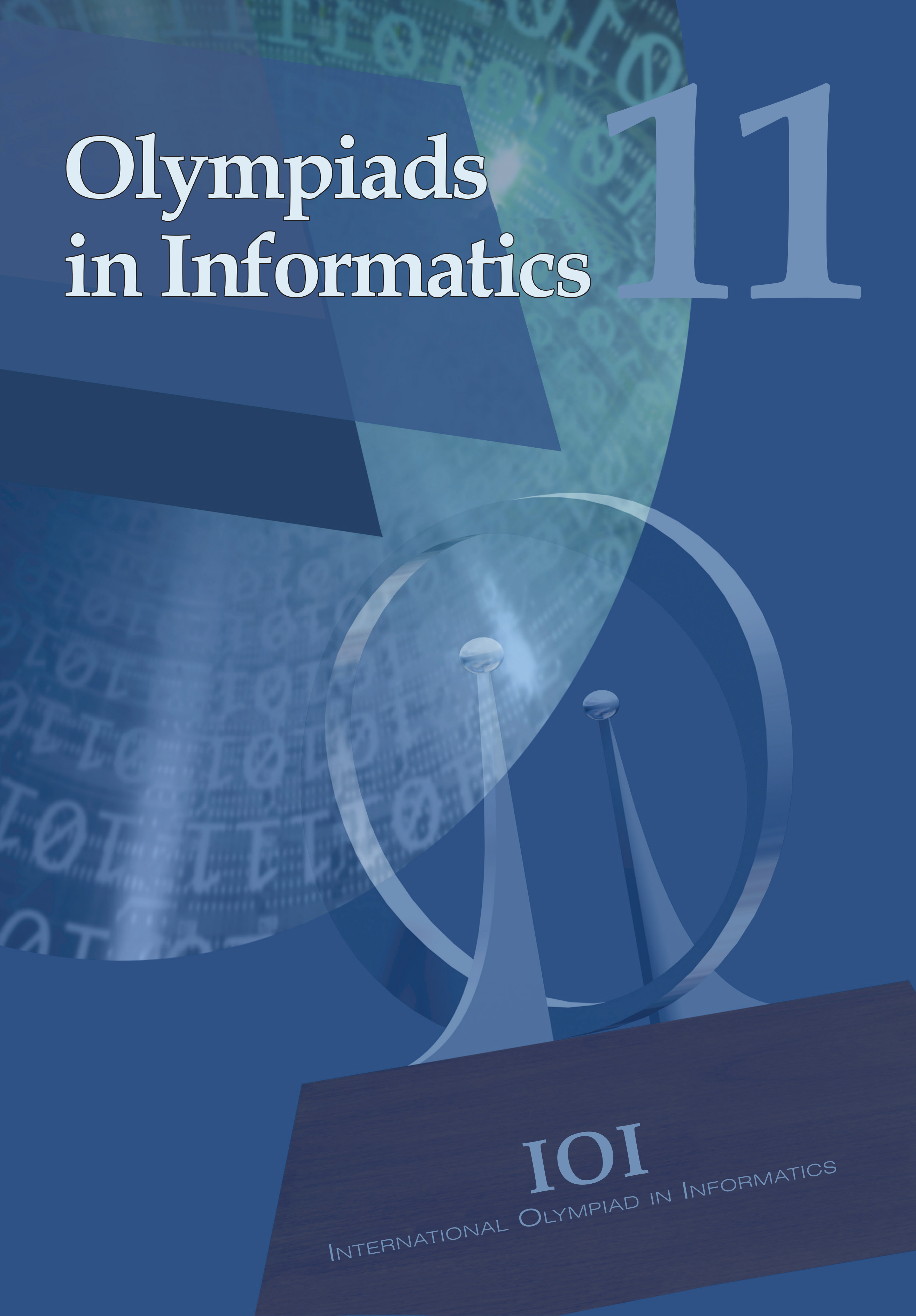
# Olympiads in Informatics

11



ISSN 1822-7732

**INTERNATIONAL OLYMPIAD IN INFORMATICS**
**VILNIUS UNIVERSITY**
**INSTITUTE OF MATHEMATICS AND INFORMATICS**

# OLYMPIADS IN INFORMATICS

Volume 11 2017

Selected papers of
the International Conference joint with
the XXIX International Olympiad in Informatics
Tehran, Iran, 28 July – 4 August, 2017

The journal Olympiads in Informatics is an international open access journal devoted to publishing original research of the highest quality in all aspects of learning and teaching informatics through olympiads and other competitions.

`http://ioinformatics.org/oi_index.shtml`

ISSN 1822-7732 (Print)
2335-8955 (Online)

Olympiads in Informatics, 2017, Vol. 11, 1–2
© 2017 *IOI, Vilnius University*

# Foreword

The International Olympiad in Informatics (IOI) community gathers each year during the olympiad and has the opportunity to communicate during this worldwide event. Each country has many things to present and discuss. The national olympiads do not exist in isolation, and the papers from the 11th IOI conference show how similar problems arise in different countries and different environments.

The IOI Journal is focused on the research and practice of professionals who are working in the field of teaching informatics to talented students. The format for the Journal follows three tracks (since the 5th volume): the primary section of the journal focuses on research, the second section is devoted to sharing national experiences, and the last section presents book reviews and other related information.

In this volume, we publish a detailed research on the development of the curricula for Computer Science education in schools, in the paper "On a Methodology for Creating School Curricula in Computing" written by professors Krassimir Manev and Neli Maneva. The authors propose an idea for methodology for the development of school curricula in computing. These ideas were raised during the IOI Workshop, held in Bitola, Macedonia, in April 2015. The main goal of the Workshop was to review the current state of teaching computing in schools in the presented countries and to propose some ideas for development of an International School Curriculum in Computing, based on the experience of the IOI community.

The paper "Learning and Teaching Algorithms Design and Optimisation Using Contests Tasks" by Sébastien Combéfis and his colleagues presents training materials built from contest tasks to teach and learn how to design algorithms that solve concrete and contextualised problems. The first learning modules will be built thanks to a pedagogical device that will be deployed during the 2017–2018 academic year. The next step of this work, after the creation of the first modules, will be their evaluation in real conditions, that is, using them to teach algorithm design and optimisation related aspects and topics to interested learners.

The IOI conferences sustain the aim to become a part of, and to bring in, the wider pedagogical community. Each year there are typically one or two papers devoted to this purpose. In this volume we have a detailed paper titled "Learning Trajectory of Item Response Theory Course Using Multiple Softwares" written by Henri Retnawati from Indonesia. The paper describes learning trajectory of item response theory (IRT) that has been integrated to use multiple software and obstacles in an advanced IRT course. This design research study is of validation-study type with qualitative approach. The learning trajectory that had been hypothesized was based on the researcher's experience in lecturing and in identifying the necessary requirements for each chapter in the course.

Michael Dolinsky presents a paper "A New Generation Distance Learning System for Programming and Olympiads in Informatics". He proposes a new system, which supports teaching and learning of an arbitrary programming language, provides personal approach, and guarantees quality of knowledge and skills of graduates.

Antti Laaksonen discusses competitive programming in teaching university algorithm courses, and Dennis Komm presents an introduction to runtime analysis for an SOI workshop.

Some of the other papers in this volume deal with improving teaching and learning computer programming at secondary level, teaching graphs for students at lower secondary level, constructing problems in the sense of the IOI in Discrete Mathematics and Theoretical Informatics, and training world-class professionals in ICT.

We also understand the need for continuing to share our national experiences – our problems are common problems. In the second part of the volume, Bulgaria, Japan and Kazakstan present thoughtful analyses on the situation in their countries concerning informatics olympiads. IOI community members Bakhyt Matkarimov, Gregg Lee, Margot Phillips and Eljakim Schrijvers review IOI host guidelines and general aspects, which we hope will be of interest to our readership.

Finally, Antti Laaksonen presents a new book on Competitive Programming, and Krassimir Manev and Biserka Yovcheva announce the first European Junior Olympiad in Informatics.

As always, thanks are due to all those who have contributed to the current volume and the IOI conference. A lot of work is required, not only to the writing of the papers, but also to an extended period of reviewing and revising. Peer reviewing of the papers takes a significant amount of time and work, and special thanks should be given to the reviewers. Many thanks to the reviewers Ingrianni Liem, Jari Koivisto, Rein Prank, and Noa Ragonis who wrote several reports and improved quality of papers.

In particular, we would like to thank the organisational committee for IOI'2017 in Tehran and the Iranian organisation of this year's IOI for giving us the opportunity to host the conference.

Editors

Olympiads in Informatics, 2017, Vol. 11, 3–18
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.01

# Constructive Problems in the Structure of the Olympiad in Discrete Mathematics and Theoretical Informatics

Vasiliy A. AKIMUSHKIN[1], Sergei N. POZDNIAKOV[2], Anton S. CHUKHNOV[3]

[1]*Informational and methodological center of computer technologies and informatics faculty of SPbETU "LETI"*
*197376, Saint-Petersburg, prof. Popovs str., 5*
*tel. (812) 234-63-81, 26-96-493*
[2]*Department of Mathematics, Saint Petersburg Electrotechnical University*
*Mathematics and Mechanics Faculty, St. Petersburg State University*
*192281, Saint-Petersburg, Kupchinskaya str., 10-3-223*
*tel. (812) 234-63-81, (911) 76-98-492*
[3]*Department of Mathematics, Saint Petersburg Electrotechnical University "LETI"*
*194292, Saint-Petersburg, Rudneva str., 31/29 - 45*
*tel. (812) 234-63-81, (921) 976-96-87*
*e-mail: vasiliy.akimushkin@gmail.com, pozdnkov@gmail.com, septembreange@gmail.com*

**Abstract.** In the paper problems which organizers of Olympiads are faced considered. The approach to solve this problems suggested. This approach is based on activity theory and includes using rather simple constructive problems as a first step to more complicated theoretical ones. The experience of implementing this approach within the framework of the Olympiad in Discrete Mathematics and Theoretical Informatics is described. The focus is set on computer manipulators – interactive dynamic models of mathematical and informatical object.

**Keywords:** Olympiads in informatics, discrete mathematics, computer science education, constructive problems, information technologies in education.

## 1. Introduction

One of the challenges of mass participation of schoolchildren in the Olympiads in mathematics and computer science is the complexity of the Olympiad problems. Most schoolchildren are not initially motivated to solve complex problems, so the number of participants in the Olympiads is rather small. At the same time, reducing the level of complexity of the Olympiad problems would undermine the very idea of the Olympiad movement, within the framework of which the student is offered difficult tasks that require a non-standard approach and a deep understanding of the problem. At first glance it seems that the contradiction between the difficulty of tasks and the mass participation of schoolchildren is insurmountable.

The approach, suggested in this article, showed the possibility of increasing the number of participants of the Olympiad by using of constructive tasks. Those tasks are tied to computer dynamic models – manipulators dealing with objects of the problem. It allows to bring an experimental component into the solution of theoretical problems, to "touch" the idea of a solution "by hands", to transform it into a constructive form and only after that to make theoretical generalizations. Built on computer manipulators, representing opportunities for experiments with important theoretical concepts, such problems play the role of a bridge to more complex theoretical problems (Akimushkin and Pozdniakov, 2015).

Since the Olympiads are usually held in several stages (for example, training, qualifying and main stages), it is proposed to consider all stages as components of a single process of leading schoolchildren to difficult theoretical problems; to ensure continuity of the Olympiad stages, in order to introduce new theoretical concepts into the subject of the Olympiad; to begin with simple constructive tasks on the training tour and gradually complicate them, adding theoretical questions to the qualifying round and complete with serious theoretical tasks in the final round.

In this way it is possible to solve one more problem – the problem of introducing new ideas into existing mathematics and computer science courses. At the Olympiad, the topic of the tasks may be wider than the one presented in the school curriculum, but new ideas should be introduced the way that they can be understood by schoolchildren. Three-step introduction of new concepts through the use of constructive tasks on computer manipulators allows us to solve this problem.

## 2. Background and Literature Review

The approach, suggested in the article, based on the activity theory. Psychological works of L.S. Vygotsky about the role of a tool in the environment of a child state and justify that men and women obtain the control of their own intelligence only through the control of the real world objects. In what follows the names of these objects in a language used by men and women become the signs of these objects, and thus the language is inseparable from their thinking (Vygotsky, 1930);

The works of Seymour Papert actually used Vygotsky ideas in the conditions of the appearance of a computer and "smart objects" in a child environment. They are introduced into a child environment and through the interaction with them it forms his representation of important scientific ideas and analyzes its own thinking (Papert, 1980);

For the last 10–15 years, there has been rapidly growing the movement of mass competetions, which tend to occupy an intermediate place between the school course of computer science and mathematics and the Olympiad movement (van der Vegt, 2016; Kostadinov *et al.*, 2015; Sysło and Kwiatkowska, 2015). The most famous contest is the BEBRAS competition, which brought together the methodological ideas of scientists and practitioners from more than 60 countries in overcoming this challenge (Dagienė and Sentance, 2016). Starting from multiple choice test problems, which connects the form of the contest with various systems of knowledge assessment, over time the competition absorbed the ideas of other researchers and the range of possible types of answers in tasks was expanded by so-called dynamic tasks.

Dutch Olympiad of Informatics can be an example of using constructive and theoretical non-programming tasks in the framework of informatics Olympiad with positive effect (van der Vegt, 2012; van der Vegt, 2016).

Another example of an informatics competition using theoretical problems is the Open School Olympiad "Information Technologies" held by IFMO University in Saint-Petersburg.

The team represented by the authors of the article has also been working on the idea of introducing electronic tools into the process of solving problems in mathematics and computer science for more than 15 years, and during this time supports the CTE contest ("Konstruiruy, Issleduy, Optimiziruy" which means CTE "Construct, Test, Explore"), based on the idea of using computer models of meaningful ideas from mathematics, physics and informatics for cultivation of interest in science through experimental and constructive activity (Ivanov, *et al.*, 2004; Posov and Maytarattanakhon, 2014).

## 3. Olympiad Structure

Olympiad of the Olympiad in Discrete Mathematics and Theoretical Informatics consists of three stages: training, qualifying and final.

Training round serves to let participants get acquainted with the framework, with computer manipulators and with basic concepts of discrete mathematics and theoretical informatics used in the tasks. It is held on the internet, the participants are able to perform the tasks at home. Tasks of the training round are rather simple and score gained by the participants has no consequences for them. Tasks are checked automatically.

Qualifying round is open and is also held on the internet. Tasks are harder then in the training round and require some thinking. Most of the tasks use computer manipulators, they are checked automatically as in the training round. A few tasks do not require manipulators, but text solution. They are checked manually.

Russian participants of the BEBRAS competition which takes place just before the start of the training tour of the DM&TI Olympiad are invited to the Olympiad,. Through this, the number of regions of Russia in which the olympics is held is rather big. For example in the season 2016–2017 among the participants of the qualifying round were representatives of 60 regions of Russia (out of a total of 85).

The final round is organized on the sites of universities of those cities, which are convenient for participants who have taken part in this round. In 2016–2017, 18 sites were organized for the final round.

The final round takes place on the server of the Olympiad in the same environment as the training and selection rounds. Thus, all participants are already familiar with the shell, and the local co-organizers need only identify the attendees and control the independence of their work. The logging of works is carried out centrally.

Tasks of the round belong to both manipulator and theoretical types. As the number of participants is lower than in qualifying round so there is a room to increase number of tasks of the second type.

After a qualifying as well as final round the surveys were held. Questions and resuls of both surveys are presented in the «Results analysis» section.

In the table below we can see a number of participants of qualifying and final rounds as well as number of participants completed the surveys.

| Round | Participants | Surveys completed | % |
|---|---|---|---|
| Qualifying | 513 | 127 | 25% |
| Final | 97 | 35 | 36% |

## 4. Computer Manipulators as a Basis for Constructive Tasks and Experiments

We call computer manipulator a dynamic interactive model of object of greatest importance for discrete mathematics and theoretical computer science. Each manipulator gives participants enough freedom in changing parameters of a task. Thus we allow participants to get closer to objects of discrete mathematics and theoretical informatics, to make them master concepts and methods not only through reading their formal definitions, but as well through their own practical experience, throgh «feeling» and «touching» properties of those objects. This process, which could be called interiorization, allows theoretical ideas to find their way into the student's minds.

In the 2016–2017 Olympiad the following manipulators were used:

- **Graphs** – a manipulator based on the ability to build a graph, introducing the necessary notation, edge weights, coloring vertices, etc. The manipulator may be adapted to the certain task using only those instrumental capabilities that are required to complete this task to make the interface intuitively understandable and not overloaded with unnecessary operations. Also, the manipulator is equipped with a set of procedures for verifying the properties of the solution that make up the condition of the task (for example, checking the planarity or regularity of the graph being built) that the participant of the Olympiad can use for experiments in the process of solving constructive problems or investigating particular cases of the theoretical problem (Akimushkin *et al.*, 2015). These properties belongs to all the manipulators used in the Olympiad.
- **Finite state machines** – the manipulator showing the graphical representation of the finite state machine. It can be used in different contexts, for example, as a deterministic machine, as an machine with an output alphabet or a recognizing machine. Just as in the "Graphs" manipulator, the machine used in the certain task has already been adapted to the task, and the user works within the context of the task restricted by built-in constraints of the manipulator.

- **Regular expressions** – the manipulator is another form of representation of content, which, according to Kleene's theorem, can be formulated in terms of finite state machines. However, the presence of several representations of the same concept is a way of supporting the process of assimilation of this concept (Bogdanov *et al.*, 2008). Despite the fact that the solutions of constructive problems on this manipulator, as well as on the Finite State Machines manipulator, can be verified algorithmically, in both cases the estimation of particular solutions with respect to a parameter is used-the percentage of properly analyzed or generated chains.
- **The Turing Machine** – this well-known manipulator allows you to introduce algorithmic elements inherent in programming Olympiads into the Olympiad in theoretical informatics. At the same time, since the Turing machine is a theoretical tool for investigating the algorithms, there appears the natural possibility to raise questions of the existence or laboriousness of algorithms with given properties.
- **Logical schemes.** While the above manipulators are very slightly connected with the ideas of the school curriculum, this manipulator is very close to the elements of logics included in the course of informatics in Russia (proposition logic). Represented by logical schemes, Boolean functions (logical expressions) are more demonstrative and easier to set up constructive tasks with. The user can easily test the assembled circuit on all binary sets (also the percentage of sets on which the circuit works correctly, is used to evaluate partial solutions). In addition, schemes allow to offer tasks that go beyond the elements of mathematical logic studied in school and are related to real problems of constructing logical elements that can have several outputs and some parts of the circuit can use to calculate signals at several outputs (that is, the task is not reduced to the calculation of the values of several independent Boolean functions).
- **Tarski world –** this is the most complex manipulator that allows students to enter the field of first-order logic or the logic of predicates. The manipulator is based on the idea first implementatied in the well-known program with the same name (Barker-Plummer, *et al.*, 2008) and was used earlier in the CTE competition. In this manipulator, in an informal (verbal) form, you can use both logical operators and quantifiers. As a basic subject set, a checkered field with the figures placed on it is used. Figures have properties such as color, shape, size (single predicates) and a certain location relative to each other, for example, higher, lefter, side by side (two-place predicates).

## 5. DM&TI Olympiad Task Analysis

As already mentioned above, tasks are formulated in a way to create cross-cutting thematic lines through all stages of the Olympiad, except that in each round the task authors try to create internal links between tasks so that participants in a short time of the Olympiads do not scatter attention to different subjects, but, on the contrary, to see one story from different sides, in different interpretations and metaphors. Therefore, the problems of one stage, usually about 8, are generally divided into 3–4 groups of problems so that the tasks of each group have some unity.

Let us give an example of several such interrelated tasks from the qualifying round of the Olympiad in Discrete Mathematics and Theoretical Informatics of the 2016–2017 season (DM&TI-2017).

Let's consider a constructive task on regular expressions (the original task number is given) and the manipulator image with which the participant works.

***№2 (3 points)***

*Construct a regular expression describing the set of words from the letters **a** and **b**, from which all words specified by the regular expression **(ab)*** are removed. Try to give the expression as short as possible.*

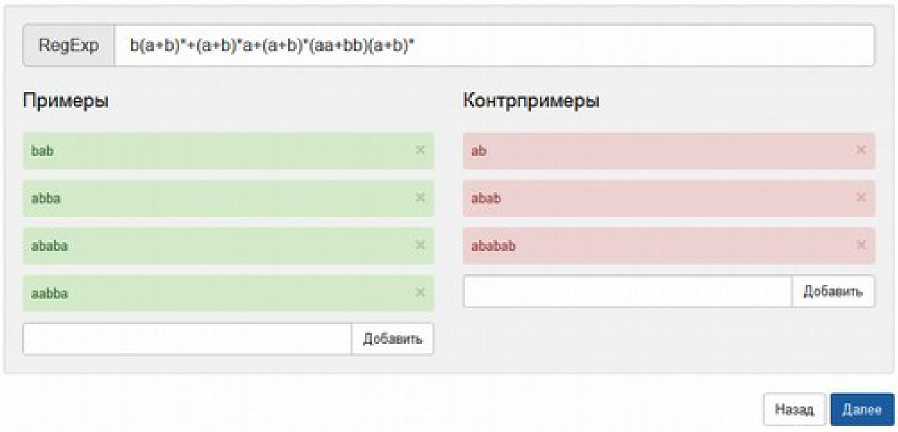
Help (can be called by the participant).

Regular expressions contain three operations: splicing strings (multiplication), selecting one of the two options (addition) and an iteration, denoted by an asterisk. The initial solution is **b*(a + b)**. It consists of two parts – **b*** denotes an arbitrary number of letters **b** (possibly none), **(a + b)** – one of the letters **a** or **b**. Below through the color highlighting you can see which words do satisfy this expression, and which do not.

***Solution***

If the word does not satisfy the regular expression **(ab)***, i. e. does not have the form **abab ... ab**, it means that either this word starts with **b**, or ends with **a**, or contains two identical letters in a row. The first term in the formula corresponds to the first case, the second term to the second case, the third one to the third.

The answer is shown on the Fig. 1:



RegExp: b(a+b)*+(a+b)*a+(a+b)*(aa+bb)(a+b)*

| Примеры | Контрпримеры |
|---|---|
| bab | ab |
| abba | abab |
| ababa | ababab |
| aabba | |

Fig. 1. User interface for working with constructive tasks on regular expressions. A. participant can extend a number of examples and counterexamples to check the input of the response (the top line shows the answer of the problem).

Immediately after this task is the combinatorical problem, based on the same interpretation – representing certain set of words in the form of regular expression.

***№3 (4 points)***

*How many different words does the regular expression* ***(a + ab) (b + ab) (a + ab) (b + ab) (a + ab)****? Do not forget to explain your answer.*

***Solution***

Each expression in braces can have one of two possible values, which means that there are 32 variants in all. However, some words are thus constructed two times. Only two words are given in two ways, the others by one. Thus, the answer for this problem is 32 – 2 = 30.

The next problem does not differ from Problem 2 in essence, but is proposed in a different interpretation – the interpretation of finite state machine. Obviously, the participant will not fail to notice the same condition in these tasks. Thus, he will be indirectly get acquainted with the concepts necessary for the formulation of Kleene's theorem on the equivalence of regular expressions and automaton languages.

***№4 (4 points)***

*Below there is a finite state machine recognizing all words from the letters* ***a*** *and* ***b*** *that match the regular expression* ***(ab)****. *Rework it to make it recognize all the words in the same alphabet, except these. Try to keep the machine as small as possible.*

***Solution:***

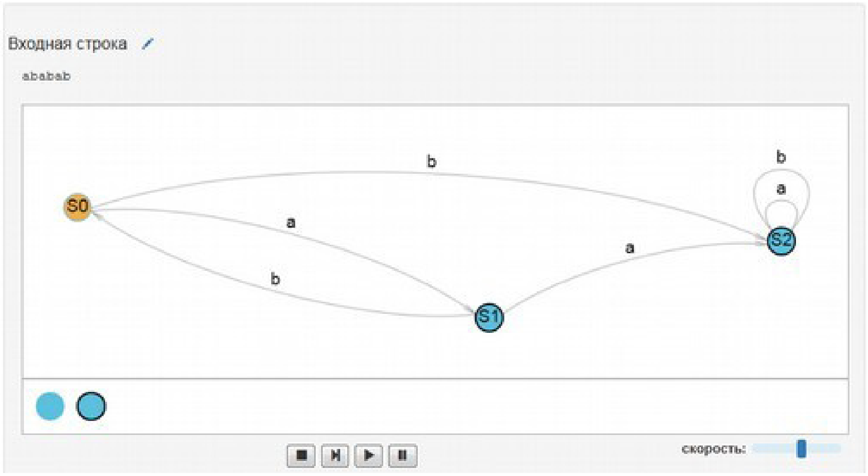


Fig. 2. User interface for working with finite state machines constructive tasks (the figure shows the solution of the task).

Now let us consider the tasks of the final round. In the 2016–2017 season, some of the tasks were gathered with the concept of the "voting machine" (note that this task was used in the popular interpretation in the CTE competition). Here are the tasks texts and some solutions.

The first series of problems is based on the manipulator "Logical schemes" and represents an alternation of constructive tasks and theoretical problems.

In the core of all tasks there lies a fact that goes beyond the school curriculum. From the school curriculum schoolchildren can learn the fact implicitly presented in it about the completeness of the set {negation, disjunction, conjunction} (or even with the exception of a disjunction or a conjunction according to Morgan's law). However, in these problems we consider a self-dual, monotonic Boolean function that preserves 0 and 1, which must be expressed in terms of monotone functions preserving 0 and 1. The last two problems represent a significant complication of the original problem, since in this case the function through which the function given in addition to the listed properties also possesses self-duality.

***1. Logical schemes: "Voting machine"***

*Let's denote a **voting machine** for an odd number **n** a logic scheme with **n** inputs that return «true» when more than half of the inputs have «true» values, and returns «false» value otherwise.*

*This is a very natural definition: if **n** inputs are **n** people, each of which votes either «for» (true) or «against» (false), on the output we get the option for which the majority has voted.*

***1.1. (3 points)***

*Construct a voting machine for three people using the AND and OR logical elements.*

***Solution:***

Possible solutions are shown below:

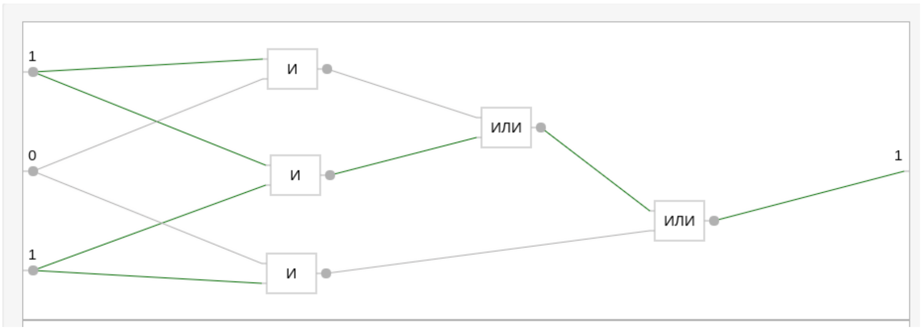


Fig. 3. User interface for working with constructive tasks on logic circuits. One solution for the task.

Also on the diagram you can see how this logical scheme works on a certain set of values (TRUE, FALSE, TRUE).

At the Fig. 3 we sort all possible pairs of inputs, and if at least one of them takes a true value, the logical scheme produces the true result.

***1.2. (3 points)***

*Prove that it is possible to construct an automaton for voting of any odd number of elements from the AND and OR elements.*

***1.3. (4 points)***

*Prove that $2^n$ elements AND and OR are sufficient to construct a voting machine for odd number n people.*

***1.4. (4 points)***

*There is a logical element with three inputs, which gives output «1» if the number of «1» at the inputs is greater than the number of «0» (implements the voting function for three people).*

*Construct a voting machine for five people, using only voting elements for three people.*

***Solution:***

We enumerate the voters using numbers from 1 to 5. Let's take the voting elements of three people with numbers 1, 2 and 3; 1, 2 and 4; 1, 2 and 5. The outputs of each of these elements we submit to the inputs of the fourth voting element for three people. Notice that the scheme obtained gives the correct result for voting of five people for all situations except two: when people with numbers 1 and 2 vote «for», and the other three «against», and the inverse situation.

We will create two more similar schemes: a scheme that is «mistaken» only in the distribution of the voters 2 and 3 against 1, 4 and 5, and a scheme that «mistakes» in the distribution of 4 and 5 against 1, 2 and 3. Note that at least two of these schemes will give the correct result for any distribution of votes. So, if we submit their outputs to the inputs of the final voting element for three people, the result will always be correct.

Let us pay attention to the solution of the previous problem (the corresponding scheme can be easily constructed and is not given in the article). This constructive problem leads us to the solution to the next, theoretical problem. This allows us to make an assertion that has long time been accepted by mathematicians and is confirmed by psychologists that theoretical knowledge is based on very certain tasks, solved by the person himself.

***1.5. (5 points)***

*There is a logical element with three inputs, which gives output «1» if the number of «1» at the inputs is greater than the number of «0» (implements the voting function for three people).*

*Prove that it is possible to build a voting machine for any odd number of people, using only voting elements for three people.*

***Solution:***

We prove this statement using mathematical induction. The basis n = 3 is contained in the problem text.

To make induction step form 2n −1 to 2n + 1 we use the inductional hypothesis. Regard the following voting elements for 2n – 1 people: all but 1st and 2nd; all but 2nd and 3rd; all but 1st and 3rd. Their outputs submit to the inputs of the element for three people.

The scheme obtained will give us correct results in almost all cases. The only exception is the situation when majority contains exactly n+1 people including 1st, 2nd and 3rd.

Creating another two schemes which are «mistaken» in two another situations. Then when we submit outputs of all 3 schemes to the input of the final three people element, the result provided by this element will be always correct.

In tasks of other manipulators, the idea of modeling the voting scheme was also used. So, for problems on regular expressions and finite state machines, it was suggested to describe the structure of input data, in which voting ends with a positive result.

