(u, c) + |c| \mid c \text{ is a long cycle in } H_u\}$$

    for every vertex $u \in s$. H_u is the u-constituent of s. If there are no long cycles in some H_u then $\psi(u) = \infty$.
  - The set U of vertices from s that are not train vertices, plus x_0.
  - The vertex α from s such that ω is in the α-constituent of s. A boolean variable T is set to TRUE iff α is a train vertex.
  - If T is TRUE, U' is the set of vertices $u \in U$ such that the arc of s with endpoints α and u containing x_0 has length $\leqslant \lceil \frac{s}{2} \rceil - q - 1$.
  - If T is FALSE, U'' is the set of vertices $u \in U$ such that the arc of s with endpoints x_0 and u avoiding x_1 has length $\leqslant \lceil \frac{s}{2} \rceil - q - 1$.

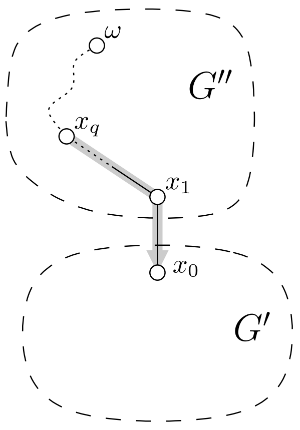
- The length m of the arc of s with endpoints x_0 and α that does not contain x_1.
- $\forall u \in U$, the value $\mathrm{arc}'(u)$: the length of the arc of s with endpoints x_0 and u that avoids vertex x_1.
- $\forall u \in U$, the value $\mathrm{arc}(u)$: the minimum of the lengths of the two arcs of s with endpoints u and α.

Both U' and U'' are defined by differences. If the respective quantity is negative then the set is empty. It is easy to see that PREPROCESSING can be implemented by a modified DFS.

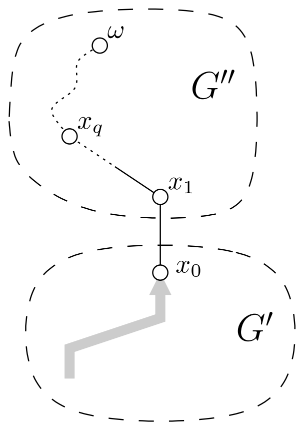
```
TRAIN IN A CACTUS $G$: cactus; $x_1 \dots x_q$: train in $G$; $\omega$: the target
1.  PREPROCESSING$(G, (x_0, x_1 \dots x_q), \omega)$
2.  if $x_0 = \omega$
3.      return $0$
4.  if $q = 0$ or $(q > 0$ and $S)$
5.      return $D[\omega]$
6.  if $(x_0, x_1)$ is tree edge
7.      return $\delta + D[\omega]$
8.  if $T$
9.      if $q \geqslant |s|$
10.         $\mathrm{tmp} \leftarrow \min\{\mathrm{arc}'(u) + \psi(u) + \mathrm{arc}(u) \mid u \in U\}$
11.     else
12.         $\mathrm{tmp} \leftarrow \min\{\mathrm{arc}'(u) + \psi(u) + \mathrm{arc}(u) \mid u \in U'\}$
13.         $\mathrm{tmp} \leftarrow \min\{\mathrm{m}, \mathrm{tmp}\}$
14. else
15.     $\mathrm{tmp} \leftarrow \min\{\mathrm{arc}'(u) + \psi(u) + \mathrm{arc}(u) \mid u \in U''\}$
16.     $\mathrm{tmp} \leftarrow \min\{\mathrm{m}, \mathrm{tmp}\}$
17. return $\mathrm{tmp} + \mathrm{dist}(\alpha, \omega)$
```

4.2. *Verification*

Assume that PREPROCESSING is correct. If the locomotive is at the target, the length of the shortest trajectory is obviously 0. Line 3 takes case of that case. If the train consists of a single vertex, namely the locomotive, the problem is the same as computing a shortest path in an undirected graph. By assumption, the array $D[1 \dots n]$ is stores the lengths of shortest paths in G with respect to x_0, so $D[\omega]$ is the answer (line 5). If the train is longer than a single vertex but the target is in some fragment H relative to x_0 such that x_1 is not in H (and consequently, $x_2, \dots, x_q$ are not in H), the answer is again $D[\omega]$. To see why, note that any shortest path between x_0 and ω lies solely in H, which follows from the fact that x_0 is a cut vertex; if x_0 was not a cut vertex there would be only one fragment relative to it. By assumption, S is set to TRUE by PREPROCESSING iff the target and x_1 are in different fragments. Therefore, the returned value $D[\omega]$ is the correct answer in this case, too.



(a) The initial train.



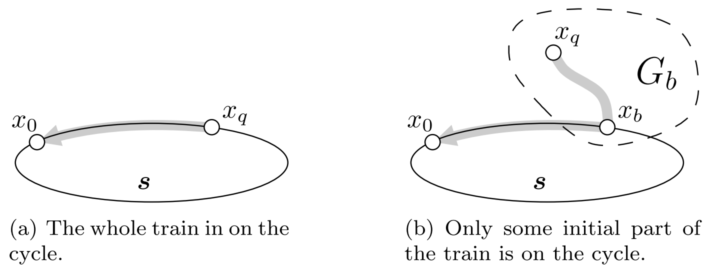
(b) If the train can make a U-turn in G', the locomotive can get back to x_0, this time the body of the train not blocking the paths to ω.

Fig. 6. (x_0, x_1) is a tree edge.

In the remainder of the proof the train is longer than a single vertex and x_1 and ω are in the same fragment relative to x_0. Suppose the train's first edge (x_0, x_1) is a tree edge (see Fig. 6(a)). Let G' be the connected component of G that remains after the removal of (x_0, x_1). In other words, G' consists of all fragments relative to x_0, except for the one containing x_1, being "glued together" at x_0. We argue if there are no long cycles in G' then the locomotive can never reach the target. Indeed, suppose there are no long cycles in G' and consider any possible sequence of moves. If the locomotive traverses a tree edge it only gets further from the target. If it moves along a cycle it still cannot make a U-turn and, if it exits the cycle, it exits via a vertex different from the vertex of entry, again finding itself even further from the target. As G is finite, at some moment either the locomotive will be stuck in a vertex of degree 1 or in a cycle vertex of degree 2 but being blocked by the body of the train. In this case the variable δ is set to ∞ by PREPROCESSING and therefore the returned value (line 7) is correct.

On the other hand, if there is at least one long cycle in G', the locomotive can reach the target. Because of the nature of cactus graphs, the locomotive has to get back to vertex x_0, with its body in G' (see Fig. 6(b)). Now it can reach ω via a free path in G''. The cost of the solution is the length of the trajectory used to reach a long cycle, make the U-turn and get back to x_0, plus the length of the chosen free path from x_0 to ω. A minimum is obtained when both the trajectory and the free path are minimum. The minimum length of a such a free path is the value stored in $D[\omega]$. The minimum length of such a trajectory is the value δ precomputed by PREPROCESSING. It follows the assignment at line 7 is correct in the current case, too.

It remains to consider the subcase when (x_0, x_1) is a cycle edge. Because of the nature of cacti that edge cannot belong to more than one cycle. We call the cycle s. It is obvious that the overlap between the train and s is a contiguous sequence of vertices from s. If G



(a) The whole train in on the cycle.

(b) Only some initial part of the train is on the cycle.

Fig. 7. (x_0, x_1) is a cycle edge.

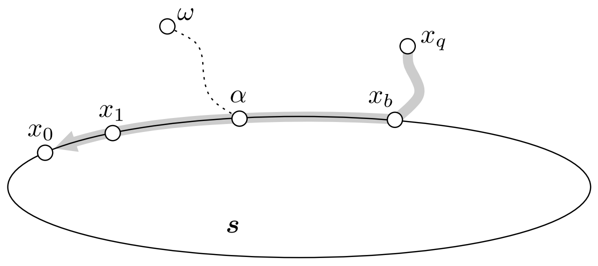


Fig. 8. α is a train vertex. The body of the train at the initial moment blocks all paths between x_0 and ω.

were a general graph that would not be necessarily true but it is true for cacti: either x_q is a vertex from s in which case the whole train lies over an arc of s (Fig. 7(a)), or there is some intermediate train vertex x_b, $2 \leqslant b \leqslant q-1$, such that all train vertices after x_b are in the x_b-constituent – call it G_b – of s (Fig. 7(b)). We distinguish the following two subcases. Call α the cycle vertex such that ω is in the α-constituent of s. Such an α exists because the graph is connected. So, we distinguish the subcase when α is a train vertex from the subcase when α is not a train vertex.

First assume α is a train vertex. Consider G (Fig. 8). If the train is longer than or equal to the length of the cycle, there is only one way to get the locomotive to ω: use a long cycle inside some u-constituent of s for a U-turn so that to bring the locomotive back to the cycle, namely at vertex u, with the body of the train being in the u-constituent. An additional limitation is that the cycle vertex u must not be one of $x_1, x_2, \ldots, x_b$. Figure 9 shows the train having performed the U-turn and gotten back to u. If such a u does not exist then the algorithm must return ∞. Now note that if the case is the current one, the variable T (line 8) has been set to TRUE on the assumption that PREPROCESSING is correct, and further the condition at line 9 is TRUE. So, the assignment at line 10 takes place: if such a u does not exist, the algorithm assigns ∞ to u and then returns ∞ at line 17. On the other hand, if such a u exist, it is a vertex from U (as defined in the description of PREPROCESSING). The overall trajectory consists four legs, as seen from the locomotive:

- From x_0 to u. There is only one choice for the direction along s. By assumption, $\mathrm{arc}'(u)$ stores the length of this leg.

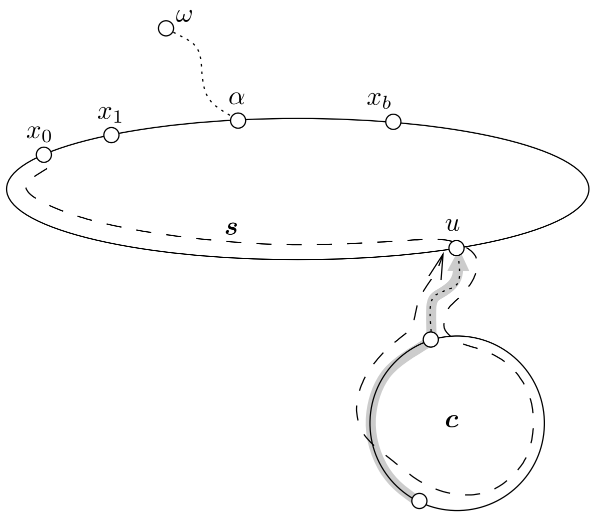


Fig. 9. The train is longer than the cycle but still there is an ω-trajectory. Using some cycle vertex u and a long cycle c in the u-constituent, the train performs a U-turn and the locomotive is again at u – see the dashed line. Now the body of the train does not block the paths between x_0 and ω.

- From u via c, making the U-turn, and back to u. By assumption, that is the value $\psi(u)$.
- From u to α. There are two choices for the direction along x, corresponding to the two arcs of s with endpoint u and α. By assumption, $\mathrm{arc}(u)$ stores the minimum of their lengths.
- From α to ω.

The last leg is independent of the choice of u and its contribution to the length of the overall trajectory is the $\mathrm{dist}(\alpha, \omega)$ term at line 17. The sum of the other three legs depends on the choice of u and therefore seeking the minimum at line 10 is the correct thing to do.

Now suppose that α is a train vertex and the train is shorter than the cycle. In this case the train does not have to perform a U-turn necessarily. If it just goes along the cycle there will be a free path at the moment the locomotive is in $N(x_b)$. It follows that now we always have a solution that is at least $m + \mathrm{dist}(\alpha, \omega)$ (recall the definition of m in the description of PREPROCESSING), as shown on Fig. 10. It is guaranteed that the moment before the locomotive gets at α, vertex α will be free.

However, that is not the only way to reach the goal under the current assumptions. Even if the cycle is long enough relative to the length of the train, it may still be possible to get the locomotive to α by performing a U-turn within some u-component of s (Fig. 11(a)) and going backwards along the cycle to x_1 and then to α (Fig. 11(b)). The overall trajectory may be shorter than the trajectory that the other way round s. Note that if using a U-turn is to be beneficial, after getting to u for the second time, the locomotive should go along s in the opposite direction to the one before – if it goes in the same direction as before, the U-turn maneuver is just a useless detour (Fig. 11(c)).

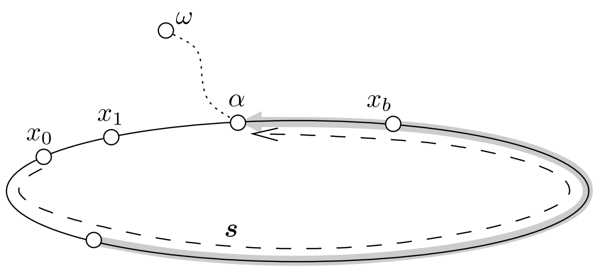
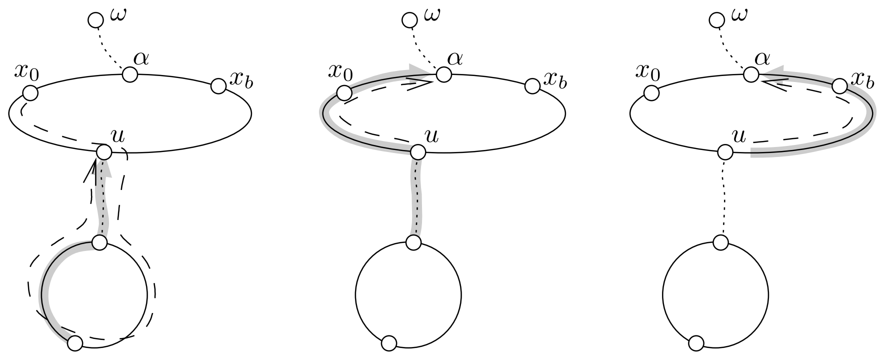


Fig. 10. The train is shorter than the cycle. One way to reach the goal is to slide the locomotive along the cycle until it reaches α – see the dashed line – and then perform the obvious thing.



(a) Having made the U-turn and gotten back to u.

(b) Going backwards to x_0 and then to α may yield a shorter way than the one shown on Figure 10.

(c) However, going forward from u to α is overall always a worse way than the one shown on Figure 10.

Fig. 11. The train is shorter than the cycle. Another way to reach the goal is to make a U-turn inside some u-constituent.

Now consider the possible locations of that vertex u in s. Let us fix a direction on s such that x_0, α, and x_1 appear in that order in that direction. On all our figures that direction is counter-clockwise. Let *the antipodal vertex of* α be the vertex in s that is at distance $\lceil \frac{s}{2} \rceil$ away from α in the said direction. Informally speaking, it makes no sense to choose vertex u past the antipodal vertex because if the locomotive gets to the antipodal vertex, the best thing to do is to let it slide in the same direction along the cycle until it hits α. Furthermore, we can improve the maximum distance from α that the potential vertex u can be. Since a U-turn inside the u-constituent costs at least $q + 1$ moves, the u vertex should not be further away from α (in the said direction) than $\lceil \frac{s}{2} \rceil - q - 1$. Indeed, the definition of U' in the description of PREPROCESSING provides that.

Note that the u vertex cannot possibly be past x_b (in the said direction). Assume the opposite: u is past x_b but before α as shown on the figure on the left. Let k, l and t be the distances shown there. Obviously, $k + l + t = |s|$. In order the U-turn in the u-constituent to make sense, it must be the case that $k + q + 1 + k + t < k + l$, because it has to

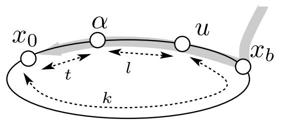


Fig. 12. Vertex u cannot be past vertex x_b in the counter-clockwise direction from x_0.

more beneficial the train to go to u in the said direction (cost k), make the U-turn (at least $q+1$), go back to x_1 in the opposite direction (cost k) and then go to α still in the opposite direction (cost t), rather than go in the said direction from x_1 to α (cost $k+l$). But

$$k+q+1+k+t < k+l \leftrightarrow k+q+1+t < l \leftrightarrow k+q+1+t+l < 2l$$
$$\leftrightarrow |s|+q+1 < 2l$$

and it is impossible to be the case that $|s|+q+1 < 2l$ because both $|s| > l$ and $q > l$; $q > l$ because $q > |s|$. It follows that u cannot be a vertex outside U, thus the definition of U' as a subset of U is correct.

It remains to consider only one more subcase: α is not a train vertex. In this case the relevant part of the algorithm is lines 15 and 16 because by assumption, the variable T is FALSE. The locomotive can always reach ω in this case because there is an obvious solution with cost $m + \operatorname{dist}(\alpha, \omega)$. Note that the assignments at lines 16 and 17 provide the output is at most $m + \operatorname{dist}(\alpha, \omega)$. That value correspond to a solution in which the locomotive slides directly from x_0 to α along s. However, there may be a better way to reach α. The locomotive can reach α from the other direction. To accomplish that, the train has to make a U-turn using some long cycle c inside some u-constituent of s first. That maneuver makes sense only if u is one of the vertices from U''; for any vertex from s past U'' (in the direction away from x_0, avoiding x_1) it is better not to try a U-turn even if there is a long cycle in the u-constituent. And indeed, the code at line 15 ties to find a suitable U-turn only in the constituents of vertices from U''. The addition $\operatorname{arc}'(u) + \psi(u) + \operatorname{arc}(u)$ precisely corresponds to going from x_0 to u (in the appropriate direction), making the U-turn and going back from u via x_0 to α. Because u is a vertex from U'', the shorter arc between u and α is the one containing x_1. Figure 13 illustrates the possibility we just discussed.

That concludes the proof of correctness.

4.3. *Time Complexity Analysis*

Let G be the cactus we run our algorithm on with $n = |V(G)|$ and $m = |E(G)|$. Clearly, $m = \Theta(n)$. PREPROCESSING runs in time $\Theta(n+m)$ because that is the running time of DFS in general and PREPROCESSING can be implemented as a modified DFS, keeping the running time in the same asymptotic bound. For cacti, $\Theta(n+m)$ is $\Theta(n)$. So, PREPROCESSING takes $\Theta(n)$ time. The three minima at lines 10, 11, and 15 take $\Theta(n)$ time at worst, and the other computations are constant-time. It follows TRAIN IN A CACTUS runs in time $\Theta(n)$.

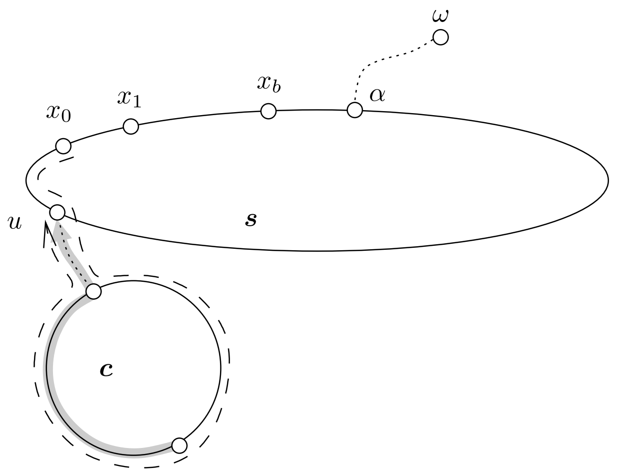


Fig. 13. Although α is not a train vertex initially it makes sense to perform a U-turn and then reach α "backwards" via the shorter cycle arch containing x_0 rather then simply follow the cycle forward to α.

5. Conclusions

We have constructed a linear-time algorithm for novel computational graph problem on a restricted graph class, the problem being $\mathcal{NP}$-complete in general. Our algorithm is relatively straightforward and easy to grasp and implement, which makes it practical.

There are numerous possibilities for future research stemming from this one. Since cacti are outerplanars, one may try to develop fast algorithms for the same problem on more general graph classes, for example outerplanars or even on partial 2-trees. Another way to generalize this result is to modify the rules of the train movement, for example letting the train gain or lose length or move bidirectionally in some circumstances.

References

Garey, M., Johnson, D. (1979). *Computers and Intractability*. W.H. Freeman and Co.

Pankov, P. (2008). Naturalness naturalness in tasks for olympiads in informatics. *Olympiads in Informatics*, 2, 115–121.

Torii, R. (2008). Path transferability of graphs. *Discrete Mathematics*, 308(17), 3782–3804.

Tutte, W.T. (1998). *Graph Theory As I Have Known It*. Oxford University Press, Inc.

M. Markov is an assistant professor of discrete mathematics and algorithms at Sofa University, Sofia. Bulgaria, PhD in computer science.

K. Manev is a professor of discrete mathematics and algorithms at New Bulgarian University, Sofia, Bulgaria, PhD in computer science. He was a member of Bulgarian National Committee for olympiads in informatics since 1982 and president of NC from 1998 to 2002. He was member of the organizing team of IOI 1989, IOI 1990, vice president of IOI 2009 and a leader of Bulgarian team for IOI in 1989, 1998, 1999, 2000 and 2005. From 2000 to 2003 he was elected member of IC of IOI, since 2005 to 2010 represented in IC the Host country of IOI 2009, and during IOI 2010 was elected as a member of IC of IOI again for the period 2010–2013.

Olympiads in Informatics, 2013, Vol. 7, 78–89
© 2013 *Vilnius University*

Latvian Olympiad in Informatics – Lessons Learned

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

Abstract. The paper gives an insight in the format of annual (since 1988) Latvian programming competition for secondary school students – Latvian Olympiad in Informatics (LIO) – as well as describing findings and drawbacks of the current, 26th LIO. As an illustration a couple of task examples are given. Problems concerning grading process are discussed.

Key words: olympiads in informatics, competition tasks, grading, jury responsibility.

1. Introduction

The introduction of computers in Latvian schools started in the middle of 1980s as an ex-USSR program of school computerization. After the restoration of independence of Latvia in 1990 this process got more power and after joining the European Union in 2004 several EU projects were also devoted to updating the ICT equipment in schools and educating teachers. According to the newest European Commission survey (EC, 2013), in the benchmarking group "digitally equipped school", Latvian schools are ranked over the EU average having also very good Internet availability and speed marks. Today, two ICT related disciplines are taught in Latvian schools – the mandatory discipline "Applied Informatics" (starting with grade 5) and a mandatory "Basics of Programming" (for grades 10–12). Often, the naming of these disciplines is a source of confusion, because in the early years of computers, in schools, there was just one discipline "Informatics", usually used as a synonym for "Programming". Even now statisticians do not try to distinguish between these disciplines and from the national statistics registers (Latvian Statistics) you can get only the total number of school students attending these two disciplines: 84,802 in the season of 2011/2012 (53 222 in grades 1–9, and 31 580 in grades 10–12).

Teaching algorithms and programming is an essential part of the ICT education process and now the process is started to improve the teaching of algorithms (or "bring them back") in Latvian schools. To encourage teaching of programming, since the very beginning of computerization era, algorithic programming competitions have been organized. Traditionally, at that time, the competitions for high school students in disciplines like as mathematics and physics, with unusual tasks going beyond the ordinary school programme, were called "olympiads".

The first programming competition in Latvia which can be named "olympiad" was organized by young scientists of Computing Centre of Latvian State University (now Institute of Mathematics and Computer Science, University of Latvia) in February, 22nd 1986. Roots of this competition go back directly to the famous Latvian mathematics olympiads conducted at that time by professor Agnis Andžāns (Ramāna and Andžāns, 2002). Prof. Andžāns also participated in setting up of programming (or, in the early years more appropriate name would be – algorithmic) olympiads.

Formally the Latvian Olympiads in Informatics (*Latvijas informātikas* (*programmēšanas*) *olimpiāde*, LIO) started in 1988. Today the LIO together with olympiads in mathematics, physics, chemistry, geography and biology is one of 6 "big"(having a corresponding international olympiad) science olympiads held annually. In the season of 2012./2013 the 26th LIO took place (`http://www.lio.lv`).

The LIO is the most popular individual programming competition in Latvia. Besides the LIO regular team competitions, like the team competitions in mathematics and informatics "Ugāle" (Opmanis, 2006) and Lattelecom IT Olympiad (`http://olimpiade.lattelecom.lv/olimpiade2011/`) are organized.

Winners of the LIO participate successfully in the International Olympiads in Informatics (IOI; `ioinformatics.org`) and their achievements were recognized by the authors of an international survey (Nedkov, 2012).

2. Format of the LIO

In the history of the LIO we can find essentially different formats – in the early days it was a one day theoretical pen-and-paper algorithm design and analysis round followed by a practical round on computers. Later, the format of two rounds of different lengths (five and four hours) was used, and smaller differences were introduced – like different maximum points for a solved problem. Current the LIOs are strongly connected to such international events as Baltic Olympiad in Informatics (BOI; Bulotaite, 1997; Poranen *et al.*, 2009) and the IOI and they try to follow the standards of these competitions.

Today the LIO is a four level programming competition. Competitions of all levels are graded by using an automated grading system. The first level – school competition – is aimed at newcomers but is open as well for all students willing to brush up their competition participant skills. There are two divisions – junior division (grades 8–10) and senior division (grades 11–12). For each division three tasks are prepared and it is supposed that tasks must be solved in the standard time for a single olympiad round – 5 hours. Participation in the school level competition is compulsory – it has no qualifying meaning. It is intended rather for testing further competitor willingness to participate in the next years' LIOs. It must be mentioned that despite the formal lower bound of grade 8, younger participants are allowed to participate as well. The youngest participant at this year's LIO was from grade 5. Also a lot of interested teachers take part as contestants in first level competitions.

The second level are regional competitions which have qualifying status and the main goal of participants is to qualify for the next level competition. Since 2006 when LIO's jury was seriously accused for incorrect grading, all LIO procedures were described at a very detailed level and nowadays the text of the LIO Regulations is approximately four times longer than regulations of any other Latvian scientific olympiad. Such strong regulations give no way for manoeuvres even if some of contestants are not able to participate at regional level. If the student due to any reason could not participate at the regional level competition and did not win a medal year before (and therefore qualifies for competition), there is no way to get to the next stage and therefore road to the IOI this year is closed.

Second level competitions are organized in one round, they have the same two age divisions as the first level and there are also three tasks given for five hours. Competitions are organized across the entirety of Latvia in a so called "supervised online" way. This means that all competitors from each region come together in a particular school and compete under the supervision of a teacher responsible for the competition in the region. The real situation in Latvia is such that regional competitions *de facto* are organized only in places where there are enthusiastic teachers willing to spend their free time for such extra-curricular activities. It seems, there are no other serious factors(such as population of a particular city) influencing popularity of programming and participation rate in the LIO. At the regional level grading is done by the central grading system and so all the participants despite their geographical distance are in virtually almost the same contest situation. This year, the number of contestants was lower than in the previous year – 66 in the junior division (78 at 2011) and 96 in the senior division (130 at 2011). Only representatives of 41 secondary school (out of 835 having Internet access) participated in the second level competition.

For this year, because of legal matters it was strictly stated that no more than 40 participants from each group would qualify to the next level competition. These participants were to be chosen according to four criteria (in the given order):

1. participant's result in his/her region is the best and he/she is the only one in the region with such a result, and it is at least 25% from the 10 best overall participants average result;
2. previous year LIO medalist;
3. previous year Baltic or international OI participant;
4. best participants according to their overall results not qualified according to the previous criteria.

The third level – country competition – is sometimes also called the "finals" and it is organized as an on-site competition. In two consecutive days two five-hour rounds with three tasks each are organized. Age groups are the same as in the previous levels. Already for the fifth year in a row the LIO finals were organized outside the capital of Latvia – Riga. This tradition started in 2009 following a similar pattern used in the neighbour country Lithuania. In these five years, four times the competition was organized in the local universities and university colleges (Ventspils, Rēzekne, Jelgava, Daugavpils) and one time in a state gymnasium (Cēsis). This experience was highly positive, it showed that outside of Riga the event becomes a real festival and local organizers are even better (definitely because of a willingness to demonstrate their ability). Again, because of legal issues this year it was the first one when for all Latvian scientific olympiads a strict medal allocation algorithm was established. The main idea comes from the IOI medal allocation algorithm. It is stated that at least 1/24 of all participants will be awarded with gold medals, at least 1/8 of all participants with gold or silver medals and at most 1/4 of all participants with any of the medals. Such an algorithm allows knowledge in advance of the number of medals required (there can be some deviations only due to equivalent numbers of points received).

After the third level competition all medal winners are invited to the fourth level competition – the selection round for the participation in Baltic OI. The selection round is organized as an on-site two consecutive days competition in a single age group. The six best participants are included in the Latvian team for participation in Baltic OI. Interestingly, in this year the three best participants from the youngest group of the country competition qualified for Baltic OI showing the first, second and fifth result in the selection round.

3. Lessons Learned from the Current LIO's Country Competition

Despite almost the same structure year-by-year, the previous experience is analyzed every year and some innovations are tried.

3.1. *Introducing "First Subtask"*

An important innovation at the State level competition was introduced in order to treat the so called "0-frustration" (i.e., scoring of 0 points at the State level competition) of contestants and their teachers. From time to time this issue was raised during discussions, more or less openly blaming the problem setters for making the competition "too hard". From the viewpoint of organizers, it was not always clear whether gaining 0 points is caused by technical problems like impossibility to technically submit solution to the grading system, a total misunderstanding of the task or the complexity level of the task being really too high.

For this year's competition it was decided to add into every task description a special part – the so called "first subtask". Namely, for one or several test cases there were given exact input data. To get points for this subtask (2 points out of the maximum possible 100) it was enough to solve these particular test cases on a paper or by using simple tools available on contestant's computer, and after this write a simple program which solves these test cases only.

The intention of this invention was distinguishing clearly between contestants who did not understood the idea of the task at all from that the ones who cannot solve the task because of its hardness. The experiment was successful – for the first time, there were no participants with 0 scores at all in both age groups. However, instead of 12 points which could be collected theoretically in such an easy way during two competition days, there were still contestants scoring just 2 or 4 points. This clearly shows that the "first subtask" approach should be continued and the exact reasons of the weak performance of particular students must be studied.

3.2. *Jury Mistake(s)*

Following the last years' IOI tradition at LIO partial feedback is also provided – for every accepted submission there is a possibility for several tests see the exact result of grading, whether it is correct or not. From the early years of innovation of particular or full feedback the author warned the IOI community about the impossibility of correcting jury mistakes in case of erroneous grading during contest. However, till now such warnings were not heard.

Taking in account the author's sceptical attitude to using feedback in competitions, it is more ironic, that the situation which can be easy defined as *force majeure* took place in the competition conducted by the author as head of the jury.

The rationale of the case was the following: on the first competition day one of the tasks needed a checker for grading due to non-unique correct results. During the competition the checker was run with incorrect parameters using a file with correct jury solution instead of a file with actual contestant's solution. This led to incorrect grading of huge number of solutions because all solutions which fitted in the time limits and ended with normal exit code were accepted and even more – it was reported back to contestant that all "visible" task groups were solved correctly.

Afterwards it was quite interestingly to analyze how contestants proceed during competition and afterwards. It was obvious, that most of contestants receiving message "all is correct" stopped working on this particular task. Clear psychological parallels with computer games can be seen there – if a level of the game is completed, you proceed to the next one and do not bother whether you were smart or just lucky. The dangerous part is that among these participants were also such participants whose programs were so trivial that they simply could not be correct! In general a serious alarm signal is that contestants take too much account of the grading system and are too lazy or too weak in testing. Even more – full or even partial feedback stimulates contestants not to test their programs which was definitely not the case without feedback. One way to cope with this would be to give the contestants the possibility of creating an appropriate test case and checking whether execution result is the same for contestants and jury programs. Then level of program testing will directly depend on the content of test case created by the contestant. Further these tests may also be used in the full test set as suggested before (Opmanis, 2006).

But let us come back to story about the jury mistake. After the competition round the jury got no official protests. From different sources came unofficial rumours that several contestants have decided that the "grading system is broken" because full score were given for very weak programs and intuitively contestants felt that "something is wrong". After getting even such unofficial impulses, the jury quite fast recovered the source of problem and regraded all submissions of that particular task. Regrading influenced 23 out of 36 contestants lowering their results (in some cases from 100 to 0 points). Regrading was the only reasonable solution seen by jury and corresponds to previous practice at OI. However, adult participants of the LIO were not united in support of the jury decision – other options like exclude results for this particular task or even all competition round were mentioned. One more theoretically possible option is “leave results as is” – so “keeping promise about information submitted during contest”.

This uncomfortable situation predicted in theory, also in practice showed its very ugly face – there simply was no fair-for-all decision. If you try to analyze benefits and drawbacks, you can easy see that always some contestant can be placed in “unfair” position. I would like to emphasize that this took place in a situation with particular feedback. The author’s feeling is that in case of full feedback things will be even worse, because the feeling “all is correct” must be even stronger. And if there is willingness to keep feedback in its current form, there simply must be some paragraph in the rules – what to do in the cases of jury mistake that are found after the competition round.

Besides the problem that contestants are not testing their solutions and, in the worst scenario, are losing this essential ability necessary for software developers, this episode also raises the question about the role of the grading system. Some authors argue that grading system is just a simple technical tool like a stop-watch in the field and track competitions. This is almost true for the “old” approach where results of main grading were not available during contest. In the case of immediate feedback the grading system acts more like a referee in sports and therefore is a serious player at the playground. If we try to follow this analogy and find out what is said about referee’s (or more general, “officials”) errors and mistakes in the rules for various sports, we find a lot of common together with several quite different principles. For example, in football (`http://www.fifa.com/aboutfifa/footballdevelopment/technicalsupport/refereeing/laws-of-the-game/index.html`) referee has the authority to decide on all points and his decision is final. There is no way how to influence referee’s decision even when it is wrong. In the basketball rules (`http://www.fiba.com`) there is section named “Correctable errors” and similar section “Correcting errors” is in tennis rules (`http://www.itftennis.com/officiating/rulebooks/rules-of-tennis.aspx`). In these sections a limited number of possible judging errors is listed and clear algorithms how to proceed are given. In ice hockey rules there are guidelines for simple situations like “In cases of an obvious error in awarding a goal or an assist which has been announced, it should be corrected promptly” (`http://www.iihf.com/iihf-home/sport/iihf-rule-book.html`). Quite obvious, that there may be also errors not listed in the rules. In the tennis rules error situations are mentioned where the decision whether to correct this error or not depends on the moment when the error is discovered. In basketball and ice hockey there are clear deadlines like signing a game protocol after what no errors can be corrected at all.

Trying to use similar principles in the programming contests the question must be answered how to proceed in case when a grading system error is found after the competition round. Ask contestants to repeat their performance? Cancel the competition results? At the programming competitions the possibility of running into problems with testing is much more higher than in sports because of the complexity of grading systems and it would be wise to be prepared for such situations. In general, the absence of an overall good analogy with sports is one of the reasons why it is hard to obtain clear and relatively simple procedure for the programming competitions.

At the end I would like to express my strong feeling that the philosophical questions like as "what exactly are the goals and contents of the competition", "does full feedback change the competition paradigm" must be answered taking in account current situation, and as soon as possible. As a source for thinking could serve the discussion raised after the "CodeForces" competition and trying to understand whether it is acceptable to change tests after seeing contestant submissions (Mitrichev, 2013).

4. Task Examples

To give insight into LIO tasks, let us discuss two examples of tasks – "Valid booklets" (authors – me and Rihards Opmanis) and "Benefit" (author Sergejs Melniks) from LIO selection round. The task "Valid booklets" was proposed for Baltic OI in 2011, and the task "Benefit" – for IOI 2012. However, both tasks were rejected by the respective scientific committees. The task "Valid booklets" is interesting because its solution uses the pigeonhole (or Dirichlet) principle present in IOI Syllabus (The International Olympiad in Informatics Syllabus, 2013) but not very often really used in task solutions. For the national competition, the story part of the task "Benefit" was completely reformulated to hide clues for the suggested model solution. Below, both task descriptions are slightly modified for publication omitting technical information about formats of input and output data. a mathematical style of solution description is chosen to allow the reader to follow the reasoning better.

4.1. *Task "Valid Booklets"*

4.1.1. *Task Description*

An exam paper consists of K pages numbered consecutively: $123\ldots K$. There were made M copies and all pages placed in one big pile: $1\ldots K1\ldots K1\ldots K$ (M times). Using a booklet stitching machine there were created booklets taking pages from the pile. Each booklet (maybe except the last one) contains N pages.