***3.1. (3 points)***

*There are N people standing in the queue for voting. It is known that next to each person (directly ahead of him in line or behind) is a person who votes «for». Prove that the number of people in the queue, who vote «for» at least half of the total amount of people.*

***3.2. (3 points)***

*We assign to each person in the queue «1» or «0», depending on whether he votes «for» or «against». It is known that next to each person (directly ahead of him in line or behind) is a person who votes «for». Construct a regular expression that describes all such sets of 0 and 1 or prove that this is impossible.*

***4.1. (3 points)***

*We assign to each person in the queue «1» or «0», depending on whether he votes «for» or «against». It is known that next to each person (directly ahead of him in line or behind) is a person who votes «for».. Construct a finite state machine that recognizes all such sets of «0» and «1» or prove that this is impossible.*

## 6. Results Analysis

During the Olympiad we were evaluating the hypothesis: solving simple constructive tasks may be an important step to more complicated theoretical ones; theoretical tasks become avainle to be solved by more participants due to existance of constructive ones;

constructive problems are helpful participants not only in solving specific tasks but in deeper understanding of the whole subject as well.

Of course the optimal way to make an experiment is to check the theoretical tasks with and without constructive ones on different groups; unfortunately, it is completely impossible within the same competition. So the main thing we should analyze as a results of the experiment is participants feedback: whether constructive problems were crucial for them or not.

Constructive tasks are being solved by much more participants than theoretical tasks, at the same time, participants who have passed into the final round generally solve the theoretical tasks of the qualifying round.

## 6.1. *Qualifying Round Results Analysis*

The survey of participants of the qualifying round (127 people fulfilled the survey, which is about 20% of all participants in the qualifying round) shows that participants liked the tasks (96% of participants liked the participation in the Olympics of the DM&TI). This answer can be interpreted as the fact that the tasks did not cause rejection reactions, were understandable and "accepted" by the participants.

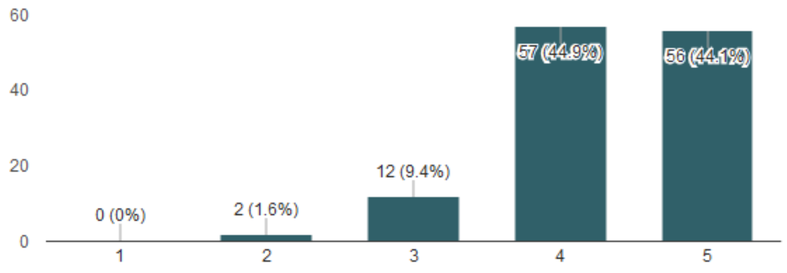


Fig. 4. Results of a survey of participants on how much they liked the tasks (on a five-point scale from 1 to 5).

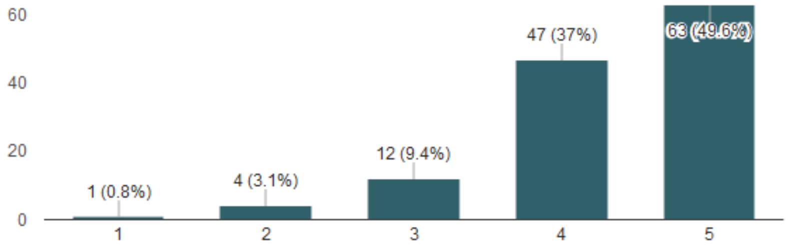


Fig. 5. Results of a survey of participants on how much they liked working with dynamic modules (on a five-point scale from 1 to 5).

It should be noted that the distribution of answers to the question, whether participants liked working with dynamic manipulator models, has a similar structure.

Some statements of the participants of the qualifying round:

> *«Most of all I liked interactive tasks, such as graphs, logic schemes, etc. I liked them because I could interact with the task «visually», without text and commands. Visual contact is very important.»*

> *«You could put the dynamic modules in permanent access, both for training and just as a useful application.»*

> *«It's quite interesting and unusual that you can use more opportunities of the online Olympiad (computer manipulators). The content of the Olympiad itself is also impressive, there were no special questions on the tasks or places where it takes more time to think about the problem than to solve it.»*

These data show that the problem of attracting more participants to participate in the qualifying round is solved by the use of dynamic modules and constructive tasks built on them.

## 6.2. *Final Round Results Analysis*

In the final round of the 2016–2017 season people participated (of 104 passed). 13 tasks, divided into 6 groups by reference to various dynamic manipulators were offered to participants. Initially, the number of tasks was assumed redundant, since for 3 hours of a final round it is really possible to solve 6–7 tasks. The total amount of score of all tasks

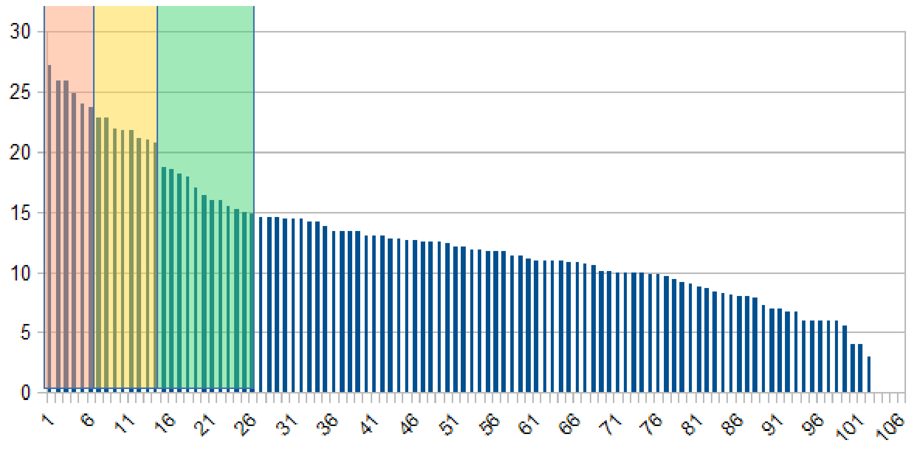


Fig 6. Score distribution of the final round. Diploma of different degrees areas are coloured: rose – I degree; yellow – II degree; green – III degree.

was 51 points. Half of these points is the expected best result. After the final round, the following criteria were determined: more than 23.5 – I degree diploma, 20 to 23.5 points – II degree diploma, 15 to 20 points – III degree diploma. The median score obtained is 25% of the maximum.

35 of 97 participants of the final round took part in the survey the targeted to determine the role of constructive problems in solving theoretical ones and the influence of identical manipulators and theoretical ideas attached to them throughout all rounds of the Olympiad over preparation for the final round. Results of the survey are presented in the figure 7.

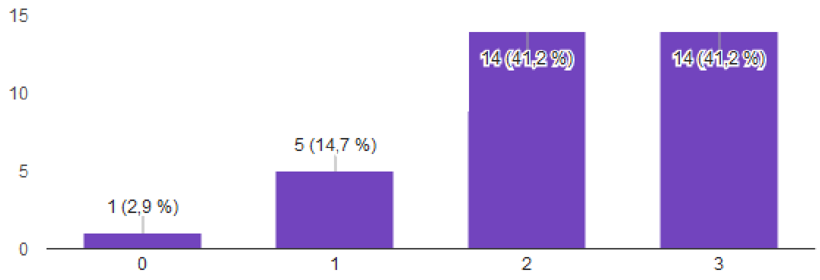


Fig. 7. Results of a survey for the participants of a final round about the importance of constructive tasks for understanding theoretical problems associated with them (on a four-point scale: 0 – do not influence, 3 – are necessary for solving theoretical problems).

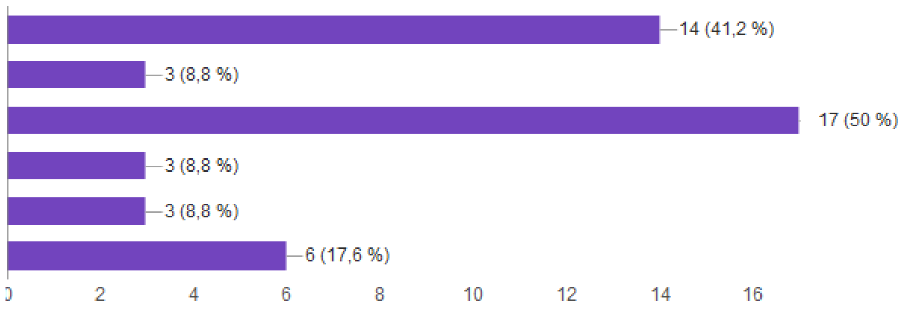


Fig. 8. Results of the survey of the final round participants about how difficult the problems of the Olympiad revealed to be:

line 1 – «these problems are difficult, because they are not in the school curriculum»
line 2 – «tasks are more difficult than in other Olympiad»
line 3 – «thanks to training and qualifying rounds I got acquainted with the new subjects of the problems and began to solve them successfully»
line 4 – «I am familiar with this kind of tasks through additional activity in the school»
line 5 – «I participated in this Olympiad more than once and got used to the specific tasks of the Olympiad»
line 6 – «other».

Statistical analysis shows that the confidence interval for the average estimation of the importance of constructive problems for the successful solution of the theoretical is (2.01, 2.39). With a probability of 0.95, it can be argued that the average value when sampling a larger volume will not go beyond the interval found. This means confirmation of the assumption that the constructive tasks that precede theoretical problems play an essential role for their successful solution.

Figure 8 shows the results of a survey about the difficulty of the tasks. The third line of the chart is highlighted most clearly in which the participants report on the role of the training and qualifying round for the preparation for a final round: «Thanks to the training and selection round, I got acquainted with the new subjects of the problems and began to solve them successfully».

The estimayed average per cent of participants, for whom the training and qualifying rounds became important elements of preparation for the successful completion of the final one, ranges from 43% to 57%. With a probability of 0.95, it can be argued that the average percent when sampling a larger volume will not go beyond the interval found.

## 7. Conclusions

Based on five years of experience in organizing the Olympiad in Discrete Mathematics and Theoretical Informatics and in analyzing the results of the Olympiads and questionnaires, the following conclusions can be drawn:

1. Using constructive tasks built on computer manipulator models allows to increase in the number of participants in the training and qualifying rounds of the Olympiad.
2. The absence of important concepts used in Olympiad problems in the school curriculum in mathematics and computer science can be compensated by considering three rounds of the Olympiad as a single process of immersing the participants of the Olympiad in new subject areas. The latter can be considered as a pre-process for introducing new elements into the school curriculum.
3. Using a series of tasks, the first ones of which have a constructive form and allow you to experiment with solutions of problems in a computer-modeled domain, allow participants to successfully solve theoretical problems, removing the barrier of fear of difficult tasks.

## References

Akimushkin, V.A., Maytarttanakhon, A., Pozdnyakov, S.N. (2015). Olympiad in theoretical computer science and discrete mathematics. In: Brodnik, A., Vahrenhold, J. (Eds.) *Informatics in Schools: Curricula, Competences, and Competitions. 8th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives: ISSEP 2015, Ljubljana, Slovenia, September 28 – October 1, 2015, Proceedings*. Springer, LNCS 9378, 94–105.

Akimushkin, V.A., Pozdniakov, S. (2015). Computer enhanced Olympiad in theoretical informatics and discrete mathematics In: Vassiliev N.N. (Ed.), *International Conference Polynomial Computer Algebra 2015. St.-Petersburg, April 13–18*. Euler International Mathematical Institute, VVM. Publishing, 80–83.

Barker-Plummer, D., Barwise, J., Etchemendy, J. (2008). *Tarski's World*. Stanford, Calif: CSLI Publications.

Bogdanov, M., Pozdnyakov, S., Pukhov, A. (2008). Multiplicity of the knowledge representation forms as a base of using a computer for the studying of the discrete mathematics. In: *The 9th International Conference "Teaching Mathematics: Retrospective and Perspectives"*. Vilnius Pedagogical University, 16–17 May 2008.

Dagienė, V., Sentance, S. (2016, October). It's computational thinking! Bebras tasks in the curriculum. In: *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer International Publishing, 28–39.

Ivanov, S., Mamaeva, S., Pozdnyakov, S., Stepulenok, D., Entina, S. (2004). Computer support of educational research on mathematics. *Computer Tools in Education*, 2, 5–18. (In Russian. Иванов, С., Мамаева, С., Поздняков, С., Степулёнок, Д., Энтина, С. Компьютерная поддержка дистанционного учебного исследования по математике. *Компьютерные инструменты в образовании*, 2, 5–18). `http://window.edu.ru/resource/340/24340/files/2004_2_05.pdf`

Kostadinov, B., Jovanov, M., Stankov, E., Mihova M., Risteska Stojkoska, B. (2015). Different approaches for making the initial selection of talented students in programming competitions, *Olympiads in informatics*, 9, 113–125. DOI: `http://dx.doi.org/10.15388/ioi.2015.09`

Papert, S. (1980). Mindstorms: Children, Computers, and Powerful Ideas.

Posov, I., Maytarattanakhon, A. (2014). Automation of distance contests based on research problems in mathematics and informatics. *Computer Tools in Education*, 6, 45–51. (In Russian. Посов, И.А., Майтараттанакон, А. Автоматизация проведения дистанционных соревнований, основанных на исследовательских сюжетах по математике и информатике. *Компьютерные инструменты в образовании,* 6, 45–51).

Sysło, M.M., Kwiatkowska, A.B. (2015). Introducing a new computer science curriculum for all school levels in Poland. *LNCS*, 9378, 141–154.

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

van der Vegt, W. (2016). Bridging the gap between Bebras and Olympiad; experiences from the Netherlands. *Olympiads in Informatics*, 10. 223–230.

Vygotsky, L., Luria, A. (1930). Tool and symbol in child development. In: Jaan Valsiner and Rene van der Veer (eds.), *The Vygotsky Reader*.

**V. Akimushkin.** Graduated from Saint-Petersburg State Electrotechnical University "LETI", Computer Science Faculty  and then from Saint-Petersburg State Universtity. Working at Informational and methodological center of computer technologies and informatics faculty of SPbETU "LETI" as a leading programmer.

**S. Pozdniakov.** Graduated from Leningrad university, Faculty of Mathematics and Mechanics. Doctor of sciences in theory and methodology of mathematics and computer science 1999 (Moscow). Doctor thesis title: “The modelling of information environment as technological base for teaching mathematics”. Working at Saint-Petersburg State Electrotechnical University “LETI” from 1986 and at Saint-Petersburg State Universtity from 2007. Teaching courses for university students: “Discrete Mathematics”, “Mathematical Logic”, “Theory of Algorithms”, “Construction of Computer Programs for Education”, “Information Technologies in Mathematics”, “Models of Intellectual Processes”. Teaching courses for school teachers: “Methodology of Using Computer for Supporting of Math Education”.

**A. Chukhnov.** Graduated from Saint-Petersburg State Universtity, Faculty of Mathematics and Mechanics at 2004. Working at Saint-Petersburg State Electrotechnical University “LETI” since 2005, specialized in Discrete Mathematics and Mathematical Logics. Also since the same time working at school №239 with talented schoolchildren who are interested in additional education.

Olympiads in Informatics, 2017, Vol. 11, 19–28
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.02

# Learning and Teaching Algorithm Design and Optimisation Using Contests Tasks

Sébastien COMBÉFIS, Saïkou Ahmadou BARRY, Martin CRAPPE,
Mathieu DAVID, Guillaume de MOFFARTS,
Hadrien HACHEZ, Julien KESSELS
*Electronics and IT Unit, École Centrale des Arts et Métiers (ECAM)*
*Promenade de l'Alma 50, 1200 Woluwé-Saint-Lambert, Belgium*
*e-mail: s.combefis@ecam.be, saikouah@gmail.com, martin.crappe@gmail.com,*
*mathieudavid@mathieudavid.org, guillaumedemoff@gmail.com,*
*hadrienhachez@hotmail.com, julien.kessels@gmail.com*

**Abstract.** It is important for a future computer science engineer or scientist to master algorithm design and to know how to optimise algorithms to solve real-world problems. Most programming and IT contests require their contestants to design algorithms to solve problems and to optimise their code to get the best temporal and spatial performances. This paper presents training materials built from contest tasks to teach and learn how to design algorithms that solve concrete and contextualised problems. The first learning modules will be built thanks to a pedagogical device that will be deployed during the 2017–2018 academic year at ECAM in the frame of the LADO project. All the produced materials will be open sourced and available in English.

**Keywords:** algorithm and optimisation; pedagogical device; learning modules.

## 1. Introduction

Computer science engineers and scientists must have skills in algorithm design and optimisation. While scientists may be more interested in a theoretical understanding of algorithms, their complexities and how to optimise them, engineers need to know how and which algorithm to apply for a given real-world problem.

One way to learn how to write optimised algorithms is to make experiments. Given an implementation of the algorithm, it can be executed on several instances of a problem to measure temporal and spatial performances. Different implementations can therefore be compared, to identify the best ones. Basic algorithms and related data structures can be easily learned with textbooks or other similar non-interactive resources. What is more difficult to learn is how to chose an algorithm and how to implement it to efficiently solve a given problem. It is even more difficult when dealing with real-world problems for which new algorithms have to be devised and optimised to compute a solution within a reasonable amount of time.

Programming contests and some other more general IT contests typically ask their contestants to solve one or several challenging tasks. In addition to finding a relevant algorithm, contestants must optimise their implementation to obtain the best running time and memory consumption to get a good position in the contest ranking. Contest tasks therefore provide good case studies to learn algorithm design and optimisation. But such tasks are not designed and suited for learning, as is. They are generally devised to be difficult to solve and so that a solution to solve them is not immediate.

This paper presents how learning materials can be built from contest tasks to teach algorithm design and optimisation. Various tasks from programming and more general IT contests are considered. The paper also presents a pedagogical device that will make students building the first learning modules. It will be settled the next academic year and all the produced materials will be open source.

The remainder of this paper is structured as follows. Section 2 briefly browses related works. Section 3 presents how tasks from contests are used to build materials suited for teaching and learning algorithm design and optimisation. Section 4 discusses the LADO project that will implement the proposed pedagogical device at ECAM during the 2017–2018 academic year. Finally, the last section concludes the paper with perspectives and future work.

## 2. Related Work

Online programming and IT contests can be used to build trainings to teach and learn programming skills as detailed in previous works. (Combéfis *et al.*, 2014) highlights the necessity to provide additional information to the contests tasks, to make them suited for teaching purposes. In particular, offering a feedback to the learners is very important to support their learning, as detailed in (Combéfis *et al.*, 2012). Finally, as highlighted in (Combéfis *et al.*, 2013), learning algorithm design is important for young people but also for students in higher education institutions, as future professionals. This paper builds on these previous works to propose concrete learning materials.

Some other works uses contests to teach and learn programming and other IT related skills. In (Garcia-Mateos, *et al.*, 2009), the authors explain how to make courses more entertaining to increase the motivation of students by using programming contests. The main difference with the work of this paper is that they are developing their own contest-style tasks fitted for education purposes. Evaluation of students' submissions is made by Mooshak, an on-line judging system used to manage programming competitions (Leal, *et al.*, 2003). As indicated by the authors in the conclusion, having a better feedback in case of a "wrong answer" has still to be done. Competitive programming should be introduced early as highlighted by several authors (Leal, *et al.*, 2008; Ribeiro, *et al.*, 2008) as it helps to get better skills in algorithm design and optimisation.

Finally, (Booth, 2001) discusses the need and importance of context while learning computer science and engineering. The author draws up theoretical foundations and considerations about the move to highly constructivist pedagogies in higher education, which is in line with the method proposed in this paper.

## 3. Teaching and Learning with Contests Tasks

This paper proposes learning materials based on tasks from programming and IT contests. It also presents a pedagogical device that will result in the production of these learning materials by a selected group of students. The developed learning modules will be about how to design and optimise algorithms. Contests tasks are used as illustrating and motivating real-world problems and also as case studies to assess the algorithms choice to solve the given problem and to test and experiment with their implementations.

As previously mentioned, this paper goes one step further than (Combéfis *et al.*, 2014). It proposes a concrete way to build learning materials from contest tasks, to teach one aspect of programming, namely algorithm design and optimisation.

### 3.1. *Algorithm Design and Optimisation*

As highlighted by (Skiena, 2008), most professional programmers are not prepared enough to tackle algorithm design problems. It is therefore important that future professionals, namely current students, are taught how to correctly and efficiently implement algorithms for real-world problems as testified by a survey presented in (Lethbridge, 2000).

Algorithm designers have to deal with several processes. According to (Kant, 1985), there are six main processes when designing an algorithm. The designer starts with the understanding of the problem and then proposes a kernel idea to solve the problem. The next steps consist in executing and analysing the algorithm in order to formulate any difficulties or opportunities to improve the current solution. Finally, the last two steps is firstly a verification of the correctness, that is, assessing whether the obtained result is as good solution for the problem and secondly an evaluation of efficiency and other quality criteria.

A simplified algorithm design and optimisation process is depicted on Fig. 1. The two first steps are similar to these of (Kant, 1985), namely the understanding of the problem from which a first kernel idea emerges, typically based on a brute-force algorithm. The kernel idea should also explore the basic data structures that can be used to model the problem. All these ideas are then implemented into a program before jumping to the next two steps. The third step consists in the execution and correctness analysis of the algorithm, which can be made on small instances of the problem with a manual check, with the help of an online grader if available or with a solution check program. Finally, in the last step, the algorithm is evaluated according to its efficiency and improvement points are identified. The process then loops between the two last steps until the obtained algorithm is efficient enough.

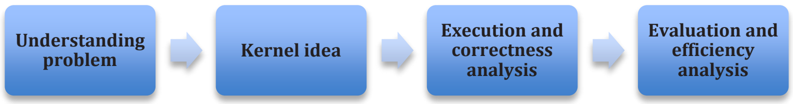

Understanding problem
Kernel idea
Execution and correctness analysis
Evaluation and efficiency analysis

Fig. 1. The four main steps of algorithm design and optimisation process.

Students must be able to implement the algorithms they are taught thanks to the learning modules by themselves, for the situation provided by the contest task, but also for other new similar situations. It means that they should have a deep understanding of these algorithms. In particular, for a given algorithm, students must be able to:

1. Understand the algorithm, its precise specifications and all the used data structures the algorithm relies on.
2. Know the class of problems for which it is relevant to apply the algorithm.
3. Implement the algorithm.
4. Compare and test different implementations, in particular in terms of time and space complexities.

These needs are related to the four mains steps of Fig. 1 and are clearly delineated in the proposed training materials. During the last two steps, students must be able to justify all their choices and argue about why they are relevant. For that, they can use complexities analysis, research papers, reference books, etc.

For example, (Levitin, 1999) describes four general design techniques (brute force, divide-and-conquer, decrease-and-conquer and transform-and-conquer) that are qualified as more general than usual classification (greedy, dynamic, backtracking, branch-and-bound, etc.) which suffers from different levels of generality and fails to classify many classical algorithms. This classification can, for example, be used to compare different algorithms and ensure that several tracks have been explored.

Concerning the optimisation that can be done at each iteration of the last two steps, they can be done at different levels. Students must be aware of these possibilities along with their advantages and disadvantages. An algorithm can be mainly optimised by:

1. Selecting appropriate data structures.
2. Choosing the appropriate parameters and configuration for the algorithm.
3. Exploiting programming language related techniques.

It is important for the students to be able to test different optimisations. The contest task can be used as a direct case study for that. Some tasks are already prepared for such experiments, while some work has to be done for others.

### 3.2. *Tasks from Contests*

As detailed in (Combéfis, *et al.*, 2014), there is a great variety of IT contests, at least for online ones. Most of them involve choosing algorithms and writing programs. Whereas these two activities are obvious for programming contests, it is not necessarily obvious for some computer security contests, innovation contests, etc. but yet possibly existing. Of course, not all the IT related contest tasks can be used to build learning materials as proposed in this paper, since they should include a possible programming activity.

The three main kinds of tasks that are considered in this work are:

- Programming tasks where the contestant has to choose an algorithm and implement it to solve a problem;

- Security tasks for which the contestant has to analyse data, a protocol, a file, etc. with scripts and hack a system, decrypt a file, etc. with selected algorithms.
- Innovation tasks in the frame of which the contestant may be asked to implement a prototype of an application.

For programming tasks, contestants are often given instances of the problem they have to solve. These instances can exhibit two kinds of differences:

- They can be of gradual complexities: small ones can be solved by hand while large ones require an optimised algorithm so that to be solved within the imposed time and memory limits.
- They can have different structures: a single algorithm may not run efficiently for all the instances and instances peculiarities must be identified to correctly chose a relevant algorithm for each instance.

These kinds of tasks are the best ones for the proposed learning modules since they naturally drive their contestants towards algorithm optimisation.

Using security and innovation tasks is less obvious but not impossible since they often involve writing code based on an algorithm that has first to be identified. Also, such tasks are often better more directly related to a real-world issue to solve and could therefore be more motivating for students.

## 3.3. *Learning Modules*

Combining algorithms to be taught and their possible optimisations with contests tasks can result in the creation of learning modules, which is the purpose of this paper. The idea is to build learning materials, which is then used in classroom with students following the "learning by doing" of (Dewey, 1938). Several kinds of learning modules, with different learning objectives, can be designed:

- They can focus on one specific algorithm and possibly on how it can be tuned and optimised.
- They can present different types of algorithms that are used to solve one given class of problems, and make it possible to compare them.
- They can present one class of algorithms suited for the given problem and compare their performance.

One given contest task can be used in several learning modules with different objectives, but it can sometimes be more suited for one specific kind. On the contrary, one learning module should be limited to only one contest task. It makes it possible to ensure that the focus is put on a single context.

The structure of a learning module is always the same, regardless of its type. The description and material of a module contain the following elements:

1. An introduction to the learning module that draws up what the learning objectives are and what will be concretely learned.

2. A brief presentation of the contest task and any other additional information that can help the learners to better understand the problem, and in particular a precise specification and description of the given inputs and expected outputs.
3. A presentation of a brute force solution, and possibly some expected solutions for small instances of the problem.
4. A detailed presentation of the algorithm/types of algorithms/class of algorithms that is the subject of the learning module.
5. A discussion about the implementation of the presented algorithms to solve the given contest task and more generally the associated problem.
6. A detailed presentation of the optimisation techniques that can be applied to improve the performances to solve the various instances of the problem.

A learning module contains both theoretical and practical activities. One module can be used to organise an activity spread on one or two weeks, for a programming course, for example. Fig. 2 shows a possible sequence to be followed, which is similar to the four main steps shown on Fig. 1. This sequence is supported by the description of the learning module just presented here above.

In the first step, the learners receive the problem statement and the associated contest task. They read it by themselves at home, and they try to understand precisely the problem and the given specifications. The second step takes place in the classroom with the professor. He or she starts by answering questions from learners about their understanding of the problem and then presents the basic algorithm(s) that can be used to solve it, and gives some tips about the possible optimisations that could be done. Then, the learners work by themselves, implementing their ideas and testing them on the contest task. They will go through several iterations between the last two steps, optimising their solution. At the end, a final exchange with the professor is organised, after he or she looked at the learners' solutions, to make a collective debriefing and feedback session.

Implementation examples contained in the learning module can be available in several programming languages. A note is associated to them, describing briefly the implementation and any language-specific features that have been used to solve the contest task. It is indeed also important to take into consideration the actual implementation since it may influence performances. Moreover, learning modules are not intended to focus on one single programming language. The material associated to a learning module should be language-agnostic, except for elements 5 and 6 mentioned above that may contain discussions related to the actual implementation with given programming languages.

To be able to use the proposed learning modules to teach algorithms design and optimisation, they have to be related in some way. Two elements can be used to classify and relate the learning modules:

- Keywords can be used to search for learning modules that are related to the same content. For example, all the modules concerning dynamic programming or all the modules with algorithms dealing with trees can be identified.
- References between learning modules can be used to build dependency trees grouping modules into learning blocks with a common theme and following given learning paths. For example, a sequence of modules about dynamic programming from simple to advanced techniques can be identified.

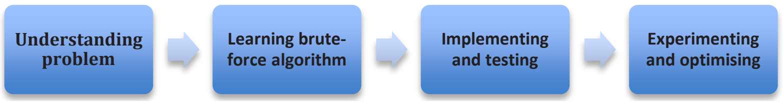


Fig. 2. A possible sequence of activities to follow in order to use proposed learning modules to teach a course.

### 3.4. *Evaluation*

Since this paper proposes materials to teach and learn, it should also take the evaluation part into account. In particular, learning modules should contain materials that can be used to assess whether the learners successfully achieved the four steps shown in Fig. 2. For that purpose, the learning materials contain several simple exercises that can be proposed to the learners to check whether they got a good understanding of the newly learned concepts. These exercises can go from simple multiple choices questions to open questions requiring writing small codes. For the latter, future work includes the possibility to provide automated evaluation and feedback through the use of the Pythia platform (Combéfis, *et al*, 2012; Combéfis*, et al.*, 2015). Also, several contest tasks could be used for the same learning modules since they are dealing with the same algorithmic techniques. They can be used to assess whether the learner is able to transfer its new knowledge to a different context.

To ease the use of the learning modules to support a course, their associated materials are split in two parts, that is, presented in two different ways. The complete description as presented in the previous section is dedicated to the professor. Learners get a partial description not showing all the example solutions. The proposed materials can of course be used in other contexts. It is, for example, possible for autodidacts to use the description intended to professors to learn the concepts by themselves.

## 4. The LADO Project

The *LADO project* (Learn and Teach Algorithm Design and Optimisation) consists in the design and construction, with a group of students, of several learning modules as proposed in this paper. It will start during the 2017–2018 academic year at ECAM with a small team composed of six students. The produced material will be in English and made open source on GitHub at the following address: https://github.com/ECAM-Brussels/lado.

This project will follow a special pedagogical device and is therefore a learning experiment for the involved students. They will indeed have to learn algorithm design and optimisation techniques to be able to create learning materials. This work will be done under the supervision of a coach-professor. Involved students will work collaboratively, which is eased thanks to the use of a Git repository.