We will say that booklet is *valid* if it contains full set of exam paper pages in the right order (pages $1\ldots K$ consecutively).

Write a program which for given values of K (the number of exam paper pages), M (the number of copies) and N (the number of booklets, $N \leqslant 9 \times 10^{18}$) calculates number of valid booklets. It is known that $K \times M \leqslant 9 \times 10^{18}$. See Table 1.

4.1.2. *Solution*

Let's start with several observations:

1. if $N < K$, then the result is 0,
2. if $N = K$, then the result is M,
3. if $N \geqslant 2K - 1$, then each full group definitely is valid booklet, because there always is a group of K consecutive pages starting with the first page. The number of booklets therefore is. The last booklet with less than N pages can add one more valid booklet if.

Table 1

Examples

Input data	Output data	Comment
2 6 3	4	The following booklets were created (exam paper sets are underlined): ($\underline{1\ 2}$ 1) (2 $\underline{1\ 2}$) ($\underline{1\ 2}$ 1) (2 $\underline{1\ 2}$)
4 6 5	3	The following booklets were created (exam paper sets are underlined): ($\underline{1\ 2\ 3\ 4}$ 1) (2 3 4 1 2) (3 4 1 2 3) (4 $\underline{1\ 2\ 3\ 4}$) ($\underline{1\ 2\ 3\ 4}$) Despite the fact that the last booklet contains only four pages, it still contains full set of exam paper pages and therefore is valid.

The only remaining case is $K < N \leqslant 2K - 2$. According to pigeonhole principle there are one or two first pages in each full group. a booklet is valid if there are two first pages or group ends with page number K (only one of these two situations can take place).

The number of first pages located in full groups is
$N_{FP} = \begin{cases} M, & if\{\frac{KM}{N}\} < K, \\ M-1, & if\{\frac{KM}{N}\} \geqslant K. \end{cases}$.

The number of full groups containing two first pages can be calculated as
$N_{FP} - [\frac{KM}{N}]$.

The number of full groups ending with page number K can be calculated as $[\frac{KM}{LCM(K,N)}]$, where $LCM(K,N)$ is the least common multiple of K and N. Using the formula $KN = LCM(K,N) \times GCD(K,N)$ where $GCD(K,N)$ is the greatest common divisor of K and N, the previous formula can be transformed as follows:

$$\left[\frac{KM}{LCM(K,N)}\right] = \left[\frac{M \times GCD(K,N)}{N}\right] = \left[\frac{M}{\left[\frac{N}{GCD(K,N)}\right]}\right].$$

This transformation may be helpful to limit the intermediate results.

An uncompleted group can still be valid booklet if $\{\frac{KM}{N}\} \geqslant K$.

Summing up all these calculations we obtain a surprisingly simple formula:

$$M - \left[\frac{KM}{N}\right] + \left[\frac{M}{\left[\frac{N}{GCD(K,N)}\right]}\right]$$

4.2. *Task "Benefit"*

4.2.1. *Task Description*

Given N cards, having one side coloured in green and the other side in a red colour. On each side some integer is written. When any two cards are chosen (lets denote them by

Table 2

Example

Input data	Output data	Comment
5 **9 –1** **7 8** **–2 4** **9 –6** **3 5**	**114**	The maximum difference of benefits is between the fourth and the second card: $9 \times 8 - 7{\times}(-6) = 114$

A and B), it is possible to calculate *the benefit of A against B*. This benefit is calculated as the product of the numbers written on the green side of card A and on the red side of card B.

For example, if 10 is written on the green side of the card A, and 3 on its red side, and 7 is written on the green side of the card B, and (-2) on the red side, then the benefit of A against B is $10 \times (-2) = -20$ and the benefit of B against a is $7 \times 3 = 21$. When both benefits are calculated, it is possible to find difference between these benefits. In the given example it is -41. If the same cards would be chosen in the reverse order, the difference of benefits would be 41.

Write a program, which, for a given description of N ($N \leqslant 2 \times 10^5$) cards finds the maximum possible difference of benefits. It is known that all the numbers written on cards are between -2×10^9 and 2×10^9. See Table 2.

4.2.2. *Task Solution (by Sergejs Meļņiks)*

We begin with some observations that reveal the geometric meaning of the task. Let us denote by s_{ij} the benefit of card with index i against the card with index j, and by x_i and y_i – the numbers written on the green and red sides of the card with index i respectively. We can observe that:

1) $s_{ij} = -s_{ji}$, i.e., the maximum cannot be negative, because for any negative s_{ij} we can choose $s_{ji} = -s_{ij} > s_{ij}$ Therefore, it suffices to find the maximum value of $|s_{ij}| = |x_i y_j - x_j y_i|$.
2) Replacing x_i by $-x_i$ and y_i by $-y_i$ does not change the value of $|s_{ij}| = |x_i y_j - x_j y_i| = |-(x_i y_j - x_j y_i)| = |(-x_i) y_j - x_j(-y_i)|$.
3) Consider a triangle OAB in the coordinate plane, where O – the point of origin having coordinates (0,0), A – the point with coordinates(x_i, y_i) and B – the point with coordinates (x_j, y_j), then $|x_i y_j - x_j y_i|$ equals to doubled area of the triangle OAB.

Therefore, the original task can be reduced to the following one:

Let us construct the set of points $P = \{P_0, P_1, \ldots, P_{N-1}, P_N\}$, where the coordinates of P_0 are (0,0), and for all $i = 1, \ldots, N$ the coordinates of P_i correspond to the "card" (x_i, y_i) as follows:

$$\begin{cases} (-x_i, -y_i) & \text{if } y_i < 0 \text{ or } (y_i = 0 \text{ and } x_i < 0), \\ (x_i, y_i) & \text{otherwise.} \end{cases}$$

The triangle must be found with the largest area from all triangles such that one vertex is the point P_0 and two other vertices are placed in the two different points P_i and P_j.

Let us denote the convex hull of P by $\mathrm{conv}(P)$. By construction, P_0 belongs to $\mathrm{conv}(P)$ because P_0 is the leftmost of all the lowest points of P.

By purely geometrical means we can prove that all vertices of the desired triangle with maximum area are located in the vertices of $\mathrm{conv}(P)$.

Let the points A, B, C be located in the interior of a convex polygon M.

We want to construct a triangle ABV such that V is the vertex of M, and the area of $ABVS_{ABV} > S_{ABC}$.

The ray AC crosses the boundary of the polygon M at some point D. Obviously, $S_{ABD} > S_{ABC}$.

If D is a vertex of M, we set $V = D$. Otherwise, D is located on the some side UW of M (see Fig.1). If UW is parallel to AB, then $S_{ABU} = S_{ABW} = S_{ABD} > S_{ABC}$, otherwise one of the points U and W(let this be point W, for definiteness) is located closer to AB than the point D, and the other vertex (point U) is located further from the AB than point D. Then, choosing $V = U$we obtain that $S_{ABV} > S_{ABC}$.

Now we can extend the ray VB till the intersection with the perimeter of the polygon $M$and use the same reasoning as above. Afterwards, in the same way we can extend the ray VA.

To construct $\mathrm{conv}(P)$ let's sort all points P_i except P_0 by the polar angle and then apply the Graham scan. Note that all necessary comparisons can be easily implemented in integer arithmetic. The time complexity of constructing the convex hull is $O(N \log N)$ with required memory is linear in N.

Let $\mathrm{conv}(P)$ to be the polygon $B_0B_1 \dots B_K$, where $B_0$is the point of origin $(0, 0)$.

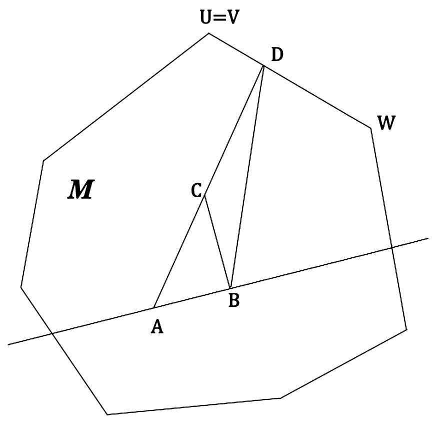


Fig. 1. Triangle with maximum area within polygon.

Let denote by $q(i)$ the number j such that $j > i$ and $S_{B_0B_iB_j} = \max_{i<m\leqslant K} S_{B_0B_iB_m}$.

It can be proved that $q(i)$ is a non-decreasing function: if $i_1 < i_2$ then $q(i_1) \leqslant q(i_2)$.

These considerations allow us to find a triangle with the maximum area via a single traversal of the convex hull by consecutive processing related values i and $q(i)$.

5. Conclusions

During its 26 years, the idea of Latvian Olympiad in Informatics has shown its vitality. However, changes made since the early years show that there is always space for new ideas and improvements in wide range of directions.

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

References

Bulotaite, J., Diks, K., Opmanis, M., Prank, R. (1997). *Baltic Olympiads in Informatics*. Institute of Mathematics and Informatics, Vilnius.

EC (2013). *Survey of Schools: ICT in Education, Final Report*. `http://ec.europa.eu/digital-agenda/en/news/survey-schools-ict-education`

International Olympiad in Informatics. `http://www.ioinformatics.org`

Lattelecom IT Olympiad. `http://olimpiade.lattelecom.lv/olimpiade2011/`

Latvian Olympiads in Informatics. `http://www.lio.lv`

Latvian Statistics. Databases of the Central Department of Statistics. `http://data.csb.gov.lv/DATABASE/Iedzsoc/Ikgadējie statistikas dati/Izglītība unzinātne/ Izglītība un zinātne.asp`

Mitrichev, P. (2013). *blog at CodeForces*. `http://codeforces.ru/blog/entry/6928`

Nedkov, P., (2012). Young talent in informatics. In: *An AICA – IT STAR Survey, International Conference on Young Talents and the Digital Future*, Milan. `http://www.aicanet.it/eventicontestuali/allegati-eventi-2012/IOI_ Survey-final.pdf`

Opmanis, M. (2006). Some ways to improve olympiads in informatics. *Informatics in Education*, 5(1), 113–124.

Opmanis, M. (2009). Team competition in mathematics and informatics "Ugāle" – finding new task types. *Olympiads in Informatics*, 3, 80–100.

Poranen, T., Dagiene, V., Eldhuset, Å., Hyyrö, H., Kubica, M., Laaksonen, A., Opmanis, M., Pohl, W., Skūpiene, J., Söderjhelm, P., Truu, A. (2009). Baltic olympiads in informatics: challenges for training together. *Olympiads in Informatics*, 3, 112–131.

Ramāna, L., Andžāns, A. (2002). Advanced mathematical education in Latvia. Creativity in mathematics education and the education of the gifted students. In: Andžāns, A., Meissner, H. (Eds.) *Proceedings of the International Conference*, Riga, University of Latvia.

The International Olympiad in Informatics Syllabus (2013). `http://people.ksp.sk/ ~misof/ioi-syllabus/ioi-syllabus.pdf`

M. Opmanis is researcher at the Institute of Mathematics and Computer Science of University of Latvia. Since the very first Latvian Olympiads in Informatics in 1986 he is involved in its organization at various positions (last years – head of jury). He is deputy or team leader of Latvian IOI teams since 1996 and team leader of Latvian team at Baltic Olympiads in Informatics since the first Olympiad in 1995 (except 2008 and 2010). M. Opmanis was head of jury of Baltic Olympiad in Informatics at BOI 1996, 1999, 2004 and 2012. Since 2012 he is member of IOI International Committee.

Olympiads in Informatics, 2013, Vol. 7, 90–100
© 2013 *Vilnius University*

Where to Use and How not to Use Polynomial String Hashing

Jakub PACHOCKI, Jakub RADOSZEWSKI
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
e-mail: {pachocki,jrad}@mimuw.edu.pl

Abstract. We discuss the usefulness of polynomial string hashing in programming competition tasks. We show why several common choices of parameters of a hash function can easily lead to a large number of collisions. We particularly concentrate on the case of hashing modulo the size of the integer type used for computation of fingerprints, that is, modulo a power of two. We also give examples of tasks in which string hashing yields a solution much simpler than the solutions obtained using other known algorithms and data structures for string processing.

Key words: programming contests, hashing on strings, task evaluation.

1. Introduction

Hash functions are used to map large data sets of elements of an arbitrary length (the *keys*) to smaller data sets of elements of a fixed length (the *fingerprints*). The basic application of hashing is efficient testing of equality of keys by comparing their fingerprints. A *collision* happens when two different keys have the same fingerprint. The way in which collisions are handled is crucial in most applications of hashing. Hashing is particularly useful in construction of efficient practical algorithms.

Here we focus on the case of the keys being strings over an integer alphabet $\Sigma = \{0, 1, \ldots, A-1\}$. The elements of Σ are called *symbols*.

An ideal hash function for strings should obviously depend both on the multiset of the symbols present in the key and on the order of the symbols. The most common family of such hash functions treats the symbols of a string as coefficients of a polynomial with an integer variable p and computes its value modulo an integer constant M:

$$H(s_1 s_2 s_3 \ldots s_n) = (s_1 + s_2 p + s_3 p^2 + \cdots + s_n p^{n-1}) \bmod M.$$

A careful choice of the parameters M, p is important to obtain “good” properties of the hash function, i.e., low collision rate.

Fingerprints of strings can be computed in $O(n)$ time using the well-known Horner's method:

$$h_{n+1} = 0, \quad h_i = (s_i + p h_{i+1}) \bmod M \quad \text{for } i = n, n-1, \ldots, 1. \tag{1}$$

Polynomial hashing has a *rolling* property: the fingerprints can be updated efficiently when symbols are added or removed at the ends of the string (provided that an array of powers of p modulo M of sufficient length is stored). The popular Rabin–Karp pattern matching algorithm is based on this property (Karp and Rabin, 1987). Moreover, if the fingerprints of all the suffixes of a string are stored as in (1), one can efficiently find the fingerprint of any *substring* of the string:

$$H(s_i \dots s_j) = (h_i - p^{j-i+1}h_{j+1}) \bmod M. \tag{2}$$

This enables efficient comparison of any pair of substrings of a given string.

When the fingerprints of two strings are equal, we basically have the following options: either we consider the strings equal henceforth, and this way possibly sacrifice the correctness in the case a collision occurs, or simply check symbol-by-symbol if the strings are indeed equal, possibly sacrificing the efficiency. The decision should be made depending on a particular application.

In this article we discuss the usefulness of polynomial string hashing in solutions of programming competition tasks. We first show why several common ways of choosing the constants M and p in the hash function can easily lead to a large number of collisions with a detailed discussion of the case in which $M = 2^k$. Then we consider examples of tasks in which string hashing applies especially well.

2. How to Choose Parameters of Hash Function?

2.1. *Basic Constraints*

A good requirement for a hash function on strings is that it should be difficult to find a pair of different strings, preferably of the same length n, that have equal fingerprints. This excludes the choice of $M < n$. Indeed, in this case at some point the powers of p corresponding to respective symbols of the string start to repeat. Assume that $p^i \equiv p^j \bmod M$ for $i < j < n$. Then the following two strings of length n have the same fingerprint:

$$\underbrace{0\dots0}_{i}a_1a_2\dots a_{n-j}\underbrace{0\dots0}_{j-i} \quad \text{and} \quad \underbrace{0\dots0}_{j}a_1a_2\dots a_{n-j}.$$

Similarly, if $\gcd(M,p) > 1$ then powers of p modulo M may repeat for exponents smaller than n. The safest choice is to set p as one of the generators of the group $U(\mathbb{Z}_M)$ – the group of all integers relatively prime to M under multiplication modulo M. Such a generator exists if M equals 2, 4, q^a or $2q^a$ where q is an odd prime and $a \geqslant 1$ is integer (Weisstein, on-line). A generator of $U(\mathbb{Z}_M)$ can be found by testing a number of random candidates. We will not get into further details here; it is simply most important not to choose M and p for which $M \mid p^i$ for any integer i.

A slightly less obvious fact is that it is bad to choose p that is too small. If $p < A$ (the size of the alphabet) then it is very easy to show two strings of length 2 that cause a collision:

$$H(01) = H(p0).$$

2.2. *Upper Bound on M*

We also need to consider the magnitude of the parameter M. Let us recall that most programming languages, and especially the languages C, C$++$, Pascal that are used for IOI-style competitions, use built-in integer types for integer manipulation. The most popular such types operate on 32-bit or 64-bit numbers which corresponds to computations modulo 2^{32} and 2^{64} respectively. Thus, to be able to use Horner's method for fingerprint computation (1), the value $(M-1)\cdot p + (M-1) = (M-1)\cdot(p+1)$ must fit within the selected integer type. However, if we wish to compute fingerprints for substrings of a string using the (2), we need $(M-1)^2 + (M-1)$ to fit within the integer type, which bounds the range of possible values of M significantly. Alternatively, one could choose a greater constant M and use a fancier integer multiplication algorithm (which is far less convenient).

2.3. *Lower Bound on M*

On the other side M is bounded due to the well-known birthday paradox: if we consider a collection of m keys with $m \geqslant 1.2\sqrt{M}$ then the chance of a collision to occur within this collection is at least 50% (assuming that the distribution of fingerprints is close to uniform on the set of all strings). Thus if the birthday paradox applies then one needs to choose $M = \omega(m^2)$ to have a fair chance to avoid a collision. However, one should note that not always the birthday paradox applies. As a benchmark consider the following two problems.

Problem 1: ***Longest Repeating Substring.*** Given a string s, compute the longest string that occurs at least twice as a substring of s.

Problem 2: ***Lexicographical Comparison.*** Given a string s and a number of queries (a, b, c, d) specifying pairs of substrings of s: $s_a s_{a+1} \dots s_b$ and $s_c s_{c+1} \dots s_d$, check, for each query, which of the two substrings is lexicographically smaller.

A solution to Problem 1 uses the fact that if s has a repeating substring of length k then it has a repeating substring of any length smaller than k. Therefore we can apply binary search to find the maximum length k of a repeating substring. For a candidate value of k we need to find out if there is any pair of substrings of s of length k with equal fingerprints. In this situation the birthday paradox applies. Here we assume that the distribution of fingerprints is close to uniform, we also ignore the fact that fingerprints of consecutive substrings heavily depend on each other – both of these simplifying assumptions turn out not to influence the chance of a collision significantly.

The situation in Problem 2 is different. For a given pair of substrings, we apply binary search to find the length of their longest common prefix and afterwards we compare the symbols that immediately follow the common prefix, provided that they exist. Here we have a completely different setting, since we only check if specific pairs of substrings are equal and we do not care about collisions *across* the pairs. In a uniform model, the chance of a collision within a single comparison is $\frac{1}{M}$, and the chance of a collision occurring within m substring comparisons does not exceed $\frac{m}{M}$. The birthday paradox does not apply here.

2.4. *What if* $M = 2^k$*?*

A very tempting idea is not to select any value of M at all: simply perform all the computations modulo the size of the integer type, that is, modulo 2^k for some positive integer k. Apart from simplicity we also gain efficiency since the modulo operation is relatively slow, especially for larger integer types. That is why many contestants often choose this method when implementing their solutions. However, this might not be the safest choice. Below we show a known family of strings which causes many collisions for such a hash function.

This family is the ubiquitous Thue–Morse sequence (Allouche and Shallit, 1999). It is defined recursively as follows:

$$\tau_0 = 0;\ \tau_i = \tau_{i-1}\bar{\tau}_{i-1} \quad \text{for } i > 0$$

where $\bar{x}$ is the sequence resulting by negating the bits of x. We have:

$$\tau_0 = 0,\ \tau_1 = 01,\ \tau_2 = 0110,\ \tau_3 = 01101001,\ \tau_4 = 0110100110010110, \ldots$$

and clearly the length of τ_i is 2^i.

Let $W(s)$ be the fingerprint of a string s without any modulus:

$$W(s_1 s_2 s_3 \ldots s_n) = s_1 + s_2 p + s_3 p^2 + \cdots + s_n p^{n-1}.$$

We will show the following property of the Thue–Morse sequence that uncovers the reason for its aforementioned especially bad behavior when hashing modulo $M = 2^k$.

Theorem 1. *For any* $n \geqslant 0$ *and* $2 \nmid p$, $2^{n(n+1)/2} \mid W(\bar{\tau}_n) - W(\tau_n)$.

Proof. By the recursive definition – $\tau_n = \tau_{n-1}\bar{\tau}_{n-1}$ and, similarly, $\bar{\tau}_n = \bar{\tau}_{n-1}\tau_{n-1}$ – we have:

$$\begin{aligned} W(\bar{\tau}_n) - W(\tau_n) &= W(\bar{\tau}_{n-1}) + p^{2^{n-1}} W(\tau_{n-1}) - W(\tau_{n-1}) - p^{2^{n-1}} W(\bar{\tau}_{n-1}) \\ &= W(\bar{\tau}_{n-1})(1 - p^{2^{n-1}}) - W(\tau_{n-1})(1 - p^{2^{n-1}}) \\ &= (1 - p^{2^{n-1}})(W(\bar{\tau}_{n-1}) - W(\tau_{n-1})). \end{aligned}$$

Now it is easy to show by induction that:

$$W(\bar{\tau}_n) - W(\tau_n) = (1 - p^{2^{n-1}})(1 - p^{2^{n-2}}) \ldots (1 - p).$$

To conclude the proof, it suffices to argue that

$$2^i \mid 1 - p^{2^{i-1}} \quad \text{for any } i \geqslant 1. \tag{3}$$

This fact can also be proved by induction. For $i = 1$ this is a consequence of the fact that p is odd. For the inductive step ($i > 1$) we use the following equality:

$$1 - p^{2^{i-1}} = \left(1 - p^{2^{i-2}}\right)\left(1 + p^{2^{i-2}}\right).$$

The second factor is even again because p is odd. Due to the inductive hypothesis, the first factor is divisible by 2^{i-1}. This concludes the inductive proof of (3) and, consequently, the proof of the whole theorem. □

By Theorem 1, the strings τ_n and $\bar{\tau}_n$ certainly yield a collision if $n(n+1)/2$ exceeds the number of bits in the integer type. For instance, for 64-bit integers it suffices to take τ_{11} and $\bar{\tau}_{11}$ which are both of length 2048. Finally, let us note that our example is really vicious – it yields a collision regardless of the parameter p, provided that it is odd (and choosing an even p is surely bad if M is a power of two). This example can also be extended by sparsifying the strings (i.e., inserting segments consisting of a large number of zeros) to eliminate with high probability a heuristic algorithm that additionally checks equality of a few random symbols at corresponding positions of the strings if their fingerprints are equal.

The same example was described in a recent blog post related to programming competitions (Akhmedov, 2012).

3. Usefulness of String Hashing

In the previous section we have described several conditions that limit the choice of constants for string hashing. In some applications it is difficult to satisfy all these conditions at the same time. In contrast, in this section we show examples of problems that are more difficult to solve without using string hashing. Two particularly interesting problems we describe in detail, and at the end we list several other examples of tasks from Polish programming competitions. In the algorithms we do not describe how to handle collisions, that is, we make an optimistic assumption that no collisions occur and thus agree to obtain a heuristic solution.

We have already mentioned that string hashing can be used to check equality of substrings of a given string. For the same purpose one could use the Dictionary of Basic Factors ($O(n \log n)$ time and space preprocessing for a string of length n) or the suffix tree/suffix array ($O(n)$ time and space preprocessing for a constant-sized alphabet in the basic variant, both data structures are relatively complex). More on these data structures can be found, for instance, in the books Crochemore *et al.* (2007) and Crochemore and Rytter (2003). However, in the following problem it is much more difficult to apply any of these data structures instead of string hashing. This is a task from the final round of a Polish programming competition Algorithmic Engagements 2011 (see `http://main.edu.pl/en/archive/pa/2011/bio`).

Problem 3: ***Computational Biology.*** We are given a string $s = s_1 \dots s_n$ over a constant-sized alphabet. A *cyclic substring* of s is a string t such that all cyclic rotations of t are substrings of s [1]. For a cyclic substring t of s, we define *the number of cyclic occurrences* of t in s as the total number of occurrences of distinct cyclic rotations of t within s. We would like to find a cyclic substring of s of a given length m that has the largest number of cyclic occurrences. We are to output this number of occurrences.

For example, consider the string $s =$ `BABABBAAB` and $m = 3$. The string `AAB` is its cyclic substring with 3 cyclic occurrences: one as `AAB`, one as `ABA` and one as `BAA`. The string `ABB` is also a cyclic substring and it has 4 cyclic occurrences: one as `ABB`, two as `BAB` and one as `BBA`. Thus the result is 4.

On the other hand, consider the string $s =$ `ABAABAABAABAAAAA` and $m = 5$. Here the result is just 1: a single cyclic occurrence of a cyclic substring `AAAAA`. Note that none of the strings `ABAAB` and `AABAA` is a cyclic substring of the string and therefore they are not included in the result.

Using string hashing, we solve this problem as follows. We start by computing fingerprints of all m-symbol substrings of s: $s_1 \dots s_m$, $s_2 \dots s_{m+1}, \dots$ – this can be done in $O(n)$ time using the rolling property of the hash function. Using a map-like data structure storing fingerprints, for each of these substrings we count the number of times it occurs in s.

Now we treat these substrings as vertices of a directed graph. The vertices are identified by the fingerprints. Each vertex is assigned its multiplicity: the number of occurrences of the corresponding substring in s. The edges of the graph represent cyclic rotations by a single symbol: there is an edge from $u_1 u_2 \dots u_m$ to $u_2 \dots u_m u_1$ provided that the latter string occurs as a substring of s (see Fig. 1). Note that the fingerprint of the endpoint of any edge can be computed in $O(1)$ time, again due to the rolling property of the hash function.

Thus our problem reduces to finding the heaviest cycle in a graph in which each vertex has in-degree and out-degree at most 1. This can be done in linear time, since such a graph is a collection of cycles and paths. The whole solution has $O(n \log n)$ time complexity due to the application of a map-like data structure.

The part of the above solution that causes difficulties in using the Dictionary of Basic Factors or the suffix tree/suffix array data structures is the fact that if $u_1 u_2 \dots u_m$ is a substring of s then $u_2 \dots u_m u_1$ is (almost never) a substring of s that could be easily identified by its position in s.

We proceed to the second task example. Recall that in the pattern matching problem we are given two strings, a pattern and a text, and we are to find all the occurrences of the

[1] *A cyclic rotation* of a string is constructed by moving its first letter to its end, possibly multiple times. For example, there are three different cyclic rotations of `ABAABA`, namely `BAABAA`, `AABAAB` and `ABAABA`.

$$\begin{tikzcd}
& \overset{1}{\texttt{ABA}} \arrow[dr] & & & \overset{1}{\texttt{BBA}} \arrow[dr] & \\
\underset{1}{\texttt{AAB}} \arrow[ur] & & \underset{1}{\texttt{BAA}} \arrow[ll] & \underset{1}{\texttt{ABB}} \arrow[ur] & & \underset{2}{\texttt{BAB}} \arrow[ll]
\end{tikzcd}$$

$$\begin{tikzcd}
\underset{3}{\texttt{ABAAB}} \arrow[r] & \underset{3}{\texttt{BAABA}} & \underset{1}{\texttt{AAAAA}} \arrow[loop above] \\
\underset{3}{\texttt{AABAA}} \arrow[r] & \underset{1}{\texttt{ABAAA}} \arrow[r] & \underset{1}{\texttt{BAAAA}}
\end{tikzcd}$$

Fig. 1. The graphs of substrings for: $s = \text{BABABBAAB}$ and $m = 3$ (above) and $s = \text{ABAABAABAABAAAAA}$ and $m = 5$ (below)

pattern in the text. This problem has a number of efficient, linear time solutions, including the Morris–Pratt algorithm, Boyer–Moore algorithm, and a family of algorithms working in constant space. Also suffix trees/suffix arrays can be used for this problem. It is, however, rather difficult to extend any of these algorithms to work for the 2-dimensional variant of the problem:

Problem 4: ***2-Dimensional Pattern Matching.*** Let the pattern be an array composed of $m \times m'$ symbols and the text be an array composed of $n \times n'$ symbols. Find all occurrences of the pattern as a subarray of the text.

The Rabin–Karp pattern matching algorithm that is based on string hashing can easily be extended to two dimensions. In each row of the text we compute the fingerprint of each substring of m symbols, this is done in $O(nn')$ time using the rolling property of the hash. We create a new array of size $n \times (n' - m' + 1)$ that contains these fingerprints. Thus the problem reduces to 1-dimensional pattern matching in the columns of the new array with the pattern composed of the fingerprints of the rows of the original pattern (see Fig. 2). This problem can be solved in linear time using the standard Rabin–Karp algorithm.

We obtain a linear time algorithm that appears simpler than, e.g., the known Bird/Baker 2-dimensional pattern matching algorithm which generalizes the Morris–Pratt algorithm to the case of multiple patterns, see Crochemore and Rytter (2003).

Below we list four other examples of tasks in which string hashing can be applied to obtain efficient solutions. The examples are ordered subjectively from the easiest to the hardest task. For each task a short comment on its solution is provided.

Problem 5: ***Quasi-Cyclic-Rotations.*** We are given two strings s and t of the same length n over English alphabet. We are to check if we can change exactly one letter in s so that

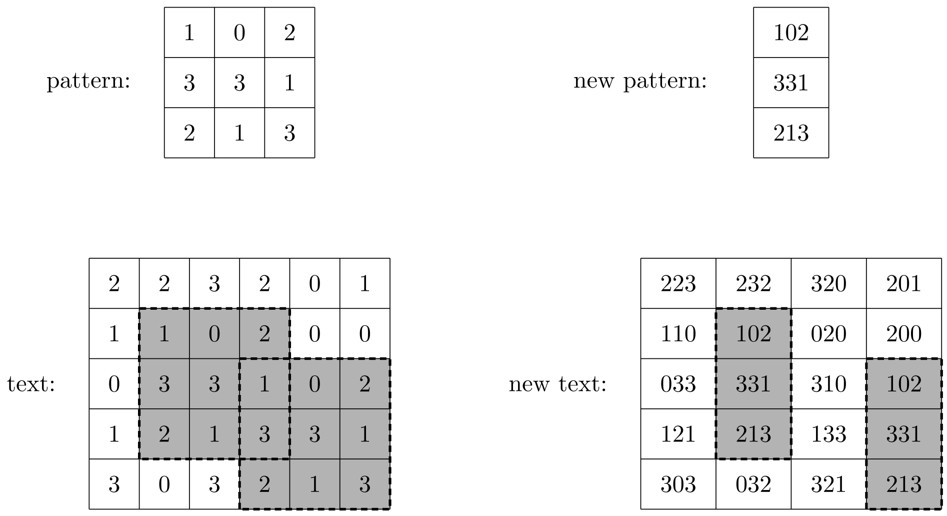


Fig. 2. Reduction of 2-dimensional pattern matching to 1-dimensional pattern matching in columns via string hashing (in this example $p = 10$, $M > 1000$)

it becomes a cyclic rotation of t. This is a task from Algorithmic Engagements 2007: `http://main.edu.pl/en/archive/pa/2007/pra`.

To check if s is an exact cyclic rotation of t, one could check if s occurs as a substring of the string tt – this is a pattern matching problem. For quasi-cyclic-rotations we can modify this approach: for each position $i = 1, 2, \dots, n$ in tt we need to check if the substring u of length n starting at this position differs from s exactly at one position. For this it suffices to find the longest common prefix and the longest common suffix of u and s. The model solution computes these values in linear total time for all u's using the *PREF* table, also called the table of prefixes (see Section 3.2 of Crochemore and Rytter (2003)). An alternative solution applies binary search for the longest commom prefix/suffix and checks a candidate prefix/suffix using string hashing. This solution requires storing of fingerprints for s and tt and works in $O(n \log n)$ time.

Problem 6: *Antisymmetry.* We are given a string $t = t_1 \dots t_n$ over $\{0, 1\}$ alphabet. For a substring u of t, by u^R we denote the reversed string u and by $\bar{u}$ we denote the negation of u obtained by changing all the zeroes to ones and ones to zeroes. A substring u of t is called *antisymmetric* if $u = \bar{u}^R$. We are to count the number of substrings of t that are antisymmetric (if the same antisymmetric substring occurs multiple times, we count each of its occurrences). This is a task from 17th Polish Olympiad in Informatics: `http://main.edu.pl/en/archive/oi/17/ant`.

Antisymmetric strings resemble palindromic strings. Recall Manacher's algorithm that finds in linear time, for each position i, the radius $R[i]$ of the longest even-length palindromic substring centered at this position (see Section 8.1 of Crochemore and Rytter (2003)). The model solution is based on a modification of Manacher's algorithm that finds, for each position i, the radius $R'[i]$ of the longest antisymmetric substring centered at this position. However, $R'[i]$ could alternatively be computed by applying binary search and checking if a candidate radius is valid via string hashing. Here string fingerprints for both t and $\bar{t}^R$ need to be stored. The solution based on string hashing has $O(n \log n)$ time complexity.

Problem 7: *Prefixuffix.* We are given a string $t = t_1 \dots t_n$ over English alphabet. A *prefixuffix* of t is a pair (p, s) of a prefix and a suffix of t, each of length at most $n/2$, such that s is a cyclic rotation of p. The goal of the task is to find the longest prefixuffix of t. This is a task from 19th Polish Olympiad in Informatics: `http://main.edu.pl/en/archive/oi/19/pre`.

The notion of a prefixuffix generalizes the notion of a *border*, which is a pair formed by an equal prefix and suffix of t. The model solution for this task works in linear time and is really tricky. Assume $2 \mid n$ and let $t = xy$, where x and y have the same length. Consider the word $Q(x, y)$ (a "crossover" of x and y) defined as $x_1 y_{n/2} x_2 y_{n/2-1} \dots x_{n/2} y_1$. Then the result is $l/2$, where l is the length of the longest prefix of $Q(x, y)$ that is a concatenation of two even-length palindromes. The value of l can be found in linear time using the output of Manacher's algorithm (already mentioned in Problem 6).

This solution deserves a short explanation. Note that (p, s) is a prefixuffix if $p = uv$ and $s = vu$ for some words u and v. Then $t = uvwzvu$ for some words w and z of equal length. Thus $x = uvw$, $y = zvu$ and $Q(x, y) = Q(u, u)Q(v, v)Q(w, z)$. Now it suffices to note that $Q(u, u)$ and $Q(v, v)$ are palindromes. E.g., if $t =$ `ababbabbabbaab` then $x =$ `ababbab`, $y =$ `babbaab` and

$$Q(x, y) = \texttt{abbaaabbbbaabb} = \texttt{abba} \cdot \texttt{aabbbbaa} \cdot \texttt{bb}.$$

Here $l = 12$ which yields a prefixuffix of t of length 6: (`ababba`, `abbaab`).

An alternative solution using string hashing was much simpler to come up with. Recall that we need to find a representation $t = uvwzvu$ with uv as long as possible. To find u, we consider all the borders of t, and to find v, we need to know the longest border of each substring of t obtained by removing the same number of letters from the front and from the back of t, that is, $a[i] = \mathit{border}(t_i t_{i+1} \dots t_{n-i} t_{n-i+1})$. All the requested borders can be found using string hashing in linear time. In particular, the latter ones, $a[i]$, can be computed for $i = 1, 2, \dots, n/2$ by observing that $a[i] \geqslant a[i-1] - 2$.

Problem 8: *String Similarity.* We consider a family of strings, $S_1, \dots, S_n$, each of length l. We perform m operations, each of which consists in swapping a pair of letters across some pair of the strings. The goal is to compute, for each $i = 1, \dots, n$, how many strings among $\{S_j\}$ were equal to S_i at some moment in time, at maximum. This is a task from 15th Polish Olympiad in Informatics: `http://main.edu.pl/en/archive/oi/15/poc`.

Here the first idea that comes to mind is to use string hashing to identify which pairs of strings are equal at respective moments in time. Note that fingerprints of the strings can be updated easily when the letters are swapped. However, in the case of this task string hashing is only the beginning of the solution. An efficient, $O((nl + m) \log (nl + m))$

time solution requires keeping track of groups of equal strings. Roughly speaking, for each valid fingerprint we compute the number of strings with this fingerprint at each moment in time and then for each single string we find the maximum size of a group of equal strings it belongs to at any moment in time.