The first steps that will be taken for the creation of a learning module, in a classroom supervised by the coach-professor, are as follows:

1. Reading and understanding the problem to solve, identifying the given inputs and the expected outputs to express the problem specifications.
2. Finding a brute-force algorithm that solves the problem, so that to have a correct reference implementation to use for comparisons.
3. Analysing the given instances to see whether some of them does exhibit any peculiarities that can be exploited so that to solve them more efficiently.
4. Optimising the problem solving by trying several algorithms or with any other optimisation techniques and by comparing with the reference implementation.

After these experiments the students should have a better understanding of the task and should have discovered new algorithms and optimisation techniques. In particular, they are also able to implement the learned algorithms and try them on the case study, namely the contest task.

The next step is related to the construction of the material for the learning module, following the template presented in the previous section. This step is done remotely and collaboratively thanks to GitHub. Involved students share the work among them, make edits on their fork of the repository and submit them with pull requests. The coach-professor then uses the review mechanism provided by GitHub to provide students with feedbacks. When everything is correct, changes are merged into the master branch by the coach-professor, making a new learning module available for the community.

Tasks that will be used come from several contests, which make their tasks openly available such as the IOI, ACM-ICPC, Google HashCode, Cyber Security Challenge Belgium, etc.

## 5. Conclusion

To conclude, this paper proposes a way to build learning materials to teach and learn algorithm design and optimisation. The learning modules are built with tasks coming from programming and IT related contests. They are used as a context for the learning module, as a motivation and also as a concrete case study with which the learner can make experiments.

The LADO project, that will start the next academic year, will result in the creation of the first learning modules implementing the propositions from this paper. In this pedagogical device, a group of six students, supervised by a coach-professor, will produce several modules that will be open source. This project is itself a learning experience since the involved students will learn as a side effect of the modules creation.

The next step of this work, after the creation of the first modules, will be their evaluation in real conditions, that is, using them to teach algorithm design and optimisation related stuff to interested learners.

## References

Booth, S. (2001). Learning computer science and engineering in context. *Computer Science Education*, 11(3), 169–188.

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., Van den Schrieck, V., Nootens, A. (2013). Growing algorithmic thinking through interactive problems to encourage learning programming. *Olympiads in Informatics*, 7, 3–13.

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

Combéfis, S., Paques, A. (2015). Pythia reloaded: an intelligent unit testing-based code grader for education. In: *Proceedings of the 1st Int'l Code Hunt Workshop on Educational Software Engineering (CHESE 2015)*. 5–8.

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

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

García-Mateos, G., Fernández-Alemán, J.L., (2009). Make learning fun with programming contests. *Transactions on Edutainment II*, LNCS Volume 5660. Springer-Verlag, 246–257.

García-Mateos, G., Fernández-Alemán, J.L. (2009). A course on algorithms and data structures using on-line judge. In: *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2009)*, 45–49.

Kant, E. (1985). Understanding and automating algorithm design. *IEEE Transactions on Software Engineering*, SE11(11), 1361–1374.

Leal, J.P., Silva, F. (2003). Mooshak: a web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6), 567–581.

Leal, J.P., Silva, F. (2008). Using Mooshak as a competitive learning tool. In: *Proceedings of the ACM-ICPC Competitive Learning Institute Symposium (CLIS 2008)*.

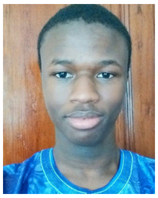
Lethbridge, T.C. (2000). What knowledge is important to a software professional? *Computer*, 33(5), 44–50.

Levitin, A. (1999). Do we teach the right algorithm design techniques? In: *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 1999)*, 179–183.

Ribeiro, P., Guerreiro, P. (2008). Early introduction of competitive programming. *Olympiads in Informatics,* 2, 149–162.

Skiena, S. (2008). *The Algorithm Design Manual*. 2nd edition, Springer.

**S. Combéfis** obtained his PhD in engineering in November 2013 from the Université catholique de Louvain in Belgium. He is currently working as a lecturer at the École Centrale des Arts et Métiers (ECAM Brussels), where his courses mainly focus on computer science. He also obtained an advanced master in pedagogy in higher education in June 2014. Co-founder of the Belgian Olympiad in Informatics (be-OI) with Damien Leroy in 2010, he later introduced the Bebras contest in Belgium in 2012 and at the same time founded CSITEd. This non-profit organisation aims at promoting computer science in secondary schools.

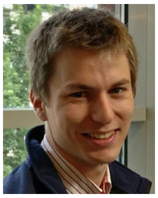


**S.A. Barry** is a bachelor student in computer science engineering at the École Centrale des Arts et Métiers (ECAM, Brussels). He has been interested in electronics and programming for many years. In particular, he likes computer security and wants to improve his hacking skills to detect potential flaws in computer systems to protect them. The LADO project could help him to improve his computing skills.

**M. Crappe** is a bachelor student in electronics engineering at the École Centrale des Arts et Métiers (ECAM Brussels). Born with the entrepreneurial spirit, he is actively involved in innovative extra-curricular projects in electronics and software developments. He hopes that the LADO project will allow him to grasp new skills in algorithm design and optimisation in order to further push his projects.

**M. David** is a bachelor student in electronics engineering at the École Centrale des Arts et Métiers (ECAM Brussels). He shares a great passion for both electronics and programming, as he is involved in the Rust programming language community and enjoys embedded systems development and the internet of things. Joining the LADO project is for him an opportunity to continue to improve his programming skills while pursuing his studies in electronics.

**G. de Moffarts** is a bachelor student in computer science engineering at the École Centrale des Arts et Métiers (ECAM Brussels). He is interested in computer science and electronics, and very curious about engineering and new technologies, such as 3D printing, artificial intelligence and the internet of things. He joined the LADO project to get an early hands-on approach in algorithm design, and hopes to put his training to use in future personal projects.



**H. Hachez** is a bachelor student in computer science engineering at the École Centrale des Arts et Métiers (ECAM, Brussels). Naturally curious, he wants to increase his knowledge in computer science. In particular, he is interested in artificial intelligence and data sciences. He also likes to shine in all the projects he is involved in, and he hopes that the LADO project will help him to improve his skills to drive him among the best.

**J. Kessels** is a bachelor student in computer science engineering at the École Centrale des Arts et Métiers (ECAM Brussels). He is passionate about software development, web technologies and electronics. Having already created several mobile applications and websites, he seeks to improve his coding skills, optimise his applications and learn new algorithms, as a contributor to the LADO project.

Olympiads in Informatics, 2017, Vol. 11, 29–39
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.03

# A New Generation Distance Learning System for Programming and Olympiads in Informatics

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

**Abstract.** The article describes concept of new generation system to teach for programming and olympiads in informatics. The new system is oriented to support teaching and learning of arbitrary programming language, to provide personal approach, to guarantee quality of knowledge and skills of graduates. Special attention is devoted for supporting new programming language course authors as well as teachers and tutors. Many base functions of the new system already are implemented at DL.GSU.BY site. From July 2016 we are teaching C++ programming on the base of the new possibilities.

**Keywords:** programming teaching, olympiad in informatics, distance learning tools.

## 1. Introduction

At June 16, 2016 the author got an e-mail from IOI (International Olympiad in Informatics):

> *From: IOI-announce [mailto:ioi-announce-bounces@lists.ioinformatics.org] On Behalf Of Bernard Blackham*
> *Sent: Thursday, June 16, 2016 9:24 AM*
> *To: ioi-announce@lists.ioinformatics.org*
> *Subject: [IOI-announce] Proposed changes to allowed languages at IOI 2017*
> *Dear IOI community,*
> *The ISC and ITC will propose to the GA at IOI 2016 to remove Pascal and C as accepted languages from IOI 2017 onwards. We will also experiment with allowing Python as an accepted language at IOI 2017, but cannot guarantee solutions in Python will score 100%. The accepted languages at IOI 2017 would therefore be: C++, Java and Python. We will ask the GA to vote on the proposals at IOI 2016.*

*Why drop Pascal? The number of competitors using Pascal has been very small for several years. In IOI 2015, four contestants submitted solutions in Pascal. The effort for the host country to support Pascal each year is quite significant, as the host writes solutions and graders for every task in every language. Dropping support for Pascal will make it feasible to experiment with newer languages at the IOI by moderating the workload on the scientific committee.*

*Why drop C? Even fewer C submissions were received (3) than Pascal at IOI 2015. Although supporting C requires less effort than Pascal due to its syntactical and run-time similarities with C++, it still requires model solutions in C to be written and validated. Solutions in C can generally be adapted with few changes to compile as C++ and perform similarly, so we do not expect students to be realistically affected by this.*

*Why add Python? Python is widely used as an introductory programming language in many countries. The intention of allowing Python is to make IOI tasks more accessible to newcomers to programming.*

*How will you handle the difference in execution speed of Python? Code written in Python is notably slower than the equivalent compiled C, C++ or even Java. The scientific committee will guarantee that it is possible to solve at least one sub-task of every task using Python, but will not guarantee that Python can score full marks.*

*Why not drop Java? Java has also had similarly low submission rates as C and Pascal last year. However, as Java was introduced quite recently, it is too early to assess whether we should continue supporting it at the IOI.*

*Kind regards,*
*Bernard*
*(on behalf of the ITC)*

To be short, it's proposed to remove Pascal and C as well as to add Python as programming language at International Olympiad in Informatics starting from IOI 2017.

Meanwhile solely Pascal was key language for successful programming teaching and learning (Dolinsky, 2016) in the author's courses at DL.GSU.BY site since 1996.

So we are forced to think about elaboration of a new system oriented on some programming language allowed at IOI. On the other hand many university students and sometimes schoolchildren skip olympiad programming and move to professional programming before they finish studying in university and school accordingly. In addition, it often happens that people with another professional education ask author to help them to study programming to get a chance to find a more profitable employment.

Reflections on these themes led to the concept of a new generation distance learning system that is described below. Moreover we have successfully started to realize it. C++ was chosen by the author as new programming language to prepare for olympiads in informatics.. C++ programming teaching piloted since July, 2016 and is in stable use since September, 2016.

Chapter 2 presents the base paradigms of the new generation system.

Chapter 3 describes content of the base education course.

Chapter 4 contains author's analysis of advantages and disadvantages of learning C++ vs Pascal in common sense as well as at DL.GSU.BY site.

Finally Chapter 5 contains conclusions.

## 2. Base Paradigms of the New Generation System

**Supporting of arbitrary programming language.** Currently there are three programming languages at IOI: C++, Java, Pascal (that will be replaced by Python soon). In the national olympiads Pascal will remain for some time. Many other programming languages are used in professional programming. At the same time there is tremendous layer of common base skills obligatory for everybody, who starts to learn any programming language. What are those skills? First of all the following:

- To understand what need to be done (from the problem's description).
- To elaborate appropriate algorithm.
- To accumulate own library of known algorithms.
- To properly code the chosen algorithm.
- To write readable and structured programs.
- To check program correctness on test data including corner cases.
- To debug code and fix possible errors.
- To train practical skills to code in fast and reliable manner.

Such knowledge and skills are marginally connected with programming language. At the same time it's desirable to give student and teacher the possibility to choose arbitrary programming language they like to study.

Our existing courses based on Pascal contain thousands of exercises, some of that have be prepared manually or generated automatically (Dolinsky, 2013). This was a huge amount of work (that lasted for 20 years) that just can't be timely repeated for another programming language (or several languages). This led to the idea to generate "on the fly" all needed preliminary exercises, using a dictionary of chosen programming language's keywords, reference solution in that language, and first three tests (input/output data) of the problem.

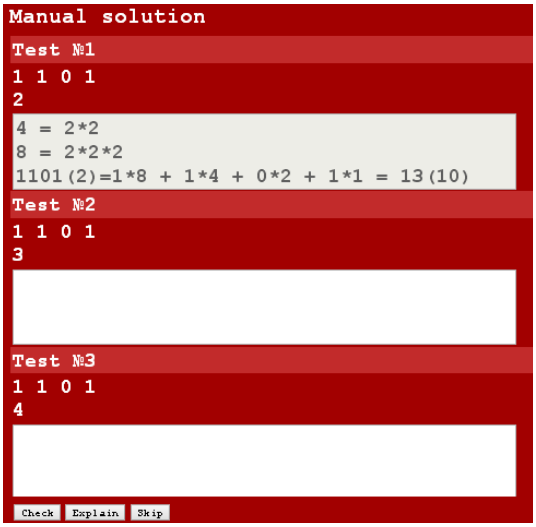
First three tests are used in the first exercise where student need to manually calculate output data for the proposed input data. This exercise ensures that students understand what exactly the program should do.

The dictionary is a list of keywords of programming language and their translation into native language (Russian in our case). These words can be used both to better remember their translation and to train novice pupils' typing skill. The dictionary is also used to translate reference solutions' algorithms to native language which are used in auto-generated preliminary exercises.

Currently we use the following kinds of auto-generated on the fly exercises that pupils can go through in case they are stuck at difficult problem:

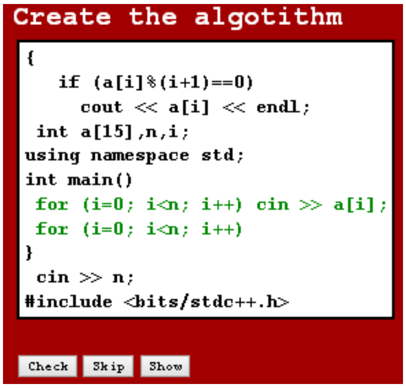
- Manually calculating and typing answers for first three tests of the problem (output data for first test is shown as example, Fig. 1).
- Reconstructing the program from its lines permutation given the correct screenshot (Fig. 2).
- Reconstructing the program from its line permutation without a hint.
- Typing the program line by line given correct code example (Fig. 3).
- Typing the program line by line given native algorithm description.
- Typing the program without hints but with error highlighting (Fig. 4).

Figures Fig. 1–Fig. 4 show examples of these exercises:



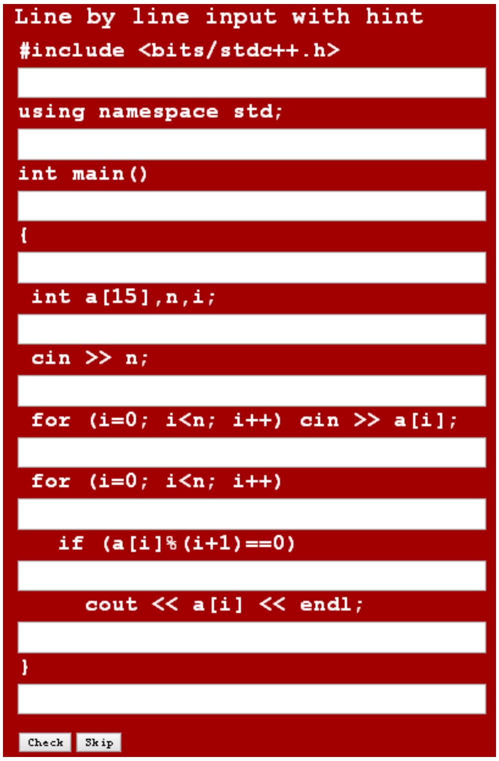
```
Manual solution
Test №1
1 1 0 1
2
4 = 2*2
8 = 2*2*2
1101(2)=1*8 + 1*4 + 0*2 + 1*1 = 13(10)
Test №2
1 1 0 1
3
Test №3
1 1 0 1
4
```

Fig 1. Calculating and typing the answers without programming.



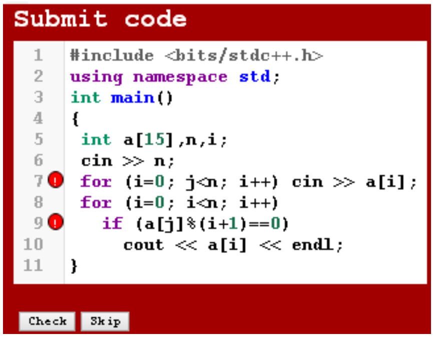
```
Create the algotithm
{
   if (a[i]%(i+1)==0)
     cout << a[i] << endl;
 int a[15],n,i;
using namespace std;
int main()
 for (i=0; i<n; i++) cin >> a[i];
 for (i=0; i<n; i++)
}
 cin >> n;
#include <bits/stdc++.h>
```

Fig 2. Reconstructing the program from permutation of its lines.


Line by line input with hint

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
 int a[15],n,i;
 cin >> n;
 for (i=0; i<n; i++) cin >> a[i];
 for (i=0; i<n; i++)
   if (a[i]%(i+1)==0)
     cout << a[i] << endl;
}
```

Fig 3. Typing the program line by line with hints.


Submit code

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
 int a[15],n,i;
 cin >> n;
 for (i=0; j<n; i++) cin >> a[i];
 for (i=0; i<n; i++)
   if (a[j]%(i+1)==0)
     cout << a[i] << endl;
}
```

Fig 4. Typing the program with errors highlighting.

Note that colour hints are widely in all exercises. Correct lines are marked green in typing and permutation exercises and wrong lines are marked red in typing exercises.

A pupil can navigate preliminary exercise hierarchy using “I don’t know” and “Skip” buttons. When “I don’t know” button is pressed pupil passes one level deeper in the tree. When “Skip” button is pressed, pupil passes to the next exercise at the current level.

Error highlighting in typing exercises is accomplished with help of standard C++ spell checker. Typed program is at last automatically sent to grading system and then the following problem is shown to pupil.

There are plans to generalize such typing exercises to a powerful WDE (web development environment) with incremental syntax analysis, displaying server-executed program's output and even step-by-step debugger. Web debugger implementation is easier for JavaScript but it should also be achievable for the chosen minimal subset of C++.

Eventually, it's sufficient to add language dictionary and base problems' reference solutions in that language to implement support for a new language in existing courses and training system.

**Considerations.** New learners can come with wide variety of previous experience and cognitive skills. One may need to complete just one exercise but another would need to go through ten of them to gain same educational effect. One may be familiar with some topics of the course but need to study deeper the other topics. And a good learning system should strive to provide everybody his own versatile effective educational path. For example, a newcomer could get a series of qualification tests on each topic. If test successfully passed then corresponding topic is skipped, otherwise in-depth learning starts to that topic.

**Differential study.** Base learning course is composed of topics. Each topic consists from four blocks: Technical Minimum, Control, Control Advanced, Practice.

Technical Minimum block contains obligatory exercises with new theory with optional in-depth fine-grained preliminary exercises. .

Control block contains minimal number of exercises to thoroughly check the student's understanding of the theory and ability to make use of it.

Control Advanced block contains advanced exercises that although don't depending on another theory, yet demand smart approach to be solved. Usually we move here exercises from Control Advanced block that are difficult for majority of students.

Practice block contains demonstrative problems from various programming contests where current theory is involved.

Every of Technical Minimum, Control, Control Advanced, Practice blocks can have structured subtopics.

If a student knows how to solve the problem then he types the solution, sends it, scores and gets the new task. Otherwise he can press button "I don't know" button and get a series of preliminary simpler tasks according to one of the following differential study strategies (some already implemented and others planned):

a. **Linear.**

   When "I don't know" button is pressed, student gets a series of described above exercises based on reference solution of the problem (manual answer calculation, code lines permutation, etc.). It is doing so in any from blocks Technical Minimum, Control, Control Advanced, Practice.

   The system remembers for each student the list of tasks where "I don't know" button was used. When student passed the last problems of a topic he is automatically presented with a random series of tasks from his past "Don't know" list. If he presses "I don't know" button again, then the problem remains in the list. Transition to the next topic happens only after "Don't know" list is cleared. Teacher cans advice student to start from Practice block before Control block.

b. **Accelerated.**

As was written above each topic has 4 blocks: Technical Minimum, Control, Control Advanced, Practice.

After Technical Minimum a student is directed into Control block. If he can't solve a task there and press "I don't know" button, he is automatically redirected to Practice block. He proceeds in "linear" mode there. But in any moment he can return in Control block and continue there. But if student can't solve next Control task, he is again redirected to Practice where he stopped last time. If a student solves all tasks in Practice (and also cleared his "Don't know" list if it wasn't empty) then he is also redirected to last unsolved Control task and further he continues like "linear" strategy is turned on. When student finishes "linear" strategy of the Control block he is then directed to Control* block, with "linear" strategy which also allows skipping too difficult tasks.

c. **Adaptive.**

Each task in the Control block is connected by topic name with a group of tasks in the Practice block. If a student can't solve Control task he is automatically redirected to corresponding Practice tasks and returns back after hi has them finished.

d. **Dynamic difficulty levels.**

We define task's difficulty metric as number of days that it's open for submission divided by number of students that solved it. For example, task difficulty of 1 means that it is solved by somebody averagely every day, value of 30 means average monthly success submission rate and difficulty of 365 implies that the task surrenders only once a year.

Taking into account that the course has been open for many years so far, this automatically calculated metric nicely reflects real difficulty of solving the task for students.

The author of the course can observe difficulty metrics of all problems and pointed difficulty levels. For example, three difficulty levels: 1–10, 11–100, 100+. These levels are used to automatically subdivide tasks of Technical Minimum, Control and Practice blocks into subsets of different difficulty. Each subset can be alone passed along as in "adaptive" strategy.

e. **Diagnostics (for input/intermediate/output control).**

Pupils are presented with a sequence of several tasks in order of ascending difficulty peeked from each topic of Control block. Only passed topics are used in case of intermediate screening. Tasks can be skipped, time may be restricted.

Entry diagnostics can be used for exact determination of entry point to study. Intermediate diagnostics can be used for regular control of the education quality. Exit diagnostics can be used as a base for graduate certification.

**Guaranteed knowledge and skills of graduates.** Use of intermediate and exit control guarantees the quality of education.

**Support of teachers and tutors.** We name a teacher the person who will teach group of students "off-line" (in the classroom, for example). Teacher are provided with convenient tools to view the studying results of their students alone and among all other students. We name a tutor the person who will help to teach one or more students not being with them physically in the same classroom or even in the same city. Convenient communication tools for tutors and their students need to be provided.

**Gamification**. Adding game elements into educational process stimulate  increase in motivation to learning. For example:

- Hall of Fame.
- Public rating.
- Prizes for best students from interested companies – T-shirts, icons, notebooks, and pens.
- Time spent and efficiency metrics.
- Dynamic forecast of education results (interpolation based on statistics of all other students).
- Seasonal competitions: Autumn, Winter, Spring, Summer Cups and Person of the Year.

**Support of the course authors.** Currently we support automated testing of programs in the following languages: Pascal, C++, Java, Kotlin, Python, Perl, Ruby, JavaScript. For a new programming language to become supported it is first of all needed to negotiate installation of its compiler on testing servers.

Then an author interested in particular new language would have to provide the dictionary of its keywords and their translations in native language. It is necessary to translate existing task descriptions into desired native language too in case it's not Russian. In case native language is English, dictionary can be absent and all exercises with translation involved would be automatically hidden from course tree. An author then needs to prepare reference solutions for course tasks in the desired programming languages and validate them in test system. There is a tool to automatically apply chosen correct submissions as reference solutions for a range of tasks.

## 3. Content of the Base Course

A new "Accelerated course-2016" was created for described new style education model. This course currently contains 8 topics:

1. Introduction to programming.
2. One dimensional array.
3. Two dimensional array.
4. Geometry.

5. Sorting.
6. String.
7. Text problems.
8. Research.

Each of the topics includes Technical Minimum, Control, Control Advanced, Practice blocks.

The themes have the following subthemes:

**Introduction to programming:** formatted input/output, built-in function, calculus systems.

**One-dimensional array:** input and output, sum of elements, counting of elements with given conditions, maximum value and its index, minimum value and its index, search for elements with given conditions, combined and modified algorithms.

**Two-dimensional array:** input and output a whole array or just one row, one column or diagonals; application of base algorithms for one-dimensional array described above for two-dimensional array and it components (row, column, diagonals).

**Geometry:** distance between two points, distances from one point to an array of points; neighbouring distances; distances between all pairs of points; base and modified algorithms on one- and two-dimensional arrays in application to geometry problems.

**Sorting:** ascending and descending, coordinated sorting of two arrays.

String: reverse, counting, maximum, search, strings of brackets, all different symbols of the string, shift of strings, string generation, array of strings, splitting in tokens.

**Text problems:** one-dimensional array, two-dimensional array, geometry, strings.

**Research:** for (brute force of variable values, brute force on a numeric range); nested for loops (brute force of two variables; brute force of three variables; brute force of digit combinations); while loops; elements of number theory.

## 4. About Transition From Pascal to C++

Switching students who already study Pascal to another language should be considered wisely. The following variants are possible in author's opinion, but every student take the decision himself:

- Don't switch to C++ at all.
- Switch to C++ after getting national diploma.
- Switch to C++ after passed the "Accelerated Course-2013".
- Switch to C++ from the "Accelerated Course-2016".
- Start learning C++ from scratch.

Advantage of switching to C++:

- Capability to participate in IOI since 2017.
- C++ STL provides ready-to-use algorithms and data structures.
- Reference solutions at USACO, COCI, Codeforces, etc. are mostly in C++.
- Saving time on don't learning Pascal.

- At the beginning is easier to learn keywords:
  C++: main int { cin>> cout<< }.
  Pascal: program var longint begin readln writeln end.
- [Some years later] capability to participate in national Olympiads in Informatics.
- Brighter employment prospects (perhaps?).

Disadvantage of switching to C++ (at least for now):

- More sophisticated IDE for coding and debugging.
- More difficult to ensure code correctness.
- Less advanced learning path for C++ for kids since primary school in our system as Pascal courses are more mature and elaborated.

At that moment author is going to act as follows:

- All beginners since grade 5 and above should start with C++ in the "Accelerated Course-2016".
- The beginners of 1–4 grades from other school start with "Learning to think" course. Further learning path would depend on the progress.
- The beginner of 1–4 grades from our base school start with the language and course preferred by their direct teacher.

## 5. Conclusion

This paper represents a concept of a new generation distance learning system for programming and olympiads in informatics. The new system is oriented to support teaching and learning of arbitrary programming language, to provide personal approach, to guarantee quality of knowledge and skills of graduates. Special attention is devoted for supporting the new programming language course authors as well as teachers and tutors. Many base functions of the new system are already implemented at DL.GSU.BY site. From July 2016 we are teaching C++ programming on the base of new possibilities.

## References

Dolinsky M. (2013). An approach to teach introductory-level computer programming. *Olympiads in Informatics*, 7, 14–22.

Dolinsky M. (2014). Technology for the development of thinking of preschool children and primary school childrens. *Olympiads in Informatics*, 8, 63–68.

Dolinsky M. (2016). Gomel training school for Olympiads in Informatics. *Olympiads in Informatics*, 10, 237–247.

Dolinsky M. (2005). *Algorithmization and Programming with TURBO PASCAL: From Simple to Olympiad Problems: Tutorial*. Sankt-Petersburg: "Piter" (In Russian: *Алгоритмизация и программирование на Turbo Pascal: от простых до олимпиадных задач: Учебное пособие*. СПб.: Питер).

Dolinsky M. (2006). *Solving of Sophisticated Olympiad Programming Problems: Tutorial*. Sankt-Petersburg: "Piter" (In Russian: *Решение сложных* и *олимпиадных задач* по *программированию: Учебное пособие*. СПб.: Питер).

Performance Statistics of Gomel pupils in international and national olympiads in informatics from 1997 to 2016. (In Russian).
`http://dl.gsu.by/olymp/result.asp`

**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, 2017, Vol. 11, 41–53
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.04

# Teaching Graphs for Contestants in Lower-Secondary-School-Age

Ágnes ERDŐSNÉ NÉMETH
*Batthyány High School, Nagykanizsa, Hungary*
*Doctoral School, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary*
*e-mail: erdosne@blg.hu*

**Abstract.** A didactically interesting question is how to familiarize lower secondary school children with abstract modelling tools – especially graphs – and how problem-solving paradigms can be developed in their minds while using the structure of graphs and the operations defined on them. In this paper, we make an overview of introducing graph models and basic graph algorithms to lower-secondary-school-age pupils and to older students, who are new to programming.

**Keywords:** graph theory, teaching informatics in primary and secondary schools, preparing for algorithmic contests.

## 1. Overview

A didactically and methodologically interesting question is how problem solving can be developed in children's minds while teaching computational thinking (CT) for all or computer science (CS) for competitors; what steps and tasks lead through from understanding the idea to its professional usage; at what age and abstraction level they can use specific tools and methods.

In this paper, we examine these questions in connection with maybe the most important modelling tool: graphs and also the operations defined on them.

There are many problems in real life where we can draw graphs naturally: like networks of roads, social networks, family tree through relationships, kinship relations and supply chains. There are problems where drawing graphs doesn't appear so naturally, but after simplification we use them for modelling and understanding the core of the problem, like map-colouring, counting the number of elements with certain properties in a specific set, packing and sorting efficiently, describing the states of games. Using graphs is a very simple and intuitive tool for modelling ideas in the solution of various mathematical and CS problems.

The aim of this paper is to examine graphs from a methodologist's perspective, so it is not intended to introduce new algorithms but rather to discover the applicability and limits of using graphs as a modelling tool and data structure for students in lower-secondary-school-age (K 6–10).

## 2. Place of Graphs in Algorithms Textbooks

There are many textbooks about algorithms. They discuss all relevant algorithms sequentially and directly as they are written for university students. The structure of these books is wide-ranging. There are not two textbooks, in which the place of graphs is the same in the sequence of algorithms. However the important role of graphs – as a modelling tools, data structure and problem-solving strategy – agrees in them:

- Cormen: separate chapters about graph data structures and algorithms.
- Dasgupta: two whole chapters plus three subchapters.
- Halim: one chapter out of nine is about graph algorithms and many subchapters are about modelling something with graph structures.
- Kleinberg: almost each chapters contains a subchapter about graphs.
- Rónyai: two whole chapters out of overall ten are exclusively about graphs.
- Sedgewick: one chapter out of overall six is about graph algorithms and 2 subchapters are about trees.
- Skiena: four whole chapters about graph algorithms and four subchapters about graphs as data structures.

These textbooks are almost useless for primary and secondary school students because of their advanced mathematical and informatical contents. Some parts of them may be useful, but just in upper secondary, for contestants preparing for IOI. However, they are very good resources for teachers about some important themes: data structures, specific procedures and different wording of practice tasks.