4. Final Remarks

In general, polynomial string hashing is a useful technique in construction of efficient string algorithms. One simply needs to remember to carefully select the modulus M and the variable of the polynomial p depending on the application. A good rule of thumb is to pick both values as prime numbers with M as large as possible so that no integer overflow occurs and p being at least the size of the alphabet.

There is a number of string processing problems in which hashing enables to present solutions that are competitive with the ones obtained using non-randomized algorithms. This includes pattern matching in one and multiple dimensions and searching for specific patterns, which includes palindromic substrings, cyclic rotations and common substrings. The major virtue of hashing is its $O(n)$ time and space consumption. However, one should always keep in mind that hashing is a heuristic algorithm.

References

Akhmedov, M. (2012). Anti-hash test. *Codeforces Blog*. http://codeforces.com/blog/entry/4898.

Allouche, J.-P., Shallit, J. (1999). The ubiquitous Prouhet-Thue-Morse sequence. In: Ding, C., Helleseth, T., Niederreiter, H. (Eds.), *Sequences and Their Applications, Proceedings SETA'98*, New York, Springer-Verlag, 1–16.

Crochemore, M., Hancart, C., Lecroq, T. (2007). *Algorithms on Strings*, Cambridge University Press.

Crochemore, M., Rytter, W. (2003). *Jewels of Stringology*, World Scientific.

Karp, R.M., Rabin, M.O. (1987). Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development – Mathematics and Computing*, 31(2), 249–260.

Weisstein, E.W. Modulo multiplication group. *MathWorld – a Wolfram Web Resource*. http://mathworld.wolfram.com/ModuloMultiplicationGroup.html.

J. Pachocki (1991), computer science student at Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland. Silver medalist of IOI'2009 in Plovdiv and winner of BOI'2009 in Stockholm, member of the runner-up team at the ACM ICPC World Finals 2012 in Warsaw. Problem setter for Polish Olympiad in Informatics and CEOI'2011 in Gdynia, Poland.

J. Radoszewski (1984), teaching assistant at Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland. Chair of the jury of Polish Olympiad in Informatics, Polish team leader at IOI 2008–2010 and 2012, former chair of the jury of CEOI'2011 in Gdynia and member of the HSC of IOI'2005 in Nowy Sącz, CEOI'2004 in Rzeszów, and BOI'2008 in Gdynia, Poland. His research interests focus on text algorithms and combinatorics.

Olympiads in Informatics, 2013, Vol. 7, 101–112
© 2013 *Vilnius University*

Tasks in Informatics of Continuous Content

Pavel S. PANKOV
International University of Kyrgyzstan
e-mail: pps50@rambler.ru

Abstract. Almost all tasks at informatics olympaids are of discrete content. Tasks of continuous content are rare; moreover, some of them are not algorithmic in nature or it is not possible to score their solutions strictly because of using approximate calculations. We propose to involve such tasks with strict formulations and discrete (in integer numbers) solutions by means of ideas of interval analysis and present some ways to create and to solve them. We hope that some classes of such tasks would enlarge scope of tasks for use in informatics olympiads at various levels.

Key words: olympiads in informatics, tasks, continuous content, validating computations, interval analysis.

1. Definitions

First of all, note that all tasks of types proposed below can be written without special terminology, for example:

Task 1.1. Given an integer number $N,\ 10 < N < 10^{1000}$. Find an integer K that the solution of the equation $x^3 + x = N$ fulfills the assertion $K \leqslant x < K + 1$.

The knowledge of the formula for solution of cubic equation is a disadvantage in solving of this task because it is impractical while the common bisectional search method yields result easily.

The following considerations are developing of idea of this task:

Denote the space of real numbers as R. We will consider continuous objects, such as real numbers (basic objects), finite ordered sets of real numbers (vectors in R^n), polygons, polynomials, and continuous functions defined in any way.

To make such definitions strict, they will be presented (in the task conditions) by means of integer numbers.

It is known that not all $x \in R$ can be represented by any algorithm (by constructive methods). Hence, the notion of computable number was introduced (see, for instance, Shanin, 1968):

DEFINITION 1.1. Any $x \in R$ is said to be computable if there exists an algorithm which reworks any natural number n in such integer m that $|x - \frac{m}{n}| < \frac{1}{n}$.

Certainly, such definitions will not be used in tasks but properties of computable numbers ought to be kept in mind to produce correct statements of problems.

In our opinion, computability of real numbers arising in tasks is obvious for the common contestant.

By strict solution we mean the following

DEFINITION 1.2 (Pankov *et al.*, 1979). Computations are said to be validating if they are conducted in such a way that their results being interpreted as sets of continuous objects contain the true objects (solutions of proposed tasks).

The term "reliable computations" is also used.

One of ways to implement validating computations is interval analysis (Moore, 1967). Let $X = [x_-, x_+] \subset R$. If $x \in X$, then X is said to be an interval presentation of x; if x_- and x_+ are rational numbers and are presentable in a computer in any standard way then X is said to be a machine interval presentation of x.

For an interval $X = [x_-, x_+\ ]$, the difference $Wid(X) = x_+ - x_-$ is said to be the width of X; let the width of an interval vector be the maximum of widths of its components.

The narrower X, the better the presentation.

If the width of an interval (vector) is zero then it is said to be degenerate, i.e., it is equivalent to a single number (vector).

For intervals, the notions *Length*(X) and *Wid*(X) coincide. For interval vectors the notions *Measure* (*Area* for 2D, *Volume* for 3D ...) differ from *Wid*.

We will use the functions of an interval considered as an entire object: $X_- := x_-, X_+ := x_+$; if X is integer and $Wid(X) > 1$ then *midX* $:= int((X_- + X_+)/2)$ (for instance, *int* means rounding down to an integer);

splitting: *Lhalf* $(X) := [X_-,$ *mid*$(X)]$ and *Rhalf* $(X) := [$*mid*$(X), X_+\ ]$.

The last two operations are also applicable to interval vectors (for instance, splitting by the widest component; if there are some such ones then choose one on the least position).

Also, we will use the function of outer interval presentation of finite or other bounded sets in R^n: $Outer(W)$ is the "least" (the narrowest in all components) box (interval for $n = 1$, interval vector for $n > 1$) containing the set W. The simplest example: $Outer(\{a, b\}) = [\min\{a, b\}, \max\{a, b\}]$.

(We will use operations *min* and *max* both for finite and bounded closed infinite sets of R).

Recall the obvious formulas of interval analysis:

$$
\begin{aligned}
&[x_-, x_+] + [y_-, y_+] = [x_- + y_-, x_+ + y_+]; [x_-, x_+] - [y_-, y_+] \\
&\quad = [x_- - y_+, x_+ - y_-], \\
&[x_-, x_+] \cdot [y_-, y_+] = Outer(\{x_-y_-, x_+y_+, x_-y_+, x_+y_-\})
\end{aligned}
$$

(by means of branch on condition the last formula can be reduced to two products); if (n *is odd or* $0 \notin [x_-, x_+\ ]$) then $[x_-, x_+]^n = Outer(\{x_-^n, x_+^n\})$ else $[x_-, x_+]^n = Outer(\{x_-^n, 0, x_+^n\})$.

Interval analysis uses the triple logic. Let $x \in X \subset R$ and we try to prove that $x > 0$. If $x_+ < 0$ (*in other words*, $X < 0$) then $x < 0$ (result: *No*); if $x_- > 0$ (*in other words*,

$X > 0$) then $x > 0$ (result: *Yes*); elsewhere ($0 \in X$; result: *Uncertainty*; try to calculate a narrower interval presentation for x).

The main idea of interval analysis is deriving results on infinite sets by means of finite operations on boundaries of intervals.

Remark 1.1. Strict results of approximate calculations could be also obtained by means of strict estimation of rounding error; but implementation of such estimation during actual vast computations is impossible.

Remark 1.2. Results of validating computations (of interval analysis) can be interpreted in two main ways: as a result for a single number or as result for a set of numbers. Give a simplest

EXAMPLE 1.1. Suppose that the statement "$\pi \in [3.1, 3.2]$" is proven already. Due to rules of interval analysis, the calculation $[3.1, 3.2]^2$ yields the result $[9.6, 10.3]$ (directed rounding off to one decimal place). Thus, two statements are proven:
"$\pi^2 \in [9.6, 10.3]$"; "For all $x \in [3.1, 3.2]$, $x^2 \in [9.6, 10.3]$".

Recall definitions of interval analysis for functions.

If the inclusion $x_- \leqslant \text{ x} \leqslant x_+$ implies the inequality $p(x) \geqslant P_-(x_-, x_+)(p(x) \leqslant P_+(x_-, x_+)$ respectively) then a function $P_-(x_-, x_+)$ of two variables is said to be a minorant ($P_+(x_-, x_+)$ is said to be a majorant respectively) of the function $p(x)$ of one variable.

The interval function $P(x_-, x_+) = [P_-(x_-, x_+), P_+(x_-, x_+)]$ is said to be an interval representation of $p(x)$. If two interval representations fulfill the assertion $P_1(x_-, x_+) \subseteq P_2(x_-, x_+)$ then $P_1(x_-, x_+)$ is said to be not worse than $P_2(x_-, x_+)$.

Remark 1.3. $P * (x_-, x_+) = \mathit{Outer}(\{p(x) | x_- \leqslant x \leqslant x_+\})$ is the best interval representation for $p(x)$. But its exact calculation is practically impossible for functions arising in tasks. And the aim of interval analysis is constructing "sufficiently good" interval representations.

Interval analysis is used as follows: repetition of uniform computations by means of computer for any finite covering of a set yields a (strict) result for all points of this set. To diminish the number of elements of the covering, bisection method can be used.

We call intervals (interval vectors) with integer boundaries integer intervals (integer interval vectors correspondingly).

2. Correct Statement of Problems and Scoring

There was proven the following (see, for instance, Shanin, 1968)

Theorem 2.1. The problem of distinguishing the cases "$x < 0$", "$x = 0$" and "$x > 0$" for a computable number x is irresolvable algorithmically in general case (cf. "triple logic" above).

The narrowest non-degenerate integer interval has the width 1.

If given functions are computed exactly for integer numbers then the following statement of problem may be correct:

Statement 2.1. Find a semiopen integer interval of width 1 containing the solution, i.e., to find such integer K that $K \leqslant x < K + 1$.

But the general task to obtain such integer interval is incorrect: if the unknown number x is integer then methods of interval analysis in virtue of Theorem 2.1 can yield only a result of type $[x-\varepsilon, x+\varepsilon]$ where ε is a small positive number, and the narrowest attainable integer interval presentation is $[x - 1, x + 1]$.

So, the following two statements of problem are correct:

Statement 2.2. Find an integer interval (vector) containing the solution (real number or vector of real numbers respectively) of width not greater than 2;

Statement 2.3. Find an integer interval (vector) of width not greater than a given natural number (greater than 2).

There are two ways to score an answer given by the contestant (output-only task) or by the contestant's program:

– the jury knows the narrowest integer interval and an answer is to contain it;
– the scoring program checks an answer by a posteriori computations.

Remark 2.1. Sometimes, to substantiate an answer, using of some mathematical results (theorems) and/or some additional calculations (checking-up of some conditions) is necessary. That is why proposals to look through listings of contestants' programs arise time-by-time. But the resolutions of majorities of juries are same: such looking through of all listings is practically impossible. So, the corresponding theorems of analysis will be mentioned below but using of them, certainly, is not to be checked in listings.

Moreover, for types of tasks proposed below the contestants can improvise, invent their own algorithms including heuristic ones, use a posteriori methods, use real numbers (with rounding-off of final results) etc. We consider this as a positive effect making competitions more interesting.

Certainly, the jury is to prepare tests with strict methods:

– interval computations mentioned in this paper;
– fitting of functions.

Simplest examples of fitting: choose a polynomial $p_1(x) > 0 (x \in R)$. If $p_2(x) = a + (x - b)^2 p_1(x)$ (*after removing brackets*) then $\min\{p_2(x)\} = a$; if $p_3(x) = (x - b)p_1(x)$ (*after removing brackets*) then the solution of the equation $p_3(x) = 0$ is b.

3. Main Types of Tasks and Mathematical Methods to Solve Them

Below appropriate one of Statements 2.1, 2.2 or 2.3 is meant in contents of all tasks (under the general denotation Statement 2 with the upper boundary w for width).

General task 3.1. Given a function in a domain $G \subset R^n$

$$f : G \to R;$$

find an integer interval containing its least (greatest) value (and an integer interval containing the value of argument yielding all solutions correspondingly).

General task 3.2. Given an equation in a domain of type

$$f(x) = 0 \quad (x \in G),$$

find an integer interval (all integer intervals) containing its least (greatest) solution (all solutions correspondingly).

General task 3.3. Given a geometrical object, find an integer interval containing any measure (length, area, volume) of it.

Remark 3.1. Experience of using such tasks at olympiads gave the following paradoxical fact. Contestants who knew "formulas" for the values to be calculated demonstrated worse results than those who did not study or have forgotten such "formulas".

The explanation of this fact is following. So-called "formulas", "expressions in explicit form" are not irrevocable results (although they seem to be such ones) but they are connections between some mathematical notions only.

Therefore, we propose to teach mathematics to computer scientists with stressing the following statements:

– every mathematical method is to be considered from the viewpoint of its constructiveness;
– common (approximate) calculations do not yield guaranteed results;
– strict estimation of computational error in real tasks is impossible (what would be also useful for their forthcoming professional activity).

Such a subject was implemented by us (Pankov *et al.*, 1996, 2002). Exposition of each notion begins with conditions ensuring its computability; further, formulas and ways of better computations are given. For instance, "differentiation" is not a constructive operation (and many computer scientists lost and lose a lot of time because of this fact) while "integration" is a "good" operation. Finding values of maximum and minimum are constructive operations under wide assumptions but finding points (arguments) where these values are attained is not constructive. Also, mistakes in programming being very difficult to be found arise because of ignoring Theorem 2.1 above.

Moreover, we (Pankov, 1987) declared that "an expression in explicit form" is not a mathematical notion but a historical one. Namely, if any type of equations arose in mathematical investigations frequently then their solutions are given special denotations; if these denotations are apt then their properties are investigated thoroughly, tables of and (last fifty years) computer procedures to calculate their values approximately are developed.

To warrant this conclusion, list

constants : $\tau = \frac{\sqrt{5}+1}{2}$ (*golden ratio*), π, *e, Euler constant C*, ...

functions: *roots, trigonometric functions, inverse trigonometric functions,* $\exp$, $\log$ and being infinitely replenished list of so-called *special functions:* integrals which cannot be expressed by means of preceding functions: *integral sine, integral logarithm*, ..., *Euler* B*- and* Γ*-functions*, *Bessel functions, hypergeometric function, Riemann* ζ*-function*, ...

Also, recall a standard expression in text-books and papers: *Given a polynomial. Let* $x_1, x_2, \ldots$ *be its roots* ... But the task of obtaining roots by given coefficients is separate and complicated.

Further, mathematicians try to reduce solving of other types of equations to those. But existence of a denotation is not a solution yet. So, we proposed not "to express a continuous object in explicit form" but to try to prove computablility of continuous objects.

4. Validating Assertions for Polynomials

We will consider tasks for polynomials with integer coefficients only. Square roots can also be involved. Combinations of polynomials, operations *max, min, abs*, square roots can yield indefinitely many tasks. For example, the function

$$f(x) = |c_1x^2 - \min\{c_2x + c_3, |c_4x + c_5x^3|\}| + c_6|x|$$

with given integer constants is sufficiently complicated to avoid all "analytical" methods.

All given numbers are assumed to be integer.

We will use mathematical denotations close to ones of algorithmic languages.

Given a natural number N and numbers $A[0], A[1], \ldots, A[N]$; let $A[N] > 0$ below.

Define a polynomial

$$p(x) := \Sigma\{A[n]x^n | n = 0..N\}. \tag{4.1}$$

The simplest interval representation of (4.1) is obvious:

$$P(x_-, x_+) := \Sigma\{A[n] \cdot [x_-, x_+]^n | n = 0..N\}. \tag{4.2}$$

There is the law of subdistributivity for intervals: $X \cdot (Y + Z) \subseteq \mathrm{X} \cdot Y + X \cdot Z$. Hence, a better interval representation (also, either minorant or majorant) can be constructed by means of presenting (4.1) by Horner method

$$p_N(x) := A[N]x\textit{; for } n = N - 1 \textit{ downto } 0 \ \{p_n(x) := x(p_{n+1}(x) + A[n])\}; \\ p(x) \equiv p_0(x). \tag{4.3}$$

Consider also a function $q(x) = \frac{K}{p(x)}$ where K is a given natural number and it is given or proven that $p(x) \neq 0$; suppose that $p(x) > 0$ within a domain.

Then an interval representation of $q(x)$ is given by the formula

$$Q(x_-, x_+) := [\mathit{floor}(K/P_+(x_-, x_+)),\ \mathit{ceiling}(K/P_-(x_-, x_+))]. \tag{4.4}$$

If it is too rough then choose a natural number K_1 and define

$$Q * (x_-, x_+) := [\mathit{floor}(K_1K/P_+(x_-, x_+)), \tag{4.5}$$

ceiling $(K_1K/P_-(x_-, x_+))$];

$Q(x_-, x_+) = \frac{Q*(x_-, x_+)}{K_1}$; the factor $\frac{1}{K_1}$ is being kept mentally for forthcoming computations.

5. Examples of Tasks

Tasks will be given in two versions: on bounded domain and on unbounded one. In the last case the domain by means of some mathematical assertions must be reduced to a bounded one.

Also, we will consider only one-dimensional case for given functions and two-dimensional case for given figures. Generalization to multi-dimensional cases is reasonable only for simple data because it itself is very complicated.

Task 5.1. Given $N > 3$, a polynomial $p(x)$ of degree N and two numbers $a < b$, find an integer interval containing $p_{\min} := \min\{p(x)|a \leqslant x \leqslant b\}$.

EXAMPLE: $\min\{6x^2 - 30x + 40| - 1 \leqslant x \leqslant 8\} \in [2, 3]$ ($N = 2$ here).

Remark 5.1. Such example is necessary; otherwise many contestants will take integer values of x only: $\min\{6x^2 - 30x + 40|x = -1..8\} = 4$.

Solutions for (4.1).

1st step. Choose and build any minorant $P_-(x_-, x_+)$ for $p(x)$.

2nd step. Choose a validating algorithm of global search $\min\{p(x);\ P_-(x_-,\ x_+)|x \in [a, b]\}$.

The simplest one is of exhaustive search:

Algorithm 5.1. *Calculate* $P_{\min} = [\min\{P_-\ (k, k+1)|k = a..b-1\},\ \min\{p(k)|k = a..b\}]$.

If it does not meet Statement 2 then try to build a better minorant and repeat calculations.

More effective algorithms use bisectional search.

Describe one of them demanding the least volume of memory: successful proving of inequalities (lower bounds for $p_{\min}$).

Algorithm 5.2.

```
Denote A:= $[a, b]$;
Define array $X[]$ of intervals;
$p_{--} := P_-(A);\ p_{-+} := p(midA)$;
$X[1] := Lhalf(A); X[2] := Rhalf(A)$;
$K := 2$;
repeat
{$p_m := mid([p_{--},\ p_{-+}])$;
repeat
{$if P_-(X[K]) \geqslant p_m$
```

```
    *then K:=* $K-1$
    *else*
    $\{p_{-+} := \min\{p_{-+},\ p(\mathit{mid}(X[K])\}$; $p_m := textitmid[p_m, p_{-+}]$;
    $X[K+1] := \mathit{Lhalf}(X[K])$; $X[K] := \mathit{Rhalf}(X[K])$;
    $K := K+1\}$
    $\}$
    *until K=* $0$;
    $p_{--} := p_m\}$;
    *until* ($\mathit{Wid}(X[K]) = 1$ *or Wid*$([p_{--}, p_{-+}]) \leqslant w$ *// Statement 2//*);
```

Result. $p_{\min} \in [p_{--}, p_{-+}]$.

If the algorithm stopped due to restriction on time then a better minorant is necessary; if it done due the occurrence $Wid(X[K]) = 1$ then

- also a better minorant is to be involved;
- if we cannot do it then we are to involve narrower (fractional) intervals.

The most standard way is following. Choose a natural K_1 and substitute $x = \frac{x_1}{K_1}$, x_1 is integer, into (4.1) : $p(x) = K_1^{-N}\Sigma\{A[n]K_1^{N-n}x_1^n | n = 0..N\}$;

the factor K_1^{-N} is being kept mentally for forthcoming computations.

Such cross-partition is to be applied to intervals distinguished by Algorithm 5.2.

Remark 5.2 (cf. Remark 3.1). Those who know "formulas" would solve the task as follows. "Define $p'(x) = \Sigma\{A[n]nx^{n-1} | n = 1..N\}$, try to solve the equation $p'(x) = 0$. Let $x_1, x_2, \ldots, x_M (M \leqslant \text{ N} - 1)$ be its roots.

Calculate $p_{\min} := \min\{\min\{p(x_m)|(m = 1..M) \wedge (a \leqslant x_m \leqslant b\}, p(a), p(b)\}.''$

But this way for $N > 3(N-1 > 2)$ is much more complicated and contains practically irresolvable components.

Task 5.2. If N is even and greater than 2, find an integer interval containing $p_{\min} := \min\{p(x) | x \in R\}$.

Solution for (4.1).

1st step. Find a priori boundaries for $\arg\min p(x)$. A rough estimation is derived from the assertion: if $p(x) > p(0)$ then $p(x) > p_{\min}$. Let $|x| \geqslant 1$:

$$\begin{aligned} p(x) - p(0) &\geqslant A[N]x^N - \Sigma\{|A[n]| \cdot |x|^n | n = 1..N-1\} \\ &\geqslant |x|^{N-1}(A[N]|x| - \Sigma\{|A[n]||n = 1..N-1\}). \end{aligned}$$

Thus, we obtain the following domain for search:

$|x| \leqslant x_0$:=*max{1, ceiling* $(\Sigma\{|A[n]||n = 1..N-1\}/A[N])\}$; and the task is reduced to Task 5.1.

Consider Task 3.2. There exists the algorithm finding the number of all (real) roots of a polynomial with integer coefficients but it is too complicated. Thus, additional conditions are to be put. The simplest (and correct) version is

Task 5.3. If $A[0] < 0; A[n] \geqslant 0 (n = 1..N)$, find an integer interval containing the (unique) solution of the equation $p(x) = 0, x > 0$.

To avoid exhaustive search, $|A[0]|$ is to be very large, for example, $A[0] < -10^{40}$ for $N = 4$.

Thus, a kind of long arithmetic is to be constructed.

Solution.

```
1st step.
$x_0 = 1$;
*repeat {* $x_0 := 2x_0$ *} until* $p(x_0) > 0$.
```

2nd step. Use bisectional search (with rounding-off to integer, cf. the operation *mid*) on $[0, x_0]$ (or on $[x_0/2, x_0]$).

Task 5.4. Given $N > 2$, (large) natural K, a polynomial $p(x)(A[0] > 0, A[n] \geqslant 0 | n = 1..N)$ and two positive numbers $a < b$, find an integer interval containing the area s between the x-axis, the graph of the function $q(x) = \frac{K}{p(x)}$ and lines $x = a$ and $x = b$.

Remark 5.3. We evade the term "integral", cf. Remark 3.1.

Solution. Construct an interval representation for the function $q(x)$, see (4.4) and (4.5). The exhaustive summation for (4.5):

$$s \in \Sigma\{Q * (x, x + 1) | x = a..b - 1\}/K_1.$$

If the number $(b - a)$ is too large then bisectional partition can be used.

Algorithm 5.3.

```
*Define arrays* $X[], V[]$ *of intervals.*
$X[1] := [a, b]$;
$V[1] := Q(X[1])Wid(X[1])$;
$S := V[1]; K := 1$;
*repeat*
{
    *find one of intervals* $V[1], \ldots, V[K]$ *of the greatest width*$(V[L])$;
    $X[L] := Lhalf(X[L]); X[K+1] := Rhalf(X[L])$;
    $V[L] := Q(X[L])\, Wid(X[L]); V[K+1] := Q(X[K+1])\, Wid(X[K+1])$;
    $K := K + 1$;
    $S := \Sigma\{V[K] | x = 1..K\}$
}
*until* $Wid(S) \leqslant w$ *// Statement 2//*;
```

Result. $s \in S$.

Remark 5.4. The value of S cannot be calculated by means of using preceding value of S:

$S := S_{prec.} - V_{prec.}[L] + V[L] + V[K + 1]$ because of the following law for intervals: $Wid(A \pm B) = Wid(A) + Wid(B)$.

Hence, if the width of an interval B is positive then $Wid((A+B)-B) > \mathit{Wid}(A)$.

Remark 5.5. For given (very smooth) functions $q(x)$, the participant may venture to use any approximate formula for definite integrals (an estimation of a remainder term is practically impossible), but for given non-smooth functions it would be in vain.

Task 5.5. Given a polynomial $p(x, y)$ (such that $p(x, y) > 0$ for $|x| >> 1$ or $|y| >> 1$). Find an integer interval containing the area of the figure $F = \{(x, y) \in R^2 | p(x, y) < 0\}$.

Solution.

1st step. Build an interval representation $[P_-, P_+]$ for p.

2nd step. By means of a rough estimation find any interval vector $[a_-, a_+] \times [b_-, b_+]$ containing F (interval representation of F).

3rd step. The exhaustive summation (*Num* means the number of elements of the set):

$$\begin{aligned}
&\mathit{Area}(F) \in [Num\{(x,y)|P_+(x,x+1,y,y+1) < 0,\\
&\quad x = a_-..a_+ - 1, y = b_-..b_+ - 1\},\\
&(a_+ - a_-)(b_+ - b_-) - Num\{(x,y)|P_-(x,x+1,y,y+1) > 0,\\
&\quad x = a_-..a_+ - 1,\ y = b_-..b_+ - 1].
\end{aligned}$$

If the number $(a_+ - a_-)(b_+ - b_-)$ is too large then binary partition can be used.

Algorithm 5.4.

```
Define array Z[ ] of 2-dimensional interval vectors.
$Z[1] := [a_-, a_+] \times [b_-, b_+]$;
$A_- := 0;\ A_+ := (a_+ - a_-)(b_+ - b_-);\ K := 1$;
repeat
{find one of $Z[1], \ldots, Z[K]$ of the greatest width $(Z[L])$;
$P_Z := P(Z[L])$;
if $0 \in P_Z$ then
{$Z[L] := \mathit{Lhalf}(Z[L]); Z[K+1] := \mathit{Rhalf}(Z[L])$;
$K := K + 1$}
else
{{if $P_{Z+} < 0$ then $A_- := A_- + \mathit{Area}(Z[L])$
else
$A_+ := A_+ - \mathit{Area}(Z[L])$}};
$Z[L] := Z[K]; K := K - 1$}
}
until $\mathit{Wid}([A_-, A_+]) \leqslant w$ // Statement 2 //;
```

Result. $\mathit{Area}(F) \in [A_-, A_+]$.

6. Tasks of Discrete Content Arising from Continuity

The following types of tasks are of “common” discrete content but we did not meet such ones at informatics olympiads. They are solved by same methods.

Task 6.1. Given a polynomial $p(x)$ and two numbers $a < b$, find
$P_{\min} := \min\{p(x)|x = a..b\}$ (and find all $\{x \in a..b|p(x) = P_{\min}\}$).

Task 6.2. If N is even, find
$P_{\min} = \min\{p(x)|x\text{is integer}\}$ (and find all $\{\text{integer}x|p(x) = P_{min}\}$).

Task 6.3. Given a polynomial $p(x)$ and two numbers $a < b$, find

$$\{x = a..b - 1|p(x)p(x + 1) \leqslant 0\}.$$

Task 6.4. Given a polynomial $p(x)$, find all
$\{integerx|p(x)p(x + 1) \leqslant 0\}$.

Remark 6.1. This task is not equivalent to the task of separating all (real) roots of a polynomial. For example, if $p(x) = 9(x - \frac{1}{3})(x - \frac{2}{3}) = 9x^2 - 3x + 2$ then $p(x) > 0$ for all integer x.

Such tasks are more interesting for multi-dimensional cases where exhaustive search is evidently too slow. But a simple generalization may be incorrect. For example, the following task:

Given a polynomial $p(x, y)$, find
$\{\mathit{integer}(x, y)|0 \in \mathit{Outer}(\{p(x, y), p(x+1, y), p(x, y+1), p(x+1, y+1)\}$ is incorrect: it has infinitely many solutions for $p(x, y) = x + y$.

Task 6.5. Given a polynomial $p(x, y)$ (such that $p(x, y) > 0$ for $|x| >> 1$ or $|y| >> 1$) and natural M. Find *Num*$\{p(x, y)$ *<0 | x and y are integer}* (*mod M*).

7. Conclusion

We hope that using tasks of proposed above types would attract attention of computer science students and computer scientists to the problem of validation of common approximate caculations and would expand the scope of tasks on olympiads in informatics of various levels. Also, distinguishing constructive (i.e., realizable on computer) methods among all mathematical ones would be useful in forthcoming professional activity of contestants of olympiads.

Acknowledgements. I thank Ulan Degenbaev for correction of the text.

References

Moore, R.E. (1966). *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.Y.

Pankov, P.S. (1978). *Validating Computations on Electronic Computers* (in Russian). Ilim Publishsing House, Frunze, Kyrgyzstan. Review: MR 82a:65004.

Pankov, P.S. (1978). A combined method for proving certain theorems of mathematical analysis via a computer. *Cybernetics* (translation of *Kibernetika*, Institute of Cybernetics, Kiev, Ukraine), 14(3), 441–448.

Pankov, P.S., Dolmatov, S.L. (1979). Substantiable evaluations by electronic computers and their application to one problem in combinatorial geometry. *Information Processing Letters*, 8(4), 202–203.

Pankov, P.S., Bayachorova, B.D., Yugai, S.A. (1982). Numerical theorem proving by electronic computers and its application in various branches of mathematics. *Cybernetics* (translation of *Kibernetika*,), 18(6), 840–848. Review: MR84i:68158.

Pankov, P.S. (1987). Machine presentation of information on continuous objects. *Scientific-Technical Information*. Series 2: information processes and systems, 2, 17–23 (in Russian).

Pankov, P.S., Janalieva, J.R. (1996). *Computer Mathematics*, Part I (in Russian). International University of Kyrgyzstan, Bishkek.

Pankov, P.S., Altynnikova, L.A., Janalieva, J.R. (2002). *Computer Mathematics*, Part II (in Russian), International University of Kyrgyzstan & Osh State University, Osh.

Shanin, N.A. (1968). Constructive real numbers and constructive function spaces. *Translation of Mathematical Monographs*, Vol. 21, American Mathematical Society, Providence, Rhode Island.

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 National 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 professor of the International University of Kyrgyzstan.

Olympiads in Informatics, 2013, Vol. 7, 113–122
© 2013 *Vilnius University*

Olympiads in Informatics – the Journal's First Six Years

SIMON
The University of Newcastle
Ourimbah Campus, PO Box 127, Ourimbah, NSW 2258, Australia
e-mail: simon@newcastle.edu.au

Abstract. This paper analyses the first six volumes of *Olympiads in Informatics*, examining the papers that have been published in these volumes and the authors who have written them. It finds that the journal is truly international, and that its authors appear to conform with the pattern expected of a discipline, notwithstanding that at this point it has only a short history and a small pool of publications. With regard to the papers themselves, the analysis finds that they cover a reasonable range of topics within the overall area of informatics olympiads, and that while they are predominantly reports, over the six years they show a pleasing increase in the number of analytical research papers.

Key words: olympiads in informatics, bibliometrics, paper classification.

1. Introduction

"*Olympiads in Informatics* is a refereed scholarly journal that provides an international forum for presenting research and development in the specific area of teaching and learning informatics through competition. The journal is focused on the research and practice of professionals who are working in this field." (Dagienė *et al.*, 2008).

After six annual volumes, it is timely to examine the journal and assess its contribution to research and development in the field. Who is contributing to the journal? Where do the authors come from? How many papers has each author written? To what extent are authors collaborating, forming a community of practice? What are the papers about? How many of the papers are factual reports, which would fall into the 'practice' category, and how many are more suited to the description 'research'?

This paper reports on a bibliometric study of the papers published in *Olympiads in Informatics* and of the authors of those papers. Bibliometric analysis of this sort is reasonably common in the library sciences, and has been applied to publications in numerous discipline areas including accounting (Chung *et al.*, 1992), crystallography (Behrens and Luksh, 2006), bioinformatics (Patra and Mishra, 2006), and computing education (Simon 2009a, 2009b). It has potential merit as the journal equivalent of genealogical research – it helps us to understand who we are ('we' in this case being the journal's authors) and where we have come from. It thus strengthens the notion of the informatics olympiad community as a community and not just a set of authors who happen to present at the same conference and publish in the same journal.

The goal of the study, then, is to answer the question: what patterns can be discerned in the papers that have been published in the first six volumes of *Olympiads in Informatics*, and in the authors who have written them?

2. The Papers Being Considered

Any analysis of the papers in a particular publication must begin by clearly establishing which papers are to be considered and which are to be omitted from the analysis.

Volumes 1–4 of the journal included only a foreword or editorial and the peer-reviewed papers. A foreword is not generally considered to be a published paper, so the forewords were not included in the analysis.

Volumes 5 and 6 saw the introduction of a new section called 'Reviews, comments'. This section is described as being for "book reviews, comments on task solutions and other initiatives in connection with teaching informatics in schools". While it is not clear whether these contributions undergo the same sort of peer review as the papers, they are clearly treated in a different manner – for example, they are not listed individually in the table of contents – so it was decided to exclude them, restricting the analysis to the full papers that have clearly undergone peer review prior to acceptance in the journal.

By this criterion, there have been 101 full papers published in the six volumes of *Olympiads in Informatics*, and it is these 101 papers that will be analysed in this work.

3. The Authors

As with any other journal or conference, some papers have single authors while others have multiple authors. Likewise, there are authors who have contributed only one paper to the journal and authors who have contributed more. To deal with this many-to-many relationship we introduce an intermediate entity called the author contribution. This term should not be interpreted as suggesting that we are trying to measure *how much* each author contributed to each paper – this is something known only to the authors. Rather, the author contribution is a uniform measure of a single author's authorship of a single paper.

There are a number of different ways of counting an author's contribution to a paper (Larsen, 2008). In the system known as *complete counting*, each author of a paper is given a count of 1 for that paper, regardless of the number of authors. If a paper has four authors, each of the four will be given a count of 1, and the paper will register four contributions. In *complete-normalised counting*, on the other hand, the paper itself is given a count of 1, which is then divided equally among the authors. If a paper has four authors, each of the four will be given a count of 0.25.

Each system has its merits and its drawbacks. Complete-normalised counting recognises that the work of writing a joint paper is shared among its co-authors, and so gives each author less credit. But at the same time it could be seen as devaluing collaboration, by giving the authors of a two-author paper only half the credit that either would have earned by writing the paper alone. For this reason, this analysis will place more emphasis on complete counting, but will acknowledge the differences between it and complete-normalised counting.

Using the complete counting method, the 101 papers that have appeared in the first six volumes of the journal together comprise 213 author contributions; that is, there is an average of just over two authors per paper. The greatest number of papers (42) have single authors; nearly as many (35) have two authors; there are 14 three-author papers, six four-author papers, and one each of papers with five, nine, ten, and eleven authors. It can be argued that shared authorship is a good thing, as collaboration is one clear measure of engagement within the community (Simon, 2007). However, this should not be seen as diminishing the value of single-author papers.

There are 130 distinct authors who have contributed to papers published in the journal.

Lotka's law of author productivity (Nicholls, 1989) encapsulates the empirical observation that in a sufficiently large list of published papers within a discipline, 60% of authors will contribute to only one paper, 15% to two papers, 7% to three papers, and so on. More precisely, given a total pool of A authors, the number of authors A_n contributing to n papers will be CA/n^p, where C and p are constants that vary according to the discipline but are generally expected to be close to 60% and 2 respectively. Having validated Lotka's law on multiple diverse data sets, Nicholls suggests that values of 71% to 81% are more realistic for C than 60%.

Although this is not the most rigorous approach, C can be trivially estimated from the case where $n = 1$, that is, the proportion of authors who have contributed to just one paper. For *Olympiads in Informatics* this estimate gives a value of 68.5%, which sits comfortably between the generally quoted 60% and Nicholls's subsequent observations, suggesting that *Olympiads in Informatics* has a reasonable number of authors who have contributed two and more papers.