## 3. Teaching Graphs as a Part of Teaching CT for All – Facing to Teaching Graphs for Competitors

Teaching graphs ideally begins in primary school in mathematics and informatics lessons through using the idea without labelling it.

### 3.1. *Mathematics Lessons*

Teaching graphs in primary and secondary school begins in mathematics lessons in most of the countries, while the pupils create a simple model for combinatorial tasks, like permutations, variations and combinations. In these cases, they just draw an appropriate figure without labelling it as a graph.

## 3.2. *CSUnlugged Activities*

CSUnplugged − CS without a computer is a collection of activities, which can be used to spark interest in and give motivation to learn CT and CS in primary schools. There are four different games among these activities based on graphs.
So the next four games can be used for modelling a problem with a graph without naming the parts of the graph – and solving a small problem on the model. These games can be altered to use a slightly larger but still manageable number of elements in a related model. In this case, the elements of the graph can be named and a specific algorithm should be given to solve the problem on the specific graph model. They can be used to help develop a deep understanding of the whole idea through games for contestants later, in secondary school.

The first example in a book about graphs is an undirected and weighted graph in *Activity 9: The Muddy City (Minimal Spanning Tree)* Fig. 1. The task is:

- At first finding the shortest route between two given point.
- Second time to find the road network with minimum cost.

This is an excellent exercise in the primary school for counting and summarizing numbers systematically on the model. For the secondary-school-age children on a larger example it is a good model for understanding the concept of finding the minimal spanning tree algorithm.

The second one is an example of directed and unweighted graph in the *Activity 12: Treasure Hunt (Finite State Automata)* Fig. 2. The task is to find the way to the treasure

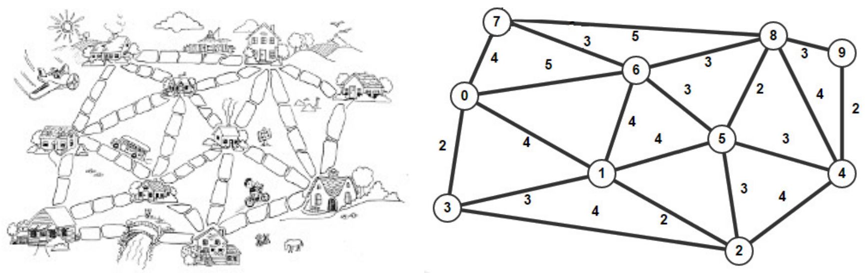


Fig. 1. Muddy City task and its model.

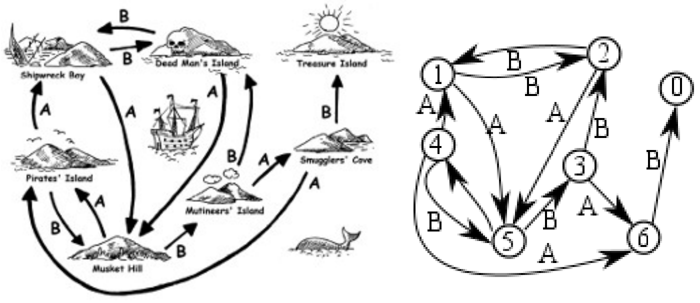


Fig. 2. Treasure Hunt task and its graph model.

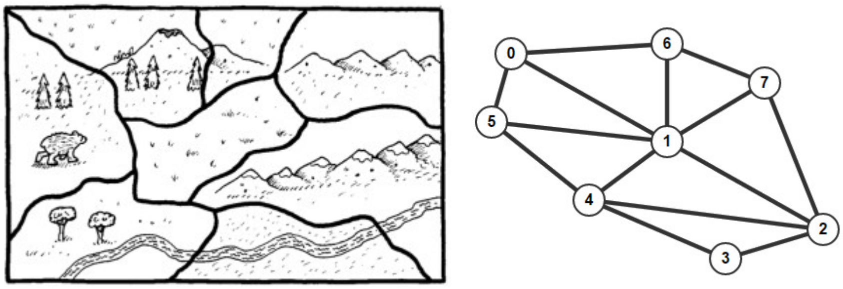


Fig. 3. The Poor Cartographer task and its graph model.

on a Treasure Island and trying to find more than one route. It is a very good example of simplification via model – the vertices (the stages of a journey) are simple numbers without drawing complex island's maps and the cruises are directed edges between these numbers.

There is another type of graph, undirected and unweighted in *Activity 14: The Poor Cartographer (Graph Coloring)* Fig. 3. The task is to colour the countries on this map with as few colours as possible, but make sure that no two bordering countries are the same colour. This task is nearly a simple colouring book with an additional special condition. It is worth to give more than one copy to the children to attempt the appropriate colouring.

For older pupils colouring is not such an interested task, but finding the smallest number of different colours needed for a specific graph model could be exciting activity.

There is also an undirected and unweighted graph model in *Activity 15: Tourist Town (Dominating Set)* Fig. 4.

The task is to mark some of the vertices in a graph model in such a way that all other vertices are at most one edge away from the marked ones. It is a good example of a mind-breaker, which is hard to solve, but it is easy to check if the answer is correct.

There may be more than one correct solution – it is a new idea in informatics tasks while learning CT. This activity leads on to many extensions and variations.

It is an appropriate task to think over the algorithm's execution time when the size of a modelling graph is increasing. It is very important for children to see problems with more than one correct solution and with more elements of the modelling graphs.

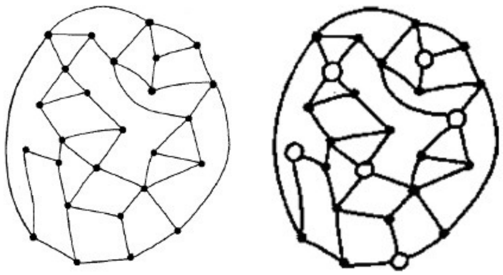


Fig. 4. Tourist Town task's model and its solution.

### 3.3. *BEBRAS*

The tasks of the BEBRAS competition is used to develop CT and to check thinking skills and abstraction level. In K 5–6 all the tasks are marked difficult about graphs, indicating that children have to use non-standard and not commonly used tools (*Shortest paths, Beaver logs, Beaver the Alchemist, Irrigation system, Many friends*). These tasks are about counting something on a task that can be described with an undirected graph model.

All the tasks using a graph model in K 7–8 are marked either medium or hard (*Street stones, Super Power Family, Pirate Hunters, Word chains, Ceremony, Storm proof network, Dice, Hobbit beaver, Cinema, Necklace, Mr. Beaver's shopping*) on undirected and sometimes weighted graphs. The children have to count or summarize a specific property of a graph. In K 5–8 the figure of the model is included in the task's description.

The upper-secondary-school-age pupils have to draw the model itself either it is undirected/directed and unweighted/weighted. In the medium and hard tasks the elements of the graphs are expressively without the exact definitions (*The magician, Word Jumble, Encounter, Expensive bridges, Control of rivers, Neighbourhood*).

The BEBRAS competition's tasks create interest and give motivation to appropriate children to learn CS.

## 4. Teaching CS for Contestants

While learning programming (and on programming contests), lower-secondary-school-age children use basic data structures (integer, boolean, one- and two-dimensional array of integers, simple strings) and basic algorithms can be applied for various problems with various wordings. Choosing, selecting, counting, searching, summarizing, selecting maximum/minimum, sorting, separating into groups are basic algorithms. Children must be proficient on these algorithms.

In the textbooks the basic tasks' wording are very easy and boring. It is possible to make them more interesting with different wording and by making the modelling part interesting as well.

Stages of solving algorithmic tasks are as follows:

- Understand the problem.
- Create an appropriate model.
- Choose the right data structure.
- Select the right algorithm.
- Verify constraints.
- Implement and test.

In addition to the basic tasks, there are many problems where children can apply advanced algorithms creatively and do the above mentioned steps over and over.

In lower-secondary-school-age the tasks can be made more complicated by phrasing them in a way that hides the underlying model, in order to make the problem interesting. Using graphs for simplification and modelling makes these difficult-looking problems easy to understand. The difficulty is supposed to be finding the right model and deriving the correct algorithm. After drawing the model, the corresponding data structure can be as simple as a one- or two-dimensional array, on which a simple, basic algorithm can be used. This method is useful for contestants, whose mind's abstraction level is higher than that of other talented students of their age. In this way graphs can be introduced as a new concept through basic algorithms, to make the comprehension of more complicated algorithms easier at a later time.

## 4.1. *Tasks*

In informatics contests in Hungary, classic graph tasks can be in every round, so the contestants have to be ready to solve them. New tasks are usually worded in a way that highly resembles the wording of former tasks. These tasks come out with another question circularly, as follows. (N is the number of vertices and M is the number of edges.)

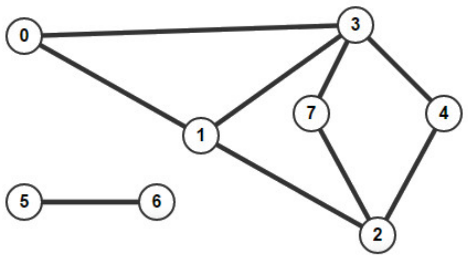
Each of the following chapters describe a teaching block. Each block is a unit of teaching, designed specially for primary school pupils, described in the intended order. At this level trees are to be handled as general graphs, no special property of them is used.

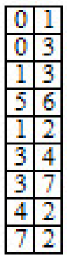
## 4.2. *Using Graph as a Model, Implementing with Edge List*

In this stage of learning the concept of vertex, edge, directed and undirected graphs can be introduced, we use unweighted graphs for modelling. Storing the graph structure is in a two-dimensional matrix as an adjacency list. The basic algorithms, like counting, searching, selecting can be used on an array.

If the modelling graph is undirected (Fig. 5), then in some cases the edge list has to be duplicate for easier handling (each edge is stored twice, and ordered by the endpoints).

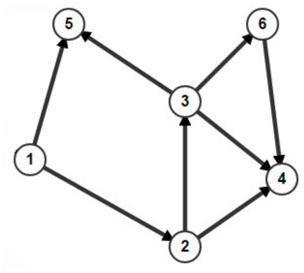
If the modelling graph is directed one (Fig. 6), the order of pairs is important.





| | |
|---|---|
| 0 | 1 |
| 0 | 3 |
| 1 | 3 |
| 5 | 6 |
| 1 | 2 |
| 3 | 4 |
| 3 | 7 |
| 4 | 2 |
| 7 | 2 |

Fig. 5. Undirected, unweighted graph and edge list.



| | |
|---|---|
| 1 | 2 |
| 1 | 5 |
| 2 | 3 |
| 2 | 4 |
| 3 | 4 |
| 3 | 5 |
| 3 | 6 |
| 6 | 4 |

Fig. 6. Directed, unweighted graph described with edge list.

Example tasks are as follows:

- **Family-tree** – *The parents of some people – identified with numbers between 1 and N – are given with an ordered pair of numbers. (1 ≤ N, M ≤ 100 000)*
  - Who are the parents of the people with the given number?
  - Who are the brothers and sisters of the person with the given number? (Somebody is one's brother or sister if they have minimum one common parent.)
- **Acquaintances** – *On a social portal there are N people. The acquaintances between the people are given and always mutual. (1 ≤ N, M ≤ 100 000)*
  - Who are acquaintances of a given people?
  - Are two given people acquainted?
- **Rumour** – *Rumours are formed by some people telling something to others and then those people passing on the rumour. People – identified by numbers between 1 and N – were asked about which person told them the rumour. It is described by an ordered pair of numbers: from to.*
  - Who passed the rumour to a given person?
  - To whom did a given person pass on the rumour?
- **Customs** – *In the middle ages certain towns made merchants who travelled through them pay customs. Of course going through towns was unavoidable, as major roads went through them. For all towns we know a list of neighbouring towns, to which merchants can directly travel, and the amount of customs that has to be paid. (1 ≤ N, M ≤ 100 000)*
  - Which city's customs is the most/less?
  - Which cities are in a one-way-connection of a city with the given number?
- **Gangs –** *The police of Johannesburg knows every criminals in the city. They know that two criminals belongs to the same gang, if and only if they have committed a crime together at least once. Gang members knows each other if they have committed a crime together. The criminals are identified by numbers from 1 to N. The police stores the identifiers each pair of criminals, who have committed a crime together. There may be lonely criminals too. (1 ≤ N, M ≤ 100 000):*
  - Who didn't commit any crime?
  - How many gangs are there?
  - Who are in the same gang with a given person?
  - Who committed a crime together with a given person?

- **Transportation** – *A company has three types of yards: production, warehousing and sales. It has N yards and none of them does two different activities. We know the transportation routes between these yards: routes can be from the production yards to a warehouse or a sales yard, or from the warehouse to another warehouse or to a sales yard. (1 ≤ N, M ≤ 100 000):*
  - Which are the production yards? (Production yard have only outgoing edges).
  - Which are the warehousing yards? (Warehousing yard has incoming and outgoing edges also).
  - How many sales yards are there? (Sales yards only have incoming edges).

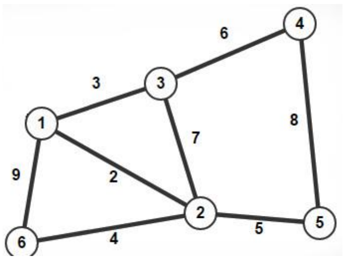
## 4.3. *Using Graph as a Model, Implementing with Adjacency Matrix*

Sometimes the graph model is weighted and easier algorithms can be coded on adjacency matrix. In this stage of studying programing, the transformation between the adjacency matrix and edge list is a challenge, regardless of the modelling graph being directed/undirected or weighted/unweighted (Fig. 7, Fig. 8).

In the next examples the children have to decide, which storage method is more efficient and which basic algorithm has to be chosen:

**Villages** – *The lengths of the roads in a county are given. The villages are identified by numbers between 1 and N. The roads are defined with a list of the numbers of the villages, which are connected by that road. (1 ≤ N, M ≤ 10 000):*

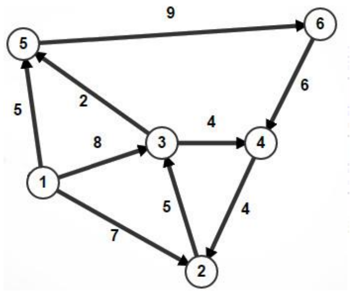
- How long is the longest/shortest road between villages?



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 3 | 0 | 0 | 9 |
| 2 | 2 | 0 | 7 | 0 | 5 | 4 |
| 3 | 3 | 7 | 0 | 6 | 0 | 0 |
| 4 | 0 | 0 | 6 | 0 | 8 | 0 |
| 5 | 0 | 5 | 0 | 8 | 0 | 0 |
| 6 | 9 | 4 | 0 | 0 | 0 | 0 |

| | | |
|---|---|---|
| 1 | 2 | 2 |
| 1 | 3 | 3 |
| 1 | 6 | 9 |
| 2 | 3 | 7 |
| 2 | 5 | 5 |
| 2 | 6 | 4 |
| 3 | 4 | 6 |
| 4 | 5 | 8 |

Fig. 7. Undirected, weighted graph described with adjacency matrix and edge list.



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 7 | 8 | 0 | 5 | 0 |
| 2 | 0 | 0 | 5 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 4 | 2 | 0 |
| 4 | 0 | 4 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 9 |
| 6 | 0 | 0 | 0 | 6 | 0 | 0 |

| | | |
|---|---|---|
| 1 | 2 | 7 |
| 1 | 3 | 8 |
| 1 | 5 | 5 |
| 2 | 3 | 5 |
| 3 | 5 | 2 |
| 3 | 4 | 4 |
| 4 | 2 | 4 |
| 5 | 6 | 9 |
| 6 | 4 | 6 |

Fig. 8. Directed, weighted graph described with adjacency matrix and edge list.

- Which villages are connected with the longest/shortest road(s)?
- Which village is farthest away from the nearest neighbour?
- Which village is the nearest to a given village?

## 4.4. *Counting the Degree of Vertices*

There are a lot of difficult tasks, in which after the simplification and modelling, the task is to count, how many edges lead to a vertex, or in directed graphs how many edges goes to and from a vertex. The algorithm is easy, but the children have to think about how the edges can be counted.

Example tasks are as follows:

- **Zoo** – *In the zoo the walking paths between animals are known. The entrance is numbered with 0 and the animals are numbered from 1 to N. The walking paths are defined with two animals' number which cages are linked by a walking path. (1 ≤ N, M ≤ 100 000):*
  - How many of the cages are in a dead end of walking paths?
  - Which animals can be visited directly from the most others?
- **Villages** – *As before. A dead-end village is a village, which is connected with other villages by just one road. (1 ≤ N, M ≤ 100 000):*
  - Which villages leads the most number of roads to? (If there is more than one, then list all!)
  - How many dead-end villages are there?
  - Which villages are dead-end villages?
- **Family-tree:** *(1 ≤ N, M ≤ 100 000):*
  - Who has the most/less children? (If there is more than one, then list all!)
  - Who hasn't got any children? (If there is more than one, then list all!)
- **Rumour:** *(1 ≤ N, M ≤ 100 000):*
  - Who didn't pass on the rumour?
  - Who started the rumour?
  - Who passed on the rumour to the most others?
- **Acquaintances:** *(1 ≤ N, M ≤ 100 000):*
  - Who has the most/less acquaintances?
- **Gangs:** *(1 ≤ N, M ≤ 100 000):*
  - How many lonely criminals exist?
  - Who is a lonely criminal?
- **Towns:** *(1 ≤ No. of towns ≤ 10 000, 1 ≤ No. of triangles ≤ 100 000)*

  *There are N towns in a convex polygon shape country. The country was divided to triangle counties, the vertices of which are towns. The towns are identified by numbers from 1 to N. The number of triangles is known. Every triangle is defined by giving the three corresponding towns.*
  - How many cities are on the border of the country?
  - How many neighbours does each city have cities have?

## 4.5. *Tasks on Special Graphs*

There are a lot of special cases, when solving the problem is based on a specific attribution of a graph, like:

- In a tree the root and levels.
- Complete graph.
- Complementer graph.
- In a directed graph the super source and the super sink.

Some examples:

- **Villages**: *(1 ≤ N, M ≤ 100 000):*
  - What is the length of the longest way to the dead-end villages, for which there is no edge to get off the path ie. the only options at any intermediate village are continuing on the path and turning back?
- **Family-tree:** *(1 ≤ N, M ≤ 100 000):*
  - Who has the most grandchildren? (If there is more than one, then list all!)
  - Who hasn't got any grandchildren? (If there is more than one, then list all!)
- **Acquaintances:** *(1 ≤ N, M ≤ 10 000):*
  - Which pairs have common acquaintances?
  - Who knows each other without any other common acquaintances?
  - Which pairs have common acquaintances without knowing each other?
- **Gangs:** *(1 ≤ N ≤ 50, 1 ≤ M ≤ 100):*
  - How many gangs are there?
  - Who has the most connection in a certain gang?
  - What is the number of members of the largest gang?
  - Who are the member of the largest gang?
  - Who are the members of a gang, where just the head knows everybody else?
  - How many "totally ordered" gangs are, where everybody knows everybody?
- **Trip** – *On a map (of size N*M) the altitude of each gridpoint above the sea level is known. The time of a hiking between two adjacent points is (1+absolut value of the height's difference). A point cannot be reached directly from another adjacent point if the difference of their high is more than given number H.*
  - What is the minimum time to reach a given Q point from a given Ppoint?
  - Give a shortest path between a given Q and P points?
- **Castle** – *There are octagonal rooms in a castle, connected with square shaped hidden doors. The visitor has to pay different amounts when entering an octagonal room. Using the hidden doors cost the same each time.*
  - What is the minimum time to reach a given point from another given point?
  - What is a way with minimum time to reach a given point from another given point?

## 4.6. *Floyd-Warshall Algorithm and its Variations*

After teaching recursion and dynamic programing concept, we can make a detour into the world of graph algorithms with a classical Floyd-Warshall algorithm and its variations. It is used to solve:

- General case of the path existence on an unweighted graphs.
- General case of shortest paths problems on weighted graphs.

It is frequently used in other problems, as long as the input graph is small. It needs to store the graph with adjacency matrix, but the code is only four lines: three nested loop. It is understandable after learning dynamic programing concept, the concept of Floyd-Warshall is on the Fig. 9.

It has many variations: maximin, minimax, safest path, minimin, maximax [Horváth2].

- **Yard:** *A company wants to sell its product in different towns (numbered from 1 to N) and looks for one town, where it is optimal to build a warehouse. The distances of the roads between towns are given. (1 ≤ N, M ≤ 400):*
  - What is the shortest distances to all the towns from a given town?
  - Which is the best place to plant a yard? (The distance of the farthest town is the shortest).
- **Trip** – *(1 ≤ N ≤ 400):*
  - What is the minimum time to reach from a given point P to a given point Q?
  - What is the path with minimum time from a given point P to a given point Q?
- **Castle:** *(1 ≤ N ≤ 400):*
  - What is the minimum cost to reach B from A? (for every pair).
  - What is the way with the minimum cost from A to B?
- **Duty** – *(1 ≤ N ≤ 400):*
  - What is the minimum cost to reach a given town from another given town?
  - What is a way with minimum cost to a given town from another given town?
- **Acquaintances:** *(1 ≤ N ≤ 1000):*
  - Which pairs have only common acquaintances? (at least all of them have one acquaintance).

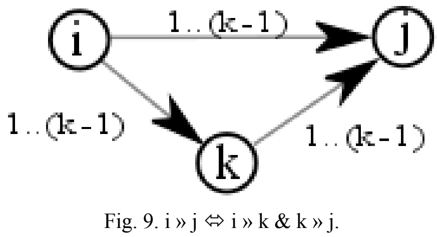


Fig. 9. i » j ⇔ i » k & k » j.

### 4.7. *Further Studies*

With these simple tasks the children can be familiar with the concept of a graph and its components, as a modelling tool and as a thinking tool as well. Next level of their studies is more complicated data structures, like list and stacks, following by more complex algorithms from this topic, like graph traversal, spanning trees and more advanced algorithms.

## 5. Conclusions

Some antecedents of the graph concept might come up in earlier mathematical and informatical studies. If you are aware of this, introducing graphs as a new modelling tool and using graph algorithms as a problem-solving strategy is much easier.

We think, if you want to teach graph algorithms you could start the whole process in the lower-secondary-school-age and circularly, and then when returning to it in higher and higher levels your students would be familiar with this concept.

Finally, graphs and graph algorithms would not be magic for the contestants, just a useful modelling tool and problem-solving strategy.

## References

Bebras–International Contest on Informatics and Computer Fluency (2007–2017). `http://bebras.org`; `http://www.beaver-comp.org.uk/`; `http://informatik-biber.de/archiv/`

CSUnplugged_OS_2015_v3.1. `http://csunplugged.org`

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2001). *Introduction to Algorithms.* MIT Press, 2nd edition.

Dagienė, V., Stupurienė, G. (2016). Bebras – a sustainable community building model for the concept based learning of informatics and computational thinking. *Informatics in Education,* 15(1), 25–44.

Dasgupta, S., Papadimitriou, C.H., Vazirani, U.V. (2006). *Algorithms*. McGraw-Hill.

Halim, S., Halim, F. (2014). Competitive Programming 3. The New Lower Bound of Programming Contests. `http://cpbook.net/`

Horváth, Gy. (2004). A programozási versenyek szerepe az oktatásban. In: *INFOÉRA Konferencia.* `http://www.infoera.hu/infoera2004/eaok/horvathgyula.pdf`

Horváth, Gy., Horváth, Gy., Zsakó, L. (2016). Variations on a classic task. In: *XXIXth DIDMATTECH.* 72–78.

Kleinberg, J., Tardos, É. (2006). *Algorithm Design*. Addison-Wesley.

MESTER online task archive and judge system. `https://mester.inf.elte.hu`

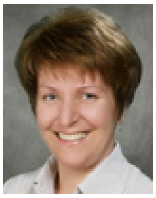
Rónyai, L., Ivanyos, G., Szabó, R. (1999). *Algoritmusok*. Typotex.

Sedgewick, R., Wayne, K. (2011). *Algorithms*, Fourth Edition. Addison-Wesley.

Skiena, S.S. (2008). *The Algorithm Design Manual*. Springer-Verlag, 2nd edition.

Szlávi, P., Zsakó, L. (2012). Informatika oktatása. *TÁMOP-4.1.2 A1 és A2 könyvei, ELTE IK.*

Zsakó, L. (2012). Variations for spanning trees. *Annales Mathematicae et Informaticae*, 33, 151–165.



**Á. Erdősné Németh** teaches mathematics and informatics at Batthyány Lajos High School in Nagykanizsa, Hungary. A lot of her students are in the final rounds of the national programming competitions, some on CEOI and IOI. She is a PhD student in the Doctoral School of Faculty of Informatics, Eötvös Loránd University in Hungary. Her current research interest is teaching computer science for talented pupils in primary and secondary school.

Olympiads in Informatics, 2017, Vol. 11, 55–75
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.05

# Improving Teaching and Learning Computer Programming in Schools through Educational Software

Metodija JANCHESKI
*University Ss. Cyril and Methodius*
*Faculty of Computer Science and Engineering*
*Rudzer Boshkovikj street, 16, 1000 Skopje, Macedonia*
*e-mail: metodija.jancheski@finki.ukim.mk, meto@ii.edu.mk*

**Abstract.** Computer programming is the universal language of our planet and a basic literacy in the digital age. There is no doubt that learning computer programming at a young age is helpful for all students at least in their everyday life. The benefits of learning programming help young students to gain advantages in thinking, processing and communication. These benefits can support acquiring, developing and improving the 21st-century skills among youth.

One of the main challenges of scientists and educational practitioners in the field is how to make computer programming attractive and interesting for the students in primary and secondary schools. The use of various educational software could have positive impact on this issue.

There are many successful examples of educational software used in schools. This paper emphasizes the usefulness of Scratch, Logo, ToolKid and other similar education software tools in teaching and learning computer programming fundamentals. Some of the most important features of such tools, including immediate feedback (instant positive reinforcement), visually, block-based, text-based and object-oriented programming are explained in details.

The author presents several practical examples of how the educational software tools mentioned above can improve teaching and learning computer programming.

**Keywords:** computer programming, Scratch, Logo, ToolKid, computational thinking.

## 1. Introduction

The first steps in computer programming are very important. Older generations were taught the languages like Basic, Fortran, and Pascal and gradually move on to C language. But these programming languages require knowledge of well symbolic expression through mathematical and logical expressions. When children jump straight into a traditional programming language, they get bored and discouraged, because one of the biggest obstacles in learning a programming language is learning the syntax of the language (Vlieg, 2016). Therefore, programming has handled older students in primary education. The modern computer languages, like Python, Delphi, C++, C# and Java, also require similar level of previous knowledge.

It was expected the appearance of new visual programming languages, including Visual Basic and Visual C to make easier the process of programming. Unfortunately, it turned out that they are not suitable for young programmers.

In today's digital world, coding is a fundamental skill alongside math and reading, but too few kids have the opportunity to learn to program because it is rarely taught in school (Tynker…, 2017). Today, at the time of the graphic interfaces and multimedia it is also hard for young programmer to work in text mode.

For a long time, the opinion prevailed that programming is necessary only for gifted math students and to those who will continue to deal with programming as their further profession. Fortunately, some leading scientists, inventors and educators recognize the benefits of programming for all students. They were aware that programming drives innovations, leading to success in other life areas. According Nicholas Negroponte from One Laptop Per Child Project (Sande, Sande, 2014):

> *"Computer programming is a powerful tool for children 'to learn learning.' ... Children who engage in programming transfer that kind of learning to other things".*

In the same spirit, Dan Shapiro, Robot Turtles inventor said (Gardner, 2014):

> *"Being able to program will make children better at whatever they do... No matter what you do, programming unlock doors for you, helps you express yourself, and helps you become more successful in anything you decide to do".*

Follows the chronological list of some notable programming languages with the year of development given in the brackets: Fortran (1957), Lisp (1958), Basic (1964), Logo (1968), Pascal (1970), C (1972), C++ (1980), Python (1991), Visual Basic (1991), Java (1995), JavaScript (1995), Delphi (1995), C# (2001), and Scratch (2003).

Before appearance of Scratch in 2003, as mentioned above, it was generally accepted that the first programming language for young programmers should be the language Logo. Because of its simplicity and features for quickly getting the effective graphical results, Logo gained great popularity in schools worldwide. Compared with Logo, the new programming language Scratch goes one step further in terms of simplicity, attractiveness and visual programming.

The question then arises as to what is the most appropriate age to start learn text-based programming languages, including Logo. The same applies to visual programming languages like Scratch. There is a continuous debate on these issues and we are witnessing various solutions worldwide. There are also many unanswered questions including whether students should learn to code in elementary schools and whether computer programming need be introduced to everyone in primary and secondary education or it should be optional.

First, there are examples where students aged eight learn text-based programming languages. Considering only the abovementioned fact that learning the syntax of the language is one of the main obstacles, this approach demands high level of patience and adaptation of teachers.

On the other side, even the major universities use visual programming languages like Scratch in the introductory computer science subjects. According analysis done by Philip Guo in July 2014, University of California Berkeley, University of Wisconsin Madison and Brown University teach either $CS_0$ or $CS_1$ using the Scratch (Guo, 2014).

Finally, considering the importance of the programming as one of the most valued 21st century's skills, the main challenge is how to enable the kids to start programming before they can read. Generally accepted opinion is that programming is best learned early (Tynker…, 2017), which is also true for learning foreign languages. In other words, no age is too early to learn programming, or 'the earlier, the better'. It is clear that learning while sitting in front of a computer for a long time is not a proper solution for young students. Short interactive multimedia applications were proved as an alternative approach that offers easier entry into programming (Ghose, 2016). This approach includes visual programming languages and educational games.

Andrew J. Ko, a researcher at the Information School at the University of Washington believes that the age of ten is crucial for learning programming languages. He said (Ghose, 2016):

> *"Once kids are about 10 years old, they may be able to work with coding languages on a computer. Right around that age, children develop a more sophisticated theory of mind and are able to predict what others are thinking and feeling – which also means they are able to make models of what their snippets of code will produce".*

The issues related to select which programming languages to teach, when and in what order remains to be decided by the educational experts, scientists and practitioners, as well as, educational institutions, in accordance with their educational strategies and politics and the best practices worldwide.

The Table 1 is not finished. It should be completed with other "serious" programming languages for higher students' ages. According the author, the suggested next steps after the young programmers are mastering Scratch programming language is programming with Logo, Python, Perl or Delphi, to reach the final phase, i.e. programming in C++ and C#, as a programming languages for professionals. In this spite, it is useful to review other findings from the abovementioned data analysis. For each university, Philip Guo looked for $CS_0$ and $CS_1$ courses in the CS, CSE, or EECS department. Follows the results of his analysis: Python (27), Java (22), MATLAB (8), C (7), C++ (6), Scheme (5), Scratch (3). Obviously, Python was the most popular language (80% of the top 10 CS departments, and 69% of the top 39 CS departments) for teaching introductory computer science courses at top-ranked U.S. Computer Science departments in 2014. Note that Scratch was the only visual programming language that made this list.

Computer programming is a great intellectual hobby; it provides the same opportunity for creative, concrete work in mathematical thinking that drama or creative writing does for verbal thinking (Harvey, 1997). It is very important to ensure kids to have fun

Table 1

Programming languages with recommended age and short description

| Programming language/game (recommended age) | Short description / Educational value |
|---|---|
| BeeBot (1+) | A simple, real-world toy that can teach kids the basics of coding. It uses simple left- and right-buttons on the robot, and kids should learn how to sequence their commands to get the BeeBot from one end of the room to the other, avoiding obstacles along the way (Ghose, 2016). |
| Robot Turtles (4+) | An actual, physical board game that surreptitiously teaches kids the basics of programming. The game teaches kids how to use directions to navigate their turtles through a maze to a tasty jewel (Ghose, 2016). |
| Light Bot (4–8) | An iPhone or Android app that teaches kids to navigate a robot through a maze, turning on lights. |
| Dash & Dot (5+) | A programmable robot pack that may be the best for slightly older kids, around age 8, who are already excited about programming (Ghose, 2016). |
| ScratchJr (5–7) | The free Android or iPhone app which allows kids to use simple icons to code their own interactive stories and games (Ghose, 2016). |
| The Foos (5–10) | The free iPhone application that uses simple icons with symbols, such as monsters, arrows and speech bubbles to solve adventures like chasing down a donkey thief or rescuing puppies lost in space. Kids can learn the basics in an hour (Ghose, 2016). |
| Tynker interactive courses (7+) | Tynker is an online platform that easily and successfully teaches students how to code through games and stories. Students learn the fundamentals of programming and design through Tynker's intuitive visual programming language (Tynker…, 2017). |
| Scratch (8 – 16) | Scratch is a simple coding language, completely free and open to use. It gets kids exposed to fundamental coding concepts, such as repeating loops and if-then statements using bright, color-blocked textual commands (Ghose, 2016). |
| Lego Mindstorms (10+) | Lego Mindstorm combines the LEGOs with motors, sensors and remote controls. With Mindstorms, young students can build robots that walk, talk and do as they command (Fichtner, 2014). |

while they learn, so they stay engaged and continue learning and creating. Nowadays, there is no doubt that programming allows kids to be creative and helps the self-confidence building and developing among them. Programming improves mathematical skills in a fun way. It teaches problem-solving skills and helps kids visualize abstract concepts (Tynker…, 2017).

## 2. Scratch

### 2.1. *What is Scratch?*

Scratch is a block-based imperative, event-driven, dynamically-typed (whether data types agree is checked during program execution) and interpreted programming language*. As a tool that offers "programming without proper programming", Scratch is simple and clear programming language. It is also object oriented, visual programming tool which does not require any prior programming knowledge and experience. Scratch provides a rich learning environment for people of all ages (Marji, 2014).

With this programming language young programmers can program (create) their own computer games, interactive stories, animations, simulations and other multimedia projects and then to share their creations online (Marji, 2014; Honey and Kanter, 2013; Pollock, 2014; Ford, 2009). Moreover, the Scratch website enable young people from around the world to learn from each other, to get and give feedback, to share interactive tutorials, guided tours, science experiments, book reports, online newsletters, and much more (Vlieg, 2016; Pollock, 2014).

This 14 years old programming language is available in more than 40 languages and used in more than 150 countries (Vlieg, 2016; Scratch…, 2017). More tinkerable, more meaningful and more social than other programming environments are the three core design principles established for Scratch (Vlieg, 2016). Creating with Scratch also encourages students to learn to think creatively, work collaboratively, and reason systematically – essential skills for success and happiness in today's world (Vlieg, 2016; Pollock, 2014). The main motto of the Scratch project is: Imagine, Program, Share.

Block programming with Scratch is relatively easy, even for young children, and it's a good way to enter the world of programming. With Scratch, young programmers easily understand the basic concepts of programming in a very effective and fun way. It has been used to introduce important computational concepts such as repeat loops, conditional statements, variables, lists, data types, events, and processes to students of many different ages, from elementary schools through universities (Vlieg, 2016). Scratch also enable students to learn important mathematical concepts and terms, including coordinates, variables, and random numbers. After learning Scratch, transition to traditional text-based languages can be done more easily (Vlieg, 2016).

According M. Resnick, one of the founders of Scratch software (Pollock, 2014),

> *"Scratch is more than a piece of software. It is part of a broader educational mission. We designed Scratch to help young people prepare for life in today's fast-changing society".*

Developed by the MIT Media Lab's Lifelong Kindergarten Group, Scratch was conceived as an educational language that would make programming fun and accessible to a new generation. The researchers at this Group believed that it was very important for all children, from all backgrounds, to grow up knowing how to design, create, and express themselves. Inspired by how kindergarteners learn through a process of experimenting, creating, designing, and exploring, the Lifelong Kindergarten Group extended this style of learning to programming in general and Scratch in particular. The primary goal of the Scratch initiative was not to prepare people for careers as professional programmers but to nurture a new generation of creative, systematic thinkers comfortable using programming to express their ideas. When you learn to code in Scratch, you learn important strategies for solving problems, designing projects, and communicating ideas (Vlieg, 2016).

* https://wiki.scratch.mit.edu/wiki/Scratch_Wiki_Home

## 2.2. *How Scratch Works?*

After launching Scratch, the students can start trying things right away. There is a default character (the Scratch cat), which already has some media to play with: two images that form a walking animation, and a "meow" sound. The students can start programming behaviors for the cat immediately: click the *move* block and the cat moves; click the *next costume* block and the cat animates; click the *play sound* block and the cat meows. The blocks start with reasonable default values for their inputs, so the user don't need to fill in any inputs (Honey and Kanter, 2013).

Scratch uses graphical blocks (puzzle-piece shapes) of code to represent programming commands. Instead of typing commands, a student can create Scratch program (project) by dragging, dropping and snapping graphical blocks of code into different sequences and combinations (stacks, scripts), much like snapping Lego bricks together (Vlieg, 2016; Honey and Kanter, 2013). The connectors on the blocks suggest how they should be put together (Vlieg, 2016). While pulling blocks from the palette, it is immediately obvious, from the shapes of the blocks, which blocks can relate to one another.

Unlike traditional text-based programming languages, there is no syntax to learn and the Scratch programmers are relieved from all worries about the syntax. Instead, the grammar is visual, indicated by the shapes of the blocks and connectors. Blocks snap together only if the combination makes sense (Honey and Kanter, 2013).

From a collection of simple programming blocks, combined with images and sounds, young students can create a wide variety of different types of projects. While creating Scratch projects, they typically engage in an extended tinkering process – creating programming scripts and costumes for each sprite, testing them out to see if they behave as expected, then revising and adapting them, repeatedly. Scratch has a range of features that allow programs monitoring as they run. Scratch scripts always highlight while they are running, so the students can see which code is being triggered when (Honey and Kanter, 2013).

## 2.3. *The Main Benefits and Advantages of Using Scratch*

The initial objective of the Scratch project was to encourage creativity, imagination and curiosity among the young students. The ultimate goal is to develop a shared community and culture around Scratch (Vlieg, 2016).

Scratch's visual programming environment enables students to explore areas of knowledge that would otherwise be inaccessible. It provides a full set of multimedia tools that can be used for creating wonderful applications, which can be done more easily than with other programming languages (Marji, 2014).

The greatest benefit of using Scratch is disclosure the interest in programming among young computer users. Today, understanding and mastery the complex machines like personal computers requires several years of work and learning. It is therefore very important to provide an opportunity for learning programming in an easy and understandable way.

One of the things that are crucial to attract and excite young developers is the feeling of ruling with animation on the computer screen. Commands for graphics management in Scratch are extremely powerful. Scratch variables can be used to manage a variety of graphics and sound effects. Therefore, it is not surprising that many people recognize Scratch as a "tool to create simple multimedia". Although Scratch is a programming language for novices, it is highly oriented to graphics, sound effects and animations.

With Scratch and its code blocks, the students can control and mix graphics, animations, music, and sound to create interactive stories, games, simulations, art, and animations and even share their creations with others in the online community (Vlieg, 2016). One of the main advantages of Scratch is that operation of simpler program modules is much better understood graphically rather than textually.

Scratch is an excellent programming language for beginners as it allows creating very attractive programs even at the first acquaintance with the programming. It introduces a quite simple philosophy, thanks to which anyone can almost immediately create their first interactive game, a cute animation or some other vivid creation. Looking like with ease mastered the computer, the young programmer encourages himself for further work. We should put an emphasis on perfecting the developer technique, and on the control over the program, and not on the animation effects. The advantage of Scratch is that its programs are displayed graphically: the scripts are complex, consisted of colorful blocks. Thanks to this, it is easy to understand how and what they work while the syntax errors (misspelled commands) are disabled. The other thing that Scratch supports is the immediate feedback which is high valuable for young students.

In many ways, Scratch promotes problem-solving skills – important in all areas of life, not just programming. The environment provides immediate feedback, allowing you to check your logic quickly and easily. The visual structure makes it a simple matter to trace the flow of your programs and refine your way of thinking. In essence, Scratch makes the ideas of computer science accessible. It makes learning intrinsically motivating; fosters the pursuit of knowledge; and encourages hands-on, self-directed learning through exploration and discovery. The barriers to entry are very low, while the ceiling is limited only by your creativity and imagination (Marji, 2014).

## 2.4. *Disadvantages of Scratch*

The possibilities of Scratch for solving mathematical and logical problems are modest. Due to lack of commands, like Input and Read, Scratch will not provide more than an effective animation. Also Scratch will not teach about modern techniques of programming, so it is necessary to continue with learning other programming languages.

## 3. Logo

Logo is a complex text-based educational programming language. It was designed in 1967 as a tool for learning – about computer programming, but also about other domains – mathematics, language, art, music… (Logo…, 2017).

Many believe Logo language is a language of children. Strictly speaking, the Logo language is also suitable for children. Others feel that the Logo is used for drawing the figures, it is also true. But most important is that the Logo is a language suitable for solving wide range of tasks of programming (Дичева *et al.*, 1996).

Logo is a dialect of Lisp, the language used in the most advanced research projects in computer science, and especially for solving tasks in the field of artificial intelligence. It was developed by artificial intelligence researchers. Their idea was to see if they could use some of their experience with the problem of trying to get computers to think in order to help human beings learn to think more effectively – at least about certain kinds of problems. Since 1984, Logo is no longer the only member of the Lisp family available for home computers. Another dialect, Scheme, has become popular in education (Harvey, 1997).

In contrast to earlier programming languages, which emphasized arithmetic computation, Logo was designed to manipulate language – words and sentences. Like any programming language, Logo is a general-purpose tool that can be approached in many ways. Logo programming can be understood at different levels of sophistication (Harvey, 1997).

Initially, the software was used in grades 6 to 8 and was mostly valued for its animation and turtle graphics. Everyday work gave the teachers a deeper understanding of Logo and confidence in using the program. This change the initial notion of Logo as a "kid's language". Logo is increasingly used at the high school level and more and more teachers consider Logo a suitable instrument for their own work. Informatics teachers, sometimes with the help of students, create Logo-projects that serve as small-scale learning programs (Papert, 1999).

When computer classes were initiated in elementary schools, the teachers noticed that even young children can create complex and interesting programs despite the fact

Table 2

The most common Logo commands

| Type of commands | The commands |
|---|---|
| Moving commands | FORWARD (FD), BACK (BK), RIGHT (RT), LEFT (LT), HIDETURTLE (HT), SHOWTURTLE (ST). |
| Navigation (position) commands | HOME, SETX, SETY, SETXY, SETPOS. |
| Color commands | SETPENCOLOR (SETPC), SETBACKGROUND (SETBG), SETFILLCOLOR (SETFC), SETFILLMODE, SETFILLPATTERN (SETFP), PENREVERSE (PX). |
| Drawing commands | CLEARSCREEN (CS), PENDOWN (PD), PENUP (PU), SETPENWIDTH (SETPW), SETPATTERN, PENERASE (PE), SHOW PEN, SHOW PENSTATE |

that they have not yet learned what “structured programming” is. It has become widely accepted that Logo is the program of choice for introducing primary schools’ students to computers (Papert, 1999).

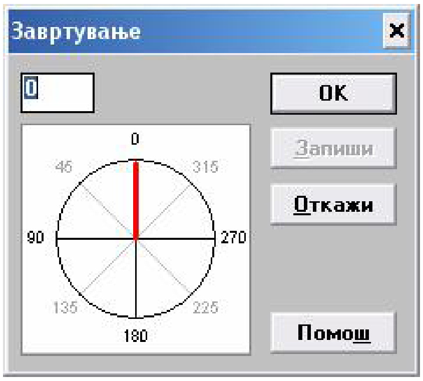
Logo should not be perceived as a mere technological tool, but as and innovative approach that encourages individuality and autonomy (Papert, 1999).

Logo language is widespread. It has some rich graphical capabilities. It’s most known feature is the turtle (or small triangle) which movement can be easily managed (Дичева *et al.*, 1996).

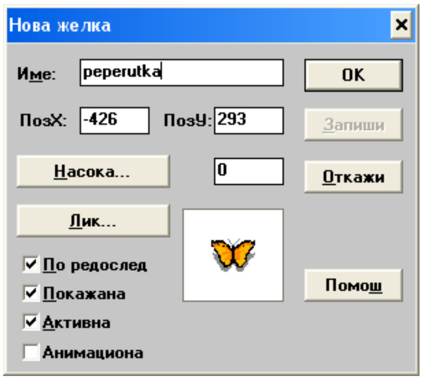
The Fig. 1 present some Logo windows.

The major role of Logo in schools, can be described as follows:

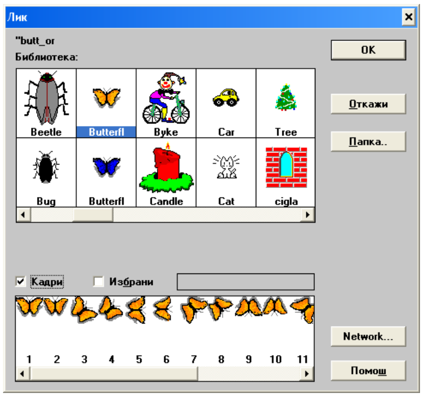
- Students learn Logo for learning’s sake.
- Students study the basics of algorithms and programming. Logo is used as either the major environment for programming or as an example of a programming language. In elementary schools the goal might be phrased more modestly as “cognitive development”.



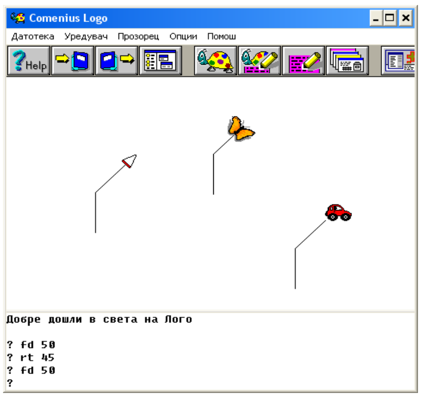
a) Selection of the angle of spinning.



b) Properties of the “new” turtle.



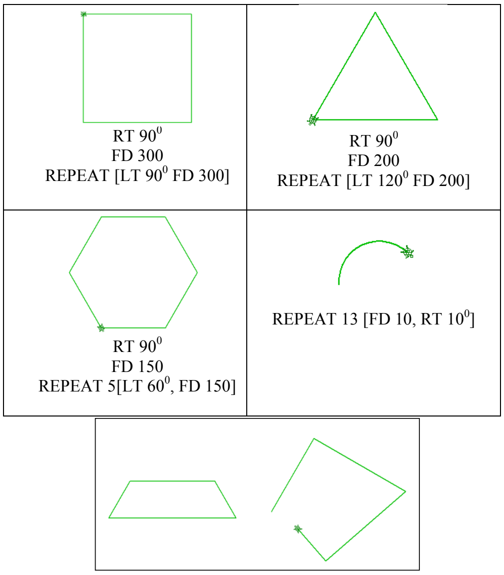
c) Selection of the favourite image.



```
Добре дошли в света на Лого

? fd 50
? rt 45
? fd 50
?
```

d) Three active “turtles”.

Fig. 1. Logo windows.



```
RT 90^0
FD 300
REPEAT [LT 90^0 FD 300]
```

```
RT 90^0
FD 200
REPEAT [LT 120^0 FD 200]
```

```
RT 90^0
FD 150
REPEAT 5[LT 60^0, FD 150]
```

```
REPEAT 13 [FD 10, RT 10^0]
```

Fig. 2. Some geometric shapes in Logo.

- Logo is used to convince students that a computer is a convenient instrument for everyday work. Students create projects on elected themes and for other school subjects (Papert, 1999).

The Fig. 2 presents several geometric shapes drawn with the Logo tools, accompanied by a list of commands used for their drawing.

## 4. ToolKid

The ToolKid software, based on Commenius Logo is used in all primary schools of Macedonia (K4) since 2006. Macedonian version of this software was updated, expanded and adapted from Bulgarian version. Relevant curricula were developed and the software was installed in primary schools. Licenses for the introduction of this software were purchased by E-School Project. The Primary Education Project continue with ToolKid implementation and development where the E-School Project ended. Nowadays, the teachers are obliged to use ToolKid software during 30% of their regular classes.

This software did not pretend to change the existing tools, methods and techniques, recently used in the education, but its primary objective was to complement them, while enabling teachers and students to work in a new environment, providing plenty of opportunities, which offer new ways for answering the placed requests (Jancheski, 2016).

The ToolKID educational software could be useful in all subjects of K-4 education (mother tongue, mathematics, nature and society, society, art education and music education). Its organization is such that enables students to acquire solid informatics literacy and culture. The primary education is the right place where the students need to obtain basic informatics knowledge. The curriculum needs to follow the informatics knowledge trends and to be redesigned in compliance with them (Jancheski, 2016).

The software package ToolKID contains 48 programs divided in 7 groups: Educational games, Drawing programs, Text programs, Sound programs, Animation and Video, Data Combining, Algorithms and Programming. These groups are presented in the main screen with images giving hints about the type of the programs included in them (Fig. 3).

The resources created with one program, like backgrounds, animation, images, sounds can be used in other in native and easy way by children. It is important and creative for them when they are going to work on different project topics.

The group Algorithms (Fig. 4) collects programs that develop the algorithm thinking (Pouring, At the river, Towers, Paths), mathematical knowledge (Figures, how many they are?) and lead to the world of the turtle geometry (Labyrinth).

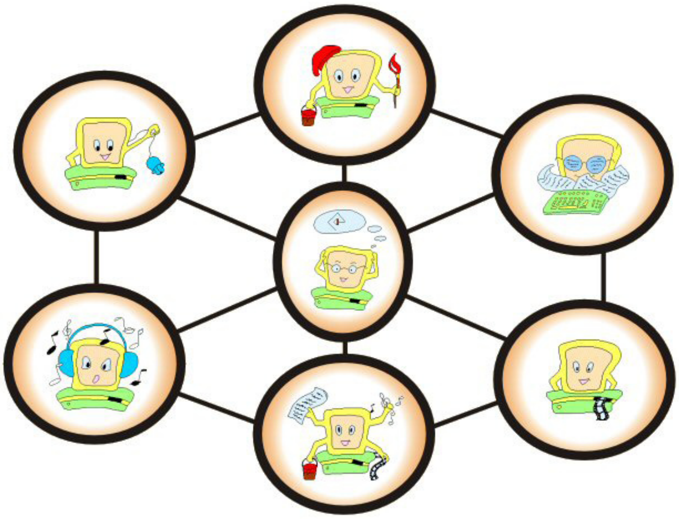


Fig. 3. Seven groups of Logo programs.

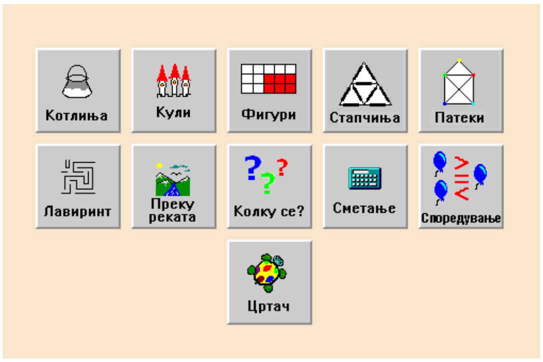


Fig. 4. Algorithms.

## 4.1. *Advanced ToolKid Usage*

ToolKid programs are designed for different class grades and they share common resources. The students can use certain ToolKid programs to create various multimedia files, which can be further shared by other ToolKID programs, or even to be used outside ToolKid environment. The modular structure of ToolKid will hopefully meet the requirements for usability at different age and different level of programming knowledge and skills.

The ToolKid software, especially the Teacher module, offers a variety of options for software development and new programs creation. By using Teacher Module, teachers can change various settings in each of the following programs: Silhouettes, Cat, Pie, Cards, Puzzles, Color, Painter, Music, Gossip, Clouds, Rebus, Notebook, and ABC. Teacher module allows teachers to share with their students these settings and other resources. This could contribute individual approach to different target groups or individuals and to improve active learning methods, including problem based learning, and project based learning. ToolKid also enables students to use the local PC resources, as well as, external multimedia files available on Internet (Jancheski, 2016).

## 4.2. *Contribution of ToolKid Software in Teaching and Learning Computer Programming*

Unfortunately, all of the above described programs, except ToolKid, are not a part of the regular curriculum in primary schools in the country. They can be used during extra work with gifted students who have an affinity for informatics and programming.

As stated above, TulKid programs are important in acquiring the contents of several school subjects. The main objectives of ToolKid software usage include:

- Acquiring the basic informatics literacy to the level of solving simple problems.
- Attaining a part of the curricula objectives for particular subjects by using information technology.
- Acquiring and developing logical and creative capabilities.
- Developing the regular attitude toward both computers and its programs using and protection.

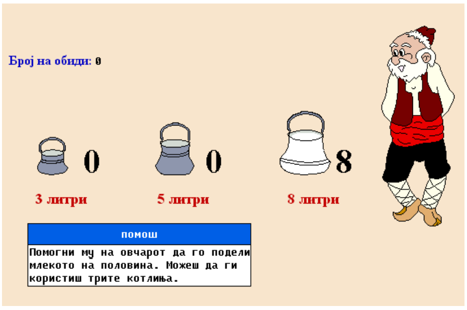
ToolKid is not a typical software for teaching and learning programming, as Scratch and Logo, but it is very effective to be used as a preparation for programming.

In order to program specific problems and tasks, students should be well knowing the nature and substance of these issues, including the entire range of possible solutions. ToolKid programs Pouring, At the river, Towers, and Paths allow students to study various problem situations and their variations through play and fun, in attractive visual environment. They are enabled to gradually reveal multiple sets of solutions and to explore for the best, quickest and most effective algorithms, programs and solutions. They should respect the rules and limitations of each of these programs. Horizons of students should be expanded by considering variations of ToolKid programs involving more complex problem situations, rules, requirements and restrictions.

The majority of the K-4 teachers interviewed by the author during the phases of implementation and monitoring of ToolKid software were highly motivated to use this software and are eager for further professional development in the field. They usually are not programming professionals but they have to be aware about their role in development of algorithmic and computational thinking among students, which is crucial, according the author.

## 4.3. *The Pouring Game*

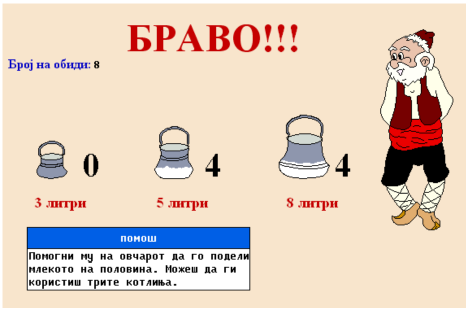
The shepherd has three bowls with the largest being full of milk. You need to help shepherd to share this milk fairly into two equal portions using only the three bowls and no other measuring devices. There are two levels of this game. The first one (Fig. 5 a) with three bowls with 8, 5, 3 liters, respectively, and the second (Fig. 5 c) with three bowls with 10, 7 and 3 liters, respectively. Each pouring from one to other bowl means one step. Determine a method of dividing with minimum steps.



Број на обиди: 0

Помогни му на овчарот да го подели млекото на половина. Можеш да ги користиш трите котлиња.

a) Initial position (level 1).

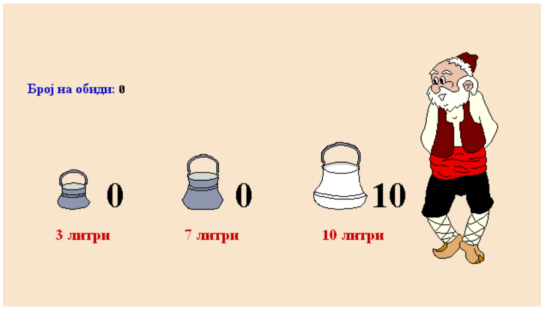


БРАВО!!!

Број на обиди: 8

Помогни му на овчарот да го подели млекото на половина. Можеш да ги користиш трите котлиња.

b) Final position (level 1).



Број на обиди: 0

c) Initial position (level 2).

Fig. 5. The Pouring Game.

Table 3

One solution on the Pouring program (first level)

| Bowl / Step | I<br>3 l. | II<br>5 l. | III<br>8 l. | Pouring |
|---|---|---|---|---|
| 1 | 0 | 0 | 8 | III → II |
| 2 | 0 | 5 | 3 | II → I |
| 3 | 3 | 2 | 3 | I → III |
| 4 | 0 | 2 | 6 | II → I |
| 5 | 2 | 0 | 6 | III → II |
| 6 | 2 | 5 | 1 | II → I |
| 7 | 3 | 4 | 1 | I → III |
| 8 | 0 | 4 | 4 | |

The Table 3 presents schematic view of possible solutions on the first level of the program.

## 4.4. *Across the River (Game)*

The six tasks (Fig. 6) contained in this educational game belong to two variant of tasks.

**Variant 1** (Fig. 7)

The shepherd and the characters (1. sheep, hay and wolf; 2. goat, cabbage and wolf; 3. rabbit, cabbage and fox) came to the bank of a river. The shepherd's challenge is to carry himself and all the characters to the far bank of the river.

The game requires the shepherd to get all the characters across a river in a small rowing boat. Only the shepherd can operate the boat. The boat can cross the river many times to get everyone across. As some of the abovementioned animals represent a real danger to other animals or objects, the sheep should be careful to do the task

Fig. 6. Menu with 6 tasks.

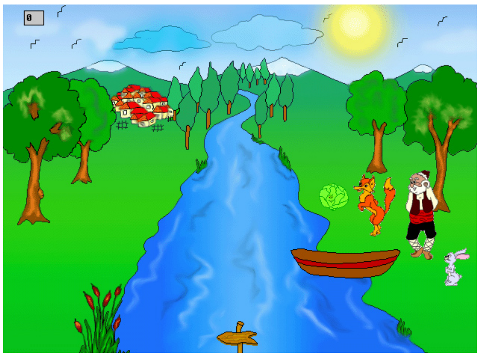


Fig. 7. Variant 1.

Table 4

Solution of Variant 1

| N⁰ of trips | Moves | Left shore | Right shore |
|---|---|---|---|
| | | S, R, C, F | |
| 1 | S, R → | C, F | S, R |
| 2 | ← S | S, C, F | R |
| 3 | S, C → | F | S, C, R |
| 4 | ← S, R | S, R, F | C |
| 5 | S, F → | R | S, F, C |
| 6 | ← S | S, R | F, C |
| 7 | S, R → | / | S, R, F, C |

*Legend*: S-Shepherd, R-Rabbit, C-Cabbage, F-Fox

keeping all characters intact. For example, if left unattended together, the wolf would eat the sheep and the goat, the fox would eat the rabbit, the sheep would eat the hay, or the goat would eat the cabbage. Solve this game in the smallest number of crossings.

*Example with no solution* (Fig. 8)**:** the girl and the characters: cat, mouse, and cheese.

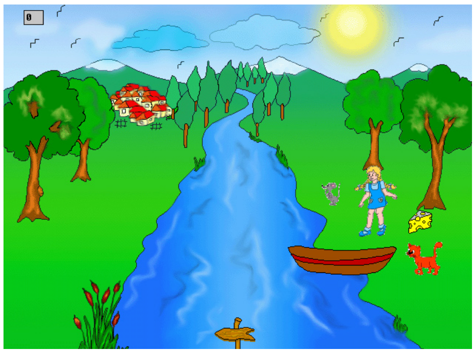


Fig. 8. Example with no solution.

**Variant 2**

A grandmother and a grandfather of equal weight (90 kg), together with their two grandchildren, each of half their weight (45 kg), wish to cross a river in a boat that only accommodate a maximum weight of one adult (90 kg). If the total weight of passengers exceeds the capacity of the boat, it will sink. It is known that all four family members can operate the boat. What is the minimum number of trips all family members need to cross the river?

*Subvariant*: Without grandfather and with the same game conditions.