Figure 1 shows the observed numbers of authors contributing to specified numbers of papers, alongside the numbers predicted by Lotka's law. The power constant p has been set at 2.0 to give a good visual match with the observed counts of contributions. If this constant were any lower the curve would drop more quickly, indicating that very few authors come back to write further papers after their first.

The message from Figure 1 is that the papers in *Olympiads in Informatics* follow reasonably closely the expectations for a large list of publications within a single discipline. This is very positive, as the journal has been running for only six years, and it is not entirely obvious that informatics olympiads can be considered a discipline. It will be interesting to see whether the pattern continues as the number of issues increases.

Table 1 shows the same observed author contributions as in Figure 1, and lists the authors who have contributed to four or more papers in the six volumes of the journal.

Table 2 shows the journal's leading authors according to the complete-normalised counting system, which counts for each author a fraction of each paper according to how many authors the paper has.

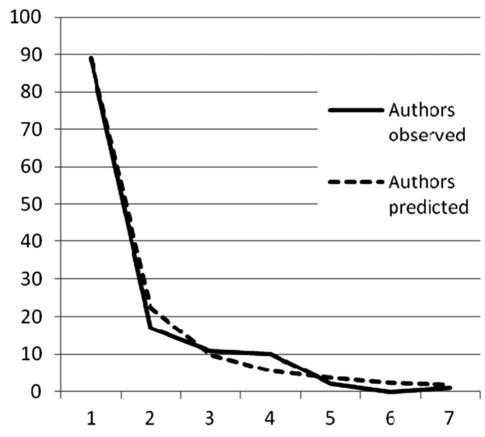


Fig. 1. Lotka's Law, with C=68.5% (from number of 1-paper authors) and p=2.0 (best fit by eye).

Table 1
Author contributions using complete counting, listing the authors with four or more papers

Papers	Author count	Authors
7	1	Pavel S Pankov
5	2	David Ginat, Marcin Kubica
4	10	Benjamin A Burton, Michal Forišek, Mathias Hiron, Emil Kelevedjiev, Vladimir M Kiryukhin, Krassimir Manev, Martin Mareš, Bruce Merry, Wolfgang Pohl, Tom Verhoeff
3	11	
2	17	
1	89	

Table 2
Leading authors using complete-normalised counting

CN Count	Authors
4.5	Pavel S. Pankov
4.0	Tom Verhoeff
3.5	Martin Mareš
3.0	Vladimir M. Kiryukhin
2.8	David Ginat, Bruce Merry
2.6	Benjamin A. Burton, Michal Forišek
2.2	Wolfgang Pohl
2.1	Willem van der Vegt
2.0	Marina S. Tsvetkova
1.9	Krassimir Manev
1.8	Emil Kelevedjiev

While Pavel S. Pankov is the clear leader by both counting systems – and also has the distinction of being the only author to have been published in all six volumes of the journal – the other rankings are quite different under the two counting systems. Tom Verhoef, for example, moves from equal fourth with complete counting to clear second with complete-normalised counting, because he was the sole author of all four of his papers. Conversely, David Ginat drops from equal second to equal fifth, and Marcin Kubica from equal second to fourteenth (not shown on Table 2), because their five papers were shared with various numbers of co-authors.

It is important to repeat that each of these counting systems has its merits and its drawbacks; and, of course, to point out that any author ranked in the top dozen by either system should be proud of that achievement.

4. The Countries

Olympiads in Informatics is a truly international journal. From the outset it has published papers from a wide range of countries. The first volume explicitly called for state-of-play reports from countries participating in the International Olympiad in Informatics, and accepted papers from Brazil, Bulgaria, Canada, China, Croatia, the Czech Republic, Germany, Italy, Kyrgyzstan, Lithuania, Macedonia, Mongolia, Poland, Portugal, the Russian Federation, Slovakia, and the USA. In subsequent issues these countries were joined by a further 23, making a total of 40 countries represented in the journal. As there are authors who have had more than one paper accepted over the years, so there are necessarily countries represented many times over (though there is one author who changed countries between publications). Table 3 shows the 40 countries represented in the 101 papers, with the number of papers that have come from each. A final row in the table notes the eight papers with authors from two or more countries; these papers are examples of the international collaboration that is almost certain to arise from a venture such as the International Olympiad in Informatics.

5. The Topics of Papers

What are the journal's papers about? Of course they are all related to the overriding theme of informatics olympiads, but within that theme they can be about quite different topics. For example, there will be papers about the national organisation of olympiads (Anido and Menderico, 2007), about the creation and choice of tasks (Burton and Hiron, 2008), and about the grading of tasks (Merry, 2010). Each of these topics could be further divided – for example, grading could be divided into automatic grading (Mareš, 2009) and manual grading (Pohl, 2008) – but this would lead to a proliferation of topics with very few papers in each, and so would make it more difficult to present a broad overview of what the papers are about.

Some papers clearly deal with more than one topic. In such cases, rather than trying to identify every topic, no matter how small its contribution, it was decided to classify each paper according to the most dominant of the topics it covers.

Volume 1 of the journal was based on the first Olympiads in Informatics Conference in 2007, which explicitly focused on organising olympiads at the national level. “Many of the issues at the national level differ from country to country. We have different educational systems and the availability and take-up of information technology varies, but even here there are as many similarities as differences. We also face many of the same problems: How do we pick our students? How do we train them? What is suitable material? etc” (Dagienė *et al.*, 2007). Therefore it is no surprise that almost all of the papers in that volume cover the topic of *organisation*.

The second conference, and therefore Volume 2, had a dual theme including task types. “Tasks are perennial issue for contests, their most visible aspect and, for many contestants, the primary reason for participation. We strive for quality, variety and suitability. We endeavour to make tasks interesting, understandable and accessible. They are used to test contestants and to train them, and perhaps even to capture the imagination of those outside the contest, be they family, sponsors or the media” (Dagienė *et al.*, 2008). Many of the papers in Volume 2 were therefore on the topic of *tasks*.

Overall, six distinct topics were identified across the six volumes of the journal, as summarised in Table 4. Each topic is briefly described below.

- *organisation*
  This is a broad topic covering many aspects of the organisation of olympiads, typically at the national level. It includes levels of competition, the logistics of getting students to suitable venues and of training them, costs and budgets, timing, and many other aspects of how an olympiad is organised.
- *tasks*
  This topic deals with many aspects of the tasks used in olympiads: task selection, examples of tasks, new styles of task, and other related matters.
- *grading*
  Papers with this topic describe or propose ways to grade the students’ submissions in an olympiad.

Table 3
The countries from which the papers have come, with a count of papers from each country

Albania	1	Finland	2	Lithuania	1	Singapore	1
Australia	3	France	2	Macedonia	1	Slovakia	3
Belgium	2	Germany	2	Mexico	1	Slovenia	1
Brazil	1	Ghana	1	Mongolia	1	Spain	1
Bulgaria	9	Indonesia	1	New Zealand	1	Switzerland	1
Canada	2	Israel	6	Poland	4	Syria	1
China	2	Italy	4	Portugal	3	Thailand	1
Croatia	1	Japan	2	Romania	1	The Netherlands	6
Czech Republic	3	Kyrgyzstan	8	Russian Federation	5	United Kingdom	3
Estonia	1	Latvia	1	Serbia	1	United States	2
International	8						

Table 4

The six identified topics and their distribution across the six volumes

Volume	Organisation	Tasks	Grading	Preparation	Infrastructure	Impact
1	13	3	1			
2	5	8	2	1		
3	3	6	3	1	1	
4	7	2	3	3		
5	7	5	2	3	1	1
6	7	5	3		4	1
Total	42	29	14	8	6	2

- *preparation*
  These papers deal with the preparation of students for olympiads, that is, preparing students prior to an olympiad. Papers about the training of students once they are in the olympiad are included in the *organisation* topic.
- *infrastructure*
  Papers with this topic describe or propose infrastructure to be used in running olympiads. They might, for example, discuss the way that wireless networks were used for submission of the competitors' work (Imajo, 2011).
- *impact*
  Only two papers were indentified within this topic, but it was not possible to ascribe those papers to any of the other five topics. These are papers that clearly deal with the impact of the olympiads in broader areas of education (Audrito *et al.*, 2012) and recruitment (Jakacki *et al.*, 2011).

6. The Natures of Papers

Independently of their topics, the papers have been divided into three categories specifying more about the nature of the work they present.

- *report*
  Report papers describe the past and/or present situation, or sometimes propose future situations. They are essentially capturing and reporting on existing factual knowledge.
- *software*
  Software papers are also reports, but of a particular kind. Rather than describing existing knowledge, they present and describe software that has been designed and constructed for a specific purpose related to the olympiads. For example, a *software* paper whose topic is *grading* will describe software that has been created to assist with the grading of students' submissions (Maggiolo and Mascellani, 2012).

Table 5
The three identified natures and their distribution across the six volumes

Volume	Report	Software	Analysis
1	16	1	0
2	12	1	3
3	7	4	3
4	12	2	1
5	11	0	8
6	6	7	7
Total	64	15	22

- *analysis*
  Analysis goes well beyond reporting, gathering data and analysing it to produce hitherto unknown results. The data might be pre-existing (Forišek, 2009) or gathered expressly for the analysis (Merry, 2010).

Table 5 shows the breakdown of papers by their natures. While there is of course great value in reporting existing facts, it is good to see the recent growth in the number of analysis papers in the journal, as these papers are more readily perceived as research (Simon, 2007), and make a clear contribution to the journal's expressed aim of publishing high-quality research.

7. Correlating Nature and Topic

Table 6 groups the papers according to both their topic and their nature. It is interesting to see that a clear majority of the analysis papers focus on the olympiad tasks, with a few focusing on organisation and a few on grading. It is not surprising that much of the software presented in the journal focuses on grading and on infrastructure. And while the bulk of the reports focus on olympiad organisation and on tasks, overall the reports cover the full range of topics.

Table 6
Papers according to topic and nature

	Organisation	Tasks	Grading	Preparation	Infrastructure	Impact
Report	39	14	3	6	1	1
Software		1	8	1	5	
Analysis	3	14	3	1		1

8. Conclusion

This analysis help to form a picture of the authors and papers of *Olympiads in Informatics*, a picture that will help members of the olympiads community to better know and understand that community.

Even though it has been going for only six years, and has published only 101 papers, the journal demonstrates a pattern of authorship that looks remarkably like a discipline when examined in the light of Lotka's Law.

The papers published in the journal are predominantly reports – not least because the first conference and the first volume explicitly called for such reports – but there is a clearly perceptible increase in the number of analytical papers being published.

This analysis supports the view of *Olympiads in Informatics* as a scholarly journal that provides an international forum for presenting research and development in the teaching and learning of informatics through competition.

References

Anido, R.O., Menderico, R.M. (2007). Brazilian olympiad in informatics. *Olympiads in Informatics*, 1, 5–14.

Audrito, G., Demo, G.B., Giovannetti, E. (2012). The role of contests in changing informatics education: a local view. O*lympiads in Informatics*, 6, 3–20.

Behrens, H., Luksh, P. (2006). A bibliometric study in crystallography. *Acta Crystallographica*, B62, 993–1001.

Burton, B.A., Hiron, M. (2008). Creating informatics olympiad tasks: exploring the black art. *Olympiads in Informatics*, 2, 16–36.

Chung, K.H., Pak, H.S., Cox, R.A.K. (1992). Patterns of research output in the accounting literature: a study of the bibliometric distributions. *Abacus*, 28(2), 168–185.

Dagienė, V., Cepeda, A., Forster, R., Manev, K. (2007). Foreword. *Olympiads in Informatics*, 1, 3–4.

Dagienė, V., Cepeda, A., Forster, R., Manev, K., Vasiga, T. (2008). Foreword. *Olympiads in Informatics*, 2, 3–4.

Forišek, M. (2009). Using item response theory to rate (not only) programmers. *Olympiads in Informatics*, 3, 3–16.

Imajo, K. (2011). Contest environment using wireless networks: a case study from Japan. *Olympiads in Informatics*, 5, 26–31.

Jakacki, G., Kubica, M., Waleń, T. (2011). Codility. Application of olympiad-style code assessment to pre-hire screening of programmers. *Olympiads in Informatics*, 5, 32–43.

Larsen, P.O. (2008). The state of the art in publication counting. *Scientometrics*, 77(2), 235–251.

Maggiolo, S., Mascellani, G. (2012). Introducing CMS: a contest management system. *Olympiads in Informatics*, 6, 86–99.

Mareš, M. (2009). Moe – design of a modular grading system. *Olympiads in Informatics*, 3, 60–66.

Merry, B. (2010). Performance analysis of sandboxes for reactive tasks. *Olympiads in Informatics*, 4, 87–94.

Nicholls, P.T. (1989). Bibliometric modeling processes and the empirical validity of Lotka's law. *Journal of the American Society for Information Science*, 40(6), 379–385.

Patra, S.K., Mishra, S. (2006). A bibliometric study of bioinformatics literature. *Scientometrics*, 67(3), 477–489.

Pohl, W. (2008). Manual grading in an informatics contest. *Olympiads in Informatics*, 2, 122–130.

Simon (2007). A classification of recent Australasian computing education publications. *Computer Science Education*, 17(3), 155–169.

Simon (2009a). Informatics in Education and Koli Calling: a comparative analysis. *Informatics in Education*, 8(1), 101–114.

Simon (2009b). Ten years of the Australasian Computing Education Conference. *Australian Computer Science Communications*, 31(5), 157–163.

Simon is a senior lecturer in information technology at the University of Newcastle in Australia. Several years ago he devised a scheme for classifying publications in computing education, and together with a small team he has published a number of papers in that area. He also conducts research in programming education and in academic integrity.

Olympiads in Informatics, 2013, Vol. 7, 123–131
© 2013 *Vilnius University*

EMAx: Software for C++ Source Code Analysis

Emil STANKOV, Mile JOVANOV, Aleksandar BOJCHEVSKI,
Ana MADEVSKA BOGDANOVA
Faculty of Computer Science and Engineering, University Ss. Cyril and Methodius
Rugjer Boshkovikj str. 16, Skopje, Macedonia
e-mail: emil.stankov@finki.ukim.mk, mile.jovanov@finki.ukim.mk,
aleksandar.bojchevski@gmail.com, ana.madevska.bogdanova@finki.ukim.mk

Abstract. Source code analysis is the process of extracting information about a program from its source code. In this paper we discuss the need for source code analysis and we present our model for a system that represents source codes as a vector of attributes. We outline the main modules of the proposed system (the parser, the executor and the parameterization module), and provide the implementation details for each module. The system that is presented has been designed to be used as a tool that provides vector representations of program solutions further used in solving the problem of source code comparison. Also, the presented system can be easily adapted and used for similar problems. It offers a unique idea for source code representation, and allows fast production of such representations.

Key words: source code analysis, source code analysis tool, programming, program evaluation.

1. Introduction

Source code analysis is an important field in computer science. Source code analysis is the process of extracting information about a program, from its source code. According to Harman (2010), the term "analysis", in this context, means any procedure that takes a source code and gives us insight into its meaning.

Considering the significant amount of code that is being produced every day, it's clear that we need more tools to analyze and understand that code. Furthermore, given the complexity of modern software, tools based on source code analysis are being increasingly used to improve performance and productivity.

Source code analysis has many applications into a variety of software engineering tasks, including: clone detection, debugging, source code optimization, source code comparison, reverse engineering, performance analysis, and many others.

Information about a program can be extracted from its source code or artifacts (e.g., from Java byte code or execution traces) generated from the source code using automatic tools (Binkley, 2007). However, given the complexity of modern software, the manual analysis of code (source code, intermediate, or machine code) is costly and ineffective. A more viable solution is to resort to tool support. Such tools provide information to programmers that can be used to coordinate their efforts and improve their overall productivity (Da Cruz *et al.*, 2009).

In this paper we present our new tool for source code analysis called EMAx. The tool has been created for the needs of a wider research in the area of source code analysis, specifically source code comparison. The paper is organized in the following manner. Section 2 briefly describes the existing models for source code analysis and illustrates one particular setting where the proposed tool can be used for evaluation of source code assessment. In Section 3 we describe the architecture and the implementation of EMAx. We outline the key modules and their specific implementations. In Section 4 we analyze the performance of the tool and describe some of its features. As a conclusion, Section 5 points out further directions for possible expanding and enhancement of its capabilities.

2. Related Work

2.1. *Existing Models for Source Code Analysis*

As stated in Binkley (2007), a system for source code analysis commonly has three components: the parser, the internal representation and the analysis of this representation. In essence, every step of the analysis converts the concrete syntax into a better abstract syntax suited to the particular analysis.

The parser parses the source code into one or more internal representations. Because of the complexities of modern programming languages, in particular those that are not LR(1), it is difficult to create an effective parser. For example, C++ is not a LR(1) language. Its grammar is ambiguous, context-dependent and potentially requires infinite look-ahead to resolve some ambiguities (Willnik, 2001).

The second major part of the model is the internal representation. There are many forms of internal representations. Some classic examples include the control-flow graph (CFG), the call graph, and the abstract syntax tree (Fischer and LeBlanc, 1988). Another popular internal representation is the static single-assignment (SSA) form, which modifies the control-flow graph in such a way that every variable is assigned exactly once, thus making def-use chains explicit (Cytron *et al.*, 1991).

The analysis of the representation can be static or dynamic. Static analysis concerns techniques for obtaining information about the possible states that a program passes through during execution, without actually running the program on specific inputs. Hence, static-analysis techniques explore a program's behavior for all possible inputs and all possible states that the program can reach (Reps *et al.*, 2004). In contrast, dynamic analysis takes into account the program's current input; however, the results are only guaranteed to be correct for the given input.

Construction of an abstract model of the program is the first step of a typical source code analysis. Clearly, it's easier to analyze a model that is not programming language specific, instead of working directly with the source code. In many cases, the first model that is created is the abstract syntax tree (AST) – a tree where each node is a construct in the source code (Kirkov and Agre, 2010). AST is usually used as a base for creating more complex graph structures (models) representing various aspects of the source code, and therefore different models are used by different source code analysis algorithms.

2.2. *Source Code Analysis in the Source Code Assessment*

Programming courses at university and high school level (especially introductory ones) often include lots of exercises in order to ease the adoption of the programming language syntax, and also to help the students to develop algorithmic way of thinking. Since programming is a compulsory course in every computer science educational curriculum, usually lots of computer science students enroll in these courses. This leads the course lecturers to the problem of mass number of solutions to exercises that have to be graded – the assessment can no longer be done manually in a reasonable amount of time.

The need for fast assessment has also been perceived in the organization of competitions in informatics. Nowadays, programming competitions require the participants to submit program source codes – solutions to concrete algorithmic problems that are given to them. Generally, the difficulty of the competition is not so much in the programming part, as it is in the design of appropriate algorithms for solving the problems at hand.

In most cases, these competitions are based on automated assessment of the submitted solutions. The automation of the assessment is necessary not only because of the large amount of solutions, but also in order to have the results in reasonable time. This is accomplished by running them on batches of input data and testing correctness of the output by comparing it to the expected output. Additionally, time and memory limits are usually enforced during this evaluation process, which allows obtaining an assessment not only in terms of the correctness of the solutions, but also in terms of their time and space complexity (Mares, 2007). Thus, the efficiency of the algorithms used is also taken into consideration.

The same or slightly similar method can be used as a solution to the previously mentioned problem of fast assessment of program codes in educational environment. There are many existing systems that are used for this purpose (Ihantola *et al.*, 2010), and the benefits are numerous, as described in Trusso *et al.* (2007).

However, the grading of the programming solutions outlined above is quite rough and strict. The grade (usually expressed in terms of number of gained points) assigned to a particular program may give a completely wrong impression about how good (and efficient) is the algorithm that it implements. As an illustration, in an extreme case, this type of automatic assessment would assign zero points to a program that, in essence, represents an implementation of a complete and 100% correct algorithm for solving the problem at hand, but uses a wrong format when printing the output data. For this reason the organizers of the international programming contests nowadays have to spend a lot of efforts to prevent such extreme cases (such as the use of feedback, and submission of multiple solutions to allow the contestants to catch these kinds of problems).

The question that we pose is the following: Is there an automated way to determine similarity of a code (the one that scored low or zero points on the grading system) to another code (that scored full score), in order to reconsider the grading of the first one?

Our wider research aims to produce an approach that can reveal a logical/semantic connection between the codes, using data mining methods, and we believe that our approach can give better results because it is not restricted by some demands as when the task is only to detect plagiarism. Our approach consists of three main stages: (1) creating parse trees for each of the source codes under consideration, (2) extracting attributes that represent key characteristics of the source codes by calculating metrics from the obtained parse trees, and (3) applying data mining clustering methods on the dataset formed by these attribute representations in order to discover the existence of similarities among them.

The EMAx software that we present in this paper has been created for the main purpose of conducting the first two steps of our procedure for source code similarity detection.

3. The Architecture of the Proposed Tool for Source Code Analysis

EMAx was created for the purpose of extraction of attributes which are used for source code similarity detection. According to Roy and Cordy (2007), source code similarity detection algorithms can be classified as based on either: strings, tokens, parse trees, program dependency graphs (PDGs), metrics or hybrid approaches. EMAx tool takes a hybrid approach.

Parse trees are used, in the first step of building the model, because they allow for high level similarities to be detected. In the second step, metrics are calculated from the parse trees. Metrics capture "scores" of code segments according to certain criteria; for instance, "the number of different variables used" or "the number of if statements".

We can analyze the proposed model as a black box, where we have a set of source codes as input and a set of corresponding vectors of integer values as output. A high level view of the proposed model for the system is depicted in Fig. 1.

According to our observations, the architecture of the proposed system should be composed of several modules, each with input-process-output structure, where the output of one module is the input for the next one. The key modules of the proposed system are:

- Parser – this module generates an abstract syntax tree (AST) for a given source code. The output of this module gives us a "static" view of the source code. The generated AST contains every little detail about the code, including: preprocessor directives, macro expansions, comments, tree of nodes representing the syntax along the C++ grammar, names for all declarations and references, scopes, etc.
- Executor – this module takes the AST generated by the parser and builds an AST representing the execution of a source code from a given starting point. The output

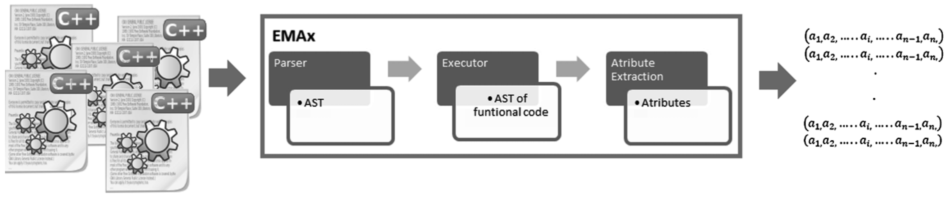

C++
EMAx
Parser
• AST
Executor
• AST of funtional code
Atribute Extraction
• Atributes
$(a_1,a_2, \ldots a_i, \ldots a_{n-1},a_n)$
$(a_1,a_2, \ldots a_i, \ldots a_{n-1},a_n)$
$(a_1,a_2, \ldots a_i, \ldots a_{n-1},a_n)$
$(a_1,a_2, \ldots a_i, \ldots a_{n-1},a_n)$

Fig. 1. High architectural view of the EMAx tool.

of this module gives us a simulated view of the possible execution of the source code. Given a starting point (usually a function), the module follows the function call hierarchy and creates the AST of the code that gets executed by the processor. It's a key module, because it provides an additional level of abstraction and bypasses the organizational decisions within a source code. It enables source code comparison on a semantic level. This means that the parts of the code that will not execute are not taken into consideration in the later analysis. This module builds an internal representation of the AST, with additional meta-data that will help the analysis.

- Attribute extraction module – this module takes the AST built by the executor and creates a vector of integer values (attributes) that represent key characteristics of the code. The attributes represent a measure for the programming structures and constructs used, as well as the relationships between them. For example, one attribute may count the number of conditional structures used (e.g., if/else statements), and another attribute may represent the relationship between a conditional structure and a loop.
- Auxiliary modules – modules for handling user interaction, displaying results, exporting data in various formats, etc.

We now describe the EMAx's implementations of each of the key modules that are outlined above. EMAx's parser uses the Eclipse CDT (C/C++ Development Tooling) parsers. CDT contains two parsers: for C and C++. These are known as DOM (Document Object Model) parsers. The parsers are compatible with the GCC suite of compilers and accept a range of GCC extensions. CDT uses the concept of a Translation Unit that represents a single source code file, as well as all the header files that are included by that file. The parser outputs an abstract syntax tree (AST) that is used as input in the next module.

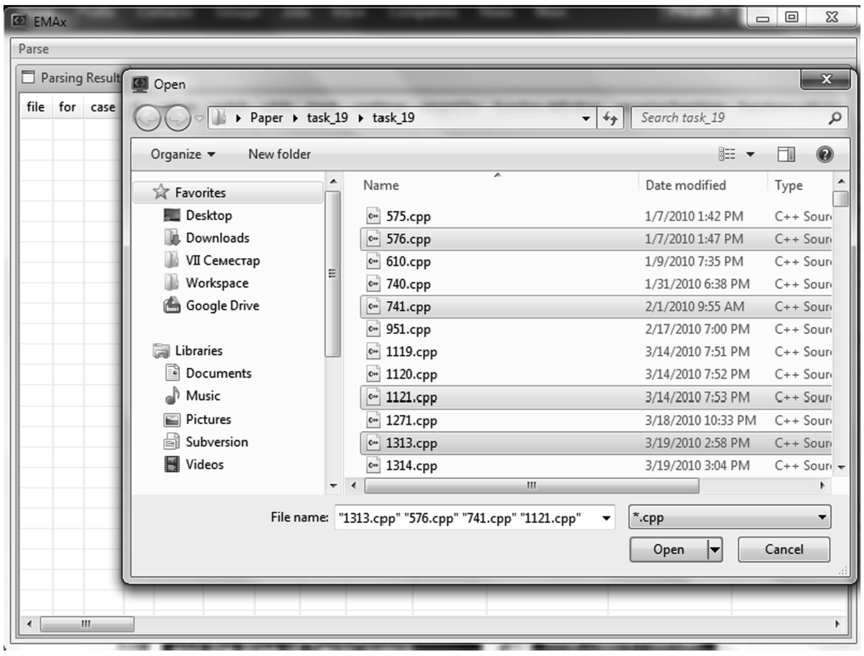
EMAx's executor uses the AST generated by the CDT parser and creates an internal representation of the AST in XML format, with additional meta-data. The XML is created and later on parsed in memory to achieve better performance. The visitor looks for a user defined starting function in the original AST. Then, it follows the function call hierarchy and combines nodes from the original AST to create the new AST with additional meta-data. The meta-data include information about the current function, and track the control structures for easier parsing. Additionally, this module includes mechanisms to avoid generating potentially infinite ASTs as a result of recursive calls or circular dependencies. Furthermore, the module takes into account function calls to external libraries (e.g., the sort function in the "algorithm" library).

EMAx's parameterization module consists of two main sub-modules. The first sub-module uses the visitor design pattern defined by Eclipse CDT. The tree traversal with a visitor is depth first and can be done in both top-down and bottom-up manners. This sub-module is used to compute some of the attributes, regarding classes, inheritance, function definitions and C++ directives. The second sub-module uses the AST generated by EMAx's executor module in XML format. It employs XPath (XML Path Language) queries for computing the values of the attributes. Most of the attributes are computed by this sub-module. Both sub-modules are written in Java.

Handling the user interaction and displaying the results is done using the Eclipse Rich Client Platform (RCP). Additionally, there are modules for exporting the results of the analysis in various formats (e.g., CSV format).

EMAx has been designed to be simple and easy to use. The specification of a starting function and the selection of input source files for analysis is a simple task. The interface for loading the source codes is shown in Fig. 2.

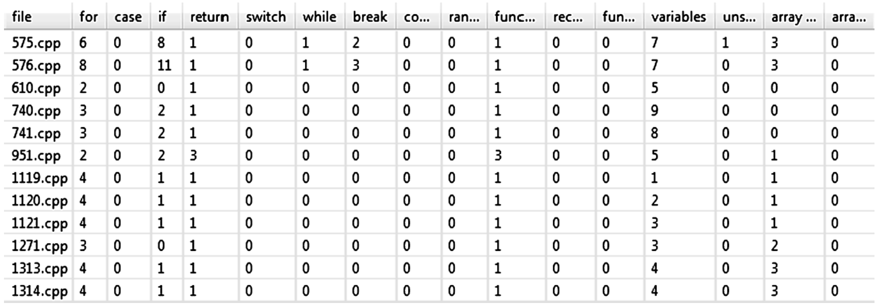
The results from the analysis are presented in a table to the user in a minimalistic manner. The user can arrange the order of the attributes and choose a subset of the attributes that he/she is interested in. Fig. 3 shows a small part of an output produced by the tool for a set of C++ source codes given as input to it.



Name	Date modified	Type
575.cpp	1/7/2010 1:42 PM	C++ Sour
576.cpp	1/7/2010 1:47 PM	C++ Sour
610.cpp	1/9/2010 7:35 PM	C++ Sour
740.cpp	1/31/2010 6:38 PM	C++ Sour
741.cpp	2/1/2010 9:55 AM	C++ Sour
951.cpp	2/17/2010 7:00 PM	C++ Sour
1119.cpp	3/14/2010 7:51 PM	C++ Sour
1120.cpp	3/14/2010 7:52 PM	C++ Sour
1121.cpp	3/14/2010 7:53 PM	C++ Sour
1271.cpp	3/18/2010 10:33 PM	C++ Sour
1313.cpp	3/19/2010 2:58 PM	C++ Sour
1314.cpp	3/19/2010 3:04 PM	C++ Sour

File name: "1313.cpp" "576.cpp" "741.cpp" "1121.cpp" *.cpp

Fig. 2. EMAx's interface for selection of input source files.



file	for	case	if	return	switch	while	break	co...	ran...	func...	rec...	fun...	variables	uns...	array ...	arra...
575.cpp	6	0	8	1	0	1	2	0	0	1	0	0	7	1	3	0
576.cpp	8	0	11	1	0	1	3	0	0	1	0	0	7	0	3	0
610.cpp	2	0	0	1	0	0	0	0	0	1	0	0	5	0	0	0
740.cpp	3	0	2	1	0	0	0	0	0	1	0	0	9	0	0	0
741.cpp	3	0	2	1	0	0	0	0	0	1	0	0	8	0	0	0
951.cpp	2	0	2	3	0	0	0	0	0	3	0	0	5	0	1	0
1119.cpp	4	0	1	1	0	0	0	0	0	1	0	0	1	0	1	0
1120.cpp	4	0	1	1	0	0	0	0	0	1	0	0	2	0	1	0
1121.cpp	4	0	1	1	0	0	0	0	0	1	0	0	3	0	1	0
1271.cpp	3	0	0	1	0	0	0	0	0	1	0	0	3	0	2	0
1313.cpp	4	0	1	1	0	0	0	0	0	1	0	0	4	0	3	0
1314.cpp	4	0	1	1	0	0	0	0	0	1	0	0	4	0	3	0

Fig. 3. Example output of EMAx (partial view).

4. Features and Performance of the Tool for Source Code Analysis

Having in mind the motivation for creating this system (source code similarity detection), it's clear that the system must be able to handle large sets of data. Furthermore, it has to produce results as fast as possible.

As described above, one of the sub-modules of EMAx's parameterization module uses XPath queries to compute the values of the attributes. Moreover, most of the queries are fixed and don't depend on the specific source code that is being analyzed. After running a few experiments, we realized that the compilation of XPath queries took most of the processing time. This gave us an opportunity for optimization. One part of the optimization was keeping a map of pre-compiled queries, which significantly reduced (nearly in half) the processing time.

Another part of the optimization was reducing the number of traversed nodes in the original AST. Specifically, we traverse only those nodes that are of interest in the process of extraction of attributes.

In order to evaluate the performance of the system, we conducted some experiments. We took large sets of source codes from algorithmic competitions, and we tested them with our system. The results from the experiments are presented in Table 1. Each entry in this table refers to a set of source codes that represent solutions to the same problem. From the results we can conclude that the processing time is highly dependable on the number of source lines per code.

In general, the complexity of the algorithm used for generating each source code's attributes is between O(N) and O(N$*$N), where N is the size of the source code in bytes. This complexity of the algorithm is a consequence of the complexities of the algorithms for creation of the parse trees, creation of the XML representation of the parse trees and evaluation of the XPath queries. Of course, the evaluation of the XPath queries on the XML representation takes up most of the processing time.

Because the system was developed in Java, using Eclipse RCP, it is able to run on various operating systems (e.g., Windows, Linux).

EMAx allows selection of a subset of attributes to be exported and analyzed which offers flexibility to any user of the tool. The system is also capable of automatic discovery

Table 1
Performance analysis of the system

Average size of source codes in SLOC (source lines of code)	Number of source files tested	Total time for completion for all source files in seconds	Average time of completion per one source file in seconds
∼ 16.2	150	∼19	∼0.127
∼ 25.3	290	∼40	∼0.137
∼ 25.1	541	∼70	∼0.130
∼ 20.17	504	∼51	∼0.101

of source codes in a directory. This feature allows easier automation of the systems that could potentially utilize this tool.

We have not found any (descriptions of) source code analysis tools that perform the same or similar function as EMAx in order to make a comparison.

5. Conclusion

In this paper we discussed the need for source code analysis. We provided an overview of systems for source code analysis, with particular focus on systems for source code comparison. Then, our model for a system that generates source code vector representations was presented. We outlined the main modules of the proposed system (the parser, the executor and the parameterization module), and we provided the implementation details for each module.

The presented system has been designed to be used as a tool that provides vector representations of source codes further used in solving the problem of source code comparison. It has been tested with real sets of source codes taken from programming competitions and has proved as very efficient in performing the task for which it is intended. Also, the presented system can be easily adapted and used for similar problems. One of the main advantages that the system offers is that it analyzes the source codes by simulating execution of the code from a given starting point. The software is highly extensible and adding support for additional languages (besides C++) is easy.

Some of our ideas for future improvement include: addition of support for extraction of user defined attributes, further optimization of the XPath queries, addition of more languages from the C/C++ family (e.g., Java), and addition of automation capabilities.

Acknowledgements. The research presented in this paper is partly supported by the Faculty of Computer Science and Engineering in Skopje.

References

Binkley, D. (2007). Source code analysis: a road map. In: *2007 Future of Software Engineering (FOSE '07)*, IEEE Computer Society, Washington, DC, USA, 2–5.

Cytron, R., Ferrante, J., Rosen, B., Wegman, M., Zadeck, K. (1991). Efficiently computing static single assignment form and the control dependence graph. *ACM Trans. Prog. Lang. Syst.*, 13(4).

Da Cruz, D., Henriques, P.R., Pinto, J.S. (2009). Code analysis: past and present. In: *Proceedings of the Third International Workshop on Foundations and Techniques for Open Source Software Certification (OpenCert 2009).*

Fischer, C.N., LeBlanc, R.J. (1988). Crafting a compiler. *Benjamin/Cummings Series in Computer Science*, Benjamin/Cummings Publishing Company, Menlo Park, CA.

Harman, M. (2010). Why source code analysis and manipulation will always be important. In: *Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM '10)*, IEEE Computer Society, Washington, DC, USA, 7–19.

Ihantola, P., Ahoniemi, T., Karavirta, V., Seppala, O. (2010). Review of recent systems for automatic assessment of programing assignments. In: *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, New York, NY, ACM.

Kirkov, R., Agre, G. (2010). Source code analysis – an overview. In: *Cybernetics and Information Technology*, 10(2), 60–77.

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

Reps, T., Sagiv, M., Wilhelm, R. (2004). Static program analysis via 3-valued logic. In: *CAV*, 15–30.

Roy, K., Cordy, J.R. (2007). A survey on software clone detection research. *School of Computing*, Queen's University, Canada.

Trusso Haley, D., Thomas, P., De Roeck, A., Petre, M. (2007). Seeing the whole picture: evaluating automated assessment systems. In: *ITALIC E-Journal of the Learning and Teaching Subject Network for Information and Computer Science (LTSN-ICS)*, 6(4), 203–24.

Willnik, E.D. (2001). Meta compilation for C++. PhD thesis, University of Surray

E. Stankov is a teaching and research assistant at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. He is a member of the Executive Board of the Computer Society of Macedonia and has actively participated in the organization and realization of the Macedonian national competitions and olympiads in informatics since 2009. Currently he is finishing his master studies at the Faculty of Computer Science and Engineering.