## 4.5. *The Tower (Mathematical Game)*

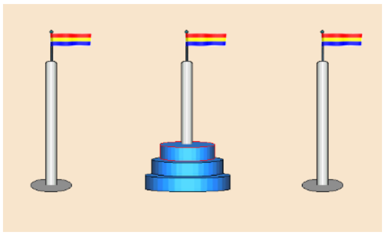
There are three vertical towers and n disks, ($3 \leq n \leq 7$). The initial position of the game is with all discs placed on one tower in ascending order of size. The goal is to move discs from this tower to another tower with minimal number of steps, while respecting the following rules: 1) each step consists of one move, 2) each move consists of taking the uppermost disk from one tower and placing in on the top of another tower, 3). No one disk may be placed on top of a smaller disk. The game provides counter of movements.

Before starting the game, the gamer has an opportunity to select the level, i.e. the total number of used disks. Allowed numbers are all elements of the set {3,4,5,6,7}.

When the gamer wants to move one of the disks, he/she need to click on it (after which its edges are stained in red), and then to click on the destination tower (where he/she want to place the disk). And without complying with the rule that it is not allowed to place greater on a smaller disk, the program will not allow it. In case the gamer wants to cancel the last selected disc before being moved, it can be done by clicking on the departure tower (the tower where this disk is placed).

The Fig. 9 provides a solution for n = 3 (three discs) where all three discs of middle tower should switch on the right tower with minimal moves.

We saw that with three disks, this mathematical game can be solved in seven moves. For a given number of disks, n, the minimum number of moves required to solve a Tower of Hanoi puzzle is $2^n - 1$.


a) Initial position.

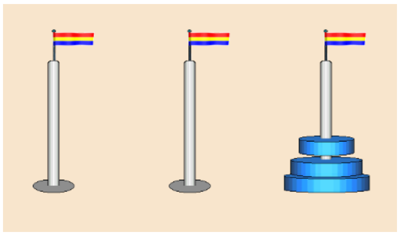

b) Final position.

Fig. 9. The Tower.

## 4.6. *The Paths (Mathematical Game)*

The program has three levels. Before choosing the level, select the desired figure out of four offered figures.

**Level 1.** Help the turtle to pass all sections of the path once by successively clicking on the colored circles. The turtle will automatically move after each subsequent click.

**Level 2.** Select the path that the turtle will pass by successively clicking on the colored circles. The turtle will carefully wait for you to press the Enter button, and then she will pass the path you have paved for her.

**Level 3.** Find out which circle should stand for the questionnaire. Then click on it and then press Enter. The turtle will start moving along the paved path, while leaving the painted trail down the path.

The first figure and the first level are predefined, as seen in the Fig. 10.

The Fig. 11 presents cases with different figures in three levels, respectively.

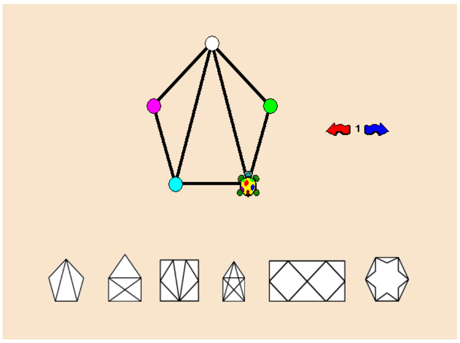
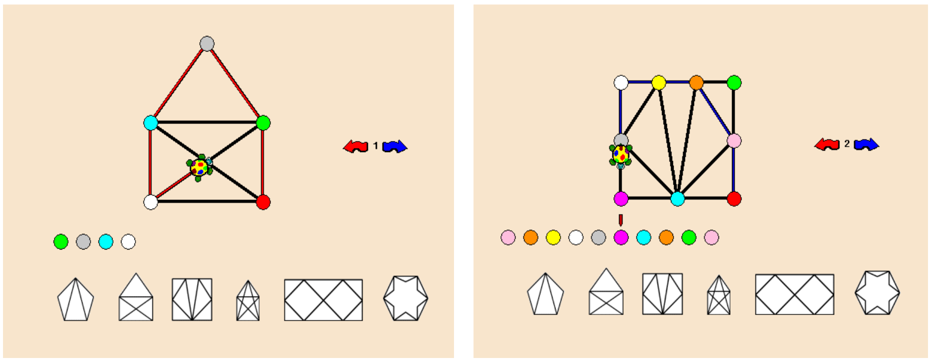


Fig. 10. Initial position of the game.



a) Level 1. b) Level 2.

Fig. 11. Cases with different figures.

## 4.7. *Labyrinth (Maze)*

Help the turtle to find her way through the maze from start to finish, while eating as much as possible food units.

The toolbar functions (Fig. 12) include:

- Moving one step forward/backward.
- Right angle rotations.
- Setting the speed of the movement (slow, normal, fast).
- Setting the step length.
- Selecting a maze from a set of mazes.
- Setting the number of fruits on the path.

Initially, the gamer determines the type of the maze, the step length and the speed of the turtle and the number of food items. The following figure shows a case with step length = 35, speed = 2, number of food items = 1.

Then the gamer uses the keys for navigation and movement to manage the turtle on the path to the goal. The turtle can never break down the walls which stand in the way in the form of black lines. If you direct to the wall, it will hit it and recover back one step.

The Fig. 13 presents a situation where the turtle successfully reached the target and delight with strawberries. The entire path the turtle passed from the start to the finish is colored in red.

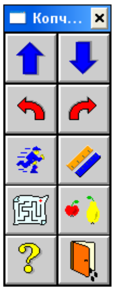
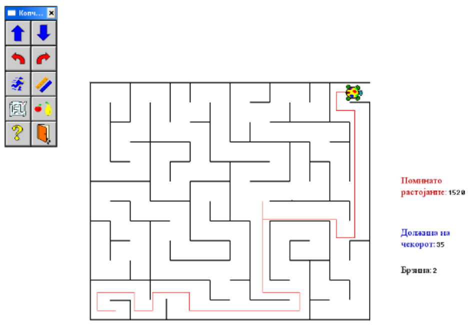


Fig. 12. The toolbar functions.



Поминато растојание: 1520

Должина на чекорот: 35

Брзина: 2

Fig. 13. Labyrinth 1.

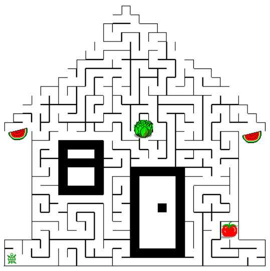


Fig. 20. Labyrinth 2.

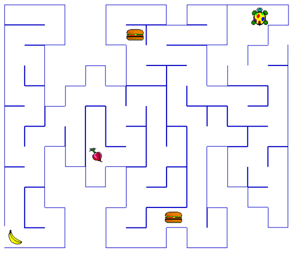


Fig. 21. Labyrinth 3.

Fig. 14. Examples with different initial parameters.

The Fig. 14 presents two examples with different initial parameters.

## 5. Conclusion

The development of the informatics industry, especially the computer technology, actualizes to a great extend the question for the promotion of the computer literacy as one of the crucial competencies of both the young people and the adults. This development every day increases the need of computer uses in the educational process and determine the criterion for literacy to be the mastery in working with computers. Enabling students for using the computer technology leads to approaching to the high educational standards which means contribution for the development of the brain freedom, knowledge, innovations and creativity as a base of the society.

Educational institutions, scientists, experts and teachers should be in line with actual achievements in science and new technologies. They also should be aware of and to promote 21st-century skills, including:

- Information and communication skills.
- Thinking and problem-solving skills.
- Communication and self-directed learning skills.
- Ability to use technology to access, manage, integrate, and evaluate information, construct new knowledge; and communicate with others effectively.
- Ability to learn academic content through real-world examples.

(Source: Partnership for 21st-Century Learning, 2004)

The programming skills are a part of skills listed above. They are widely recognized as a fundamental issue in today’s digital word. Regardless of what programming language we use, it is important to know that programming allows us not only to do experiments with the computer, but also, by developing certain (good) style of programming, can help us to think better. Only it turns programming into activity worthy to be carried by people who do not intend to become specialists in informatics. The adoption of algorithmic way of thinking is an advantage in today’s dynamic world, where people are forced to constantly plan (operations, finance, materials) and sometimes to operate in conditions of chronic lack of resources of all kinds, including the information (Дичева *et al.*, 1996).

Scratch, Logo, and ToolKid are three successful projects in the field of computer programming for novices. The first two is directly related with programming, while the role of the third is latent; it could to be used as a preparation for programming.

Well-trained experienced teachers will know how to exploit the potential of such programs to promote algorithmic way of thinking as a part of computational thinking, and to encourage creativity and competitive spirit among the students. Critical thinking, problem solving and innovation are also issues where the role of teachers is unchangeable. The author fully agrees with the statement of Ed Lazowska, adjunct professor in University of Washington (Robot..., 2017):

> *"In the 21st century, computational thinking is essential for everyone. 'Computational thinking' is a problem analysis and decomposition, algorithmic thinking and expression, functions and abstraction, fault isolation and debugging".*

## References

Дичева, Д.К., Дичева, Д.К., Николов, Р.В., Сендова, Е.Й. (1996). *Информатика в стил Лого*. Просвета.

Fichtner, A. (2014). *7 Interactive tools to teach your kids computer coding*. http://www.sheknows.com/parenting/articles/1048851/fun-ways-to-teach-your-kids-to-code

Ford Jr., J.L. (2009). *Scratch Programming for Teens*. Cengage Learning.

Gardner, A. (2014). *Teach Coding with Robot Turtles!* https://educators.brainpop.com/2014/05/23/teach-coding-robot-turtles/

Ghose, T. (2016). *The Best Coding Toys for Kids*. Live Science press. http://www.livescience.com/53957-best-coding-apps-and-toys.html

Guo, P. (2014). *Python is Now the Most Popular Introductory Teaching Language at Top U.S.* Universities. https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext

Harvey, B. (1997). *Computer science Logo style, Vol. 2. Advanced techniques*. The MIT Press.

Harvey, B. (1997). *Computer science Logo style, Vol. 1. Symbolic computing*. The MIT Press.

Honey, M., Kanter, D.E. (Eds.). (2013). *Design, make, play: Growing the next generation of STEM innovators*. Routledge.

Jancheski, M. (2016). One decade of ToolKid software implementation in Macedonian primary schools. In: *International Conference of Education, Research and Innovation Proceedings*. Seville, Spain, 2877–2886.

*Logo foundation* (2014–2017). http://el.media.mit.edu/logo-foundation/what_is_logo/index.html

Marji, M. (2014). *Learn to Program with Scratch: A Visual Introduction to Programming with Games, Art, Science, and Math*. No Starch Press.

Papert, S. (1999). Logo philosophy and implementation. Logo Computer Systems Inc.

Pollock, W. (2014). *Super Scratch Programming Adventure*. Canadá: The LEAD Project.

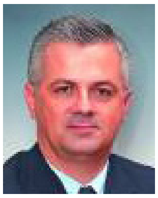
*Robot Turtles* (2014–2017). http://www.robotturtles.com

Sande, W. D., Sande, C. (2014). *Hello World! Computer Programming for Kids and Other Beginners*. Manning Publications.

*Scratch Wiki* (2017), https://wiki.scratch.mit.edu

*Tynker – Learn to Code* (2012–2017). https://www.tynker.com

Vlieg, E.A. (2016). *Lists. In Scratch by Example*. Apress, 223–248.



**M. Jancheski**, master of IT sciences, teaching/research assistant at the Faculty of Computer Science and Engineering, under the "Ss. Cyril and Methodius" University in Skopje. The leader of the Macedonian team on 7 Balkan Olympiad in Informatics and 5 International Olympiad in Informatics, since 1998. Consultant and trainer in the crucial national projects in the field of IT and informatics.

Olympiads in Informatics, 2017, Vol. 11, 77–86
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.06

# An Introduction to Running Time Analysis for an SOI Workshop

Dennis KOMM, Tobias KOHN
*Department of Computer Science, ETH Zürich*
*Universitätstrasse 6, 8092 Zürich, Switzerland*
*e-mail: {dennis.komm,tobias.kohn}@inf.ethz.ch*

**Abstract** We organize an introductory course on algorithm design and complexity analysis for prospective participants of the Swiss Olympiad in Informatics and interested high school students. The students are assumed to have some background in programming, but no formal computer science education.

Our goal for the first lesson is to introduce them to the basic tools used in running time analysis, with a particular emphasis on worst-case versus best-case analysis, uniform versus logarithmic cost measurement, and big-$O$ notation; however, we avoid being too formal and use the example of primality testing to introduce the concepts hands-on. In this paper, we describe our approach in detail.

**Keywords:** SOI workshop, running time analysis, worst-case analysis, cost measurement, big-O notation

## 1. Introduction

We organize an introductory course on algorithm design and analysis for high school students. The course was originally intended for participants of the SOI (Swiss Olympiad in Informatics). However, it has become an important part of our outreach program, and now, all interested high school students with basic programming skills are welcome. This also allows us to encourage gifted students in our course to participate in the SOI. Many prospective contestants primarily lack the necessary self-confidence rather than the skills to try and participate in an Olympic contest. Hence, opening the SOI classes to all interested students might lower the barrier for some students to take part in the SOI.

One particular challenge of the course is to give a proper introduction to algorithm analysis to an audience that has only basic knowledge in programming, and is not familiar with the formal analysis of algorithms and their running times. At the same time, we regard the issue of time complexity as a central aspect of algorithmic design and problem solving strategies.

The attending students are typically around 16 to 18 years old. In order to attend our course, we require the students to have some basic training and experience in programming. The material presented in this paper covers a class of 90 minutes. During class, the presentation of the material is interleaved with hands-on exercises. Students are given some time to work individually on the given tasks, using their own laptops.

Our approach is to use an easy computational problem to demonstrate principles such as time complexity, best- and worst-case analysis, uniform and logarithmic cost measurement, and big-$O$ notation in the first lecture of the course. In what follows, we describe the steps taken in detail, based on the teaching material supplied.

The programming languages used are both Python and C++. Since all students have experience in programming with Python (a prerequisite of the course), we decided to start with that language and switch to C++ at a later lecture; in particular, we use TigerJython, which has been developed by one of the authors (Kohn, 2017).

### 1.1. *Overview*

In order to discuss the theoretical concepts mentioned above, we chose the example of determining whether a given integer is a prime number. With some simple steps, the basic algorithm of testing all possible divisors can easily be improved. First, we can move to testing odd numbers only (after an initial test if the given integer is even). Second, the number of divisors to test can be cut down further by observing that it suffices to test up to the square root of the given integer.

This task can be used to discuss measuring the time complexity of an algorithm with respect to the input length, and introduce big-$O$ notation. Moreover, this example also allows us to discuss the notions of average-case and worst-case scenarios, giving more depth to the idea of measuring time complexity.

### 1.2. *Related Work*

While the IOI is clearly recognized for its contests, the goals and benefits of the IOI movement reach much further. Dagienė notes that “the high-level goal of the IOI is to promote computer science among the youth, and to stimulate their interest in programming and algorithms” (Dagienė, 2010). As such, a most important part are training classes and outreach programs, spurred by the IOI movement and its international contests. The high value of the IOI for CS education is also confirmed by, e.g., Sysło: “The Olympiad conducts intensive educational activities” (Sysło, 2011).

There is general agreement that programming is a good starting point for CS education, even though some noteworthy exceptions exist (e.g., Bell *et al.* presented sophisticated materials which do not require any prior programming skills (Bell *et al.*, 2009). However, programming by itself is not enough (Dagienė, 2010), but must be completed by farther-reaching topics.

When it comes to continuing CS education beyond programming, on one hand, problem solving skills are often seen as the core competence to be achieved by students (cf., e.g., Wing, 2006). On the other hand, though, the ability to evaluate a program's properties such as complexity/efficiency and correctness (cf., e.g., Sysło, 2011) is a prerequisite for improving algorithms and finding more efficient solutions. Hence, our starting point with a discussion of complexity is a first step to a more thorough discussion of algorithms and problem solving strategies in general.

For students, it is particularly important to clearly motivate the discussed topics, and build on examples and practical activities (Dagienė, 2010). A good "connection between practice and theoretical concepts" (Dagienė, 2010) is key in fostering the students' understanding of both theory itself and its relevance.

## 2. Explaining Complexity

We start our class by discussing a typical task of a computer scientist, which is to answer whether a given natural number is a prime number. It is easy to motivate the problem, as it is well known by most students, and some of them are usually already familiar with methods such as, e.g., the *sieve of Eratosthenes* (cf., e.g., O'Neill, 2009; Sedgewick, 1992). However, our aim is not to present the best solution for the problem (even if this were possible within the time boundaries of the course), but to improve the straightforward solution step by step, while introducing the aforementioned tools of algorithm analysis at the same time. Therefore, we start with the simplest possible way to determine whether a given natural number `x` is prime, i.e., we consider the algorithm shown in Algorithm 1, which "answers" whether `x` (which we assume to be at least 3) is a prime number. For such a given `x`, the algorithm simply checks whether it is divided by any number between 1 and `x`, i.e., whether it is divided by $2, 3, \ldots,$ or `x` – 1. If such a number is found, it answers "Composite," otherwise it answers "Prime." Naturally, we want to assess how long the algorithm takes to compute a result; we call this the algorithm's *time complexity*. Thus, the first question we ask is how to quantify this time.

**Algorithm 1.** Determining whether a given number is prime

```
def prime(x):
    divisor = 2
    while (divisor < x):
        rest = x % divisor
        if (rest == 0):
            print "Composite."
            return
        divisor += 1
    print "Prime."
```

A straightforward approach is to measure the absolute time taken by the program's execution in, say, milliseconds. However, a statement such as "this algorithm takes 15 milliseconds on this instance" is not very meaningful since this time obviously depends on the machine on which the algorithm is executed. It is rather undesirable to be limited to comparing two algorithms only if they are run on the same machine. This is easy to see for the students. Another point is that the programming language matters; if the students are already familiar with basic C++, they can verify this themselves by implementing the same algorithm in both Python and C++ and run it on the same input. We argue that, consequently, another more robust approach is needed, which gives better insight into an algorithm's performance. The key observation is that computational problems usually require more time to be solved if the input length increases. The main question is what this increase looks like; is it linear, quadratic, cubic, exponential, . . . ?

Now, we discuss with the students that, if we would invoke `prime(x)` with `x` being set to 100 003 (which is prime), the body of the `while`-loop is executed roughly 100 000 times; if `x` is increased and still prime, the algorithm surely takes longer and longer.

Next, we give a small introduction to encoding natural numbers in binary; to most students, this is already known and needs no detailed description. With $n$ bits, we can encode a number between 0 and $2^n - 1$, which means that encoding the number `x` takes roughly $n \approx \log_2 \mathtt{x}$ bits. Therefore, executing Algorithm 1 with prime input `x` takes a time that grows with $2^n$ with $n$ denoting the binary length of `x`. Of course, in each execution of the loop, there are a number of computational steps involved, which we need to account for, but for now we just focus on the number of times the body of the loop is executed.

It is easy to see that Algorithm 1 does the job of testing a number for primality, but it also does quite some unnecessary work. We first argue that it suffices to only test odd numbers between 1 and `x` (after testing whether `x` is divisible by 2), which reduces the number of loop executions by a factor of roughly 2.

While this improvement is easy to see, the next step, namely that we only need to test whether `x` is divisible by a number of at least 2 and at most $\lfloor\sqrt{\mathtt{x}}\rfloor$, needs a little more thinking. Most students are not familiar with the formal concept of a proof by contradiction. Yet it is possible to argue that there cannot be two divisors $a$ and $b = \mathtt{x}/a$ that are both larger than $\lfloor\sqrt{\mathtt{x}}\rfloor$. We conclude that, if the input `x` is not a prime number, we will always find a number between 2 and $\lfloor\sqrt{\mathtt{x}}\rfloor$ that divides it. If we do not find such a number, then there is no other divisor, and `x` is therefore prime. This idea is incorporated in Algorithm 2, which we usually have the students discover themselves. A typical mistake is to use "<" instead of "<=" in line 2 of Algorithm 2. Note that this implementation does not only test divisibility by odd numbers. This is done on purpose, because we want to compare this improvement to the above constant-factor improvement with respect to the naive algorithm.

We also ask the students to reason about the time complexity of their new algorithm in terms of how often the loop is executed now, which is roughly $\sqrt{\mathtt{x}}$ times, and hence $\sqrt{2^n} = 2^{n/2} \approx 1.414^n$.

We discuss the implications of this improvement in class with our students. To this end, we give a demonstration that is inspired by the book of Cormen *et al.* (Cormen *et al.*, 2009). Assume we run both Algorithms 1 and 2 on a computer that is able to

**Algorithm 2.** Determining faster whether a given number is prime (I)

```
from math import sqrt

  def prime(x):
      divisor = 2
      while divisor <= sqrt(x):
             rest = x % divisor
             if (rest == 0):
                    print "Composite."
                    return
             divisor += 1
      print "Prime."
```

process 1 000 000 executions of the `while`-loop per second. Suppose further that we execute Algorithm 1 with the prime number $\mathtt{x} = 100\,000\,000\,000\,031$, which means that we need roughly

$$\frac{100\,000\,000\,000\,031 \text{ iterations}}{1\,000\,000\,\frac{\text{iterations}}{\text{second}}} \approx 100\,000\,000 \text{ seconds} \approx 3 \text{ years}$$

to get a result. Using Algorithm 2 instead yields a running time of roughly

$$\frac{\sqrt{100\,000\,000\,000\,031} \text{ iterations}}{1\,000\,000\,\frac{\text{iterations}}{\text{second}}} \approx 10 \text{ seconds.}$$

Why we should care so much about efficiency can be illustrated even more impressively as follows. Assume that we run Algorithm 1 on a computer that is, say, 1 000 times faster than before. Then we get a time of

$$\frac{100\,000\,000\,000\,031 \text{ iterations}}{1\,000\,000\,000\,\frac{\text{iterations}}{\text{second}}} \approx 100\,000 \text{ seconds} \approx 1 \text{ day and } 3 \text{ hours },$$

which is therefore still a lot more than what Algorithm 2 takes on the much slower computer.

We can also do it the other way around. Suppose we have 10 minutes to spend, and ask what the maximum size is of a number $\mathtt{y}$ we can test. Using Algorithm 1, we obtain

$$\frac{\mathtt{y} \text{ iterations}}{1\,000\,000\,\frac{\text{iterations}}{\text{second}}} \le 600 \text{ seconds },$$

which, solving for $\mathtt{y}$, gives $\mathtt{y} \le 600\,000\,000$. Using Algorithm 2, we get

$$\frac{\sqrt{\mathtt{y}} \text{ iterations}}{1\,000\,000\,\frac{\text{iterations}}{\text{second}}} \le 600 \text{ seconds },$$

and hence we can test numbers `y` up to $600\,000\,000^2 = 3\,600\,000\,000\,000\,000\,000$. Given that we motivate Algorithm 2 through a time bound based on the number `x` itself rather than its length, it might not immediately be clear why one should use the input length as a measurement for the time complexity at all. Two related problems, however, can help in this matter. Determining whether a number is divisible by 2 has constant complexity as it does not depend on the length of the input at all. Divisibility by 3, on the other hand, is tested by summing over the digits of the input number, resulting in linear running time. In the context of these examples, time complexity based on the number of (binary) digits becomes quite natural.

## 3. Best-Case and Worst-Case Analysis

Next, we present Algorithm 3 to the students as a variation of Algorithm 2, but instead of terminating right after finding a divisor of `x` (performing an "early exit"), it sets a Boolean variable `isprime` to False, which was previously initialized with True. We ask which of these two algorithms is "better"? Although probably everyone would agree that Algorithm 2 is superior to Algorithm 3 since the latter may perform unnecessary operations, the answer to the question is actually not that easy. The reason is that there are different ways to analyze the time complexity of algorithms. If Algorithm 3 is executed with `x` = 100 000, its `while`-loop is executed roughly 100 000 times, whereas Algorithm 2 terminates at the very beginning, namely after finding out that `x` is even. However, if `x` is prime, both implementations take roughly the same time. Thus, we discuss with the students that, in order to answer whether one of the algorithms is "better," we first need to fix what kinds of inputs we look at, which we describe as follows.

- **Best-case analysis.** Here, we analyze the given algorithm's time complexity on inputs of given length that are in a sense as "favorable" as possible. For Algorithm 2,

**Algorithm 3.** Determining faster whether a given number is prime (II)

```
def prime(x):
    divisor = 2
    isprime = True
    while divisor <= sqrt(x):
        rest = x % divisor
        if (rest == 0):
            isprime = False
    if (isprime):
        print "Prime."
    else:
        print "Composite."
```

this would be numbers with a divisor of $2$. If Algorithm 3 is given such a number of length $n$, it still executes the body of its `while`-loop roughly $1.414^n$ times. Actually, this is the case for every input of length $n$.

- **Worst-case analysis.** In this case, we analyze the "least favorable" instances, i.e., those that make the algorithm run as long as possible. For Algorithm 2, these are prime numbers or squares of prime numbers; in both cases, the `while`-loop is executed roughly $1.414^n$ times, which is now the same time Algorithm 3 takes.
- **Average-case analysis.** We can also analyze the behavior of the algorithms on average. Since, for Algorithm 3, the time complexity is the same for every input of a given length, this is both easy and not very meaningful. Conversely, the behavior of Algorithm 2 is different on different inputs. However, here, we would first have to fix what we mean by "average." We could, e.g., assume that all inputs appear with the same probability. Then again, from a practical perspective, some inputs may be more likely than others. Therefore, it depends on the concrete environment in which the algorithm is executed to determine what a typical input looks like.

We usually design algorithms with their worst-case time complexity in mind; as noted above, with respect to this measure, Algorithms 2 and 3 are "equally good." Of course, we stress that implementing Algorithm 3 instead of Algorithm 2 is still a bad idea since Algorithm 2 is clearly faster in case of non-worst-case instances.

## 4. Uniform and Logarithmic Measurement

So far, we have only considered the number of executions of the `while`-loop when speaking about the algorithm's time complexity. As in the example above, we are interested in a function that describes how the running time grows with the input length $n$. Now we take a closer look at the work carried out by the algorithm, i.e., the number of *elementary instructions* within the loop. By this we mean arithmetic operations (treating addition and multiplication equally as one operation each) and, e.g., comparing two numbers. We briefly introduce the following two measurements on an informal level.

- **The uniform cost measurement.** When applying this measurement, we account cost $1$ to every elementary instruction carried out by the machine executing the algorithm. The above algorithms perform a number of instructions in every execution of the `while`-loop that does not depend on $n$; specifically, in every iteration, `divisor` is increased and compared with $\sqrt{\texttt{x}}$, `x` is divided by `divisor`, and then compared to $0$. We can therefore say that the number of computational steps is roughly $4 \cdot 1.414^n$ (plus a constant number of instructions at the beginning and the end of the program).
- **The logarithmic cost measurement.** The obvious problem with the uniform cost measurement is that it does not account for the sizes of the numbers that are in-

volved. To allow for a more accurate measurement, we can thus account a number of computational steps to instructions involving this number that is equal to its length. Since `x` is represented with $n$ bits and `divisor` increases until it is roughly $\sqrt{\texttt{x}}$ (which can be bounded by at most $n$ bits), the analysis yields roughly $4n \cdot 1.414^n$.

We conclude that the logarithmic cost measurement makes sense if the numbers considered are unbounded and may possibly be many times as large as the registers of the computer. In most of the examples presented later in class, such as sorting a sequence of given numbers, the input numbers can be represented with a few bits, and we will thus use the uniform cost measurement.

## 5. Big-$O$ Notation

Our last goal of the first lecture is to introduce big-$O$ notation on an informal level; the aim is not to give a mathematical rigorous definition. We again consider the aforementioned time complexity of roughly $4 \cdot 1.414^n$, and argue that, with growing $n$, $1.414^n$ is the dominant factor of the expression, and the constant $4$ becomes less and less significant with respect to its magnitude. The same is true for any other constant as well. What we do want to distinguish is whether the complexity grows, e.g., linearly, polynomially, or exponentially.

To this end, all functions that grow, say, almost quadratically, are contained in a set $\mathcal{O}(n^2)$. The important thing is for the students to really think of $\mathcal{O}(n^2)$ as an infinite set that contains, e.g., $1.52n^2$, $4n^2$, $20n^2$, or $1000n^2$. All these functions are contained in this set. Hence, we say "$56n^2$ is in $\mathcal{O}(n^2)$" and simply write $56n^2 \in \mathcal{O}(n^2)$, and so on. Furthermore, we observe that the function $n^2+n$ can be bounded from above by $2n^2$, and thus also $n^2 + n \in \mathcal{O}(n^2)$; likewise, $17n \leq 17n^2$ and thus $17n \in \mathcal{O}(n^2)$. The same is true for $2n^2 + 6n^{1.5}$, or $2.75n^2 + \sqrt{2}n + 9$, and so on. We explain that it suffices if this is true for sufficiently large $n$. Finally, we discuss that sets as above can be defined for other functions (which we always assume to be positive and monotonically increasing), giving a few examples.

It follows that the worst-case time complexity of Algorithm 1 is in $\mathcal{O}(2^n)$ with respect to the uniform cost measurement, and the complexity of Algorithms 2 and 3 is in $\mathcal{O}(1.414^n)$. The intuition is finally underpinned with a few more examples of different algorithms and their time complexity.

One way to conclude this lecture is to discuss the existence of randomized primality testing such as the algorithm of Solovay and Strassen (Hromkovič, 2008), or the deterministic AKS algorithm (Agrawal *et al.*, 2004), which achieves a running time in $\mathcal{O}(n^d)$, for a constant $d$.

## 6. Conclusion

We described an example of how to introduce the concepts of running time, best- and worst-case analysis, uniform and logarithmic cost measurement, and big-$O$ notation. To this end, we used the simple example of primality testing. We have so far tested this approach with different classes and settings, with the common property that the students had implemented algorithms before, but had no solid background in the formal analysis of algorithms.

Our experience is very positive. We found that, often, during presentation of our material, some students would quickly point out that the initial algorithm could be improved upon by focusing on odd numbers only, or even that testing possible divisors up to the square root would suffice. However, after having seen the context of the material, it became clear to these students that the goal of our class was not in finding the best solution for primality testing, but rather in discussing the basic concepts of complexity and what it means to improve an algorithm.

After this introductory lecture, the students were all able to use terms such as "asymptotic worst-case complexity" correctly on an intuitive level. Subsequent lectures discussed topics such as sorting or graph algorithms, with the students being able to understand and express the complexities of the different algorithms presented.