M. Jovanov is a teaching and research assistant at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. He is the president of the Computer Society of Macedonia and has actively participated in the organization and realization of the Macedonian national competitions and olympiads in informatics since 2001. Currently, he is preparing his PhD thesis on the topic of online collaborative ontology building in e-learning environment.

A. Bojchevski is an undergraduate student at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. His areas of interest include algorithmic programming, data mining and intelligent systems. Currently, he is finishing his studies.

A. Madevska Bogdanova is an associate professor at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. She is a member of the Computer Society of Macedonia and has participated in the organization and realization of the Macedonian national competitions and olympiads in informatics. Her research interests are in the field of intelligent systems, bioinformatics, machine learning and ICT in education. She is an author of over forty scientific papers, chapters in books in the area of computer science and presentations for popularization of informatics.

Olympiads in Informatics, 2013, Vol. 7, 132–139
© 2013 *Vilnius University*

Predicting the Difficulty Level of a *Bebras* Task

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 *Bebras* contest questions are marked as easy, medium or hard. Sometimes contestants perform other than expected. Since pretesting is impossible, we want to find out what kind of tasks were misplaced and develop some guidelines for predicting the difficulty level.

Key words: *Bebras* contest, task design, question difficulty.

1. Introduction

The *Bebras* contest is an international contest on informatics and computer fluency amongst the young (Bebras, 2013). Students from over twenty countries compete in their national contest. The questions used in these contests are chosen from an international task pool. No prior knowledge is required. The contest is about computer science, algorithms, structures, information processing and applications. Criteria for good *Bebras* tasks are formulated by Dagiene and Futchek (2008).

Contestants compete in their own age division. In the Netherlands we use a division for 12–14 years, the Cadets, for 14–16 years, the Juniors and for 16 years and up, the Seniors.[1] In the Netherlands the contest consists of 15 short questions, divided in three difficulty levels: easy, medium and hard. A right answer for a question is awarded with 6, 9 or 12 points (depending on the difficulty level); for a wrong answer 2, 3 or 4 points are subtracted. Skipping the question will not alter your score. Every contestant starts with 45 points, so no contestant will get a negative score; answering all questions correct gives the top score of 180. Contestants have 40 minutes to complete these 15 tasks. The contest runs for a week; the best performing contestants for every age division are invited at a university for a second round (Beverwedstrijd, 2013). Like in many countries, organizing the *Bebras* contest is done by the organizers of the Dutch Olympiad in Informatics.

One of the problems we face is that it is hard to predict the difficulty level of a task. In most cases we use the indications available in the task pool. But when we select questions that are best suitable for the Dutch contest, we sometimes end up with a lot of questions of the same difficulty level, so we need to move a few tasks to a different difficulty category. On other occasions we use questions that were designed for other age groups to complete a contest.

[1] In other countries the contest is also for age groups 10–12, the Benjamins, and sometimes even for 8–10, but our national organization is only connected with the schools of secondary education.

In this paper we will analyze the results of six different contests. A simple measure is used for comparing our predictions of the difficulty level. We will also discuss some research from other areas about predicting question difficulty. Finally we will develop a questionnaire that can be helpful for designing future contests.

2. The Difficulty Level of a Task

The easiest way to look at the difficulty of a task is to look at the result. A task that is solved by a majority of the contestants is definitely easier than a task that is solved by a small percentage. Two of the graphs of the percentages of good answers are presented in Figs. 1 and 2. With results like the one in Fig. 1 it is obvious that we made a couple of bad decisions regarding the difficulty level. In an ideal situation the five tasks in which contestants performed best should have been placed in the easy category, the five tasks in which they performed worst in the hard category. But looking this way, amongst the intended easy questions was at least one that should have been characterized as hard; four questions should have been moved to a higher difficulty level, five others to an easier one. For the Seniors we predicted rather well. Figure 2 shows that only one of the questions was designed to be medium, but turned out to be hard.

An easy measure for the quality of our predictions is the percentage of misplaced tasks. In the contest presented in Fig. 1 this was 60%, in Fig. 2 it was 13%. Table 1 gives an overview of this measure, applied to the six contests we organized in 2011 and 2012.

Are these proportional success rates in Figs. 1 and 2 reliable? Is it allowed to use these rates as a proper indication of the difficulty of a task for an age division? Well, let's state that the answer for a specific question is the result of a chance experiment. The chance for success for every experiment is an unknown value q. When we take a test sample of n contestants, the success rate p is an obvious estimator for q. The 95%-confidence interval has a maximum radius of $0.98/n$ for $p = 0.5$. With for example 1000 contestants this is less than 0.001 or 0.1%. So it is reasonable to say that the success rate p for a question can be used to describe the difficulty level q of that question.

How well did we predict the results of groups of questions? We may have misplaced a couple of question, but was the contest as a whole the way we intended, with a group of

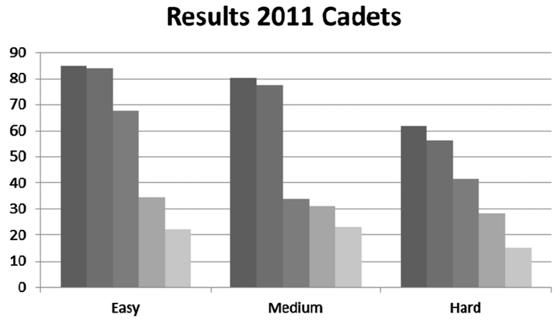


Fig. 1. Results of the 2011 contest for the Cadets.

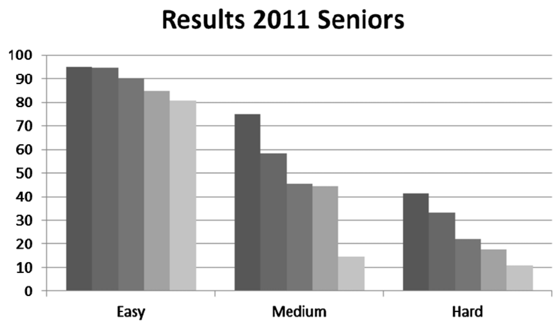


Fig. 2. Results of the 2011 contest for the Seniors.

Table 1

Percentage of misplaced tasks in 6 contests

Year	Age division	Harder than predicted	Easier than predicted	Percentage misplaced
2011	Cadets	4	5	60%
	Juniors	4	5	60%
	Seniors	1	1	13%
2012	Cadets	3	3	40%
	Juniors	1	1	13%
	Seniors	3	3	40%

Table 2

Year	Age division	Easy questions	Medium questions	Hard questions
2011	Cadets	58.64	49.18	40.74
	Juniors	57.22	67.99	30.12
	Seniors	89.13	47.61	25.11
2012	Cadets	65.14	51.48	37.08
	Juniors	72.72	58.45	29.44
	Seniors	75.09	64.93	37.17

easy questions, a medium and a hard one? The mean scores for the questions in a difficulty group are presented in Table 2. In five of the six contests the results are as expected. Only the contest for the Juniors in 2011 shows a bad result for this mean scores, where the mean score for the easy questions is much below the score for the medium intended questions. This was one of the contests where 60 % of the questions turned out to be misplaced. In this cases swapping the best answered medium question and the worst answered easy question would have made a difference of 12 % in the mean scores for the difficulty group.

3. Question Difficulty

A technique often used in test design is pretesting. A set of possible questions will be answered by a test panel, in order to compare the results, to judge about the suitability of specific questions and to get a proper indication of the difficulty level. Due to the way the international task pool for *Bebras* tasks is constructed, pretesting is not an option.

A lot of exams have a stage between doing the test and publishing he results. This gives the organizers the possibility to skip questions and to define the cut between those who passed the exam and those who did not. But the *Bebras* contest is not an exam, it is a competition. Not the score itself, but the ranking of the scores is important. Knowing the predefined difficulty level of a question is part of the contest. It is possible that contestants take this into account when answering a question, so it would be strange to change the difficulty level after the contest took place.

These considerations leave us no other option than to think ahead, and to try to estimate the difficulty level of a question in advance. Of course, it is possible to play with the parameters of a question to manipulate the difficulty level somewhat.

But this manipulation of question difficulty is a complex matter. Many question-setters try to achieve this merely on their intuition, of course based on experience with similar questions. Though *Bebras* contest designers are often right in describing and predicting the difficulty level of a task, this endeavour is an inexact science (Dhillon, 2003). Most of the research on question difficulty is done within specific subject domains. Dhillon however presents an overview of research results that are useful also for task settings like the *Bebras* contest.

Ahmed and Pollitt (1999) distinguish three kinds of difficulties in questions. Cognitive difficulty has to do with the concepts that are used in a question. The level of abstraction of these concepts will determine this difficulty. Process difficulty is about the difficulty of the cognitive operations and the degree in which they use cognitive resources. Question difficulty is connected with the linguistic and structural properties of a question. Dhillon uses these three kinds of difficulties, and she makes a further distinction between 'intrinsic' question difficulty, which is content-bound, and 'surface' question difficulty, which is format-bound. In our discussion we will stick to the intrinsic question difficulty.

Sternberg suggests that intelligence can be understood in terms of the information-processing components underlying complex reasoning and problem-solving tasks such as analogies and syllogisms. Sternberg used information-processing and mathematical modeling to decompose cognitive task performance into its elementary components and strategies (MIT Encyclopedia of Cognitive Science, 2013). He analyzes analogy item types, with the form 'A is to B as C is to ?'. A series of six roughly sequential, sometimes cyclical, cognitive steps have to be taken to find a solution of an analogy problem. The different stages are encoding ('How to think about A and B'), inference ('What is the relation between A and B?'), mapping ('What is the relation between A and C?'), application ('How to transform C according to the discovered pattern?'), justification ('Is this really the intended answer?') and response ('This is what I have found.'). Sternberg proposed that the difficulty level of a question as a whole is comprised of the sum of the difficulties of each of the individual components, multiplied by the number of times each component was executed (Dhillon, 2003). Several researchers are able to show that this approach can be useful. Increasing the number of elements involved in a question increases the time needed for solving the problem. But the error rates behaved slightly different. The number of transformations per element had the greatest impact on item difficulty (Dhillon, 2003). The assumption of linearity in analogy item-difficulty breaks down at higher levels of item complexity, due to the limits of the short-term memory capacity. Miller (1956) points out that the span of immediate memory imposes severe limitations on the amount of information that we are able to receive, process and remember. If it is impossible to solve a task using only working memory, the solution process will take much more time and will be more error-prone. Sternberg's method still offers a clear operational guide to the construction of analogy problems and related question types; the decomposition of a question in elements and operations on these elements gives a first indication of the expected difficulty level.

Ahmed and Pollitt (1999) describe a model of the question answering process for students that have to respond to a reading comprehension item. Typical stages in this answering process are reading, understanding, searching the mental representation of the text, interpreting and composing the answer. Then they observe that understanding a subject is simply a macro-level version of comprehending a language. So question answering can similarly be broken down in corresponding processes. Awareness of the processing stages will not only facilitate the identification and manipulation of legitimate sources of difficulty, but also the identification and elimination of illegitimate sources of difficulty (Dhillon, 2003). The question-setter can better foresee hurdles at which contestants will fall through no fault of their own.

Katz *et al.* (2002) investigate the predictive validity of various features of generating examples test items, algebra problems that pose constraints and ask for example solutions. Students will use a generate-and-test method to find proper answers. The difficulty level of a question can be raised by increasing the amount of testing the student must do. Both the cognitive load and the potential for error would be increased, either by increasing the number of constraints or by reducing the solution density of the item (Dhillon, 2003).

4. Two Sample Problems

We took two typical *Bebras* tasks, to analyze if we can connect the results with the theory on question difficulty.

All four beavers have a hat but something went wrong (Fig. 3).

- *All four beavers have the wrong colour hat at the moment;*
- *With the hats arranged correctly, none of the beavers have the same colour hat as the colour of the shirt;*

To which beaver belongs the green (G) hat?

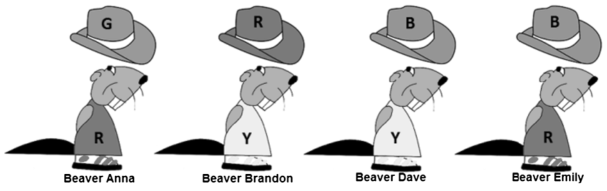


Fig. 3. Task Wrong Hat – *Bebras* Task Pool 2011.

The task Wrong Hat in Fig. 3 was used in the 2011-contest for Cadets. We changed the proposed difficulty level from medium to easy. The success rate was 83.86%. It was a multiple choice question, and in order to find the right solution a contestant has to apply two constraints on all possible answers. One could argue that the solution space has 24 different hat distributions, but application of the first constraint reduces the actual solution space to two possible answers. The solution process fits without problems in the working memory of a contestant. The story is short and easy to understand. So it makes sense to identify this task as an easy question.

The task Three Elephants in Fig. 4 turned out to be the hardest problem that we used in the 2011-contest, with a success rate of 5.60% for Juniors and of 10.95% for Seniors. The task was interactive, but only the used programming buttons were shown, not the resulting state. This task requires close reading and solving it will take a lot of different steps. The position of every elephant has to be remembered, all possible states have to be identified and checked. It is hard to do without additional utilities, like pencil and paper, because it goes beyond the boundaries of the working memory. Experts will recognize the principle behind the task and they will use a Gray code or the analogy with the problem of traversing the vertices of a cube. But since the *Bebras* is a contest that requires no previous knowledge, we have to assume that contestants are not familiar with these problems. The difficulty level of this task could be reduced in several ways. One could of course skip one of the elephants. This would reduce the search space and the number of the consecutive steps, and this would reduce the difficulty level in a serious way. It is also possible to present the problem as a multiple choice question, reducing the number of possible answers and changing the nature of the question. In that case the question is about checking the possible solutions with the given constraints, rather than producing a solution itself. A third way to change the nature of the question is by giving some visual aid in the presentation on the screen, for instance by showing the state after each programmed step. This would make it a lot easier for a contestant to keep track and it would almost certainly reduce the unwanted errors in solving this problem.

A circus has an act where three elephants perform on stage. Only one elephant at a time can enter the stage or leave the stage. The director wants to show all combinations of one or more elephants on stage exactly one time.

You get six buttons to produce a program for the elephant show. You can use a button more than once; just drag and drop a button to the program list.

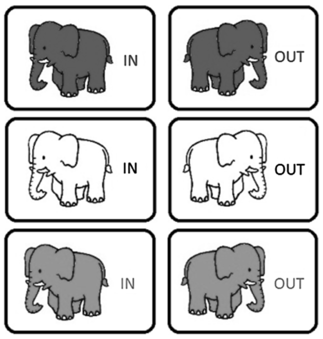


Fig. 4. Problem Three Elephants – *Bebras* Task Pool 2011.

5. Evaluation

Estimating the difficulty level of a *Bebras* task is done merely by intuition. Intuition is “the ability to acquire knowledge without interference and/or the use of reason”[2]. In some cases the intuition of tasks designers failed; tasks turned out much easier or much harder than expected. Results from research can be used to sharpen the intuition of task designers. We propose the use of a questionnaire that can be used to discover probable causes of difficulty. The answers can be used to compare questions and to discuss the estimation of the difficulty level of the question.

Table 3

Questionnaire for difficulty level estimation

I.	The question answering process
a.	Which problems will there be in reading the question?
b.	Which problems will there be in understanding the question?
c.	Which problems can arise in searching the mental representation of the text?
d.	Which problems can arise when interpreting the answer?
e.	Which problems can arise when composing the answer?
II.	The size of the problem
a.	What is the number of elements in the question?
b.	What is the number of transformations for an element in the question?
c.	What is the number of constraints in the question?
d.	How do you rate the solution density of the problem?
e.	Will it be possible to solve the problem, using only your working memory?

Next year we intend to use this questionnaire and to gather data, to investigate whether it is possible to reach a better prediction of difficulty level for *Bebras* tasks. Estimating the difficulty level will not turn into an exact science. But the use of the result of previous result should help us to improve the outcome of our intuition.

[2]Oxford English Dictionary.

References

Ahmed, A., Pollitt, A (1999). *Curriculum Demands and Question Difficulty*. Paper presented at IAEA Conference, Slovenia, May.

Bebras website (2013).
`http://bebras.org/`.

Beverwedstrijd (2013).
`http://www.beverwedstrijd.nl/` (in Dutch only).

Dagiene, V., Futschek, G. (2008). *Bebras* international contest on informatics and computer literacy: criteria for good tasks. In: Mittermeier, R.T., Syslo, M.M. (Eds.), *ISSEP 2008, LNCS*, 5090, 19–30, Springer-Verlag Berlin Heidelberg.

Dhillon, D. (2003). *Predictive Models of Question Difficulty – A Critical Review of the Literature*. Manchester, AQA Centre for Education Research and Policy.

Katz, I.R., Lipps, A.W., Trafton, J.G. (2002). *Factors Affecting Difficulty in the Generating Examples Item Type*. GRE Board Professional Report No. 97-18P. Princeton, NJ, Educational Testing Service.

Miller, G. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychology Review*, 63, 81–97.

MIT Encyclopedia of Cognitive Science (2013). *Intelligence*.
`http://ai.ato.ms/MITECS/Entry/sternberg.html/`.

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, 2010) and Enschede (2008). He also is one of the task designers for the *Bebras* contest.

Olympiads in Informatics, 2013, Vol. 7, 140–152
© 2013 *Vilnius University*

Informatics Everywhere: Information and Computation in Society, Science, and Technology

Tom VERHOEFF
Department of Mathematics and Computer Science, Eindhoven University of Technology
Den Dolech 2, 5612 AZ, Eindhoven, The Netherlands
e-mail: t.verhoeff@tue.nl

Abstract. Informatics is about information and its processing, also known as computation. Nowadays, children grow up taking smartphones and the internet for granted. Information and computation rule society. Science uses computerized equipment to collect, analyze, and visualize massive amounts of data. Scientific theories more and more involve computational models. Technology incorporates computational mechanisms, even on a nano, molecular, or quantum scale, in every imaginable product. A story.

Key words: information, computation, society, science, technology.

1. Introduction

In this article, I tell a story of why informatics is highly relevant and extremely exciting as a scientific discipline. Of course, informatics professionals already know all this (I would hope), but do IOI participants know this story? To develop and apply informatics further, we need to mobilize fresh talent and bring them up to speed. IOI participants are good candidates, but they often need some further coaxing and coaching beyond algorithms and programming.

As the title suggests, I will touch on five areas, but do so in a different order.

Society is pervaded with informatics: information and its (automated) processing – known as computation, in informatics lingo – are everywhere.

Information and computation: two key concepts united in informatics. No information without computation, no computation without information.

Science is about understanding 'the world' as it is; information and computation turn out to be indispensable tools for this; not only in analyzing data for scientific research, but also in defining models and theories. Biology, even chemistry, physics, social sciences, psychology, etc. revolve around information and computation.

Technology is about putting 'the world' to our use; most technological products, including medicines, involve ways to exploit nature for dealing with information and computation.

2. Society

Society, and I mean human society in particular, has always been centered around information and its processing. Compared to other species, human beings, as individual organisms, lack many physical characteristics – think of sheer strength, weaponry, armor, speed, camouflage – to survive among other organisms and the violence of nature. What has made us so successful is our capacity to communicate with each other and operate in organized groups. Information connects us and makes it possible to transcend our individual limitations.

In primitives times, information mostly concerned communication about the location of food and shelter, about danger and safety. It also concerned the tracking of human relationships. Group operations require organization, which typically followed blood lines, and a healthy population requires the avoidance of incest. Later, notions such as property, trading, and money got institutionalized. These require administration, which at its core concerns information. Also, farming and hunting benefit from storage and communication of information, no matter how primitive.

Information used to be handled via simple carriers, such as sounds, marks on rocks and trees, knots in ropes, signs on clay tablets, parchment, and papyrus. Information was known only in concrete physical forms. The development of writing systems and subsequently of book printing meant big leaps forward. That way, it became clear that the same language can live in diverse carriers. The information revolution was catapulted by our ability to harness information digitally and decouple it completely from physical carriers. That is, instead of just focusing on the physical information carriers, we are able to view information as something abstract and not necessarily physical. Almost any signal can be digitized and subsequently processed in the abstract digital domain. Such digital signals can then be translated either into physical actions or signals that we can understand. Informatics deals with information and computation in ways that abstract from the physical carriers.

Over time, we have also discovered and developed new ways of observing the world (sensors) and interacting with it (actuators). Information storage and processing used to be done by persons, and was therefore slow and error-prone. Through specialized equipment, the handling of information has now been automated to such an extent that, in numerous cases, it can be delegated to autonomous man-made systems. I will elaborate this point in Section 6.

Information is even more important to be successful nowadays than it was in the past. Informatics can make and break a society. For individuals, the world has become rather more complex, not so much physically, but informationally. You need the ability to tap very diverse sources of information: in order to select proper food, health care, and products, to deal with your finances and property, to plan your travels and navigate the roads, to interact with the government and companies, to connect to relatives and friends. Moreover, organizations and institutions themselves (government, military, education, law, commerce, industry, entertainment) have become more information centered. The internet, though primitive by some standards, has clearly made its mark. Cyberspace started as a of virtual world, but has become an integral part of our everyday reality. This poses new problems and dilemmas, whose understanding requires familiarity with informatics: digital ownership and copyright, privacy, identity, and (in the future) even existence. Gleick (2011) tells more of this story.

3. Information

So, what is information? Can you deal with information without dealing with some physical carrier of that information? In one sense, information is that which reduces uncertainty or, to put it differently, answers a question. The unit of information is the *bit*, which represents the answer to a question with two *equiprobable* answers. Typically, such an answer is denoted abstractly by the choice between a 0 and a 1. This notion of information does not involve specific physical carriers. Physically, a bit can be represented in many ways, for instance, the direction of magnetization of a small region in a material (such as a hard disk; soon to disappear), or the presence/absence of a package of electrons in a (semi)conductor (in your smartphone). More about this in Section 6.

Claude Shannon developed this *probabilistic information theory*. If you successively answer 100 yes/no questions where the answers are not equiprobable, you can convey the answers in less than 100 bit (data compression). If you have a communication channel that introduces some random errors (noise), then you can throw in a few well-chosen extra bits (redundancy) to protect against accidental loss of information (error correction). And when you want to prevent unauthorized persons from tapping your information, you can jumble up (encrypt) the bits to protect them against eavesdropping (cryptography). Every imaginable piece of information, for instance, numbers, lists of numbers, text, and pictures, can be encoded in appropriate sequences of bits. All of this is abstract, mathematical. In that sense, informatics is a science of the artificial (Simon, 1996), and not a natural science (but also see Section 5). Note that there is more to information than Shannon's probabilistic theory of information, (Adriaans and van Benthem, 2008).

Data compression, error correction, cryptography, and encoding all involve (abstract) operations on information. In informatics we refer to such information processing as *computation*, even if the information does not involve numbers but involves text, graphics, etc. Bits become information – get meaning – through computations that operate on these bits and use them to make decisions. Such computations could take place in a brain or a computer. Computation is the topic of the next section. Interestingly, even if you try to confine yourself to the world of information, you will discover that computation can emerge in that world. Let me illustrate that.

Consider the famous *Game of Life* (GoL) defined by Conway (Berlekamp *et al.*, 2004; Ch. 25). It is 'played' on an unbounded 2D grid of square cells, where each cell is in one of two states: dead or alive (or, if you prefer bits, 0 or 1; see Fig. 1 left). Each cell is said to have eight neighbors: two horizontal, two vertical, and four diagonal. In the next generation, a dead cell will become alive when it currently has exactly three

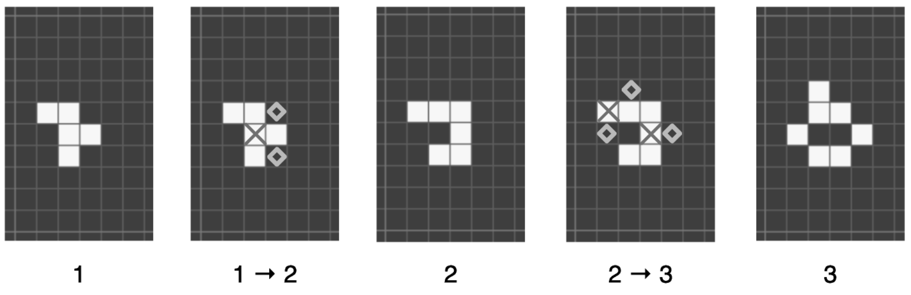


Fig. 1. Three successive generations in the Game of Life: a white cell is alive, a black cell dead, a diamond marks a newly born cell, and a cross marks a dying cell.

live neighbors, a live cell dies out when it has fewer than two or more than three live neighbors; otherwise, a cell does not change state.

Figure 1 shows three successive generations of a Game-of-Life configuration that starts with five live cells. Going from generation 2 to 3, one sees that

- three new live cells appear (diamonds);
- one cell dies from starvation, having only one live neighbor (left cross);
- one cell dies from crowding, having four live neighbors (right cross).

You might think of this grid of bits as a pure 'information world', that evolves according to a simple rule. You can experiment with the Game of Life (and similar worlds) using *Golly* (Golly, 2013). It turns out to be an interesting world. In fact, it is highly unpredictable, even though the rules are deterministic. That is, given an initial configuration, there is no general way of telling how it will evolve, e.g., whether it will die out completely, stabilize with some live cells, become periodic, or 'explode'.

How do we know this? Because we have found out that we can exploit this 'information world' to do arbitrary computations for us (Berlekamp *et al.*, 2004; Ch. 25). Given any computation, you can construct a Game-of-Life configuration that will simulate that computation. Figure 2 shows a configuration that 'generates' twin primes; that is, it 'searches' for twin primes, and whenever it 'finds' a pair it will 'report' that in a way that is recognizable from the configuration. The question whether there are infinitely many twin primes is then equivalent to whether there are infinitely many reportings during the evolution of that Game-of-Life configuration. Since we do not know whether there are infinitely many twin primes, the behavior of this configuration is not (yet) predictable.

Anything that can be computed, can be 'computed' by an appropriate Game-of-Life configuration (the Game of Life is *universal*). Thus, even in a pure 'information world' that was not designed for doing general information processing, the possibility of general computation can emerge, provided the 'information world' is 'sufficiently rich'. You can 'see' the computational aspects when you put on the right kind of interpretational 'glasses'.

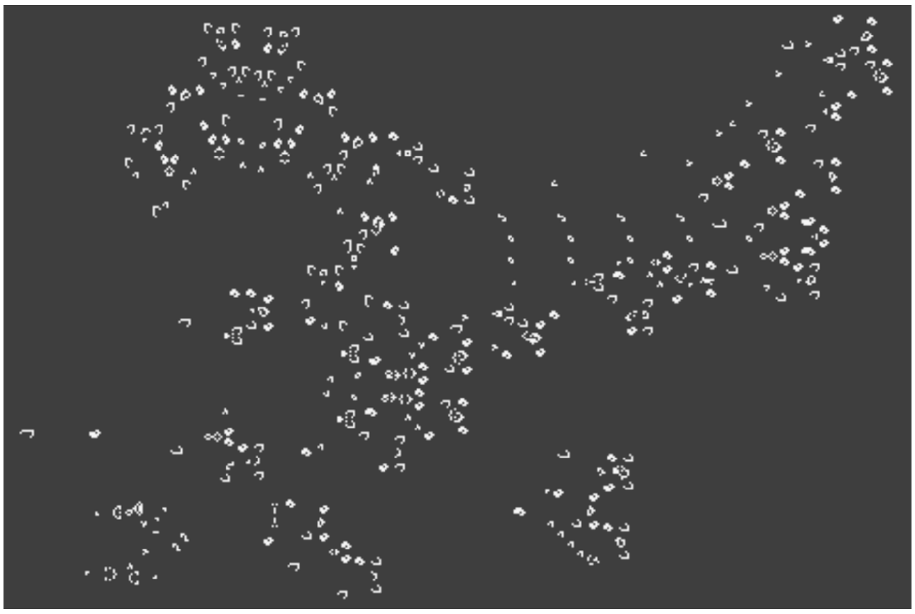


Fig. 2. Game-of-Life configuration to report twin primes; it starts with 3062 live cells (in white).

4. Computation

The counterpart of a pure ‘information world’ is a pure ‘computation world’. Can you have computation without information? The *lambda calculus* and, even more intriguing, *combinatory logic* can be viewed as pure computation worlds. In lambda calculus and combinatory logic, there are only functions, in the mathematical sense. These functions take one argument and return one result, and both the argument and the result are a function. A function of two parameters, such as $add(x, y) = x + y$ must be treated as a function of one parameter that returns a function of one parameter: $add(x)$ returns a function, say with parameter y, that adds x to its argument y. In the notation of lambda calculus:

$$add = (\lambda\, x\, .\, (\lambda\, y\, .\, x + y)). \tag{1}$$

A term of the form $(\lambda\, x\, .\, t)$ is called a *λ-abstraction*; in it, all free occurrences of x in t are said to be bound by the $\lambda\, x$.

Another term in the lambda calculus is *application of function f to argument x*, written as $(f\, x)$. Parentheses can be omitted under the assumption that function application is left-associative. That is, $f\, x\, y$ means $((f\, x)\, y)$, and thus $add\, x\, y = x + y$.

In lambda calculus, you express which functions are applied to what arguments. It abstracts completely from the nature of the ‘data’ passed in the arguments, like passing around envelopes without ever opening them. Only a few rules are needed to define the meaning of lambda terms. In particular, there is *β-reduction*, that defines the effect of applying a λ-abstraction $(\lambda\, x\, .\, t)$ to a term u:

$$(\lambda\, x\, .\, t)\, u = t[x := u], \tag{2}$$

where $t[x := u]$ denotes a *capture-avoiding substitution*: it is obtained from t by replacing every free occurrence of x in t by u, 'avoiding capture' of free variables in u by λ-abstractions occurring inside t (if needed, bound formal parameters inside t are systematically renamed through so-called α-conversion). The technical details are easy to find, so we will skip them here (Moore and Mertens, 2011; Chu-Caroll, 2013).

To illustrate this shuffling of unopened envelopes, consider the following function f defined by

$$f = (\lambda\, x\, .\, (\lambda\, y\, .\, (\lambda\, z\, .\, x\, z\, y))). \tag{3}$$

Thus, $f\, x\, y\, z = x\, z\, y$; that is, f applies its first argument to its third and then its second argument; it switches the roles of the two arguments of x.

The notation of lambda calculus involves formal parameters to help define how arguments are used by the function. However, such explicit parameters can be avoided, by starting with three special functions, called *combinators*, defined as follows.

$$\text{Identityfunction}: \quad I = (\lambda\, x\, .\, x), \tag{4}$$
$$\text{Constantfunction}: \quad K = (\lambda\, x\, .\, (\lambda\, y\, .\, x)), \tag{5}$$
$$\text{Substitutionfunction}: \quad S = (\lambda\, x\, .\, (\lambda\, y\, .\, (\lambda\, z\, .\, x\, z\, (y\, z)))). \tag{6}$$

These functions are uniquely characterized by the following properties:

$$I\, x = x, \tag{7}$$
$$K\, x\, y = x, \tag{8}$$
$$S\, x\, y\, z = x\, z\, (y\, z). \tag{9}$$

The beautiful thing is that every lambda term can be expressed by an appropriate combination of SKI combinators. For instance, function f defined above in (3) can also defined by

$$f = S(S(KS)(S(KK)S))(KK). \tag{10}$$

If you carefully carry out ten reductions, then you obtain (see Appendix A):

$$S(S(KS)(S(KK)S))(KK)\, x\, y\, z = x\, z\, y. \tag{11}$$

The SKI combinators are basic forms of passing around empty envelopes; they suffice to express any other form. In fact, with a bit more juggling, you can even come up with a single combinator that can express all lambda terms. Unfortunately, I do not know of a Golly-like program for lambda calculus.

To define this pure ‘computation world’ takes a bit more effort than defining the pure ‘information world’ Game of Life. The surprising thing is that in this ‘computation world’ without data, it is possible to ‘discover’ data. In a way similar to how certain Game-of-Life configurations can be interpreted as doing information processing, you can also interpret certain pure functions as data values. For instance, the natural number n can be represented by the function

$$\widehat{n} = (\lambda\, s\, .\, (\lambda\, z\, .\, s^n\, z)), \tag{12}$$

where s^n consists of n applications of function s. For example, the number two is represented by

$$\widehat{2} = (\lambda\, s\, .\, (\lambda\, z\, .\, s\, (s\, z))). \tag{13}$$

To understand this representation, it may help to think of parameter s as a successor function that adds one, and parameter z as a zero function. The number n is represented by a function that applies its first argument s precisely n times to its second argument z. In a way, this definition is very practical, since the function that represents the number n captures what it means to do something n times. These are known as *Church numerals*. Other functions can be defined to operate on these ‘numbers’, such as addition:

$$add\, x\, y = (\lambda\, s\, .\, (\lambda\, z\, .\, x\, s\, (y\, s\, z))). \tag{14}$$

In this definition of add, ‘number’ $y = \widehat{n}$ is used as the ‘zero’ of ‘number’ $x = \widehat{m}$, and since x applies s precisely m times to that ‘zero’, it yields the function that applies s precisely $m + n$ times to z. hence, representing the sum $m + n$: $add\, \widehat{m}\, \widehat{n} = \widehat{m+n}$. Here is $2 + 2$ in lambda calculus (see Appendix A for details):

$$\begin{array}{ll}
 & add\, \widehat{2}\, \widehat{2} \\
= & \{\text{definitions of } add \text{ and } \widehat{2}\} \\
 & (\lambda\, s\, .\, (\lambda\, z\, .\, (\lambda\, s\, .\, (\lambda\, z\, .\, s\, (s\, z)))\, s\, ((\lambda\, s\, .\, (\lambda\, z\, .\, s\, (s\, z)))\, s\, z))) \\
= & \{\text{four } \beta - \text{reductions}\} \\
 & (\lambda\, s\, .\, (\lambda\, z\, .\, s\, (s\, (s\, (s\, z))))) \\
= & \{\text{definition of } \widehat{4}\} \\
 & \widehat{4}
\end{array}$$

In a similar way, we can define boolean values and boolean operators. With some more effort, we can define an *if*-construct, and also a *loop*-construct (fixpoint combinator). That way, we have a full-blown (*universal*) programming language.

Any data that can be described, can be described by appropriate pure functions. Thus, even in a pure ‘computation world’ that was not designed for general data representation, the possibility of describing arbitrary data values can emerge, provided the ‘computation world’ is sufficiently rich. You can ‘see’ the data values (information) when you put on the right kind of interpretational ‘glasses’.

5. Science

Science aims to understand the world around (and inside) us. Ultimately, such understanding could give us the ability to find out what will happen in any given situation, that is, to predict (some aspects of) the future. For instance, whether in the next year, a specific asteroid will hit the Earth or pass by at a safe distance. Or, whether a specific molecular compound will be toxic or work as a medicine. Typically, scientific theories have a mathematical basis, and predictions are obtained through computations. The development, fine tuning, and testing of theories involves huge amounts of data. At CERN, the Large Hadron Collider generates petabytes of data (CERN, 2013).

However, it is not this kind of informatics involvement in science that I consider fundamental. More interesting is that scientific theories and models themselves have become more and more computational in nature. Such computational models involve system *states* described by relevant information, and *state changes* described by computational steps on that state information. Constructing, analyzing, and applying such models requires informatics skills. Also, the development of tools to assist in such modeling activities belongs to the domain of informatics.

In biology, this became obvious with the discovery of the genetic code and the mechanisms of self-reproduction. DNA is a carrier of genetic information, which is manipulated (transcribed, repaired, and copied) by protein structures in cells (Verhoeff, 2010); for an interactive challenge to write a self-reproducing program. These protein structures implement information processors, that is, computations. But also other sciences became more computational. Think of catalytic surface reactions in chemistry modeled by cellular automata like the Game of Life. In *digital physics* ('it from bit', as Wheeler phrased it (Gleick, 2011; Wikipedia, 2013); the universe is modeled as one big computation, with discrete information 'particles' as fundamental building blocks. This theme is also pursued in Wolfram (2002), Rucker (2006) for a readable summary. Investigating nature is very much like investigating the (artificial) Game of Life. Information and computation turn out to be inherent in nature: *nature is universal*, in a computational sense. Thus, informatics is not just a science of the artificial, but a natural science as well (Rosenbloom, 2013).

More philosophically, when it comes to contemplating the predictability of the future, again, informatics is highly relevant. What does it mean that something is predictable? That we have a model that enables us to deduce information about a future state, given sufficient information about the current state. Note that it is not necessarily the case the we can deduce the future state in full detail. To be more precise, the method for deducing information about that future state must be *effective*, that is, the method must be unambiguously executable in finite time. And that is precisely what, in informatics, we call an *algorithm*. A model is capable of predicting the future, when there is an algorithm that, given a current state as input, outputs information about a future state. Interestingly, from informatics, we know that for certain questions about the future of certain models, there exist no algorithms to decide those questions. Actually, any model that is sufficiently 'rich' (*universal*, in informatics terminology) is inherently unpredictable. Furthermore, some models are such that any algorithm to predict its future necessarily runs longer than just letting reality unfold (and bring about its own future).

This brings me to *algorithmic information theory* (AIT), as opposed to the probabilistic information theory of Section 3. In AIT, the information content of an object (such as a bit string), is defined as the length of a shortest program that generates the object. If a shortest program to generate a given bit string has 'nearly' the same size as the object itself, then that object is said to be *random*. A bit string consisting of one million zeros is clearly not random. There is no shortcut to storing or communicating a random bit string: you might as well store or communicate (a copy of) it. When observing the bits of a random bit string one by one, each next bit cannot be predicted efficiently. Discovery of patterns in the structure or behavior of nature is, in this sense, proof that nature is not completely random. However, there is no inherent reason why nature would not be random, at least in some respects. Indeed, 'most' bit strings turn out to be random, and indeed many processes in nature turn out to be 'chaotic', that is, their behavior is highly unpredictable.

6. Technology

Technology aims to provide artifacts that can help us live better in this world. Through technology, we twist nature to our intent. The artifacts vary over a wide range of products, including tools to assist in design and production. They interact with the world and with us. Some artifacts mostly have a physical purpose, like a hammer, a bow and arrow, a bridge, or a steam engine. Other artifacts, although physical in their construction, (also) concern information and computation, like a planetarium, a clock, a lock-with-key, or an abacus.

Technological artifacts somehow involve control, sensors, and actuators. Control and sensing used to be exclusively in human hands. But this turns out to be tedious, time consuming, costly, and error prone. Hence, there is a long history of automating the control. Initially, mechanical control mechanisms were invented and discovered, such as the centrifugal governor on a steam engine to stabilize its operating speed, the camshaft in a combustion engine to time the ignition, and the punched cards that controlled Jacquard's loom. Then, control became electro-mechanical, e.g., the rotating drum sequencers in old telephone exchanges and dish washers.

In hindsight, control centers around information and its processing. However, early controllers involved particular physical information carriers, and they were designed with a focus on the physical carriers, rather than on information as an abstract concept. This changed with the advent of electronics. Controllers became computers based on binary logic.

We have used (and still use) all kinds of natural phenomena as information carriers and processors in automated controllers (computers): electromagnetism (solenoid relays, core memory, hard disks), electrons (vacuum tubes, radio valves, transistors), photons (fiber optics), organic molecules (Adleman's DNA computer), quantum states of matter (quantum cryptography, experimental quantum gates and computers). The possibility for nature to compute is an emergent property. Like the Game of Life, nature appears to be *universal*, when it comes to computing. For any imaginable computation, there is a way to 'hardwire' nature to do this computation. Moreover, this universality also implies that we can have *programmable* machines, where we can change the computation by changing a program rather than changing the 'wiring'. Informatics has contributed powerful programming techniques that enable us to solve complex computational problems, thereby paling the primitive controllers of the past.

Modern technology would be impossible without an abstract treatment of information and its processing. A key informatics concept is the *algorithm*. It needs no explanation in the IOI community, but for the general public it remains somewhat mystical. Fortunately, informatics is slowly becoming accepted as a serious topic in general education. I can recommend a book like (Cormen, 2013).

An overwhelming number of products nowadays contain computing devices and software. We have fully automated factories, autonomous robots, chemistry labs on a chip, and 3D-printers. The scale at which technology operates is shrinking: macro, micro, nano, molecular, quantum level. Who knows what is still to come.

7. Conclusion

The basic building blocks of informatics are surprisingly simple. To describe any kind of data, all you need is a bunch of bits, and to describe any kind of computation, all you need is an SKI combination. Information can be studied without emphasis on computation, and computation can be studied without emphasis on information. But, in the end, information and computation always go together.

Those basic building blocks are not convenient for practical application. Over the years, we have acquired an awesome arsenal of algorithms and data structures to solve many computational problems efficiently. This is well known to IOI participants. Some limits of computability have been mapped out clearly, see for instance the challenging and inspiring book by Hofstadter (1979), or Harel (2000), Moore+Mertens (2011), Cormen (2013).

The power of informatics is in abstraction, also see (Verhoeff, 2011). Informatics treats information and its processing without reference to concrete physical carriers (and certainly not to concrete computers). In that sense, it is like mathematics, where numbers are treated without reference to concrete physical objects. Guo (2010) defines informatics as "efficiently implementing automated abstraction". You might call informatics a branch of mathematics, but informatics goes beyond mathematics: *informatics automates mathematics*.

Informatics has earned the status of a separate scientific discipline, as eloquently argued by Rosenbloom (2013). It has changed the way we look at society, science, and technology. This concludes my story of informatics. However, the story of informatics is unfinished, with numerous exciting open problems; see, for instance, (Adriaans, 2013).

I hope this article

- inspires IOI coaches to broaden the view of IOI contestants on informatics,
- encourages discussion on informatics as a true scientific discipline, and
- persuades you to read some interesting literature on informatics topics.

Acknowledgments. I would like to thank Ruurd Kuiper for critically reading a draft version of this article, and trying to keep me honest.

References

Adriaans, P., van Benthem, J. (2008). *Philosophy of Information*, Vol. 8 of the Handbook of the Philosophy of Science, Elsevier.

Adriaans, P. (2013). *Fundamental Problems in the Study of Information and Computation*. http://www.pieter-adriaans.com/information/fundamental-problems-in-the-study-of-information-and-computation.html (accessed April 2013).

Berlekamp, E.R., Conway, J.H., Guy, R.K. (2004). In: Peters, A.K. (Ed.), *Winning Ways for Your Mathematical Plays*, Vol. 4 (2nd edn.).

CERN. *Computing*. http://home.web.cern.ch/about/computing (accessed April 2013).

Chu-Caroll, M. C. (2013). *Good Math: A Geek's Guide to the Beauty of Numbers, Logic, and Computation*. The Pragmatic Bookshelf.

Cormen, Th.H. (2013). *Algorithms Unlocked*. The MIT Press.

Gleick, J. (2011). *The Information: a History, a Theory, a Flood*. Pantheon.

Golly (2013). *An Open Source, Cross-Platform Application for Exploring Conway's Game of Life and Other Cellular Automata*. http://golly.sourceforge.net (accessed March 2013).

Guo, P. (2010). *What is Computer Science? Efficiently Implementing Automated Abstractions*. http://www.pgbovine.net/what-is-computer-science.htm (accessed April 2013).

Harel, D. (2000). *Computers Ltd: What They Really Can't Do*. Oxford University Press.

Hofstadter, D.R. (1979). *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books.

Moore, C., Mertens, S. (2011). *The Nature of Computation*. Oxford University Press. http://www.nature-of-computation.org/

Rosenbloom, P. (2013). *On Computing: The Fourth Great Scientific Domain*. The MIT Press.

Rucker, R. (2006). *The Lifebox, the Seashell, and the Soul: What Gnarly Computation Taught Me About Ultimate Reality, the Meaning of Life, and How to Be Happy*. Basic Books. http://www.rudyrucker.com/lifebox/

Simon, H.A. (1996). *The Sciences of the Artificial*. The MIT Press, (3rd edn.).

Verhoeff, T. (2010). An enticing environment for programming. *Olympiads in Informatics*, 4, 134–141.

Verthoeff, T. (2011). On abstraction in informatics. In: *ISSEP 2011: Proceedings of Selected Papers*, on CD-ROM. http://pubshop.bmukk.gv.at/detail.aspx?id=444. Download: http://www.win.tue.nl/w̃stomv/publications/issep-2011-on-abstraction.pdf (accessed March 2013).

Wikipedia. *Digital Physics*. http://en.wikipedia.org/wiki/Digital_physics (accessed April 2013).

Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media.

A Some Details

Here are the reductions to derive (11). Note that different reduction orders are possible.

$$\begin{aligned} & \underline{S(\underline{S(KS)(S(KK)S)})(\underline{KK})\,\underline{x}}\,y\,z \\ = \quad & \{\text{property (9) of S}\} \end{aligned}$$

$$
\begin{aligned}
& S(KS)(S(KK)S)\,x\,(\underline{KK\,x})\,y\,z \\
= \quad & \{\text{property (8) of K}\} \\
& \underline{S(\underline{KS})(\underline{S(KK)S})\,\underline{x}}\,K\,y\,z \\
= \quad & \{\text{property (9) of S}\} \\
& \underline{KS\,x}(S(KK)S\,x)K\,y\,z \\
= \quad & \{\text{property (8) of K}\} \\
& S(\underline{S(\underline{KK})\underline{S}\,\underline{x}})K\,y\,z \\
= \quad & \{\text{property (9) of S}\} \\
& S(\underline{KK\,x}(S\,x))K\,y\,z \\
= \quad & \{\text{property (8) of K}\} \\
& \underline{S(\underline{K(S\,x)})\underline{K}\,\underline{y}}\,z \\
= \quad & \{\text{property (9) of S}\} \\
& \underline{K(S\,x)\,y}\,(K\,y)\,z \\
= \quad & \{\text{property (8) of K}\} \\
& \underline{S\,\underline{x}(\underline{K\,y})\,\underline{z}} \\
= \quad & \{\text{property (9) of S}\} \\
& x\,z\,(\underline{K\,y\,z}) \\
= \quad & \{\text{property (8) of K}\} \\
& x\,z\,y
\end{aligned}
$$

Here is $2 + 2$ in lambda calculus with all the details:

$$
\begin{aligned}
& add\,\widehat{2}\,\widehat{2} \\
= \quad & \{\text{definition of } add\} \\
& (\lambda\,s\,.\,(\lambda\,z\,.\,\widehat{2}\,s\,(\widehat{2}\,s\,z))) \\
= \quad & \{\text{definition of } \widehat{2}\} \\
& (\lambda\,s\,.\,(\lambda\,z\,.\,\underline{(\lambda\,s\,.\,(\lambda\,z\,.\,s\,(s\,z)))\,s}\,(\underline{(\lambda\,s\,.\,(\lambda\,z\,.\,s\,(s\,z)))\,s}\,z))) \\
= \quad & \{\beta - \text{reduction on underlined applications}\} \\
& (\lambda\,s\,.\,(\lambda\,z\,.\,(\lambda\,z\,.\,s\,(s\,z))\,\underline{((\lambda\,z\,.\,s\,(s\,z))\,z)})) \\
= \quad & \{\beta - \text{reduction on underlined application}\} \\
& (\lambda\,s\,.\,(\lambda\,z\,.\,\underline{(\lambda\,z\,.\,s\,(s\,z))\,(s\,(s\,z))})) \\
= \quad & \{\beta - \text{reduction on underlined application}\} \\
& (\lambda\,s\,.\,(\lambda\,z\,.\,s\,(s\,(s\,(s\,z))))) \\
= \quad & \{\text{definition of } \widehat{4}\} \\
& \widehat{4}
\end{aligned}
$$

T. Verhoeff is an 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, 2013, Vol. 7, 153–162
© 2013 *Vilnius University*

REPORTS

International Olympiads in Informatics in Kazakhstan

Artem IGLIKOV[1], Zaza GAMEZARDASHVILI[2],
Bakhyt MATKARIMOV[3]
[1]*Kazakh-British Technical University*
59 Tole bi str., 050000 Almaty, Kazakhstan
[2]*Tbilisi Free University*
Microdistrict 1, Nutsubidze str., 0183 Tbilisi, Georgia
[3]*Nazarbayev University*
Kabanbay batyr ave 53, 010000 Astana, Kazakhstan
e-mail: artem.iglikov@gmail.com, gam_zaza@yahoo.com, bakhyt.matkarimov@gmail.com

Abstract. The Republic of Kazakhstan will host the International Olympiad in Informatics and Asia-Pacific Informatics Olympiad in 2015. Organization of the on-site International contest is a challenging task owing to the various scientific, technical, organizational, and even political issues. This paper discusses the Kazakhstan experience of hosting International programming contests from the viewpoint of the technical and scientific committee. We will briefly cover issues of contestant's workplace preparation, overview our testing system, and define some challenges in problemset preparation.

Key words: IOI, ICPC, APIO, IZhO, EOI/VKOSHP, scientific committee, CMS.

1. Short Overview

In Kazakhstan, a single State enterprise called the Republican Scientific Practical Centre "Daryn" (Republican Scientific Practical Centre "Daryn") of the Ministry of Education and Science (Ministry of Education and Science of the Republic of Kazakhstan) bears responsibility for organizing the National Olympiads in all subjects for secondary school students. Before 2003, we had only one programming contest – our Daryn-organized National Olympiad in Informatics for secondary school students. In 2003 our College and University students began participating in the ACM ICPC, on the base of al-Farabi National State University and Nazarbayev University.

During the last decade the number of programming problem solving contests in Kazakhstan has dramatically increased with many educational and scientific organizations offering various competitions in programming and information technologies. Many Kazakhstan Universities opened annual programming championships, among them Kazakh-British Technical University, Suleyman Demirel University, al-Farabi National State University, International IT University, and Nazarbayev University. University championships are ACM ICPC-style and open for secondary school students who compete in and often win them. In fact, ICPC and IOI activity joined together and in just a few years demonstrated commendable results – in 2007, Kazakhstan students reached the World final of ICPC and, independently, won gold medals level at IOI.

To date, Kazakhstan has created and hosted two ongoing International Informatics Olympiads for secondary school students:

– Since 2005, the International Zhautykov Olympiad (IZhO) has conducted competitions for secondary school students organized by the Ministry of Education and Science, Daryn, and O. Zhautykov Republican Specialized Physics-Mathematics Secondary Boarding School (O. Zhautykov Republican Specialized Physics-Mathematics Secondary Boarding School). The International Zhautykov Olympiad (International Zhautykov Olympiad) welcomes secondary school teams to compete in Mathematics, Physics, and Informatics (Informatics debuted in 2009). Each team comprises seven students (contestants); a student competes individually in only one subject. Contest rules are close to those of the top World Olympiads, i.e., IMO, IPhO, and IOI. Medals are allocated individually, and a weighted sum formula yields the total team score. Winning teams receive valuable prizes, like modern laptops for every contestant of the team (independent of his individual result).
– Since 2009, the Saint Petersburg National Research University of Information technologies, Mechanics and Optics (IFMO), Daryn, and K. Satpayev Kazakh National Technical University conduct the Eurasian Olympiad in Informatics. EOI is actually the All-Russian Team Olympiad in Programming for schoolboys (VKOSHP) (All-Russian Team Olympiad in Programming for Schoolboys), hosted in Almaty, Kazakhstan. This is an ACM ICPC style contest and traditionally carried out within the single week with ACM ICPC Northeastern European Regional Programming Contest for high school students. It is hosted by Saint Petersburg and Barnaul in Russia, Tbilisi or Batumi, Georgia, or Erevan, Armenia, Tashkent, Uzbekistan, and Almaty, Kazakhstan. Before 2009, Kazakhstan students participated VKOSHP at the Saint Petersburg, Barnaul, and Tashkent sites, several times winning the championship at Tashkent site, and once at Barnaul site.

Kazakhstan students will join the Asia-Pacific APIO starting from 2012.

Results of International Zhautykov Olympiads, National Olympiad, APIO, and VKOSHP/EOI are taken into account together to select Kazakhstan team to IOI. The Country report on various programming competitions in Kazakhstan will be presented in subsequent publications.

2. Contest Environment

One of the most important tasks in hosting a big contest is the construction of a contest technical environment that is familiar to all contestants and additionally performs well under high loads.

2.1. *Ideal Conditions*

To prepare an ideal contest environment, one would need:

- precisely equivalent computers: the environment should have enough computers for all contestants plus reserves for failures and the testing system;
- networking and printing: all computers must be connected to network;
- power, backup and restore: system should be stable under power and other faults;
- time to work: whole system must be available and reserved for at least one week before contest;
- permissions: the technical committee must have full system access, i.e., permission to modify hardware, to format hard drives, and reinstall the operating system and other software;
- software: there should be a reliable ready-to-work Testing and Contest management system (CMS);
- knowledge and experience: the technical committee must have good knowledge of operating system and contest system administration.

The contest problem set does not appear in the above list as it includes only technical issues. Let us consider problem set as scientific issue, since task preparation is not yet completely automated.

If all these conditions are met, then there will be a chance to run contest ideally. Technical committee staff starts preparation before the contest with enough time, cleans all computers, installs the proper operating system, prepares the workspace for contestants, prepares testing system, runs several load tests, checks all computers, runs sample contests, generates reports, etc. All the best!

2.2. *Reality*

The reality usually is different from the ideal above. Olympiads in Informatics in Kazakhstan are usually held in Universities. These Universities have offered a number of computers in one or several collocated laboratories, all computers on a unified network, and sometimes even servers and printers! But actually:

- Nobody will give full control to these computers to the technicians of the Olympiad, because there are system administrators there, and they do not want strangers to take control over their systems. In the best case, they will help you with some problems.
- Nobody will even talk about reinstallation of operating system. It's not a problem to reinstall the system before the Olympiad. The problem is to restore original

system after the Olympiad. And because there are usually classes just a day before the contest and on the next day after, they cannot lose time (a day or two) on reinstalling hundreds of computers.
- There is no possibility to reserve computers for several days before the process since the computers are in ongoing use by students.
- Sometimes there is even no possibility to reinstall operating system on jury computers used for testing.
- All universities primarily use Windows-family installations for their education process.
- There are only a few people in the technical committee, and all of them are working or studying. So we just don't have a free week for setting up all things.

These problems can be solved in different ways. Some universities propose using thin clients for virtual machines on their server cluster (but you need to be sure, that the performance of the cluster will be enough for hundred or more contestants). Some universities already have all needed software on contestant computers (but there is also other software and teaching materials available on these machines). Some universities, of course, simply don't have the needed software.

A committee survives in these conditions (like starting to work only 12 hours before contest) only by having all software ready and completely automated scripts for installation and setup. Scripting and tools like Sysinternals (Windows Sysinternals) are very helpful. One more very helpful feature is that at the moment all compilers that are used in the Olympiad (MinGW C/C++, Java, FPC), all IDE's, and other tools for Olympiad (CodeBlocks, Eclipse, FreePascal, Far Manager) do not actually require explicit installation process. It's enough to unpack them in some folder and to put shortcuts on the desktop. So we often created a self-extracting archive with everything needed and just downloaded it to all computers and run once.

Having enough time, a special windows account for contestants created. It has limited access to the system, for example, it cannot access to any other software and documents that was already installed to the computer before the Olympiad. Now contestants have limited access to computer resources that will not needed during the contest.

2.3. *Testing System*

One big challenge is that of grading the contestant solutions. A number of testing systems are now available for use, e.g., PCMS2 (NRU ITMO PCMS2 CMS), eJudge (Ejudge CMS Home Page), or others. All of them have different features: some of them work only on Linux (but we cannot reinstall operating system), some of them require huge administration experience; some of them do not support IOI rules. In the Kazakhstan quarterfinals of ICPC, we used PCMS2. Many Universities use eJudge for training and championship. For our National and International Zhautykov Olympiads, we created our own testing system. Every test system has an invoker module, responsible for executing contestant code and checking runtime limits. In our system we are using invokers running under the Widows OS family, from PCMS2 and from the testlib package of Saratov State University, Russia (Testlib).

Here are some benefits of our CMS:

- The programmer, who designed and wrote the system, is always a member of the technical committee, so problems can be fixed very quickly, and if the jury sees that it would be nice to have some feature, it can be implemented very quickly.
- Almost all parts of the system are cross-platform or potentially cross-platform (because we have used it only on Windows OS, some parts just have no adaptation for other systems), so we don't need to reinstall computers for installing the testing system.
- The system uses shared folders for distributing tasks between machines, it doesn't create any TCP connections, and so even if there is a firewall, or some limitations on the network, it will have no problems (and debugging is much easier, too).
- All system modules are very independent, so it's not a problem to reboot an invoker or add new invoker on the fly.
- The big part of the system (user interface generation and logic for tasks distribution) is written in PHP, so fixes and upgrades can be made very quickly (even during the contest).
- The invoker's part is written in plain command shell language, so preparing invokers involves just setting up compilers and running a command to install invoker scripts.
- Importantly, the system fully tests solutions during the contest, so current results are always visible to jury and are almost immediately available after the contest.

The first version of the system was written for Kazakhstan National Olympiad in Informatics. It was incrementally developed over three years to have all features of a modern scalable contest testing system: support of multiple invokers, presenting results during the contest, printing, processing clarification requests, accounts for observers and other cool things. The system is currently used in National and International Zhautykov Olympiads.

3. International Zhautykov Olympiad

When Andrey Lopatin was IZhO Jury member we used his TestSys testing system. From him we learned many interesting tricks; one of them was how to avoid possible network connections between contestant's computers. The trick based on the feature of TCP/IP protocol stack, tested for Windows OS family – it is not required for computer and default gateway to be in same subnet. Every computer is isolated in its own subnet, having to send packages to default gateway to communicate with other computers. The contestant cannot change network setting without administrative privileges, so all network communications controlled by default gateway which is also testing system interface server.

The technical conditions of IZhO were far from ideal. In the first years there were too many different types of computers and network cabling problems; currently there are no administrative privileges on software and hardware. Sometimes we divided students into two groups, foreign and ours, and give our guests the better workspaces. Despite these troubles, we did not get negative assessment from contestants. The practice shows that a strong student needs very little to work and win – just a working compiler and availability of the submit interface! We often observed that Kazakhstan student seated at the worst workstations won the contest.

4. Eurasian Olympiad in Informatics

The EOI exploited almost ideal conditions – all necessary hardware with necessary permissions was allocated. IFMO representatives of the scientific and technical committee are present on every VKOSHP site, and the contest environment is identical everywhere except the contestant computers and testing server. VKOSHP uses the IFMO PCMS2 testing system. PCMS2 supports several distributed testing servers, so every site is in fact autonomous. EOI carried out the best traditions of Russian programming competitions. Every technical task is automated, starting from operating system install and test and finishing with diploma printing.

5. Challenge of Creating Problem Set

One of the features of the Zhautykov Olympiad is a wide distribution of contestant skill level and experience. Even though this contest is an International one, it's not of IOI challenge. So, a smaller number of countries participate and those who do participate sometimes do not send the strongest contestants. The contest entrants span the range from IOI gold medallists to beginners. If we target a problem set to the stronger participants, the beginners will have zero points, which would quite probably reduce their motivation. If we target the problem set to the weakest participants, the contest will not be interesting for stronger ones. One main requirement is to avoid the situation where the number of full score results exceeds the limit of the number of gold medals.

Thus it is quite challenging for the jury to create a problem set that will be interesting for all categories of participants and at the same time will allow fair evaluation of the winners. Though several first contests always included a problem that can be fully solved by a beginner, last year we had no such problem, but almost all problems we used had easy subtasks for 30 or even 50 points that could be solved with naïve algorithms. We think that having easy subtasks is more interesting for both types of participants than having one easy problem. Because the easy problem can be solved in the beginning of the contest even by a non-experienced contestant, and after this he or she will do nothing for the remaining four hours. With several easy subtasks, a contestant will be able to show different skills and gain more satisfaction, because he was not counting flies for several hours.

5.1. *Sample Tasks of the International Zhautykov Olympiad*

Below are some example problems of past contests.

Problem B from year 2009. The contestant needed to find a bridge nearest to the given two vertices in a graph. Solution of this problem assumes solving standard graph problems:

1. Find minimal distances from two given vertices to all other vertices.
2. Find all bridges in graph and choose one of them, for which robots arrival time is minimal possible.

The effective solution of this problem assumes use of Dijkstra's algorithm ($O((N + M) * \log(N))$, for example, implemented with heap data structure) for finding distances and based on depth-first-search $O(N + M)$ algorithm for finding bridges in graph.

A hard problem C from year 2009. In this problem, we must check existence of the symmetry axis for the given set of points on the plane. Naive $O(N * N)$ solution check existence of symmetry axis for all pairs of points, assuming that they are mirror reflection and taking care that axis of symmetry may pass through one or more points in the set. Note that checking, if the given line is an axis of symmetry for given set of points, could be done in $O(N)$ time using a hash table.

Full score solutions assume to use linear time algorithms to find longest palindrome in cyclic sequence of objects. Note, that the symmetry axis, if exists, will pass through the centre of mass M of the given set of points. Now, let us consider the vectors from point M to each of the point of the set. If there is an axis of symmetry, then we can partition our vectors into two groups: for each vector X (not lying on the symmetry axis) from the first group, there is only one vector Y from the other group that has the same length and lying at the same angle as X from the symmetry axis, but in opposite direction. So, we can mark-up point vectors by unique identifier (using angles between pairs of consecutive vectors and their lengths), for example, sorting all vectors by angle and length and updating angles from absolute to relative cross points values. As result, we will get a circular sequence of vectors (circular string) that should be checked for being a cyclic palindrome-like one, i.e., if there is a symmetry axis, then reading the sequence of vectors in clockwise or opposite direction around symmetry axis gives the same result. In case of point set $\{(0, 1), (1, 0), (0, -1), (-1, 0)\}$, the cyclic sequence may look like $\{(90, 1), (90, 1), (90, 1), (90, 1)\}$ which contains four palindromes of length 3 (odd) and four palindromes of length 4 (even). To avoid consideration of cyclic palindromes, we might double initial sequence, i.e., make a new sequence with double size, containing the initial sequence twice. If the length of the longest odd/even palindrome in the new sequence is more than or equal to $N - 1/N$, then there exists in initial sequence a cyclic palindrome. Finding the longest palindrome in a string can be done in $O(N * \log N)$ or $O(N)$ with different algorithms.

An easy problem D from year 2009. In this problem, contestants had to calculate the Bth power of A modulo C for 64-bit integers A, B, C. The problem was targeted for weaker participants, so several solutions possible. The main issue of this problem is that even multiplication of two numbers when calculating the power can exceed the maximal value of 64-bit integer data type. So full score solution may use arbitrary precision arithmetic or the contestant can note, that it is possible to calculate product $X * Y$ as $X + X + \cdots + X$, replacing product by addition in the well-known fast-exponentiation algorithm. Jury noted that, despite enabling the Java programming language, contestants didn't submit solutions exploiting the *modPow* method of *BigInteger* class.

In problem E from year 2009, we have a number of watermelons, and we know their sizes W_0 at some point in time. Also we know the speed of growing S for each watermelon. The size of watermelon after K days can be calculated as $W_k = W_0 + S * K$. Given parameters of all watermelons, the contestant must find the heaviest watermelon on given days.

Naive approaches to solve this problem include an $O(N * M)$ solution which iterates through all watermelons for each request and choose the heaviest one. Another $O(N * N + M * \log(N))$ solution notes that there is not more than one segment of time when the given watermelon becomes the heaviest one, and we can find this moment by iterating through all pairs of watermelons, and then using binary search to find the heaviest one at given moment of time.

The full score solution requires $O((N + M) * \log(N)$ time where we can find the segment for being the heaviest for each watermelon in $O(N * \log(N))$ time. Note that the weight of the watermelon is the linear function of time. It could be represented as a line on the plane (time-weight). Then for each moment of time we have to find the topmost point of the given lines. It is obvious that this point's form convex polyline, which in fact is the intersection of half planes determined by given lines of watermelon's weight in time. There is a known algorithm for finding such intersection in $O(N * \log(N))$, using stack data structure. In the stack we will store vertices of a part of the convex polyline that we are constructing. Initially stack is empty. First step of algorithm is sorting lines by angle in ascending order. Then let us consider each line (half plane) by the order. Every half plane cuts a part of current polyline and adds one segment to it. To construct updated polyline we should remove points from the stack while they do not belong to added half plane and then add one point – the intersection of the last polyline segment left in stack and new line. At the end of the algorithm the stack will contain resulting polyline, and then we can use binary search to determine the heaviest watermelon at a given time moment.

Problem D from year 2013. This problem required the contestant to answer queries about a special graph. The feature of the graph is that each vertex has at most one outgoing edge. Queries revolve around deleting a given edge and finding the length of the shortest path between two vertices. There are several approaches on how to solve this problem. The main thing is to note that the graph will consist of one or more components containing one cycle along with trees suspended to this cycle.

We can do some pre-processing and for each vertex check whether it is in the cycle and, if it is, calculate the cycle number and the vertex position in it. Otherwise, we calculate the vertex level in the tree. After this we will process queries in the reverse order (avoiding edge deletion by adding edges instead). When an edge is added, the vertices need to be marked as connected, which is done using a disjoint sets structure. So, when we have a query on the shortest path, we need to check whether vertices are connected using disjoint sets and, if they are, calculate the shortest path using information about level, cycle number, and position in the cycle. As you see it's not easy to solve this problem for full constraints where the number of vertices and number of queries can be up to

10^5, but this problem has an easy subtask for 50 points where the number of vertices is not greater than $2 * 10^3$ and number of queries is not greater than $2 * 10^4$, which can be solved relatively easy.

6. Commentary

Joined ICPC and IOI activity along with the ongoing work of the educational and scientific organizations in Kazakhstan has lead us to fast and positive results: in 2007 our students reached the World finals of ICPC and the gold medal level of the IOI. One important factor of our success is the participation of Kazakhstan students in various International competitions and camps, among them we would like to emphasize the Summer school of computing for secondary school students (Summer School of Computing), Petrozavodsk training camps for high school students (Petrozavodsk Training Camps), and E. Pankratiev Open Team Programming Collegiate Cup (E. Pankratiev Open Team Programming Collegiate Cup), all these events in Russia.

Inspired by the successful development of programming contests in Kazakhstan, Informatics was introduced since 2009 into the International Zhautykov Olympiad and All-Russian Team Olympiad in Programming for schoolboys hosted in Kazakhstan as Eurasian Olympiad in Informatics.

Herewith we present statistics of IOI medals for the participants of International Zhautykov Olympiad:

IOI 2009: 1 Gold (Dzmitry Bahdanau, Belarus), 2 Silver, 3 Bronze;

IOI 2010: 2 Gold (Rumen Hristov, Vladislav Haralampiev, Bulgaria), 4 Silver, 10 Bronze;

IOI 2011: 1 Gold (Baris Kaya, Turkey), 2 Silver, 6 Bronze;

IOI 2012: 2 Gold (Maxim Akhmedov, Oleg Ivanov, Russia), 3 Silver, 3 Bronze.

7. Conclusions

In this paper we covered some aspects of conducting an international on-site contest from the position of technical and scientific committee. To sum up, holding an International contest is a big responsibility and has large number of difficult and not-so-difficult challenges, all of which are about compromise. We described only a few of them: preparing workplaces, testing system, preparing problemset. Having experienced team, all these challenges can be easily solved in ideal environment, but the reality introduces many limits, so we always need to find balancing point between ideal and possible solution, which experienced team always find! In 2015, the International Olympiad in Informatics and Asian-Pacific Informatics Olympiads will be hosted by the Republic of Kazakhstan. We will be happy to present our country in these events at the best level of International contests organization!

Acknowledgments. The authors would like to thank the Republican Scientific Practical Centre “Daryn”, personally Tildash Bituova and Sholpan Kirabayeva, and the Zhautykov

Republican Specialized Physics-Mathematics boarding school for gifted students, personally Kairosh Makishev and Arkhat Dolayev, in addition to the whole Russian ACM ICPC and IOI team! This paper was supported by the grant of the Ministry of Education and Science of the Republic of Kazakhstan.

References

All-Russian Team Olympiad in Programming for Schoolboys. `http://neerc.ifmo.ru/school/russia-team/index.html`
Ejudge CMS Home Page. `http://ejudge.ru/`
International Zhautykov Olympiad. `http://izho.kz/eng/`
Ministry of Education and Science of the Republic of Kazakhstan. `http://www.edu.gov.kz/en/`
NRU ITMO PCMS2 CMS. `http://neerc.ifmo.ru/trains/information/software.html`
E. Pankratiev Open Team Programming Collegiate Cup. `http://www.opencup.ru/`
Petrozavodsk Training Camps. `http://acm.petrsu.ru/site/?lang=eng`
Republican Scientific Practical Centre "Daryn". `http://daryn.kz/?lang=en`
Summer School of Computing. `http://olympiads.ru/sis/`
Testlib. `http://code.google.com/p/testlib/`
O. Zhautykov Republican Specialized Physics-Mathematics Secondary Boarding School. `http://fizmat.kz/`
Windows Sysinternals. `http://www.sysinternals.com`

A. Iglikov, participated at IOI from 2012 as Kazakhstan team leader. Graduated from K. Satpayev Kazakh National Technical University, currently PhD student of Kazakh-British Technical University. Successfully participated in ACM ICPC 2005–2009, advanced to the ACM ICPC 2007 World final. In top 200 of TopCoder rating.

Z. Gamezardashvili, participated at IOI from 2002 as Georgia team deputy leader. Graduated from Tbilisi State University, Georgia, in 1988. Research interests include algorithms design, software engineering, compiling techniques, digital systems and operating systems. Jury member of the Zhautykov International Olympiad from 2009.

B. Matkarimov, participated at IOI from 2005 as Kazakhstan team leader, IOI IC member from 2012. Graduated from Novosibirsk State University, Russia, in 1989. Initiator and Jury chairman of Kazakhstan quarterfinal of ACM ICPC. Secretary in informatics of Zhautykov International Olympiad and Eurasian Olympiad in Informatics.

Olympiads in Informatics, 2013, Vol. 7, 163–167
© 2013 *Vilnius University*

Bangladesh Olympiads in Informatics

Muhammad KAYKOBAD
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1000, Bangladesh
e-mail: kaykobad@cse.buet.ac.bd

Abstract. This paper details the growing popularity of olympiad-type academic competitions in Bangladesh. We concentrate on improving our students' ability to solve problems, develop programming skills, analytical skills, and emphasize in developing skills other than rote-memorization. These initiatives have already paid of dividends as is perceived through increasing participation, which emphasizes the most cost-effective method to gain excellence: competition.

Key words: IOI, informatics, BMOC, BIOC.

1. Educational Demographics

The country has very limited natural resources, with a surplus only in human resources. In 1972, we had a population of 75 million, with some 7 791 secondary schools and 1.84 million students compared to 161 million people, 19,040 schools and 7.5 million students in 2011 (Bangladesh Bureu of Educational Information and Statistics). Female students are catching up with their male counterparts both in count and performance. We have five years education in primary schools, then another five years in secondary schools after which two years college education that is prerequisite for admission into universities. Teacher–student ratios in primary, secondary, college and universities are respectively 1 to 43, 34, 30, and 35.

2. Introduction of Olympiads

The government has been providing some 300 million textbooks annually in the hope that students from disadvantaged strata of society benefit from some education. In spite of the best efforts, our schools and colleges suffer from shortages in infrastructure, laboratories, library facilities, and adequate count of teachers. Under the circumstances, some educationists had been considering ways to overcome this deficit and empower our young people with praiseworthy skill.

Our kids grow in competitive environment on limited resources. They must work hard to ensure their survival. Some 17–18 years back, we convinced dynamic editor Matiur Rahman (of the *Prothom Alo* (`http://www.prothom-alo.com/`), one of our popular daily newspapers), to allocate space to publish math puzzles and problems for our younger readers. The response was astonishing. Thousands of students stormed into the newspaper office with their solutions. They were soon joined by older people, as well! This gave us a great boost for continuing the initiative. The newspaper ultimately had to employ extra staff to evaluate the thousands of submitted solutions.

Some 12–13 years age, we organized a mathematics olympiad for secondary and college (not university) students that resulted in massive participation of students and the presence of guardians and teachers. The Bangladesh Mathematical Olympiad Committee (BMOC) was formed to give our initiative an institutional shape. Again, *The Daily Prothom Alo* came forward to publicize olympiad activities and popularize them for the masses. Additionallly, the Dutch-Bangla Bank came forward with financial support. We commenced organizing olympiad activities under the banner of the mathematics festival that was organized in 17–18 areas across the country, each with about a thousand participants.