## References

Agrawal, M., Kayal, N., Saxena, N. (2004). PRIMES is in $\mathcal{P}$. *Annals of Mathematics*, 160(2),781–793.

Bell, T., Alexander, J., Freeman, I., Grimley, M. (2009). Computer science unplugged: school students doing real computing without computers. *NZ J. Appl. Comput. Inf. Tech.* 13(1), 20–29.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2009). *Introduction to Algorithms*, 3rd edition. The Mit Press.

Dagienė, V. (2010). Sustaining informatics education by contests. In: *Proceedings of the 4th International Conference on Informatics in Secondary Schools – Evolution and Perspectives: Teaching Fundamentals Concepts of Informatics*. 1–12.

Hromkovič, J. (2008). *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms*. Springer.

Kohn, T. (2017). *Teaching Python Programming to Novices: Addressing Misconceptions and Creating a Development Environment*. PhD Thesis, ETH Zürich.

O'Neill, M.E. (2009). The genuine sieve of eratosthenes. *Journal of Functional Programming*, 19(1), 95–106.

Sedgewick, R. (1992). *Algorithms in C++*. Addison-Wesley.

Sysło, M. (2011). Outreach to prospective informatics students. In: *Proceedings of the 5th International Conference on Informatics in Schools: Situation, Evolution and Perspectives*. 56–70.

Wing, J.M. (2006). Computational thinking. *Commun. ACM* 49(3), 33–35.

**D. Komm** is lecturer at ETH Zurich and an external lecturer at University of Zurich. He studied computer science at RWTH Aachen University and Queensland University of Technology. He received his PhD from ETH Zurich in 2012. His research interests focus on algorithmics and advice complexity.

**T. Kohn** is a PostDoc researcher in computer science education at ETH Zurich. The focus of his research is programming education, particularly in high schools. He holds an MSc in mathematics, and a PhD in computer science from ETH, and has been teaching mathematics and computer science for 10 years.

Olympiads in Informatics, 2017, Vol. 11, 87–92
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.07

# A Competitive Programming Approach to a University Introductory Algorithms Course

Antti LAAKSONEN
*Department of Computer Science*
*University of Helsinki*
*e-mail: ahslaaks@cs.helsinki.fi*

**Abstract.** This paper is based on our experiences on teaching a university introductory algorithms course using ideas from competitive programming. The problems are solved using a real programming language and automatically tested using a set of test cases. Like in programming contests, there are no hints and well-known problems are not used. The purpose of such problems, compared to traditional problems, is to better improve the problem solving skills of the students.

**Keywords:** algorithms course, competitive programming.

## 1. Introduction

This paper summarizes our experiences on teaching the *Data Structures and Algorithms* course at the University of Helsinki using ideas from competitive programming. The course deals with basic data structures and algorithms, such as trees, graphs and sorting. The course is a compulsory course for computer science students, and it is usually taken during the first year of studies. The course book is *Introduction to Algorithms* (Cormen *et al*., 2009), though the course covers only a part of the book.

For some years ago, the course was purely theoretic and algorithm design problems were solved using pen and paper and discussed in exercise sessions. However, it was observed that the learning results were not good and many students had difficulties in designing even very simple algorithms. After this, some ideas from competitive programming have been used on the course to improve the student's problem solving skills.

The *competitive programming approach* is based on the following ideas:

- The algorithm design problems are presented without hints and the solutions cannot be easily found using search engines.
- The solutions are implemented using a real programming language and automatically graded using a set of test cases.
- Solutions are given points only if they work correctly and efficiently, and techniques for testing and debugging are presented.

The above ideas have been used in programming contests for a long time, but we believe that they have potential to be used much more widely also on university algorithm courses. This does not mean that the courses should be *competitive*: only the problem types and the way the solutions are evaluated resemble the practices used in programming contests.

Of course, it is not a new idea to use automatic program evaluation in a university course. For example, García-Mateos *et al*. (2009) describe a programming course where the final exam is replaced with a series of programming contests using the Mooshak system, and Enström *et al*. (2011) present their experiences after using the Kattis system in several algorithm courses.

In this paper, we focus on the features of competitive programming style problems. First, we describe the way the competitive programming approach has been used on our course. After this, we discuss some of the advantages and disadvantages of the approach.

## 2. Course Organization

The course consists of 12 weeks. Every week there are two problem sets: a set of algorithm design problems and a set of other problems. The algorithm design problems follow the competitive programming approach: they are submitted in an online course system and evaluated automatically. The other problems include simulation problems and mathematical problems.

The competitive programming approach was introduced in the course in 2011. Initially, the DOMjudge system[1] was used to evaluate the solutions. However, this system was not optimal for the course, because it is intended for ICPC style contests and only shows if a solution is correct or not.

Since 2012, the TMC system (Pärtel *et al*., 2013) has been used on the course. This system is also used in the introductory programming courses at the University of Helsinki. TMC allows students to create their Java solutions in the NetBeans IDE and evaluates the solutions using JUnit tests. The tests are written by the course staff and can be downloaded using the TMC plugin.

The algorithm design problems are renewed every now and then so that the students do not get too familiar with them. For example, one of the first problems in the fall 2014 iteration of the course was as follows:

> Given a string, your task is to create a palindrome by removing exactly one letter from the string. For example, the string `ABCBXA` can be turned into a palindrome by removing the letter `X`.
>
> Your task is to implement the following method:
>
> `boolean almostPalindrome(String s)`
>
> The parameter `s` is a string that consists of at most $10^5$ letters. The method should return `true`, if it is possible to create a palindrome by removing exactly

[1] `https://www.domjudge.org/`

one letter, and `false` otherwise.

Time limit: 2 seconds.

The above format is used in all algorithm design problems. First there is a short problem statement, then a template for a Java method and an explanation what the method should do. In each problem, there is also a certain time limit for a single test case.

To get points for the problem, the student has to implement a method that corresponds to the problem statement and works efficiently in all test cases. An important detail in the above problem statement is that the string can be quite long ($10^5$ letters). Thus, the intended solution should work in $O(n)$ or $O(n \log n)$ time.

The benefit in using JUnit tests is that if the algorithm does not work correctly, the tests can give a friendly message to the student. For example, in the above problem, a message could be "The string `AAABAA` can be turned into a palindrome, but your method returned `false`." Depending on the problem configuration, the tests can be either fully or partially public.

## 3. Discussion

### 3.1. *Improving Problem Solving Skills*

Algorithm design problems are difficult, and it requires a great deal of work to improve one's problem solving skills. As algorithm design is difficult, it is a common practice to include *hints* in problems. For example, consider the following problem (Cormen *et al.*, 2009, page 42):

Give an algorithm that determines the number of inversions in any permutation on $n$ elements in $\Theta(n \lg n)$ worst-case time. (*Hint:* Modify merge sort.)

It is definitely easier to solve the above problem using the hint. However, at the same time, the hint almost completely spoils the problem, and after reading the hint, there is not much problem solving needed.

Such hints are never seen in competitive programming and there is a good reason for it: it is an important step in problem solving to consider different ideas how to approach the problem before finally finding a way to solve it. Using hints it may be possible to solve problems quickly, but this does not improve one's problem solving skills.

Still, it is clear that the above problem is difficult and without the hint, the problem would be impossible to solve for many students who are beginners in algorithm design. However, we do not think that presenting a hint is a good way to overcome this. If a problem is too difficult, it should be presented later when the students really can solve it.

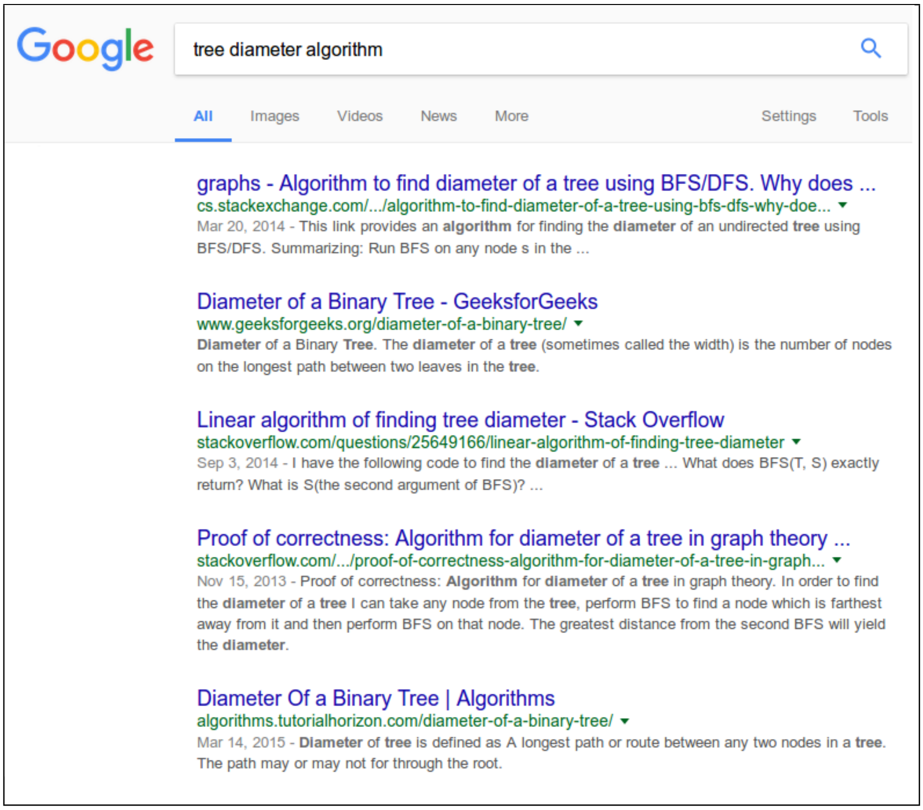
Thus, the challenge is to find problems that require problem solving but are not too difficult. Fortunately, the easiest problems in many programming contests have those properties. It is better to solve an easy problem using one's own skills than to solve a difficult problem using hints.

## 3.2. *Originality of Problems*

Another benefit in competitive programming problems is that they are – or should be – *original*. This ensures that it is not too easy to find solutions to problems just by using search engines. For example, consider the following problem (Cormen *et al.*, 2009, page 602):

> The *diameter* of a tree $T = (V, E)$ is defined as $\max_{u,v \in V} \delta(u, v)$, that is, the largest of all shortest-path distances in the tree. Give an efficient algorithm to compute the diameter of a tree, and analyze the running time of your algorithm.

A simple Google search (Fig. 1) can be used to find complete tutorials how to solve the problem. Of course, nobody is forced to use search engines to solve problems. However, in practice, many students do this, and this is very harmful to their problem solving skills. Problems like the tree diameter problem are good *examples* how to design algorithms, but they should not be used as course problems, because they are too well-known problems.



graphs - Algorithm to find diameter of a tree using BFS/DFS. Why does ...
cs.stackexchange.com/.../algorithm-to-find-diameter-of-a-tree-using-bfs-dfs-why-doe...
Mar 20, 2014 - This link provides an **algorithm** for finding the **diameter** of an undirected **tree** using BFS/DFS. Summarizing: Run BFS on any node s in the ...

Diameter of a Binary Tree - GeeksforGeeks
www.geeksforgeeks.org/diameter-of-a-binary-tree/
**Diameter** of a Binary **Tree**. The **diameter** of a **tree** (sometimes called the width) is the number of nodes on the longest path between two leaves in the **tree**.

Linear algorithm of finding tree diameter - Stack Overflow
stackoverflow.com/questions/25649166/linear-algorithm-of-finding-tree-diameter
Sep 3, 2014 - I have the following code to find the **diameter** of a **tree** ... What does BFS(T, S) exactly return? What is S(the second argument of BFS)? ...

Proof of correctness: Algorithm for diameter of a tree in graph theory ...
stackoverflow.com/.../proof-of-correctness-algorithm-for-diameter-of-a-tree-in-graph...
Nov 15, 2013 - Proof of correctness: **Algorithm** for **diameter** of a **tree** in graph theory. In order to find the **diameter** of a **tree** I can take any node from the **tree**, perform BFS to find a node which is farthest away from it and then perform BFS on that node. The greatest distance from the second BFS will yield the **diameter**.

Diameter Of a Binary Tree | Algorithms
algorithms.tutorialhorizon.com/diameter-of-a-binary-tree/
Mar 14, 2015 - **Diameter** of **tree** is defined as A longest path or route between any two nodes in a **tree**. The path may or may not for through the root.

Fig. 1. Solving the tree diameter problem using Google.

## 3.3. *Focusing on Correct and Efficient Algorithms*

The important question after designing an algorithm is: does the algorithm really work? One way to answer this question is to give a *proof* which shows that the algorithm works. However, it is not realistic to expect such proofs in an introductory algorithms course. Just giving a verbal description or a pseudocode of the algorithm is not satisfactory either, because it is easy to make wrong assumptions or skip important details.

In the competitive programming approach, all algorithms are *implemented* using a real programming language and the implementations are *tested* using a comprehensive set of test cases. This has two important benefits: the students have to precisely describe how their algorithms work, and after that, they will see if their algorithms are correct or not.

When learning to design algorithms, it is not only important to find correct algorithms to problems but also see why certain approaches do not work. In particular, it is easy to sketch intuitive greedy solutions to many problems, but such solutions often do not work in reality.

A side effect of the competitive programming approach is that it also improves programming, debugging and testing skills of the students. In competitive programming, a common way to find a bug in an algorithm is to use *stress testing*, which involves generating a large set of random test cases and checking if a brute force algorithm and an efficient algorithm always agree with each other. Many students are not familiar with this kind of comprehensive testing, even if they have attended courses on software engineering.

Another important factor is the efficiency of algorithms. By implementing and testing algorithms it is possible to see how efficient they are in reality and what is the connection between time complexities and real running times of algorithms. Moreover, it becomes evident which kind of optimizations are important. Typically, during the first weeks, the students try to improve their solutions using micro-optimizations without success, and later understand the importance of time complexities.

The responsibility of the course staff is to create challenging sets of test cases that ensure that accepted solutions are correct and efficient. This is sometimes difficult: for example, optimized brute force solutions can be surprisingly efficient. Still, if an algorithm is worth learning, it should be possible to find a test case where it works better than a brute force algorithm.

## 3.4. *Limits of Competitive Programming*

It is clear that some features of algorithms cannot be automatically tested using the competitive programming approach. For example, consider the following problem (Cormen *et al.*, 2009, p. 223):

> Describe an $O(n)$-time algorithm that, given a set $S$ of $n$ distinct numbers and a positive integer $k \leq n$, determines the $k$ numbers in $S$ that are closest to the median of $S$.

It is easy to solve the problem in $O(n \log n)$ time: just sort the array and take the $k$ middle elements. Designing an $O(n)$ algorithm is a more difficult task. Unfortunately, it is not possible to automatically test whether the time complexity of an algorithm is $O(n)$ or $O(n \log n)$ by measuring the running time because the difference may be very small. In general, logarithmic factors in time complexities cannot be reliably detected.

In addition, sometimes it may be a problem that it is possible to *guess* a solution to a problem and just test if it works. Of course this also happens in real programming contests. It is difficult to prevent this, but we do not think it is a problem in an introductory course. In fact, even in algorithm research, guessing is a valid way to design an algorithm, though it is required to later prove that the algorithm is correct.

## Acknowledgements

The author would like to thank Toni Annala, Matti Luukkainen, Pekka Mikkola and Kristiina Paloheimo for useful discussions.

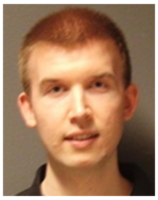
## References

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2009). *Introduction to Algorithms*. MIT Press.

Enström, E., Kreitz, G., Niemelä, F., Söderman, P., Kann, V. (2011). Five years with Kattis – using an automated assessment system in teaching. *IEEE Frontiers in Education Conference*.

García-Mateos, G., Fernández-Alemán, J.L. (2009). A course on algorithms and data structures using on-line judging. *ACM SIGCSE Bulletin*, 41(3), 45–49.

Pärtel, M., Luukkainen, M., Vihavainen, A., Vikberg, T. (2013). Test my code. *International Journal of Technology Enhanced Learning*, 5(3–4), 271–283.



**A. Laaksonen** received his PhD in Computer Science from the University of Helsinki. He is one of the organizers of the Finnish Olympiad in Informatics, and a coach and a team leader of Finnish BOI and IOI teams. He is the author of the book *Competitive Programmer's Handbook* and one of the developers of the CSES contest system.

Olympiads in Informatics, 2017, Vol. 11, 93–107
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.08

# On a Metodology for Creating School Curricula in Computing*

Krassimir MANEV[1], Neli MANEVA[2]
[1]*Department of Informatics, New Bulgarian University,*
*21, Montevideo str., 1618 Sofia, Bulgaria*
[2]*Bulgarian Academie of Science, Institute of Mathematics and Informatics*
*Acad. G. Bonchev Str., bl. 8, 1113 Sofia, Bulgaria*
*e-mail: kmanev@nbu.bg, neman@math.bas.bg*

**Abstract.** It is obvious that the scientific and practical human activity *domain* having computers as main objects – called further Computing – became crucial for development and well-being of the Humanity. That is why education in Computing has to find its adequate place in the curricula of the school all over the world. The paper proposes a methodology for development of school curricula in Computing. The main feature of this methodology is that it is extracted from the guidance for creating university curricula in Computing of the most respected professional associations in the domain – ACM and the CS section of IEEE. As a sample that illustrates application of the methodology, a part of a specific Curriculum is created.

**Keywords:** computing, education in Computing, curriculum, curricula development, methodology.

## 1. Introduction

As defined in The Glossary of Education Reform (Glossary, 2017): “The term ***curriculum*** refers to the lessons and academic content taught in a school or in a specific course or program“. Very frequently the word is used to denote just the list of courses offered by a school, but such understanding of the term is more appropriate for some external use – to demonstrate briefly to the community or some institutions the essence of the proposed education. Here we will consider the notion in its depth. As the above mentioned Glossary underlines: “… curriculum typically refers to the knowledge and skills students are expected to learn …” which includes:

- The ***learning standards*** or ***learning objectives*** the students are expected to meet.
- The ***units*** and ***lessons*** that teachers teach.
- The ***assignments*** and ***projects*** given to students.

* This work was a result from the discussions during the IOI Workshop dedicated to teaching of Informatics in secondary schools, held in Bitola, Macedonia, April 2015.

- The ***learning resources*** – books, materials, videos, presentations and readings, used in a course.
- The ***assessment resources*** – tests, tasks, projects and other materials used to evaluate student's learning.

The authors of Glossary go beyond the above mentioned items and append that as inevitable parts of the modern curriculum have to be considered also:

- ***Curriculum philosophy***, i.e. the educational philosophy of the institution which developed it.
- ***Curriculum packages*** that have been developed by an outside organization as an example of usage of best practices in the domain.
- ***Curriculum standardization*** dedicated to increase teaching quality with greater curricular consistency throughout schools, regions and states.

In this paper we would like to propose an idea for methodology for development of curricula for the schools in the very large domain of the science and the technologies, which is dedicated to computers, their programming and usage called below *Computing*. These ideas raised during the IOI Workshop, held in Bitola, Macedonia, in April 2015 (called briefly *the Workshop*). Its main goal was to resume the current state of the teaching Computing in schools of the presented countries and to propose some ideas for development of an International School Curriculum in Computing (ISCC) on the base of the experience of the IOI community.

In Chapter 2 we briefly describe the status quo of teaching Computing in the secondary schools on the base of the presentations of the participating in the Workshop countries. We resume the results of some well-known efforts for creating an ISCC for schools over the world and define the goal of the paper – to transform the methodology of ACM-IEEE for creating university curricula in Computing into an agile methodology for creating curricula for the primary and secondary schools. In Chapter 3 we present the methodology of ACM-IEEE for development of university curricula. The transformed methodology is given in Chapter 4. Chapter 5 contains a sample of applying the methodology for development of a part of a specific curriculum. In Chapter 6 there are conclusion and perspectives for future work on the methodology.

## 2. Teaching Computing in Secondary Schools

The outcomes of the Workshop were presented in (Ackovska *et al.*, 2015), so let us briefly resume the status quo of education in Computing, as shown in the presentations of the participating 12 countries: Belgium, Bolivia, Brazil, Bulgaria, Croatia, Estonia, Hungary, Lithuania, Macedonia, New Zealand, Serbia and Slovenia. The dominating conclusion is that no one of the participating countries has officially approved national curriculum for teaching Informatics and only few of the countries have some curriculum for teaching Information Technologies. Compulsory courses in programming are given in some special (math and science oriented) schools. Because all participating in the Workshop countries are IOI members more serious teaching of Informatics (Programming, Data Structures and Algorithms) is proposed for pupils that take part in Olympiads in Informatics and other programming contests.

Having as one of the objectives of the Workshop the discussion of the possibilities for creation an international curriculum for primary and high school for teaching the basics of Computing, some existing attempts for elaboration of such curricula as well as guidelines for elaboration of such curricula were considered.

It is important to mention here the efforts of the colleagues from ACM. They started long years ago to discuss teaching of Computing (ACM Curriculum Committee, 1968). But since 2001 their activity become persistent together with colleagues from CS section of IEEE. And the goal is not just to create a curriculum but a **methodology for elaboration university level curricula** in all area of the domain that is linked to creating, programming and using computers, called here ***Computing*** (Computing Curricula, 2001; Computer Science Curricula, 2013; Information Technology, 2008). This significant work is in the base of this paper.

Working on this paper we considered also some available official and stable curricula for primary and secondary school – recommendations of Computing at School Working Group for United Kindom, endorsed by British Computer Society, Microsoft, Google and Intellect - UK trade body for the hi-tech sector, (Computer Science, 2012), the Australian curriculum guidance (Information and communication technology, date of publishing is unavailable), the USA recommendation K-12 (A Model Curriculum, 2003), the recent K-12 CS framework () and the Russian Secondary school Curriculum in Informatics (Kiryukhin and Tsvetkova, 2016). We do not use them in this paper but if the proposed here methodology is adopted then some ideas from the mentioned curricula will be very helpful. In this work we also rely on our more than 15 years experience as authors and co-authors of textbooks for the universities, as well as for Bulgarian primary and secondary school Curriculum in Computing, which is too dynamic, to be referred here.

After presenting the national reports of the participants in the Workshop a natural question raised: *Is it possible to have universal and unique ISCC?* The opinion of the most of the participants was that **it seems impossible**. Some arguments for this are listed below:

- The general *educational structure* of different countries presented in the Workshop is very different! We expect that if a larger number of countries decide to apply such ISCC then the differences will be even more drastic.
- The Countries have various history of introducing education in Computing and so – different level of experience in teaching these disciplines, different traditions, different quantity and quality of teachers, different computing resources, etc.
- The specific structure of the economy of the countries supposes different *models* of school education in Computing. For example, some countries with developed software industries will need a model which is different from the model of countries in which there is no such industry and there no intentions to develop it.

That is why it seems more realistic that **not a single Curriculum but many different Curricula** have to be created even in one specific country.

The question is: are we able to predict what exactly will be necessary for education in Computing in each country, depending of its specific needs and to predefine some fixed number of Curricula, developing them in depth. The answer again is – rather not! That is why it seems more appropriate for our goals a **set of *curriculum elements*** to be developed as well as ***guidance*** for using these elements in order to create many specific curricula.

Good example for such kind of activity are the mentioned above efforts of the professional organizations ACM and IEEE-CS to elaborate and maintain during the years a system of curriculum elements and guidance for creating specific curricula in Computing for the university stage of education. That is why these efforts are resumed in the next chapter as well as a proposal how the methodology of ACM-IEEE curricula guidance group could be tuned for our goals.

## 3. ACM-IEEE Computing Curricula Guidance

ACM-IEEE Computing Curricula Guidance is not a single methodical act. It has about 20 years history. It is a persistent work of the ACM-IEEE Computing Curricula Guidance work group, which started with a single recommendations updated on a regular base through the year that recently led to specific recommendation for the different fields of the Computing domain. That is why this important work could be a model for creating and future maintenance of corresponding guidance for development of the ISCC.

### 3.1. *Principles*

The ACM-IEEE curriculum guidance work groups are based on some principles, that seem to be applicable to our goals:

- Curriculum guidance should not only identify the **fundamental knowledge** and **skills** of the domain but should provide a **methodology** for creating specific courses, defining their content and the appropriate order of teaching them.
- The required body of knowledge must be **as small as possible**.
- Curriculum guidance must strive to be **international** in scope, **broadly based** and must include **professional practice** as an integral component.
- The rapid evolution of Computing requires an **ongoing review** of the corresponding curricula recommendations through the years.
- Curriculum guidance must be **sensitive to progress** in technology, pedagogy and lifelong learning.

### 3.2. *Structure*

The ACM-IEEE Computing Curricula Guidance has the following structure:

- *Computing* is a broad, scientific and practical human activity *domain* including a corpus of *teaching elements* – both theoretical (knowledge) and practical (skills), having computers as main objects – their creation as well as their programming and usage.

- *Body of knowledge* of the domain describes it‘s fundamental concepts, theories and notions. *Core of knowledge* specifies the body of knowledge elements, for which there exists a broad consensus that they are essential for the education in the domain. Body of knowledge is hierarchically structured in *fields*, *areas*, *units* and *topics*.
- *Learning objectives* are composed of two parts – a set of requirements that all students should be able to meet and set of requirements to promote individual assessment of each student achievements.
- *Curriculum models* present different approaches for organizing the educational process among which the creator of a specific Curriculum could choose. ACM-IEEE guidance considers three level of models – *introductory*, *intermediate*, and *advanced*.
- The part *Course descriptions* contains detailed description of the *courses* – both compulsory and elective. Course is the main structural object of each curriculum. One discipline could be covered by one or more courses and one course could be dedicated to one or more disciplines.

Some other parts of the guidance structure could be also considered on the next stages of development of guidance for ISCC.

## 3.3. *Body of Knowledge*

*Body of knowledge* of the Computing domain is hierarchically organized in four levels. Level 1 contains the *fields* of the domain; on Level 2 the fields are divided into *areas*; on Level 3 each area is divided to *units;* on Level 4 each unit is composed from *individual topics*.

ACM-IEEE group considered 5 **fields**: *Computer Engineering*, *Computer Science* (CS), *Information Systems*, *Software Engineering* and *Information Technologies* (IT).

For example, the field CS is composed of the following **areas**: Discrete Structures, Programming Fundamentals, Algorithms and Complexity, Architecture and Organization, Operating Systems, Net-Centric Computing, Programming Languages, Human-Computer Interaction, Graphics and Visual Computing, Intelligent Systems, Information Management, Social and Professional Issues, Software Engineering, Numerical Methods. The field of IT includes the **areas**: Information Technology Fundamentals, Human Computer Interaction, Information Assurance and Security, Information Management, Networking, Programming Fundamentals, Systems Administration and Maintenance, Social and Professional Issues, Web Systems and Technologies, etc.

As a first example we would like to mention the unit Discrete structures, which is the mathematical foundation of the domain. It includes the following units:

- Sets, relations, and functions (6).
- Basic logic (10).
- Proof techniques (12).
- Basics of counting (5).

- Graphs and trees (4).
- Discrete probability (6).

The underlying of a unit means that it is included in the Core of the corresponding area. In this case all items in both units are in the Core of the area. The numbers in brackets are the predicted in the ACM-IEEE guidance class hours.

As a second example let us to consider the **units** that ACM-IEEE guidance includes in the area Programming Fundamentals, common for the fields Computer Science and Information Technologies:

- Fundamental programming constructs (9/10).
- Algorithms and problem-solving (6/6).
- Fundamental data structures (14/10).
- Recursion (5/0).
- Object-Oriented Programming (0/9).
- Event-driven programming (4/3).

The numbers in brackets, separated by slash in this case are the predicted in the ACM-IEEE guidance class hours for the area when included in CS or IT field, respectively. As it could be seen the very small differences in the two areas are in the different number of class hours of the same units and that the unit Recursion is included in one of the area instead the unit OOP in the other. In this case all items in both units are in the Core of the area too, but it is not the case in all areas.

For each unit the guidance of ACM-IEEE contains description of the individual **topics** included in the unit and the corresponding learning objectives. As example, these topics for the unit Sets, relations and functions of the area Discrete structures are:

- Sets (Venn diagrams; Union, intersection, complement, Cartesian product, Power set; Cardinality of finite sets).
- Relations (Reflexivity, symmetry, transitivity; Relations of equivalence and Partial orders).
- Functions (Surjections, injections, bijection; Inverses; Composition).

## 3.4. *Learning Objectives*

Learning objectives are less formal but important part of the guidance in the structure of created curricula. In narrative form they not only show the vision of the creator of the curricula for achievement of the students but are the media for unifying the topics of the unit in something complete. Learning objectives of the topic Sets, relations and functions of the unit Discrete structures could be:

- Explain with examples the basic terminology of sets, relations, and functions.
- Perform the operations associated with sets, relations, and functions.
- Relate practical examples to the appropriate set, or relation or function model, and interpret the associated operations and terminology in the context.
- Demonstrate basic counting principles, including uses of diagonalization and the Pigeonhole principle.

## 3.5. *Models*

The different *models* in ACM-IEEE curriculum guidance are strongly connected to the university degree of education. A model is some leading approach of organizing of the teaching material, i.e. splitting the core units in courses, ordering the courses in time etc. The model includes different quantitative parameters of the specific university – how many semesters covers the curriculum, how many weeks per semester, how many class hours per week, the ratio of classes for lectures toward classes for practice, the ratio of classes for compulsory courses toward elective courses, etc.

But the model includes some other parameters, too. For example, on the introductory level, different models could be identified depending of the preferred paradigm of the introductory programming course: Imperative language first, Object-oriented language first or Functional language first. The chapter of ACM-IEEE Curricula Guidance that discusses models contains some other organizing concepts and could be used for some helpful ideas.