Winners were given tickets to participate in national mathematics olympiad. Unlike the international version, we started the olympiad in four groups:

- Primary (up to grade 5);
- Junior (up to grade 8);
- Secondary (up to grade 10);
- Higher secondary (up to grade 12).

with the hope of familiarizing younger students with these sorts of competitions.

The first Bangladesh team to the International Mathematics Olympiad participated in Mexico in 2005. In a short time, Mathematics Festivals have become extremely popular thanks significantly to the involvement of a celebrated writer, Professor Md Zafar Iqbal, who is also an idol of the young generation. We are successfully exploiting this platform not only to raise the level of analytical and problem-solving skills but also to encourage students to be good human beings with patriotism, honesty, and tolerance, to make friendships in different localities, to learn collaboration skills, to obey parents, and maintain good health. In order to inspire students, we frequently bring celebrities like popular singers, literateurs, TV personalities, actors of all kinds, and scientists. Since 2005, we have a team every year and our students are winning medals from this event.

Alongside the mathematics olympiad, we also initiated an informatics olympiad as well. In fact, our students had been competing in the World Finals of International Collegiate Programming Contests with reasonable success since 1998. Students of BUET have an enviable record of qualifying for each World Finals since they started participation. In 2000, the BUET team outperformed teams of many world-famous universities and occupied the 11th position (The 24th ACM International Collegiate Programming Contest World Finals). In 2006, some 100 top programming students were selected from around the world, and a representative from Bangladesh occupied the 79th position! Moreover, many students who participated in ICPC contests have been offered coveted jobs at Microsoft, Google, and other famous ICT based companies.

These successes gave impetus for students to sharpen their programming skill first by participating in online contests organized by the University of Valladolid (Valladolid Online Judge Site) and then in inter-university contests organized by individual universities.

Nowadays, we have a Bangladesh National Computer Programming Contest organized every year, and then some 4–5 additional contests organized by universities.

We started programming contests for school students a little while later with the first team participating in Mexico in 2006. Since then, we have been sending teams to the IOI annually. While our school curriculum is not as supportive and up to the level for participation in IOI, we have been inspiring our school students to participate in university-level contests. In 2005, when our team very sadly failed to get Polish visas, they participated online and won a silver medal. The team participated as an IOI team in ICPC regional contest unofficially and came on top of the ranking. This gave us further impetus for consolidating with our initiatives.

Winning a silver medal at IOI 2009 held at Plovdiv, Bulgaria by Md Abirul Islam boosted our school students' interest in sharpening their programming and problem-solving skills. Our students have also started participation in International Physics Olympiad and have won medals there, too!

3. Preparation for Olympiads

Apart from publishing mathematics problems in a popular daily newspaper, BMOC began organizing math camps with the participation of winners of National Mathematics Olympiad not only for selection for IMO but also for enabling more junior students to sharpen their math skills. Students are usually given some books specially meant for participants of IMO. Winners of olympiads and participants of IMO work as mentors in these camps. Finally, based upon performance in various tests, the final team is selected for IMO. Students also participate in Asia Pacific Mathematical Olympiad on a regular basis for sharpening their analytical skills.

Professor Md Zafar Iqbal leads The Bangladesh Informatics Olympiad Committee (BIOC) to organize Divisional Informatics Olympiads in five divisions. Unlike BMOC, though, BIOC organizes these olympiads on the same day. Usually, local universities come forward to shoulder the responsibility of hosting these contests while responsibility of problem-setting and judging are shouldered by ICPC world finalists and university contestants. Usually, Divisional Informatics Olympiad problem sets are based on math problems some of which can also be coded.

While Mathematics festivals are attended by thousands (due to unavailability of computers), usually some 50 students participate in the informatics olympiad in each division with some 50 winners chosen from different divisions to participate in National Informatics Olympiad. While BIOC was not as fortunate to have sponsor from its beginning, we have now a wise and generous sponsor in the PHP family, headed by the philanthropist Sufi Md Mizanur Rahma. They have shouldered the financial responsibility of BIOC activities since the 2012 IOI.

Every year, after NIO, we invite high-ranking students to participate in selection contests for ultimate selection to the IOI team. Moreover, throughout the year, we encourage our school students to participate in USACO (USACO Training Program Gateway), Valladolid (Valladolid Online Judge Site), Chinese (ACM-ICPC Online Judge), and Russian sites (Timus Online Judge), Codeforces (Codeforces), Codejam (Google Code Jam 2013) and Topcoder (ToCoder) contests. Students also participate at the online site (Light Oj) created by one of our own computer programming enthusiasts.

This year, we could do a little more. A seven-day camp for informatics olympiad participants was organized alongside a university level contest at a university. Then again, in the month of March, we organized yet another camp at BUET for some 20 students in which they were taught aspects of programming and different algorithms. They sat live tests and three more tests were taken online for final selection of our national IOI team. Students are taught problems similar to those in IOI.

4. Conclusion

We can see a bright future for Bangladeshi students in these academic contests. Students have enormous interest in participating in live contests as well as in online ones that are organized by different sites. They are incrementally demonstrating tremendous problem-solving skills, occasionally outperforming their seniors in universities.

We have been planning to refine the overall procedure now that we have ongoing sponsors to support our activities. There is very much a possibility that soon all olympiads will come under a common banner like TIFR in India. For a limited-resource country like Bangladesh to meet the challenges of the 21st century globalized world and its competition, needless the human resources should be adequately educated and gain the skills of science and engineering – in particular those of ICT in the context that our present Government has declared to create Digital Bangladesh. The only cost-effective way of earning this level of excellence is through competition. We are striving to earn excellence in the field of education through the introduction of olympiad-type popular competitions among young population of the country.

References

ACM-ICPC Online Judge. http://acm.hust.edu.cn/
Bangladesh Bureu of Educational Information and Statistics. http://www.banbeis.gov.bd/webnew/
Bangladesh Informatics Olympiad Committee. https://sites.google.com/site/bioc2009org/
Bangladesh Mathematical Olympiad. http://www.matholympiad.org.bd
Codeforces. http://codeforces.com/
Google Code Jam 2013. https://code.google.com/codejam/
http://www.prothom-alo.com/
Light Oj. http://lightoj.com/login_main.php
The 24th ACM International Collegiate Programming Contest World Finals. http://icpc.baylor.edu/ICPCWiki/attach/History%20-%20ICPC%202000/Standings00.html
Timus Online Judge. http://acm.timus.ru/
ToCoder. http://www.topcoder.com/
USACO Training Program Gateway. ace.delos.com/usacogate.
Valladolid Online Judge Site. http://acm.uva.es
All references were accessed on 31 March 2013.

M. Kaykobad, dr., is a professor in the CSE Department of Bangladesh University of Engineering and Technology, Dhaka. He has been one of the pioneers of introducing academic competitions (like Olympiads) in Banglades and has been leading BUET teams to the World Finals of the ACM ICPC since 1998 in addition to his association with the IOI teams of Bangladesh. He is a fellow of the Bangladesh Academy of Sciences and is a distinguished alumnus of the Fllinders University of South Australia. He is a prolific column writer in the popular daily newspapers always emphasizing great importance of education for developing nations. His fields of interest are algorithms and theory of computational complexity.

Olympiads in Informatics, 2013, Vol. 7, 168–174
© 2013 *Vilnius University*

Olympiads in Informatics: the Georgian Experience

George MANDARIA
International Black Sea University
Tbilisi, Georgia
e-mail: geositu@softhome.net, mandariag@gmail.com

Abstract. This paper describes the history of participation, training, and team selection for the Georgian olympiads, which feed the International Olympiad in Informatics.

Key words: algorithms, online contests, national olympiad in informatics.

1. History of Georgia's International Olympiad Participation

Georgia's first national olympiad was held in 1989 with only a theoretical contest; in 1990, the contest had both theoretical and practical components. Since 1991, the contest has included only practical elements.

The Georgian national team (three students) participated in the olympiad of the former Soviet Union for the first time in 1988 and again in 1989 and 1990. Georgia joined the olympiad of the CIS, Georgia, and Baltic countries in 1992.

From 1989 through 1992 (skipping 1991), the Georgian national team won several diplomas (equivalent of medals) at the Soviet olympiad, including two of II degree and five of III degree in addition to a special prize. One Georgian student, who became a member of Soviet Union's national team owing to his scores in the 1989–1990 Soviet olympiad, won the gold medal at the second international olympiads.

In 1993–1995 the Georgian national team abstained from international olympiads. In 1996, Georgia was invited to the international olympiad in Hungary. Of course, as a country, we had no history of this level of competition, the syllabus, or the types of problems. We had no literature. We worked intensively over the next four years on literature and materials, some of which was shared by our foreign colleagues (from Lithuania, Latvia, Estonia, Russia, Ukraine, Byelorussia, Kazakhstan, and others).

We used those materials to prepare and publish books in the Georgian language. Then, in the second half of the 1990s, the Internet's influence began to grow.

This focus brought us our first silver medal at the IOI in Beijing (China) in 2000. Since then, our students have never returned from an IOI without any medals. Table 1 shows our performance.

In 2001, one of our students won the bronze medal at the age of 11, a feat unrepeated to this day.

The Main Center of Informatics of the Ministry of Education and Science of Georgia started to hold Georgian high school student olympiads and continued to hold them

Table 1

Historical medal counts

Year	Medals			
	Gold	Silver	Bronze	Special
1990	1	–	–	
2000	–	1	–	
2001	–	–	2	
2002	–	1	1	
2003	–	–	2	
2004	–	1	1	Youngest Medalist (IFIP)
2005	–	–	2	Youngest Medalist (IFIP)
2006	–	2	2	
2007	–	–	2	
2008	–	–	3	
2009	–	1	3	
2010	1	–	–	
2011	–	–	1	
2012	–	–	2	
Total	2	6	21	2

until 2005. In 2006–2008 the "club komaroveli" was led to organizing these olympiads. (In both cases these organizations were responsible for organizational support for the selecting and preparing process of the national team and for their participation at the international olympiads). From 2009 till today these olympiads are held by National Examination Center of the Ministry of Education and Science of Georgia. As for organizational support for national team selection, training and participation, the Rustaveli National Science Foundation of the Ministry of Education and Science of Georgia is responsible for the international olympiads.

2. History of Georgia's National Olympiads

The goals and objectives of the national olympiads in informatics are:

- identifying talented, prospective students interested in informatics;
- deepening students' learning of informatics at schools;
- improving students' computer literacy skills;
- the promotion of information technologies;
- raising the interest of study and motivation of young generation in this important area of scientific – technical progress;
- promoting the development of students' professional orientation;
- establishing analytical and creative research, as well as fostering the new computational thinking, which focuses on optimal decisions, including:

- o preparing the ground for adoption in Youth's Information Society and for the future practical activities;
- o growing the country's intellectual potential;
- o raising the Informatics tutors' qualification skills.

The original Georgian national olympiads had three stages. The first stage, school competitions (usually held in January), were the most widespread and enabled all students of public schools to show their practical skills in the field of programming. Winners of the School Olympiads attended the second stage at the regional level, in late February or early March. The third and final stage, the olympiad finals, occurred in late March or early April.

Since the 2004–2005 academic year, the courses for basic programming and study of algorithms were abolished as a compulsory school subject in secondary schools, thus removing the school level contests as well. Accordingly, only students studying in specialized Physics & Mathematical schools or in groups at specialized olympiad training centers participated in olympiads.

These actions, of course sharply reduced the number of students participating in olympiads so, in 2009 it was decided to recreate a school stage of olympiad – which is mostly a theoretical exam – to identify talented students from non-specialized schools and pique their interest in informatics olympiads.

Nowadays, this stage takes place in November and the regional and the final stages are held in first half of February and in the second half of March. Because of the shortage of qualified teaching staff, students who win the theoretical stage generally cannot participate at the next level because they lack programming skill, but for the future we are trying to transfer those students to specialized schools where the informatics olympiad teams operate so the students can exploit their talent and realize their dreams.

In 2010, a set of olympiad alumni now working as professional programmers at various companies independently acquired funds from different sponsors to start the online olympiad (GeOlymp) which now plays a great role in expanding our olympiad and identifying new talents. Additionally, this online olympiad offers opportunities to the students to compete more frequently (along with other international online contests), test their knowledge and abilities, and identify gaps in their knowledge so they can work with the teachers and improve.

3. Competition Levels

There are three groups for the olympiad according to the student's age: 8th grade and younger, 9th–10th grade, and 11th–12th grade. Theoretical and regional contests feature just one contest for every age group, but final round includes a two parts of the olympiad for senior age group.

Younger and middle age group participants are free to participate in more senior age groups' contests if no scheduling conflicts exist. Many students do this and are very successful in the higher-level contests.

Despite the current challenges in our country in the field of teaching of informatics, the format for olympiads sees continuous improvement. Recent innovations cover testing and evaluating participant solutions.

Right now the process of holding olympiads is strictly regulated, beginning from the selection of tasks and ending with the procedures of testing solutions and of summarizing the results. We proudly believe these satisfy international standards. National olympiad contests and national team selection contests, as well as the contests of international olympiads, are strictly automated. A group of experts is responsible for the selection of olympiad tasks, their high quality, and for the proper working of the automated system. This group comprises mainly ex-olympians, through anyone may compose and submit a problem for contests.

4. Georgia Olympiad Training

Preparation for the informatics olympiads spans several years. It starts, often grades 6–7, with learning of the basics of programming and continues with acquisition of algorithmic methods (perhaps during grades 7–8). Success at the national and international level relies on these algorithmic methods. Of course, students continue deeper study of programming and code-writing in parallel in order to facilitate speedy, correct solution-creation in contests.

Georgian computer science literature includes a variety of books for the study of programming; of course, foreign language texts are also available but the language barrier and cost issues for paper textbooks preclude their widespread use. We've found that most university-level and higher education literature are too difficult for our secondary school students and often for their teachers, as well.

As an additional challenge, these books do not include teaching methodologies or methodical instructions. Thus we have created and piloted several of our own textbooks to work with students. They include a collection of tasks with solutions and methodical instructions, an appropriate syllabus (that satisfies all the requirements of the international olympiad's syllabus), and a set of teaching methods that are constantly being refined.

These books include sequential learning challenges and the necessary theoretical and practical material to support the challenges. The above-mentioned "GeOlymp's" team has use their site to publicize the source codes of tasks they have used in addition to text and video versions of solutions. The regime of "Upsolving" is also included.

We believe it is best to begin learning the basics of programming (what we call the 'zero level') in sixth or seventh grade. After that, successful students are enrolled in new groups, where they begin the four-year cycle of learning algorithmic methods. Teaching proceeds according to the student's age, physical, and mental abilities.

The first two years include simple issues that progress over time; the third and fourth years cover algorithms more thoroughly. Two-hour classes are held two to three times a week.

After the first year of study (after students have studied programming and the basic issues of algorithms), those students who do not display appropriate abilities (could not overcome the difficulty level of the olympiad problems) due to teacher's recommendation take orientation to study certain programming languages and participate in the Computer Project Olympiads, where they achieve important successes in most cases. Of course, students who overcame successfully the first year program and continue to study on next level are also allowed to participate in this type of olympiad.

Those students, who are in eleventh grade after finishing the four year course, stay in the group with an individual program of work. In particular, they take an active part in the conducting of lessons with the teachers (which is very useful for them) and are intensively involved in various online olympiads (the number of which is more than enough) among high school students as well as among university students and have achieved significant results.

5. National Team Selection

The IOI is the ultimate goal of the national team aspirants. The winners and revealed perspective students of the final round are enrolled as candidates of the Georgian national team.

The 10 best students will be selected from senior and middle age groups along with the five best students from the youngest group. The four IOI representatives are chosen from this group after two additional selection rounds. They undergo a month-long intensive government-sponsored training period before the IOI, along with four reserve competitors.

The team leaders prepared tasks and conduct the contests. The Rustaveli National Science Foundation finances one additional team leader to travel with the team, who is officially considered the team's third coach in Georgia and takes part in the team preparation process. An adjunct teacher also participates.

Former National Team members, when they are nearby and have time, conduct some of the training sessions. These sessions include both theoretical and practical training that span the set of olympiad topics.

Instructors carefully evaluate the efficiency of algorithms, source code testing methods, working techniques with computers, and attach great importance to clear understanding of task conditions and analysis of tasks' restrictions.

In the last week of preparation before the IOI, the four members of the team travel to the Georgians Mountain resort for final training and a bit of relaxation. They must be ready both physically and mentally for the IOI.

As for the 17-year history of the Georgian team's participation in the International olympiads: In 1996, when our team started to participate in international olympiads, no one school in Georgia didn't work in this direction. We, the leaders of the national team, had to choose students in Georgian olympiads with old criteria and after that we started to train our national team from the zero level. It means that we trained them with the program, which is planned over 4 years. So it was really impossible for us to do this for only one month. So, at first we started to focus a bit relatively on young grade 8–9 but smart students, since they would have sufficient time (before leaving high school) to gain relevant knowledge and experience for achieving success in the international olympiads. This tactic paid off because in 2000 at IOI Beijing our team won the first silver medal and, since then, our National Team has never returned in Georgia without medals.

We wanted to spread our training components more widely than just for IOI team selection so we recruited a number of schools operating their own olympiad groups in informatics and shared our educational programs, relevant literature and recommendations, and constantly communicated. These schools were mainly physics/math schools because they have many gifted children from different regions across the country. Of course, all this yielded positive results, growing the number of candidates for the national team. Competition has increased as well.

Accordingly, since 2000–2001 the team's preparation program was changed, and it suffered some changes annually. Today, the national team training focus has moved to the analysis of more complicated algorithms, improving the knowledge of new team members, on participating in online olympiads, carrying out the daily training competitions and especially on the team members' preparation tactically and psychologically.

We invite students to the team's training camp who will not be team members until the next year or year after that so that they can grow along with the national team (who, of course, too soon graduate to university studies).

6. Conclusion

Informatics olympiads are one of the most efficient means in discovering talented young people and preparing them for future employment in the information technology field. These olympiads perfectly demonstrate their efficiency in search of young talents and the formation of high quality specialists in the field of computer technology.

Additionally, the olympiads provide preparation of high quality staff in this field.

The aim of informatics olympiads is to help students to learn and introduce them to the XXI century professions, when a person's professional and social mobility has decisive importance.

Finally, we note that the Georgian team's achievements at the international olympiads have brought about some very positive emotions between Georgian teachers and students and instilled in them the motivation to work even harder.

Acknowledgments. The National Olympiad in Informatics is conducted by the National Centre of Examination of the Ministry of Education and Science of Georgia, but for organizational support for the national team selection, training process and participation in the international olympiad is the responsible the Shota Rustaveli National Science Foundation Ministry of Education and Science of Georgia.

References

Vasiliev, V.N., Asanov, M.O., Voyakovskaya, N.N., Evstigneev, V.V., Elizarov, R.A., Kiryukhin, V.M., Mikhalev, A.V., Parfenov, V.G., Stankiewicz, A.S., Fedorov, A.G. (2004). *Development of Concept and Creation of Organizational Structure, Teaching-Methodical Ensuring and Software of Innovative Systems for Training of Highly Qualified Personnel in the Field of Information Technology*. Saint Petersburg.

Dagiene, V., Skupiene, J. (2001). *Lithuanian Olympiads in Informatics*, Vilnius.

GeOlymp. `https://geolymp.org`

Ministry of Education and Science of Georgia. `http://www.mes.gov.ge`

National Examinations Center. `http://www.naec.ge`.

Shota Rustaveli National Science Foundation. `http://www.rustaveli.org.ge`

G. Mandaria, Dr. is an associate professor of Tbilisi International Black Sea University (Faculty of Computer Technologies and Engineering). He was a head of Information Technologies Department of Main Centre of Informatics of the Ministry Education and Science of Georgia and a head of Scientific Committee of Georgian National Olympiads in Informatics from 1992 to 2005. He is team leader of Georgian High School Students since 1990 (and for IOI since 1996). He is the coach of students olympiad teams of International Black Sea University in informatics (the world championship (ACM) between universities). His current research interests include programming, data structures and algorithms, algorithmic methods.

Olympiads in Informatics, 2013, Vol. 7, 175–179
© 2013 *Vilnius University*

The Indian Computing Olympiad

Madhavan MUKUND
Chennai Mathematical Institute
H1, SIPCOT IT Park, Siruseri, Kelambakkam 603103, India
e-mail: madhavan@cmi.ac.in

Abstract. The Indian Computing Olympiad is a multi-stage national contest used to select the Indian team for the International Olympiad in Informatics. This article describes the structure of the Indian Computing Olympiad. It also highlights some of the challenges that have been faced and some steps that have been taken to overcome them.

Key words: Indian computing olympiad, IOI, zonal computing oplympiad.

1. Introduction

India started participating in the International Olympiad in Informatics (IOI) in 2002, in Korea. The Indian Computing Olympiad (ICO) was set up in 2001–2002 to select students to represent the country at IOI and has been an ongoing activity since then.

2. Structure

The Indian Computing Olympiad is made up of three stages. The first stage consists of the Zonal Olympiad in Informatics and the Zonal Computing Olympiad, held each year in late November. Both of these contests feed into the Indian National Olympiad in Informatics (INOI), held in January. The top students from INOI are selected for a 10–15 day residential training camp held in May–June, at the end of which the Indian team to IOI is chosen.

2.1. *Zonal Informatics Olympiad (ZIO)*

The Zonal Informatics Olympiad is conducted at about 40 centres across the country. The contest is open to all students in classes 8–12. The number of participants ranges from 5000 to 8000.

ZIO is a pencil and paper exam consisting of 4–5 questions. The questions are problems that require algorithmic or combinatorial insight. Instead of asking students to write out their solutions to the abstract problem, they are provided with three concrete inputs of moderate size. These inputs are expected to be too complex to work out manually without identifying and applying an efficient algorithm. Students get 5 marks for each input correctly solved and a bonus 5 marks if they get all three parts of a question correct. A typical ZIO question is shown in Figure 1. All ZIO question papers and solutions from 2002 onwards are available online (IARCS, 2013b).

2.2. *Zonal Computing Olympiad (ZCO)*

The Zonal Computing Olympiad is an online programming contest. ZCO is held at a fixed time on a Saturday with two relatively simple IOI-style problems to be solved and submitted via an online judge. Like ZIO, the contest is open to all students in classes 8–12. The number of participants is quite small and ranges from 100 to 200. Students can write the exam at any location but must submit a certificate from their school and parents that

Crazyman has decided to tile the floor of his lab in Siruseri with red and green square tiles. He is a very organized person and wants the tiling to be symmetric. His lab floor requires M tiles from north to south and $2N$ tiles from east to west. He has decided to use $M \cdot N$ red tiles to tile the western half of his lab and $M \cdot N$ green tiles to tile the eastern half.

The workers had tiled the lab to perfection, but when Crazyman went out to have lunch, a mischievous person came in, noticed that the cement had not yet set, and swapped some red and green tiles.

When Crazyman came back from lunch, he was heartbroken. He decided to fix the problem himself. Being crazy, however, he decided to restore the symmetry by a sequence of swaps, each involving only adjacent tiles.

For instance, suppose the tiles have been rearranged as follows, where R denotes a red tile and G a green tile. In this case, Crazyman needs 12 swaps of adjacent tiles to restore the tiles to their original arrangement.

```
        R G R R G G G G
West    R G R R G G G G    East
        R R R R G R G G
        R R R R G R G G
```

In each of the following cases, find the minimum number of adjacent pairs of tiles that Crazyman has to swap to restore the symmetry.

```
(a) R G R R G G R G     (b) R G R R G G G R R G
    R G R R G G G G         R R G R R R G G G R
    G G R R G R G G         G G R G G G R G G G
    R R R R G R R G         R R G G R R R R G R

(c) G G R R G G R R
    G G R R G G G G
    G G R R G R G G
    R R R R G R R R
```

Fig. 1. Sample question from ZIO

they attempted the exam without external assistance. ZCO was initiated in 2009 and all question papers are available online (IARCS, 2013b).

2.3. *Indian National Olympiad in Informatics (INOI)*

The Indian National Olympiad in Informatics is an offline programming contest. About 250–300 students are selected from ZIO and ZCO to take part in INOI. INOI is organized at a subset of the centres used to conduct ZIO. The reason to organize INOI offline is because many schools lack reliable Internet connectivity.

The INOI question paper consists of two IOI-style questions. Printed question papers are sent to all the centres. Each student works on a separate computer and leaves his or her solutions on the local hard disk. These solutions are collected and submitted by each centre coordinator via the Internet for automated centralized evaluation.

The level of difficulty is not very different from ZIO. The main purpose is to test for basic programming skills. All INOI question papers from 2002 onwards are available online (IARCS, 2013b).

2.4. *International Olympiad in Informatics Training Camp (IOITC)*

About 25 students are selected through INOI to attend the International Olympiad in Informatics Training Camp (IOITC). This is a fully residential training camp of 10–15 days held in May–June. Students attend lectures before lunch where algorithmic techniques are taught through interactive problem solving sessions involving questions from IOI and other olympiads. During the rest of the day, students work in the lab to solve problems. The lab is set up with a local server that runs an online judge. There are periodic practice tests. The last three days consist of five hour IOI-style exams with three problems on each day. The top four students from these three final exams are chosen to represent India at IOI.

3. Meeting the Challenges

Though 5000–8000 students participate in the first round of ICO each year, this is not a large number in the context of India's student population. The corresponding numbers for other olympiads such as mathematics or physics are typically around 15,000–20,000.

The main obstacle is that computer science (informatics) is not a core subject in secondary school. In those schools that do offer subjects related to computing, the emphasis is on courses that teach computer applications such as the usage of word processors and spreadsheets, rather than algorithmic problem solving. As a result, most students are not exposed to the subject. Also, there is a general lack of awareness about the ICO because there isn't a reliable network of school teachers to publicize this activity, unlike in core subjects like mathematics and science.

Another factor that deters participation is that most students who have an aptitude for mathematics and science are focussed on preparing for competitive entrance examinations to join various engineering courses after school. This preparation effectively takes up all their free time and any activity that is not directly aligned to the process is discouraged. There is, unfortunately, almost no formal incentive in the Indian education system for students who excel in the various international olympiads. A handful of institutions provide direct admission or some kind of preferential treatment, but this is not enough to attract large scale participation.

The school system in India is quite decentralized, so it is almost impossible to directly intervene and implement any schemes to introduce an effective computer science curriculum in schools. The only viable approach seems to be to provide online resources outside the school system for self-learning, accompanied by competitions and other activities to attract the attention of students.

IARCS has begun compiling lecture notes, problems and other material from the annual IOI training camps into an online training archive (IARCS, 2013b). In addition, IARCS also runs an online judge where students can practice their solutions (IARCS, 2013a). This judge currently runs on a software platform developed internally, but it will soon be ported to use the Contest Management System (CMS Development Team, 2013) that was used at IOI-2012 and will be used at both IOI-2013 and IOI-2014 as well. IARCS has intermittently run a monthly online programming contest in the past. There are plans to revive this activity, in cooperation with the CodeChef team (CodeChef, 2013), which has recently started hosting regular online programming contests for Indian students.

It is worth noting that the Indian Computing Olympiad has played an important role in encouraging student interest in algorithms and programming beyond IOI. For instance, in the last 4–5 years, almost every Indian team participating in the World Finals of the ACM Inter-Collegiate Programming Contest (ICPC) has been built around students who have been through the IOI training camp.

4. Organizers

The ICO is coordinated by the Indian Association for Research in Computing Science (IARCS), whose members are drawn from the leading research and teaching departments in Computer Science across the country. IARCS is responsible for the entire academic content of the ICO. This includes setting questions and evaluating answers for all the exams, conducting the training camp and maintaining online resources for students to prepare for the Olympiad.

IARCS conducts ICO in coordination with the Central Board for Secondary Education (CBSE), which is the largest national school board in India. CBSE provides the infrastructure to host ZIO and INOI at multiple centres across the country.

The ICO has been fortunate to have a steady corporate sponsor, Sasken Communication Technologies (Sasken, 2013), that has underwritten the cost of hosting the training camp as well as the international travel for the team to IOI since 2003. The training camp itself has been hosted at The International School Bangalore (TISB, 2013) since 2003.

References

CMS Development Team (2013). *Contest Management System (CMS).*
`https://github.com/cms-dev/cms`
CodeChef (2013). *CodeChef—A Platform for Aspiring Programmers.*
`http://www.codechef.com`
IARCS (2013a). *IARCS Problems Archive.*
`http://opc.iarcs.org.in`
IARCS (2013b). *Indian Computing Olympiad.*
`http://www.iarcs.org.in/inoi`
Sasken (2013). *Sasken Communication Technologies.*
`http://www.sasken.com`
TISB (2013). *The International School Bangalore.*
`http://www.tisb.org`

M. Mukund has been on the Faculty at the Chennai Mathematical Institute since 1992, where he is presently professor and dean of studies. His main area of research is formal methods for specification and verification of computing systems. He is the national coordinator of the Indian Computing Olympiad.

Olympiads in Informatics, 2013, Vol. 7, 180–185
© 2013 *Vilnius University*

Arbiter: the Evaluation Tool in the Contests of the China NOI

Qiyang ZHAO[1], Fan WANG[1], Baolin YIN[1], Hui SUN[2]
[1]*School of Computer Science and Technology, Beihang University*
100191 Beijing, China
[2]*Information School, Renmin University of China*
e-mail: zhaoqiyang@nlsde.buaa.edu.cn, wangfan@nlsde.buaa.edu.cn, yin@nlsde.buaa.edu.cn, sun_h@ruc.edu.cn

Abstract. In this paper, we introduce the evaluation tool, Arbiter, adopted in all contests of the China NOI(National Olympiad in Informatics). Due to the diversity of contest environments, the Arbiter family consists of three types of evaluating tools, ACC, AOC and ALS, in order to fit the camp contests, online contests, and large scale contests respectively. These tools are distinguishable both in technical details and user interfaces. Since 2002, Arbiter has been widely used in all NOI contests as the unique official evaluation tool.

Key words: grader, China NOI, evaluation.

1. Introduction

The contests of the China National Olympiad in Informatics (NOI) include the National Olympiad in Informatics in Provinces (NOIP), NOI Camps, and NOI Online contests (China National Olympiad in Informatics – about). These contests differ both in scale and type. In order to provide comprehensive support, a group of program grading and evaluation tools, the Arbiter family, was developed to fit the different contest environments. These tools run on most popular Linux OSs, including *Ubuntu*, *Redhat*, *Debian*, etc.

The Arbiter family, named after its role, has supported all NOI contests since 2002, from large scale contests of nearly 100,000 junior contestants in 31 provinces to camp contests involving 400 top contestants competing in a single hall (China National Olympiad in Informatics).

2. Arbiter for Camp Contests (ACC)

The *ACC* works in the *Client-Server* mode on a local network as shown in Fig. 1. The entire system comprises a control server and all the machines that the contestants use in the competition along with their submitted programs on those machines. The main parts of the ACC run on the control server, while the contestants' programs are evaluated

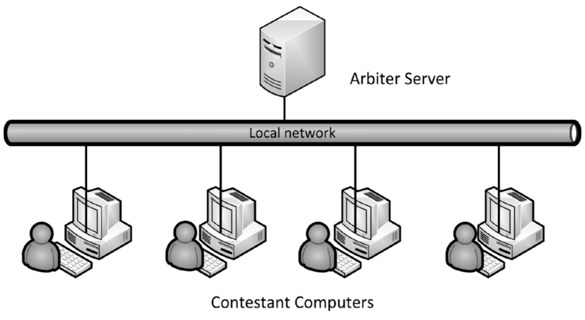


Fig. 1. Arbiter for camp contests.

on their local computers in a highly parallel way. In general, all evaluation tasks can be finished in couple of minutes, and no extra machines are needed.

The control server directs and monitors contest machines in all stages: transferring data, evaluating contestants, and so on. All Arbiter clients work on contestant machines in the daemon mode, receiving and executing commands from the control server, evaluating the contestant answers independently, and filtering all suspicious network packages sent to/from host machines. Arbiter servers and clients communicate with each other by UDP broadcasting and TCP to obtain highest efficiency and reliability as possible.

The evaluation of a camp contest is conducted in four stages: *preparation*, *contesting*, *evaluation*, and *appealing*.

(1) In the **preparation stage**, the server randomly assigns all contestants to the computers. The contest settings and contestant accounts are set, and the contest data, including problem descriptions and sample data, are sent to all contest machines.
(2) In the **contest stage**, all the Arbiter clients monitor contestant behaviors and network packages. Any exceptional actions are reported to the control server to notify contest supervisors so they can deal with it in a timely manner.
(3) In the **evaluation stage**, the server commands all clients to evaluate the contestants' answers on their own machines. Since the evaluation tasks are performed concurrently, all the work of this stage can be finished in minutes. The evaluation scores are sent to the control server, and the results can be checked and forms can be generated for the administration use. After approval, the evaluation results are sent to the corresponding machines of the contestants.
(4) In the **appeal stage**, contestants read the evaluation reports on their own machine, and create appeals for re-evaluation if they see a problem. The ACC is capable of documenting all evaluation histories and solidly supporting the judgment on whether the appeals are reasonable or not.

The control server adaptively chooses suitable schemes for efficiently transferring the contest data. All schemes are based on UDP broadcasting or TCP-based point to point, which is similar to the popular TCP-based BT mode. Data sized of hundreds of MB (being zipped) will be distributed to $400+$ contestant machines in couple of minutes. This is very efficient and meets requirements of all the contests, even one with extraordinarily large testing data.

3. Arbiter for Online Contests (AOC)

In online contests, all contestants are required to submit their answer files to a remote server. The evaluation tasks are automatically assigned to and graded by several evaluating computers, as shown in Fig. 2.

All online contests run in the *Web-Server* mode via Internet instead of a local network. The Arbiter tools are installed only on the contest server, and there are no special requirements on contestant machines (except a web browser, of course). The contestants are required to register on the remote server, read/download the tasks, and upload their answers. Behind the contest server, a cluster of machines evaluates all contestant answers and reports the results to the server. Adding additional machines to or removing existing machines from the cluster is simple. The contestants can get their scores/ranks immediately or in 1–2 days after the check and approval conducted by the administrator, according to the configuration in advance.

An online contest proceeds in four stages: *registration*, *contesting*, *evaluation*, and *appealing*.

(1) In the **registration stage**, contestants register themselves on the Arbiter server and get their accounts and passwords.
(2) In the **contest stage**, contestants log into the Arbiter server and read the tasks on the contest web pages. If configured, the contestants can run their answers on the sample data provided and obtain the feedback from the server.
(3) In the **evaluation stage**, the Arbiter server assigns contestants' answers to the evaluating machines and gets the evaluation results from them. Contestants can read their scores and evaluation specifications on contest web pages.
(4) In the **appealing stage**, contestants submit their appeals to the Arbiter server and wait for further processing. After proper processing, the final scores will be published on official NOI web pages.

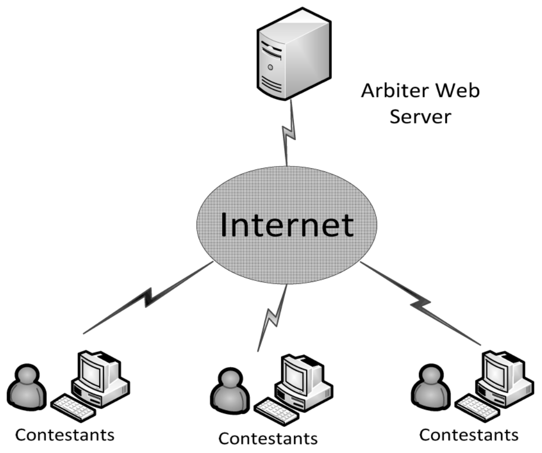


Fig. 2. Arbiter for online contest on Internet.

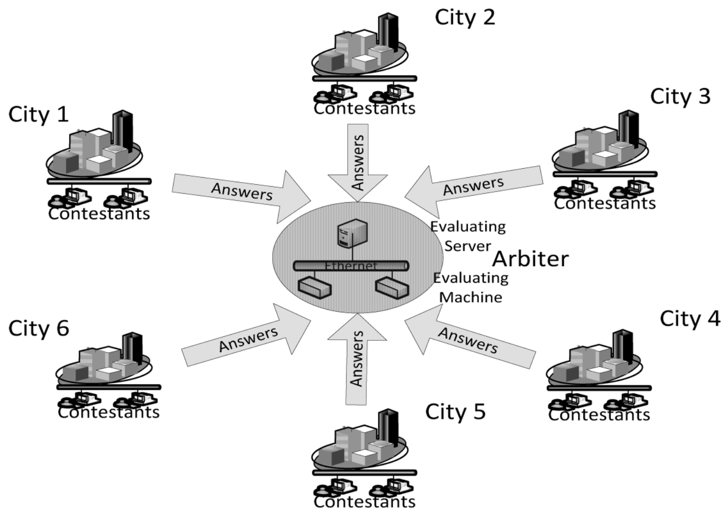


Fig. 3. Arbiter for large scale contests in different cities.

4. Arbiter for Large Scale Contests (ALS)

The NOIP contests are held simultaneously in 31 provinces for about 100,000 contestants. In order to keep the unique standard of evaluation over the provinces, the answers of all contestants are gathered together via email and evaluated in an off-line mode by the ALS system, as shown in Fig. 3.

The ALS runs on an evaluation server in either parallel mode or distributed mode, or both, according to its hardware environment it. At the initiating stage, the ALS tests for the CPU architecture and the network environment. For the multi-core CPUs on evaluating machines, the ALS will assign an evaluating process for each core. If the evaluation server is connected to a cluster of evaluating machines on which the Arbiter evaluators are installed, the ALS will send them the proper number of evaluation tasks according to their capacities, as shown in Fig. 4. In this way, the ALS makes full use of the hardware environment so that the evaluation tasks can be done in high speed.

5. Features of the Arbiter Family

In general, the Arbiter tool family exhibits the following features:

(1) *Flexibility in Usage*
It affords great convenience to contestant administrators in contest administration in all stages. The administrators can easily configure and control contests as required, such as changing the number and the data of the test data, setting the weight of each test case, publishing the results, and so on. The Arbiter server is developed with popular languages and tools, such as C, PHP, JSP, and QT Designer, which ensures compact views, convenient operating interfaces, and comprehensive shortcuts. When designing and developing

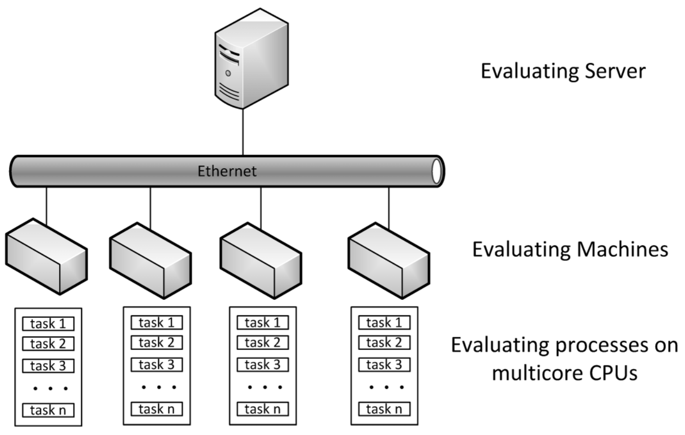


Fig. 4. An evaluating cluster of ALS.

Arbiter tools, a fundamental principle is maintained firmly: there should be fewer than four operations to resolve any complicated administrative affair. This simplifies the operating flows and makes the Arbiter easy to use.

(2) *Stability and Efficiency in Evaluation*
In our experiments, Arbiter is very stable with an error rate less than 0.1%. All contestants' answers are evaluated on contestants' machines or evaluating clusters concurrently, during which Arbiter records and checks all resources used, including CPU time and memory usage. The system will stop the evaluation processes if the program being evaluated runs 100ms more than the specified time. Therefore, little time is wasted if the constant's program runs beyond the time limits.

(3) *Easy to Migrate*
Almost all Arbiter tools are developed with C programming language, while the UI pages are written with QT libraries. Therefore, the Arbiter family is easily migrated to other operating systems with little cost. Furthermore, Arbiter is not designed for evaluating programs in a specified language. It is independent to the compliers of the target programs being evaluated. Therefore, programs in any programming languages can be evaluated, provided the corresponding compilers or interpreters can be found. It is an advantage for the contest organizers when attempting to provide new languages in the contests.

(4) *Standard Interfaces for Data Exchange*
In order to accommodate popular word-processing software such as Open Office, MS Office and so on, Arbiter restores data in standard formats, such as CSV, PostScript, etc. Therefore all configurations, contestant information, and score/rank lists can be easily imported from external data sources or be exported for further use.

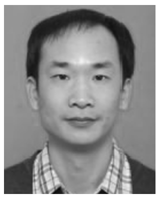
(5) *Secrecy in Data Storage*
Arbiter protects all critical data by encryption and access controls. The contestants are unable to access the data files directly, and all data operations must be performed using Arbiter tools, in accordance to the contest regulations.

6. Conclusions

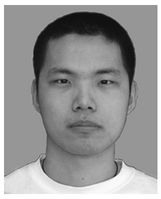
Up to now, the Arbiter tools have evaluated about 700,000 contestants, 5,000,000 problem answers, on about 60,000,000 test cases. According to official statistics published by the NOI Scientific Committee, the error rates of these evaluating tasks are about 0.05%–0.25%. It shows that Arbiter family is stable and efficient, especially in large scale contests. Therefore, the Arbiter family has become the official evaluating system for all the NOI contests since NOI 2002 Summer Camp.

References

China National Olympiad in Informatics. `http://www.noi.cn/about/summary/`
China National Olympiad in Informatics. `http://www.noi.cn/noi2009-noip2009/`

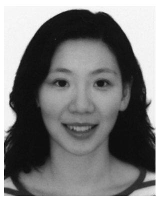


Q. Zhao, dr., is associated with the School of Computer Science and Engineering (SCSE), Beihang University (BUAA) of China. His major interests are in computer vision and image processing. He is a member of NOI science committee since 2009.



F. Wang is now pursuing his PhD degree in the School of Computer Science and Engineering (SCSE), Beihang University (BUAA) of China. There he received his master degree in 2011. His major interest is in computer vision.

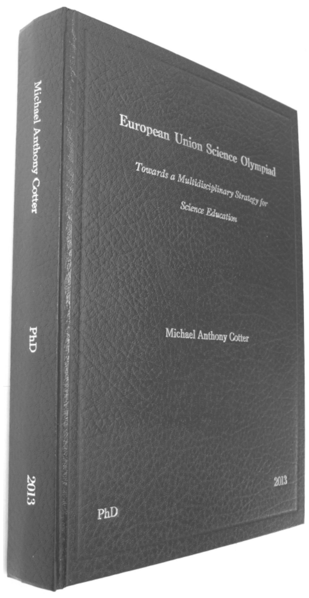
B. Yin is a professor at the School of Computer Science and Technology, Beihang University of China. He received his PhD degree in computer science from the Department of Artificial Intelligence, the University of Edinburgh in 1984. He has served as the vice chairman of Scientific Committee of National Olympiad in Informatics since 1999.



H. Sun, dr., is associated with Information School, Renmin University of China. She received her PhD degree from Tsinghua University in 2005. Her research interests are in data mining. Since 2005, dr. Sun is a member of NOI Science Committee and she will be deputy leader of China Team in IOI 2013.

Olympiads in Informatics, 2013, Vol. 7, 186–190
© 2013 *Vilnius University*

REVIEWS, COMMENTS



European Union Science Olympiad
Towards a Multidisciplinary Strategy for Science Education

Author: Michael Anthony COTTER
Publishing house: Dublin City University
Country: Ireland
Year of edition: 2013
Language: English
Number of pages: 303
ISBN: N/A

This PhD thesis, by Michael A. Cotter, is primarily practice-focused research on a progressive version of science education with historical, social and political dimensions. It tracks the development and organisation of a unique science competition for sixteen year old students, the European Union Science Olympiad (EUSO) from its conception to the end of its tenth year. The motivation for establishing the EUSO was to develop, organise and maintain over an extended period of time a science competition which would develop "Team Science Tasks" that would integrate science, be problem-based, be connected to the real world and involve the construction of knowledge, higher-order thinking, alternative solutions, depth of knowledge and sophisticated communication between the team members. The EUSO was designed to fill a gap in the science olympiad landscape, whilst simultaneously functioning as a stimulus for increased interest in and enjoyment of scientific phenomena at a critical period in students' education.

A crucial element in this success story has been the methodology used in the organisation of the EUSO – *Participatory Action Research* (*PAR*). Through this democratic process, participants were released from the constraints of the established olympiad structures and enabled to function in an environment which allowed them to investigate their own reality in order to change it. Such actions have resulted in the development of a new science olympiad model, the EUSO model.

The idea of the EUSO crystallized for the researcher following his attendance at and his experience of organising Ireland's participation in the 6th International Olympiad in Informatics (IOI), in Haninge, Sweden in 1994. At this event, he met and spoke with the then President, Dr. Yngve Lindberg (1900–1966) and with the Chairman of the Scientific Committee for IOI 1994, Dr. Håkan Strömberg. This enabled him to organise Ireland's participation in the IBO, IChO and IPhO in subsequent years.

In the early years, Irish students' success stories emanated from the IOI, primarily through the establishment of a junior section in the Irish Informatics Olympiad and to the setting up of online computer programming clubs and training programmes by Mr. Charlie Daly, the Irish team leader. As a result, Eóin Curran was awarded a silver medal at IOI 1997 and Martin Orr a gold medal at IOI 2003. These results suggested that Ireland's performance at other olympiads could be enhanced through early intervention.

Much of the documentation emanating from the EU and elsewhere signalled a consensus regarding the features of science education that EU member countries had in common. These included:

- decline in the numbers studying science in secondary schools;
- sharp decline in the numbers studying chemistry and physics;
- steep decline in the number of girls studying chemistry and physics;
- lack of interest among students in school science;
- poor attitude towards science and scientists;
- shortage of suitably qualified science teachers, particularly in chemistry and physics;
- ineffectual teaching of science;
- single subject science curricula;
- lack of integration in school science;
- lack of experiments / practicals in school science;
- shortage of university science graduates.

The researcher was fully aware that his personal perspectives and preconceptions would have a bearing on the research process and his approach to answering the research question. In this regard he realised that he subscribes to the view that the kind of knowledge that is valid and satisfactory is that which is created through the subject's interactions with the world. However, subjects construct their own meaning in different ways and experience the world from different perspectives. Such a constructivist stance calls for an interpretivist approach to knowledge creation. In interpretivism, the *'world is interpreted through the classification schemas of the mind'* (Gray, 2006, p. 20); the emphasis is on understanding the real workings behind 'reality'. For the researcher, attempts to understand this reality were grounded in people's experiences of that reality. The task was to explore people's multiple perspectives in the natural settings.

The researcher was also mindful of the fact that his chosen methodology should align with his epistemological stance and theoretical perspective, i.e., constructivism and interpretivism, respectively. In this regard, the postmodernist approach to knowledge creation promoted by Action Research (AR) was considered appropriate. AR enables the researcher to:

'Develop a context in which individuals and groups with divergent perceptions and interpretations can formulate a construction of their situation that makes sense to them all – a joint construction' (Stringer, 1999, p. 45).

For the researcher, the particular AR model chosen, namely Kemmis and McTaggart's (2003) Participatory Action Research (PAR) was selected due to its primary focus: authentic participation. In addition, the emphasis on investigation of actual practices and the concentration on transformation of practitioners' practices in an egalitarian manner enabled adoption of PAR as the model most likely to have the capacity to handle the variety of challenges that the EUSO concept would inevitably generate.

However PAR does not cover all aspects of this research. The historical aspect of the EUSO is a crucial aspect and a vital component of the narrative.

The researcher's direct involvement in the IBO, IChO, IPhO and the IOI gave him a unique insight into their structures, organisation and management and gave him access to a wide range of influential people who gave him their views on aspects of the events and the profile a new olympiad might adopt. This raised ethical issues around privileged information and how it might be used. However the researcher was open and not clandestine or covert in any way and as many opinions as possible were sought. Cooperation and collaboration were ensured and dissenting opinions were listened to. Gray (2006) citing Badger (2000) suggests that

"At least superficially Action Research seems to pose few ethical dilemmas because it is based on a philosophy of collaboration for the mutual benefit of the researchers and participants" (p. 388).

While Action research is not a "smash and grab" approach to research or what Lather (1986) calls "rape research" (p. 261), it requires negotiated access, confidentially and the right to withdraw.

This thesis reviews the limited literature on the history of the international science olympiads from the Leningrad Mathematical Olympiad (LMO) in 1934 to the present day olympiads by using primary sources where possible. Literature on relevant topics raised by the research question such as, why teach science are explored. The central role of science education in the education systems across the EU is investigated, in particular the perceptible decline in participation in the Senior Cycle of the secondary education systems in chemistry and physics by all students, but especially girls. This has raised concerns about the future supply of well qualified scientists, engineers and science teachers. This has led to much research on the pivotal role of the teacher, their qualifications and training, the curriculum content of the subjects they teach and the teaching methods that are employed and how this in turn influences the interest and attitude of students. The critical role of assessment is reviewed.

Chapter 4 of this thesis outlines the research methodology chosen. The Epistemological Stance and Theoretical Perspective of the researcher are presented as is the rationale for choosing Participatory Action Research (PAR), because of its aim to transform and to be participatory, practical, collaborative, emancipatory and reflexive. While acknowledging the overarching importance of the PAR approach, the import of the setting and historical account of the EUSO over a ten year period is emphasised. The advancement of the Action Research methodology from the early work of Kurt Lewin (1890–1947) through its many modifications and developments to PAR developed by Kemmis and McTaggart (2003) is described. The challenges faced and overcome, the data collection method of informal and formal meetings, diaries and interviews are graphically presented in five cycles over a five year period. The cyclical nature of the research is further described in the development of the EUSO constitution from 2002–2012

The thesis presents a brief summary of the twenty EUSO Tasks developed over a ten year period 2003–2012. It also serves as a historical record which describes how these unique Team Science Tasks, a central feature of the EUSO were created. As well as each summary it also illustrates how each Task contributes, in its own way, to the concept of "Rich Tasks" in a progressive version of science education. Each task was reviewed to see if it conforms to Progressive Pedagogies (Hayes *et al.*, 2006). In line with PAR, it describes how the Scientific Committee from the host country chooses the topics and designs the Task before presentation to the mentors at a General Assembly (GA) meeting. All aspects are discussed before finalisation by task designers and mentors in a collaborative manner. The need for future research into the amount of interaction that takes place between team members is alluded to. The elimination of high-stakes assessment is emphasised as a feature of the EUSO in the presentation of the results. This feature developed during the PAR process, by getting the mentors to emancipate themselves from the familiar concept of olympiads where the emphasis is on winning and not on taking part.

The results from the EUSO are presented in two five-year periods because of EU enlargement in 2004 and 2007. This contributes to the historical record of the EUSO and highlights some of the trends that have developed. It was expected that the former Soviet Bloc countries, because of their success at the international olympiads would perform well. The interviewing, which was one of the research methods used in PAR was continued to help the researcher comment on the results. In the first five years the former Soviet Block countries featured strongly in the gold and silver medal categories while the early EU members performed well in the silver category and were dominant in the bronze category. In the years 2008–2012 the former Soviet Bloc countries were dominant in the gold and silver categories while the early EU members featured strongly in the bronze category.

The final chapter, Conclusions & Recommendations, points to the success of the EUSO as the product of Participatory Action Research (PAR). Ten years of the EUSO has seen the number of participating countries and the numbers of students steadily increase. Ten EU governments have hosted the EUSO between 2003–2012 and the science faculties of the universities in these countries have cooperated to develop twenty, integrated, content rich, team science tasks. Finally recommendation including the promotion of an integrated science curriculum and the possibility advanced programmes in science. Areas where additional areas of research might be undertaken are highlighted.

References

Cotter, M., Petersen, S. (Eds.) (2013). Tasks of the European Union Science Olympiads 2003–2007. In: *Challenging Interdisciplinary Science Experiments*, Vol. 1, Münster, Waxmann.

Dillon, J., Osborne, J. (2008). *Science Education in Europe: Critical Reflections*. London, The Nuffield Foundation.

EACEA/Eurydice, Eurostat (2009). *Key Data on Education in Europe 2009*. Brussels, Eurydice.

European Commission. (2007). *Science Education Now: A Renewed Pedagogy for the Future of Europe*. Brussels, European Commission.

Gray, D. (2006). *Doing Research in the Real World*. London, SAGE Group.

Hayes, D., Mills, M., Christie, P., Lingard, B. (2006). *Teachers and Schooling*. Sydney, Allen and Unwin.

Kemmis, S., McTaggart, R. (2003). Participatory action research. In: Denzin, N., Lincon, Y. (Eds.) *Strategies of Qualitative Inquiry*. London, Sage.

Lather, P. (1986). Research as Praxis. *Harvard Educational Review*, 56(3), 157–277.

OECD (2009). PISA 2009. *Assessment Framework – Key Competencies in Reading, Mathematics and Science*. Paris, OECD.

Stringer, E.T. (1999). *Action Research*, 2nd edn. Thousand Oaks, CA, Sage.

Dr. Carmel Mulcahy

Looking for a Challenge?
The Ultimate Problem Set from the University of Warsaw Programming Competitions

Looking for a Challenge

*Editors: Krzysztof Diks, Tomasz Idziaszek,
Jakub Łącki, Jakub Radoszewski*
*Publishing house: Faculty of Mathematics,
Informatics and Mechanics, University of Warsaw*
Country: Poland
Year of edition: 2012
Language: English
Number of pages: 428

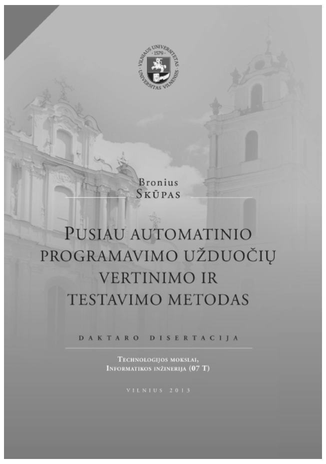
One of the constants during my involvement with programming contests is the fact that there are always good problems from Polish contests. Problems from X and XI OI (2003 and 2004 Polish IOI team selections) were among the first and most memorable hard problems shown to me by my mentors. Most of my conversations at IOIs with Chinese team members inevitably drifted towards how we solved that year's POI problems. As a coach, a significant portion of the problems that I point contestants towards are from POI/PA/AMPPZ. The book 'Looking for a Challenge' is a compilation of the best among these problems, and will be a valuable resource for contestants at all levels for at least the next decade.

The book is organized into a series of disjoint chapters, each about a problem and its solution. Each set of chapters was written separately by a long time contributor to Polish programming competitions. These authors include the founders of the Polish contest programs as well as top contestants throughout the years. As the problems were chosen separately, the solution analyses often reflect the contributors' unique problem solving approaches. These insights are perhaps even more valuable than the problems themselves. They represent a large sample of top problem solvers' way of approaching, breaking down, and solving a problem. The problems selected cover a wide range of difficulty levels. Problems such as Canoes (p. 338), Triangles (p. 352), and Party (p. 395) are solved by elementary but tricky observations. On the other end of the spectrum, some problems' solutions naturally lead to key ideas from active areas of CS research. For example, Ants (p. 18), Superknight (p. 240), and Guesswork (p. 416) are related to streaming algorithms, lattice reduction, and Bayesian statistics respectively. In between, many of the problems such as Permutations (p. 91) and Cave (p. 264) represent recurring themes in Polish contests with many interesting follow-ups and variations.

The organization of the chapters makes this book more suitable as supplemental material for developing problem solving skills. In this role, the choice of problems once again shines through. Many solutions have short implementations, making them ideal for students who are just getting familiar with programming. Each chapter includes a link to an online judge where readers can test if their solutions are correct. Most of the solutions are written in a self-contained manner, but as one gain more exposure to various techniques, the analysis can also be read once again from a more systematic perspective. Perhaps a good approach to using this book is to treat it as a series of disjoint articles, each to be read individually but repeatedly over time. When I heard that a friend from Warsaw University was visiting CMU, I suggested that he replace all empty space in his luggage with copies of 'Looking for a Challenge'. It is with the same enthusiasm that I recommend this book to you. It is a wonderful resource that provides a glimpse of the past, present, and future of Polish programming competitions.

Richard Peng

Olympiads in Informatics, 2013, Vol. 7, 193–197
© 2013 *Vilnius University*



Bronius
Skūpas

Pusiau automatinio programavimo užduočių vertinimo ir testavimo metodas

Daktaro disertacija

Technologijos mokslai, Informatikos inžinerija (07 T)

Vilnius 2013

Method for Semi-Automatic Evaluation and Testing of Programming Assignments

Abstract of dissertation in English, 33 pages

Author: Bronius SKŪPAS
Publishing house: Vilnius University
Country: Lithuania
Year of edition: 2013
Language: Lithuanian
Number of pages: 110

This is a PhD thesis (dissertation) defended at Vilnius University on February 11, 2013. The reason for this review in the journal is that it describes the developed system for evaluation of programming assignments, which is very important while teaching programming in secondary schools as well as training students for informatics olympiads.

Teaching programming is a difficult process because of the association with creative thinking, strictly formalized tasks, and practical programming assignments. The students have to be trained to create not only running, but also qualitatively designed reliable, properly functioning programs. Testing and evaluation of programs developed by the students requires a lot of the teacher's efforts and time.

In the thesis, automatic program testing is defined as dynamic testing, based on black box testing with tests prepared in advance. In order to use automatic program testing, programming tasks are often specified in detail, i.e., the required data input and output formats are introduced. This allows evaluating by applying the fact-verification method. Accurately identified verifiable fact is defined as an evaluation criterion.

Automatic programming assignments testing can be used in the teaching process, during programming exams, professional programming knowledge tests during recruitment and programming competitions.

Many researchers note that fully automated testing, based on static and dynamic analysis, cannot be completely fair. Therefore common practice is to use semi-automated testing of programming assignments, which is a mixture of automatic testing and manual evaluation and provides greater flexibility in the use of automated testing benefits.

The thesis focuses on the exploration of possibilities of using a semi-automatic testing system for programming assignments in schools, distance education, as well as competitions and exams. Common evaluation errors are analysed as well. The research is focused on creating a new method of evaluation for programming assignments in order to achieve high quality evaluation in acceptable time and the justification of the evaluation to the user.

The aim of the work was to develop a method for semi-automatic evaluation and testing of programming assignments that would improve system and evaluator interaction in order to increase the efficiency and the quality of the evaluation and to implement it into a prototype of semi-automatic evaluation system.

The constructive research method was chosen as the key method for the research. Kasanen *et al.* (1993) states six phases of the constructive approach: (1) finding a practically relevant problem with research potential; (2) obtaining general and comprehensive understanding of the topic; (3) innovating, i.e., constructing a solution idea; (4) demonstrating that the solution works; (5) showing the theoretical connections and the research contribution of the solution concept; (6) examining the scope of applicability of the solution.

When designing the Maturity exam of Information Technology/Informatics in Lithuania it was decided that the submitted programs designed by graduating high school students should be evaluated even if they do not compile. It was also agreed that the evaluation should be positive, i.e., the points should only be given for the skills demonstrated by the student. From the exam requirements it was clear that evaluation should be either manual or semi-automated. Manual evaluation was excluded due to limited number of evaluators and the limited time resources.

When considering the possibilities of applying semi-automated evaluation for evaluation of the Maturity exam submissions, a raised hypothesis was that deeper analysis of the process of manual evaluation of programs designed by the students would enable the creation of a new semi-automated testing and evaluation method which would have the same quality and precision as manual evaluation but higher efficiency (Fig. 1).

Long observation of informatics teachers evaluating programs manually as well as observation of the students searching for mistakes in their programs served as a premise for stating the hypothesis above. The process of designing a short program (i.e., no longer than 100 lines) is similar to that of designing complicated software systems.

The researcher attempted to benefit from the experience of evaluation in the Lithuanian informatics olympiads. However it turned out that the demands of the exam are different from that of the olympiads:

1. The evaluation system should operate in the environment as similar as possible to that used during the exams by the students (Windows OS, FreePascal compiler; after 2010 also Mingw g++).
2. The task cannot be considered as partially solved if the program passes just some of the grading tests. The tasks are rather simple and the reasons for each failure should be identified.
3. The program with minor mistakes (missing punctuation mark, undeclared variable, etc.) cannot be assigned few or no points because of those mistakes only.

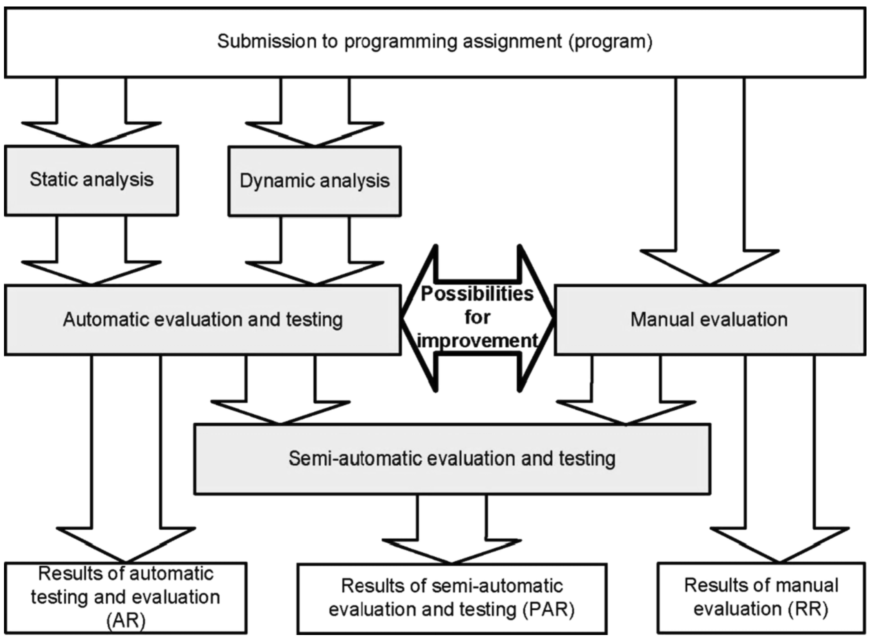

Submission to programming assignment (program)
Static analysis
Dynamic analysis
Automatic evaluation and testing
Possibilities for improvement
Manual evaluation
Semi-automatic evaluation and testing
Results of automatic testing and evaluation (AR)
Results of semi-automatic evaluation and testing (PAR)
Results of manual evaluation (RR)

Fig. 1. Scheme illustrating suggestions how to improve the evaluation process.

4. There is no need to support the role of the student in the exam evaluation system, because the student cannot observe his/her score directly during the exam.

During the evaluation the semi-automated evaluation system should provide the administrator functionality to:

- load task specifications together with tests into the system;
- modify task evaluation criteria;
- to compose, assign and present to the evaluator an evaluation package containing a set of submitted programs to be evaluated;
- to assign an evaluation package to another evaluator;
- to observe and record the outcome of evaluation process.

A semi-automated evaluation system should provide the following functionalities for evaluating a submitted program:

- to be able to execute statistical analysis plug-ins and provide the results to the evaluator;
- to automatically test the submitted program using black box testing;
- to present the results of automated testing and the program being evaluated to the evaluator;
- allow modification of the program being evaluated and to retest the modified program without changing scores assigned by automated testing; to display the successfully passed tests, to visualize the changes made by the evaluator and to allow to restore the initial version of the submitted program;

- allowing to fill/modify the scores obtained from semi-automated testing if automated testing did not assign any points to the program;
- allowing to fill/modify the manual evaluation scores.

There were also added *non-functional requirements*. Those will not be emphasized in the formalized problem statement in this thesis, because their implementation is not so abstract. Here are the most important ones:

Semi-automated evaluation system should allow:

- to evaluate incomplete programs in a positive manner;
- to help finding the exact location of an error;
- to provide clear feedback messages to the user-evaluator;
- to operate fast;
- to allow the evaluator to comfortably experiment with the program being evaluated.

After formalizing the problem and emphasizing the role of the evaluator (i.e., dissociating from many roles common in various grading systems) the prototype of semi-automated evaluation system was created, which implemented the grading of one program. In the system there was implemented safe black box testing, reviewing the program under evaluation and collecting scores from manual grading.

After some experiments with the evaluation system being developed it was observed that for the evaluators it is not easy to find a mistake in the program. In such case the evaluators used to open the program in some widely accepted IDE and started experimenting there. However most IDEs do not provide functionalities for testing the program with a group of tests. This observation suggested the idea to transfer some IDE functionalities to the evaluation system. The first and the essential improvement was the following: the program text review component was replaced by program text editing component and there was added a button allowing batch retesting of the modified program. This improvement is illustrated in Fig. 2.

In conclusion, the researcher investigated and classified the functionalities of modern automatic and semi-automatic programming assignments testing systems and discovered that this class of software still encounters a lot of problems, the systems are being improved and new testing systems are being developed.

Improvements to the semi-automatic evaluation method presented in this work allows the evaluator to use automatic testing system interactively, to carry out the experiments with the program being evaluated, to analyse interactively the consequences of the errors discovered in the programs of the students, and to consider the weights of the mistakes made by the students.

The Lithuanian State Matura exam of Information Technology practical task assessment software presented in this work implements the proposed semi-automatic evaluation method. The system has been used successfully by the National Examination Centre; it is constantly being improved, adapted to the changing needs.

The effectiveness of manual evaluation method was compared to the proposed semi-automated evaluation method: the efficiency increased from 1.4 to 2.2 times.

Qualitative analysis of evaluation results shows that results obtained by using the proposed semi-automatic evaluation method and the automatic testing differ significantly.

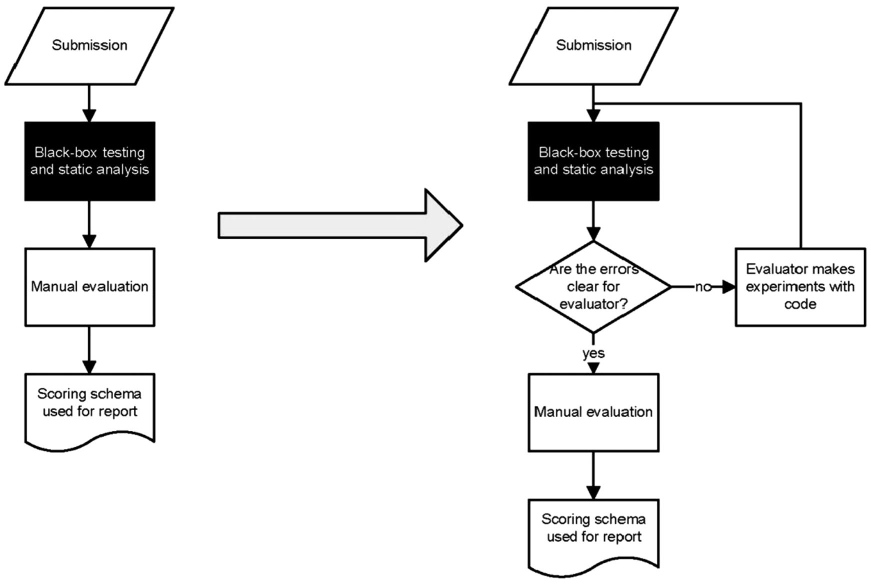

Submission
Black-box testing and static analysis
Manual evaluation
Scoring schema used for report
Submission
Black-box testing and static analysis
Are the errors clear for evaluator?
no
Evaluator makes experiments with code
yes
Manual evaluation
Scoring schema used for report

Fig. 2. Semi-automated program testing process improvement.

It was confirmed that the application of the proposed method resulted in higher quality of evaluation.

The method has also been also been implemented and tested in the EduJudge plug-in (`http://eduvalab.uva.es/en/projects/edujudge-project`) to the learning management system *Moodle*. (It is pity that this plugin is still not available for general public). The experiment confirmed that the improved semi-automatic evaluation method can be easily implemented in other systems.

References

Kasanen, E., Lukka, K., Siitonen, A. (1993). The constructive approach in management accounting research. *Journal of Management Accounting Research*, 5, 243–264.

Valentina Dagienė and Bronius Skūpas
(prepared by Abstract of dissertation)

Instructions to Authors

OLYMPIADS IN INFORMATICS is a peer-reviewed scholarly journal that provides an international forum for presenting research and developments in the specific scope of teaching and learning informatics through olympiads and other competitions. The journal is focused on the research and practice of professionals who are working in the field of teaching informatics to talented student. OLYMPIADS IN INFORMATICS is published annually (in the summer).

The journal consists of two sections: the main part is devoted to research papers and only original high-quality scientific papers are accepted; the second section is for countries reports on national olympiads or contests, book reviews, comments on tasks solutions and other initiatives in connection with teaching informatics in schools.

The journal is closely connected to the scientific conference annually organized during the International Olympiad in Informatics (IOI).

Abstracting/Indexing

OLYMPIADS IN INFORMATICS is abstracted/indexed by

- Cabell Publishing,
- Central and Eastern European Online Library (CEEOL),
- EBSCO,
- Educational Research Abstracts (ERA).

Submission of Manuscripts

All research papers submitted for publication in this journal must contain original unpublished work and must not have been submitted for publication elsewhere. Any manuscript which does not conform to the requirements will be returned.

The journal language is English. No formal limit is placed on the length of a paper, but the editors may recommend the shortening of a long paper.

Each paper submitted for the journal should be prepared according to the following structure: (1) a concise and informative title; (2) the full names and affiliations of all authors, including e-mail addresses; (3) an informative abstract of 70–150 words; (4) a list of relevant keywords; (5) full text of the paper; (6) a list of references; (7) biographic information about the author(s) including photography.

All illustrations should be numbered consecutively and supplied with captions. They must fit on a 124×194 mm sheet of paper, including the title.

References cited in the text should be indicated in brackets, e.g., for one author – (Johnson, 1999), for two authors – (Johnson and Peterson, 2002), for three or more authors – (Johnson *et al.*, 2002). If necessary, the page number may be indicated as (Johnson, 2001, p. 25).

The list of references should be presented at the end of the paper in alphabetic order. Papers by the same author(s) in the same year should be distinguished by the letters a, b, etc. Only Latin characters should be used in references.

Please adhere closely to the following format in the list of references.

For books:

Hromkovič, J. (2009). *Algorithmic Adventures: From Knowledge to Magic*. Springer-Verlag, Berlin.

Schwartz, J.E., Beichner, R.J. (1999). *Essentials of Educational Technology*. Allyn and Bacon, Boston.

For contribution to collective works:

Batissta, M.T., Clements, D.H. (2000). Mathematics curriculum development as a scientific endeavor. In: Kelly, A.E., Lesh, R.A. (Eds.) *Handbook of Research Design in Mathematics and Science Education*. Lawrence Erlbaum Associates Pub., London, 737–760.

Plomp, T., Reinen, I.J. (1996). Computer literacy. In: Plomp, T., Ely, A.D. (Eds.), *International Encyclopedia for Educational Technology*. Pergamon Press, London, 626–630.

For journal papers:

McCormick, R. (1992). Curriculum development and new information technology. *Journal of Information Technology for Teacher Education*, 1(1), 23–49. `http://rice.edn.deakin.edu.au/archives/JITTE/j113.htm`.

Burton, B.A. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.

For documents on Internet:

International Olympiads in Informatics (2011). `http://www.IOInformatics.org/`.

Bebras – International Contest on Informatics and Computer Fluency (2007–2011). `http://bebras.org/en/welcome`.

Authors must submit electronic versions of manuscripts in PDF to the editors. The manuscripts should conform all the requirements above.

If a paper is accepted for publication, the authors will be asked for a computer-processed text of the final version of the paper, supplemented with illustrations and tables, prepared as a Microsoft Word or LaTeX document. The illustrations are to be presented in TIF, WMF, BMP, PCX or PNG formats (the resolution of point graphics pictures is 300 dots per inch).

Contacts for communication:

Valentina Dagienė
Vilnius University Institute of Mathematics and Informatics
Akademijos 4, LT-08663 Vilnius, Lithuania
Phone: +370 5 2109 732
Fax: +370 52 729 209
E-mail: valentina.dagiene@mii.vu.lt

Internet Address

Olympiads in Informatics provides an early access to the articles by placing PDF files of the papers on the Internet. All the information about the journal can be found at:

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