## 3.6. *Courses Creation*

When all curriculum elements are defined and the model fixed, the last step is creation of the set of courses. For this purpose a *decision table* is created. Its rows are labeled with the units (or topics if the curriculum has to be more detailed) and columns labeled with the courses chosen by the requirements of the model. The next step is to decide which core element (unit or topic) in which course will be included, with how many class hours, and how the elements will be ordered inside the course. The final step is to choose distribution of identified courses in semesters. A part of such decision table of covering some of the elements by some courses for the model Imperative language first from ACM-IEEE Curricula Guidaence is shown on Fig. 1.

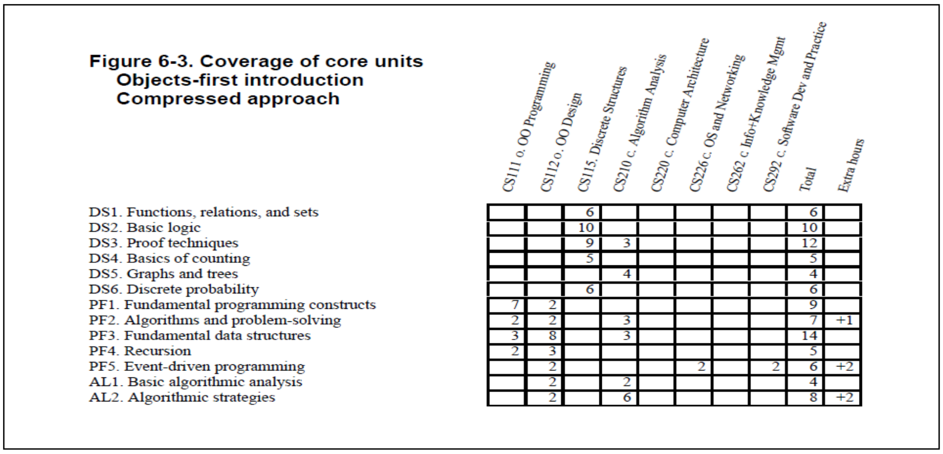


**Figure 6-3. Coverage of core units**
**Objects-first introduction**
**Compressed approach**

| | CS111 o. OO Programming | CS112 o. OO Design | CS115. Discrete Structures | CS210 c. Algorithm Analysis | CS220 c. Computer Architecture | CS226 c. OS and Networking | CS262 c. Info+Knowledge Mgmt | CS292 c. Software Dev and Practice | Total | Extra hours |
|---|---|---|---|---|---|---|---|---|---|---|
| DS1. Functions, relations, and sets | | | 6 | | | | | | 6 | |
| DS2. Basic logic | | | 10 | | | | | | 10 | |
| DS3. Proof techniques | | | 9 | 3 | | | | | 12 | |
| DS4. Basics of counting | | | 5 | | | | | | 5 | |
| DS5. Graphs and trees | | | | 4 | | | | | 4 | |
| DS6. Discrete probability | | | 6 | | | | | | 6 | |
| PF1. Fundamental programming constructs | 7 | 2 | | | | | | | 9 | |
| PF2. Algorithms and problem-solving | 2 | 2 | | 3 | | | | | 7 | +1 |
| PF3. Fundamental data structures | 3 | 8 | | 3 | | | | | 14 | |
| PF4. Recursion | 2 | 3 | | | | | | | 5 | |
| PF5. Event-driven programming | | 2 | | | | 2 | | 2 | 6 | +2 |
| AL1. Basic algorithmic analysis | | 2 | | 2 | | | | | 4 | |
| AL2. Algorithmic strategies | | 2 | | 6 | | | | | 8 | +2 |

Figure 1. Decision table for creating courses (Computing Curricula, 2001).

## 4. Implementing the ACM-IEEE Methodology

### 4.1. *Body of Knowledge*

Creating a methodology for development of different curricula, our efforts will be concentrated especially on fields Computer Science and Information Technologies. We consider teaching Computing Systems, Information System and Software Engineering not appropriate for the school students because these fields require maturity of higher level and some specific knowledge – electronic engineering, business management and software business management, which are more appropriate for universities. But some knowledge elements from these fields could find place in our guidance, too.

We propose to use the body of knowledge from the version of ACM-IEEE guidance from 2001 and, after a broad discussion/questioning in IOI community and outside, to extract the areas and the units that are appropriate for the school. It will be necessary for sure to append some units or/and topics that are considered nowadays not appropriate or not necessary for the universities, but are important for school students. As the presentation of the countries that participated in the Workshop shows, there is some corpus of knowledge in the area IT (let us call it *computing literacy*) that is traditionally studied in schools, but not as university program, and have to find place in ISCC. The units of core of knowledge have to be identified as a result of the discussion/questioning mentioned above. Further we can choose an appropriate quantity of class hours for each unit.

A similar process of deleting/appending has to be done for the individual topics of each included in the body unit (core or elective). In parallel, the corresponding educational objectives have to be edited with the respect of the age of the student. It is quite possible that in our recommendations we will have to define few alternative versions of the learning objectives connected with each individual topic, unit and even area, depending on the chosen educational model.

### 4.2. *Educational Model*

Identifying the necessary educational model in our case will be the most important and difficult task. As mentioned above the model is the element of each curriculum that has to introduce a flexibility in the process of creating curricula. This will be our instrument to cover as many as possible concepts of organization of education on different age levels of the school, as well as of the different kind of schools.

Let us start with the levels of education which exist in the educational system of each country (and could be different). In most countries there are three levels in secondary school – *primary, intermediate* and *high*. In Bulgarian system, for example, two models exist for the level of education:

- 1–4 grades as primary, 5–8 grades as intermediate, 9–12 grades as high.
- 1–4 grades as primary, 5–7, grades as intermediate, 8–12 grades as high.

The trend is the first of them to be eliminated soon or later. There is also trend to separate the high level to two sublevels 8–10 grades and 11–12 grades in order to give a possibility for some specialization in the sublevel 11–12 of the students. Similar subdivision is predicted in (Computer Science, 2012).

In addition we have to consider also existence of some special schools – mathematical, language, art, sport, professional, etc. That is why it is necessary to define very carefully the core units of the body of knowledge which to be common for the national educational system in order to give to each student, graduated in secondary school the possibility to continue her/his education in an university program of Computing does not matter in what kind of school is graduated. As well as to classify the elective units of the body in such a way that the curricula for the specialized schools to include the most appropriate for the peculiarities of the school elective units.

In Bulgarian system, for example, the courses are classified in 3 categories – *compulsory*, *compulsory-elective* (which means that the students have to take *m* among the proposed set of *n* such courses) and *free-elective* (which means that students are not obliged at all but could take as many such courses as they would like). In last category the corresponding school could include any course which is appropriate to the profile of the school (professional school of machine engineering, for example, could include learning of one CAD/CAM system, which is inappropriate for other schools).

Is it possible to define different kind of models also on the base of preferred main fields – *CS-oriented* (most appropriate for mathematical and engineering schools), *IT-oriented* (more appropriated for language, art and sport schools), *CS&IT-oriented* (more appropriated for regular schools), etc.

The most important difference between university models (we mean the *classic universities* but not the so called *liberal art*, where the educational model is more close to the models of secondary schools) and the school models is that in the university programs in the domain of Computing in each moment students take few courses from the domain. In secondary school model we will have in each moment only one or maximum two courses in the domain with 1-4 class hours per week – asking for more class hours for Computing nowadays seems not realistic. So the model has to include some „class hours per week“ *scheme*, which defines the grade, number of courses, class hours per week and distribution of the class hours between the two courses (or between IT and CS if the model includes only one course).

## 4.3. *Creating Specific Courses*

When the body of knoledge and the model are fixed this stage will be relatively easy. Because we supposed that the curricula will have usually 1 course per school year or maximum 2. That is why we propose the corresponding decision table to be composed of rows labeled with grades and columns labeled with units. If necessary the columns will be distributed between the courses when the model predicts two.

## 5. A Sample for Implementing the Methodology

In this Chapter we apply the proposed methodology for creating part of a possible School Computing Curriculum. In the sample we use also our experience of long years writing textbooks in *Informatics* (this is the traditional name of the course corresponding to CS) and IT for the Bulgarian schools, obeyng to set of so called *learning programs* (separate program for each grade‘s course), composition of which is not really curriculum regarding used in this paper notion. The principles and requirements of these learning programs could not be neglected, because they are mandatory for all schools, and to the authors of textbooks, respectively. That is why applying the proposed methodology we have to keep in mind the corresponding learning programs, too.

Traditionally for the Bulgarian schools (and, probably, the same is in many other countries) the teaching elements in the Learning programs in Computing are from two main areas from ACM-IEEE Curricula Guidance – Computer science and Information Technologies. That is why we will create our sample based on this two areas too.

### 5.1. *Model*

Our model will be dedicated for the regular schools. That is why it will be of type CS-IT oriented with 4 levels – primary level (1-4 degrees), intermediate level (4-7 degrees) and two high levels – basic high level (8-10 degrees) and advanced high level (11-12 degrees). The school year in the primary level will be 32 weeks long, 34 weeks long in the intermediate level and 36 week in the high levels.

There will be a single compulsory course per year named *Information Technologies* in the primary and intermediate level, single course per year called *Informatics and Information Technologies* for the basic high level and two courses for the advanced high level – *Informatics* and *Information Technologies*.

The number of class hours will be: 1 class hour per week for the primary level, 2 class hours per week for the intermediate level, 3 class hours per week for the basic high level and 4 class hours per week for the advanced high level – 3 for the course Informatics and 1 for the course Information technologies. The 108 class hours in the basic high level will be divided to 36 hours for Information Technologies and 72 hours for Informatics. The 144 class hours in the advanced level will be divided to 36 hours for Information Technologies and 108 hours for Informatics. I.e. our model predict 540 class hours in Information Technologies and 432 class hours in Informatics.

In addition to the single compulsory course a single compulsory-elective course named *Application of Information Technologies* will be specified for the intermediate level. A single compulsory-elective course named *Application of Informatics and Information Technologies* will be specified for the high level also. Both compulsory-elective courses will be with 2 class hours per week. The distribution of classes between Applications of Informatics and Applications of Information Technologies will be assigned to the administration of the schools.

## 5.2. *Body of Knowledge and Learning Objectives*

It is obvious that creating a complete Computing curricula is not possible in the volume of a journal paper. We will demonstrate here the application of the methodology only to some parts of the curriculum for the described above model. As mentioned above the body of knowledge for the created curriculum will be chosen mainly from the fields Computer Science and Information Technology, but for illustration we will develop only the Computer Science part of the body. We will use as a start point (Computing Curricula, 2001) having in mind that the last version is less appropriate for schools because it contains some new conceptions, which are most appropriate for the university programs.

According (Computing Curricula, 2001) the areas of the field Computer Science are: Discrete Structures, Programming Fundamentals, Algorithms and Complexity, Architecture and Organization, Operating Systems, Net-Centric Computing, Programming Languages, Human-Computer Interaction, Graphics and Visual Computing, Intelligent Systems, Information Management, Social and Professional Issues, Software Engineering, and Computational Science. The first task is to decide which areas are appropriate for the school and to distribute the 432 hours predicted by the model among the chosen areas.

For this purpose we consider the table from Fig. 5-1, page 17 in (Computing Curricula, 2001). This is the place where as broad as possible consensus is necessary in order to decide which are the areas which are appropriate for the school education in Computing and which are the units in each area that will be included in the Curriculum.

Modeling the work of the experts we proceeded in following way: we deleted from the table the units that we consider inappropriate for the school and then deleted the areas for which all units were deleted. Doing this we selected only the core units and distributed predicted by the model 432 hours among these units. The results of this stage are presented in the leftmost column of Table 1.

Table 1

Decision table for the Curriculum

| | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|
| DS. Discrete Structures (46 hours) | | | | | |
| DS1. Functions, relations, and sets (10) | 10 | | | | |
| DS3. Proof techniques (10) | | 10 | | | |
| DS4. Basics of counting (12) | | | | 12 | |
| DS5. Graphs and trees (14) | | 4 | 10 | | |
| PF. Programming Fundamentals (88 hours) | | | | | |
| PF1. Fundamental programming constructs (14) | 10 | 4 | | | |
| PF2. Algorithms and problem-solving (14) | 6 | 8 | | | |
| PF3. Fundamental data structures (30) | 10 | 10 | 10 | | |
| PF4. Recursion (12) | | | | 6 | 6 |
| PF5. Event-driven programming (18) | 6 | 6 | 6 | | |

To be continued

Continuation of Table 1

| | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|
| AL. Algorithms and Complexity (24 hours) | | | | | |
| AL1. Basic algorithmic analysis (4) | | 2 | 2 | | |
| AL2. Algorithmic strategies (6) | | 2 | 4 | | |
| AL3. Fundamental computing algorithms (12) | | | 12 | | |
| AL6. The complexity classes P and NP (2) | | | 2 | | |
| AR. Architecture and Organization (29 hours) | | | | | |
| AR1. Digital logic and digital systems (6) | | | | 6 | |
| AR2. Machine level representation of data (2) | 2 | | | | |
| AR3. Assembly level machine organization (9) | | | | 9 | |
| AR9. Architecture for networks (12) | | | | 12 | |
| OS. Operating Systems (18 hours) | | | | | |
| OS1. Overview of operating systems (2) | 2 | | | | |
| OS2. Operating system principles (2) | | 2 | | | |
| OS6. Device management (4) | | 4 | | | |
| OS8. File systems (4) | 2 | 2 | | | |
| OS12. Scripting (6) | | | 4 | 2 | |
| NC. Net-Centric Computing (33 hours) | | | | | |
| NC1. Introduction to net-centric computing (3) | | | | 3 | |
| NC2. Communication and networking (9) | | | | 9 | |
| NC4. The web as client-server computing (6) | | | | 6 | |
| NC5. Building web applications (6) | | | | 6 | |
| NC9. Wireless and mobile computing (9) | | | | 9 | |
| PL. Programming Languages (42 core hours) | | | | | |
| PL1. Overview of programming languages (2) | | | | 2 | |
| PL2. Virtual machines (2) | | | | 2 | |
| PL3. Introduction to language translation (2) | | | | 2 | |
| PL4. Declarations and types (12) | | | | | 12 |
| PL5. Abstraction mechanisms (6) | | | | | 6 |
| PL6. Object-oriented programming (18) | | | | | 18 |
| HC. Human-Computer Interaction (44 hours) | | | | | |
| HC1. Foundations of human-computer interaction (12) | 4 | | | | |
| HC2. Building a simple graphical user interface (12) | 4 | 8 | | | |
| HC5. Graphical user-interface design (10) | | | 10 | | |
| HC6. Graphical user-interface programming (10) | | | | 10 | |
| GV. Graphics and Visual Computing (20 hours) | | | | | |
| GV1. Fundamental techniques in graphics (8) | 6 | 2 | | | |
| GV2. Graphic systems (12) | 4 | 4 | 4 | | |
| IS. Intelligent Systems (not included) | | | | | |
| IM. Information Management (50 hours) | | | | | |
| IM1. Information models and systems (3) | | | | | 3 |
| IM2. Database systems (3) | | | | | 3 |
| IM4. Relational databases (4) | | | | | 3 |
| IM5. Database query languages (10) | | | | | 9 |
| IM6. Relational database design (30) | | | | | 30 |

To be continued

| | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|
| SP. Social and Professional Issues (15 core hours) | | | | | |
| SP1. History of computing (2) | 2 | | | | |
| SP2. Social context of computing (3) | | | | | 3 |
| SP4. Professional and ethical responsibilities (3) | | | | 3 | |
| SP5. Risks and liabilities of computer-based systems (2) | | 2 | | | |
| SP6. Intellectual property (3) | | | | | 3 |
| SP7. Privacy and civil liberties (2) | | | 2 | | |
| SE. Software Engineering (33 core hours) | | | | | |
| SE1. Software design (10) | | 2 | 2 | 3 | 3 |
| SE2. Using APIs (7) | | | 4 | 3 | |
| SE3. Software tools and environments (2) | 2 | | | | |
| SE6. Software validation (8) | 2 | | | 3 | 3 |
| SE8. Software project management (3) | | | | | 3 |
| SE9. Component-based computing (3) | | | | | 3 |
| CN. Computational Science (not included) | | | | | |
| | 72 | 72 | 72 | 108 | 108 |

## 5.3. *Course Creating*

The last stage of the Curriculum development is the construction of courses. Here the different developers could take into account their national traditions and concepts for the organization of the knowledge body. But the concept of arranging the material in its internal logic and with increasing the level of difficulty is inevitable.

In our decision table (rightmost 5 columns of Table 1) only the core items of the body of knowledge are included. Similar work has to be done for the elective items when the model is extended with the corresponding compulsory-elective and free-elective courses.

Additional information that is missing in our decision table and has to be appended, is the recommendation how the predicted hour classes to be distributed between lectures (theoretic classes) and exercises (practice classes). It is clear that the most of exercises will be in the topics dedicated to programming but the creator of the Curriculum has to fix carefully the ratio. Without some efforts to build the necessary practical skills the education in Computing is nonsense.

## 6. Conclusion

In this paper a methodology for creating school curricula in Computing was presented. The main feature of this methodology is that it is extracted from the methodology for creating university curricula in Computing of the most respected professional associations in the domain – ACM and the CS section of IEEE. There is no doubt that teaching

Computing has not the place it deserved in schools over the world but this could not continue infinitely. Computing is a fundamentally important for the development of the humanity, as well as the languages, mathematics and philosophy. So the Computing community and the IOI community as a part of Computing community is obliged to answer the questions: what part of knowledge/skills of the domain to be taught/practices in the schools, when and how.

It is obvious that a single journal paper could not contain all aspects of such complex activity as curricula development. For implementing the proposed here ideas a large amount of work has to be done. On the first place a broad discussion is necessary for selecting the items of the body of knowledge (both core and elective) that have to be included. The expertize of two professionals is not enough at all.

On the second place, the last level of the body of knowledge hierarchy – the topics – has to be included in consideration. Only in such way the precise distribution of the class hours and precise ratio lectures/exercises could be estimated. The proposed in the sample distribution is rather subjective and so not precise.

So, the effective and helpful guidance for development of school curricula in Computing could be created only as a result of mentioned above broad discussion. We believe that IOI community could play and have to play significant role in this process. As IOI community includes high quality proffessionals – researchers, school teachers, and university professors from different arreas in the domain, who posses huge experience in preparing programmers. We believe that introducing adequate curricula in the schools will rise the level of education, will increase the number of students that compete and, finally, will prepare better the students with interest in the domain for the university level education and the future professional carrier, which is the main goal of the school olympiads in Informatics.

## References

Ackovska, N, Németh, Á.E., Stankov, E., Jovanov, M. (2015). Creating an International Informatics Curriculum for Primary and High School Education (Report of the IOI Workshop). *Olympiads in Informatics*, 9, 205–212.

ACM Curriculum Committee on Computer Science (1968). Curriculum 68: Recommendations for Academic Programs in Computer Science. *Communications of ACM,* 11(3), 151–197.

A Model Curriculum for K–12 Computer Science: Final Report of the ACM K–12 Task Force Curriculum Committee, Computer Science Teachers Association (2003). Downloaded at June 2017: `http://marvin.cs.uidaho.edu/Teaching/K12-CS/acmK12CurriculumFinal2003.pdf`

Glossary (2017). *The glossary of Education Reform*. Downloaded at June 2017: `http://edglossary.org/curriculum/`

*Computer Science: A Curriculum for Schools* (2012). Downloaded at June 2017: `http://www.computingatschool.org.uk/data/uploads/ComputingCurric.pdf`

*Computer Science Curricula* 2013 (2013). Downloaded at June, 2017: `https://www.acm.org/education/CS2013-final-report.pdf`

*Computing Curricula* (2001). Computer Science (2001). Downloaded at June, 2017: `http://www.acm.org/education/curric_vols/cc2001.pdf`

*Information and communication technology capability*, (date of publishing unavailable), downloaded at June 2017: `http://www.australiancurriculum.edu.au/generalcapabilities/information-and-communication-technology-capability/introduction/introduction`

*Information Technology* (2008). Downloaded at June, 2017: `https://www.acm.org/education/curricula/IT2008%20Curriculum.pdf`

*K-12 CS framework* (date of publishing unavailable), downloaded at June 2017: `https://k12cs.org`

Kiryukhin, V.M., Tsvetkova, M.S. (2016). Informatics at Russian Secondary School. *Olympiads in Informatics*, 10 (special issue), 135–24.

**Kr. Manev** is a professor of Discrete mathematics and Algorithms in New Bulgarian University, PhD in Computer Science. He has published about 75 scientific papers and more than 30 textbooks in the fields of Informatics and Information Technologies. Member of Bulgarian National Committee for Olympiads in Informatics since 1982 and President of NC from 1998 to 2002; member of the organizing team of IOI'1989 and IOI'1990; Chairmen of IOI'2009; Leader/ Deputy Leader of Bulgarian team for IOI in 1989, 1998, 1999, 2000, 2005 and 2014. From 2001 to 2003 and from 2011 to 2013 he was elected member of IC of IOI, since 2005 to 2010 represented in IC the Host country of IOI'2009. Now he is a President of IOI for the period 2014–2017.

**N. Maneva** is a professor, Ph.D. in Computer Science, from Institute of Mathematics and Informatics, Bulgarian Academy of Sciences. Her major fields of scientific research are Software Engineering, Software Quality Assurance, Model-driven Software Development and Formal Methods in Software Engineering. She has published about 70 scientific papers and more than 30 textbooks in the field of Informatics and Information technology. She was a member of Bulgarian National Committee for Olympiads in Informatics from 1982 till 1993 and a Secretary of the Scientific Committee of IOI'1989..

Olympiads in Informatics, 2017, Vol. 11, 109–121
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.09

# Olympiads in Informatics as a Mechanism of Training World-Class Professionals in ICT

Oksana PAVLOVA, Elena YANOVA
*ITMO University, Saint-Petersburg, Russia*
*e-mail: pavlova.ifmo@gmail.com, yanova.ea@gmail.com*

**Abstract.** The development of information technologies has not brought only positive aspects to the society, but also negative ones: youth isolation, Internet addiction, inability to express thoughts logically, poor physical health, etc. Such problems impede the process of youth development and making professionals of them. But Olympiads in Informatics and Programming can turn the teens' obsession with computers and the Internet into huge advantage that will be very beneficial in future. The research undertaken analysed the role of Olympiads in Informatics and Programming and proved that Olympiads are effective mechanisms of investments into youth. The research covered about 300 students of Computer Technology Chair of ITMO University (St. Petersburg National Research University of Information Technologies, Mechanics and Optics) and the latest data on the factors motivating students were received. In particular, that there is interconnection between Olympiads and students' motivation in acquiring the profession; that Olympiads contribute to the effective development of youth; and that they provide opportunity to get a good education that will bring financial prosperity, career satisfaction and self-realization.

**Keywords:** informatics, system of olympiads, olympiads in informatics, ICT education.

## 1. Olympiads from Early Age to Present Time

Olympiads have a long and colourful history. The first written records of the ancient Olympic Games are traced back to 776 BC in ancient Olympia (Greece). They were devised to measure physical strength, agility and stamina and existed for nearly twelve centuries until 393 A.D. when they were banned by Emperor Theodosius. The first modern Olympiads took place in 1896 in Athens. Since then they have acquired different forms and may assess different abilities, skills and aptitudes.

With the development of human society, mental labour substitution for physical one and at last the change of priorities from physical to mental labour in human minds, there were offered competitions which measured knowledge, mental & creative abilities, the ability of long-concentrated mental activity or mental performance – Olympiads in Science. The first international Olympiad in Science appeared in 1959. It was Olympiad in Mathematics (IMO), being held annually since that time. A decade later there appeared Olympiads in Physics (1967) and Chemistry (1968). The advent of personal computers in the 80-s and their proliferation into mass audience gave rise to the development of Informatics and Programming in schools and universities which brought about the establishment of International Olympiads in Informatics (IOI) in 1989.

## 2. The Development of ICT in the Information Age

Since the emergence of the first Olympiad in Science, a lot of other subject-specific Olympiads have come into play – Olympiads in Biology, Astronomy, Astrophysics, and even Medicine. But Olympiads in Informatics (programming) still remain one of the most important, which explains the emergence of Olympiads or Challenge for the youngsters, such as "Bebras" – International Challenge on Informatics and Computational Thinking (Pozdniakov *et al.*, 2016), (`bebras.org`). One may hear a lot about Information age (also known as the Computer Age or Digital Age), where informatics is playing a crucial role. In Information age IT- industries create a knowledge-based society which is characterized by innovations, the presence of hi-tech knowledge in every service or goods. Unique knowledge became one of the most valuable assets that can bring income and benefits and make countries, companies, universities and human beings competitive on the local and international markets. Various interdisciplinary areas are coming to the forefront and the branch of science that joins diverse areas is Informatics (biotechnologies, translational medicine, nanotechnology, etc.). These facts turn Informatics into the core discipline for studies and educational system.

It is generally accepted that ICT (Information and communications technology) has become one of the most significant and lucrative areas in modern society. ICT is an extended term for information technology (IT) which covers communications and telecommunications (telephone lines and wireless signals) and computers with necessary software, which enable users to access, store, transmit, and process information (Murray, 2011).

ICT appeared in the 70-s with the invention of a packet-switched network of computers & creation of microprocessors and began its rapid development in the 1990-s when the World Wide Web became available to mass public. However, the development and proliferation of ICT has not brought only income and advantages to the society, but negative issues as well. ICT has caused such problems as youth isolation and insularity, Internet addiction, lack of wish to work, inability to express thoughts logically, the shortage and inability of long concentrated mental activity, poor physical health of young people, etc. In other words ICT brought up the generation of mindless coach potatoes and lotus eaters. Having above mentioned qualities, the youth of today is not possible to form qualified professionals with a high level of intelligence and great creative potential, which are necessary for the knowledge-based world. But there is a simple solution to this problem. Teenagers' obsession with computers, information technologies and the Internet can be turned into the huge advantage that will be very beneficial for society, IT-industry and teenagers themselves in future. The solution to this problem is involvement of youth into Olympiads in Science (including Programming and Informatics). Olympiads may help to overcome problems brought about by negative influence of ICT and turn teenagers' interaction with ICT into benefits that may lead to their well-being in future.

So the main task of universities and schools is to engage as many teenagers into Olympiads in Informatics as possible, and organize the system of their further professional education in the most effective way.

## 3. The Aims of Olympiads

The aims of school and student's Olympiads may be divided into two levels. The first level is the aims of the Ministry of Education and Science of the Russian Federation and the second level – the aims of universities.

The main objectives of Olympiads, defined by the Ministry of Education and Science are:

- To identify and develop students' creativity and interest in research activities.
- To create necessary conditions to support gifted children.
- Proliferation and popularization of scientific knowledge among young people.

The example of objectives of higher education institutions will be considered within ITMO University case. This university has a ten year experience of organizing diverse Olympiads in Science.

ITMO University identifies following fundamental aims of Olympiads. The Olympiads in exact sciences serve for identification, assessment and selection of talented and capable teenagers who are invited for further education at university. Three key components are required for knowledge society: unique hi-tech knowledge, professionals able to create this knowledge and special environment – system of education able to train them. Olympiads allow to divide the process of youth development and training into three steps: initial training, thorough professional training and improvement, which are briefly mentioned below (Pavlova and Kazin, 2016).

The first step is initial development or training. The potent tool of development at this stage is Olympiads in Informatics & Programming. Olympiads facilitate constant development, the achieving of high results and the forming of basic competences. Due to participating in Olympiads teenagers become more creative and more intelligent, they become good at STEM (science, technology, engineering and mathematics). Moreover, participation in Olympiads opens up more chances and ways in future.

The second step is intensive, profound development. Due to the first stage, universities may design and implement more thorough and advanced study programs with major in Information Technologies and as a result train best professionals in IT. Universities should turn into so-called "growth platforms". In other words, universities should create such conditions that will facilitate youth's development, formation of high professional competences by devising and adopting best available teaching methods & technologies, attracting best teachers & researches to educational process. “Talented people seek out opportunities to grow, and they will flock to organizations that provide ample opportunities to do so. Retention also becomes a non-issue; if people are developing more rapidly than they could anywhere else, why would they leave? If institutions are truly serious about attracting, retaining, and developing high-quality talent, they need to view themselves as growth platforms for talent where people can develop themselves faster than they could elsewhere. This, in turn, can create a self-reinforcing cycle as talent creates more opportunities for growth” (Global Human Capital Trends, 2014).

The third step implies constant development of graduates. Due to early involvement into Olympiads graduates get accustomed to improving professional competences and knowledge and maintaining qualification at high level, which is essential for life-long learning.

## 4. Advantages and Disadvantages of Participating in Olympiads

Olympiads may have different forms: individual & command, distant & intramural, and mono- and interdisciplinary types. But all of them have a lot of advantages as they open up more opportunities to youth and contribute greatly to effective youth development.

**Admission to university without entrance exams.** Olympiads allow teenagers to enter universities without entrance examinations. The certificate (diploma) of Olympiads’ winner or awardee equals a hundred points of USE (unified state exam). Having the title of the prize-winner or awardee, teenagers have the right to pass only one exam instead of three. And exams are second to none as anxiety-makers.

**Olympiads are quality time & positive youth development.** Quality time is time which is used in the most profitable and effective way and may bring future benefits. According to Nobel Laureate Gary Becker, the amount of time spent on education in childhood is considered to be crucial contribution to child’s well-being in future (Becker, 1993). Being engaged into Olympiads, teenagers make the most of their time and invest into their future life. And self-investments are the most effective ones as they are based on motivation and lead to self-realization which is crucial for life-satisfaction.

**Olympiads impart the interest in STEM.** They involve into studying STEM. Thorough training for Olympiads may grow into vocational interests, as Olympiads encourage to keep on their studies at University. Research in these areas (STEM) is significant for any society, as knowledge-based society requires high technologies, cutting edge knowledge for sustainable economic and social growth which is considered to be the main goals of the 21 century.

**Olympiads create competitive environment.** In general, each competition (having distant form or requiring presence) is good for development, as it facilitates the shaping of competitiveness. And accumulation of talented teenagers in one place (virtual or real) enhances competitive medium. It additionally encourages and inspires youth to learn hard and achieve higher results. In other words, high concentration of talents tells positively on development.

**Olympiads raise motivation in acquiring knowledge.** Many young people have passion for programming or mathematics, and perceive them as a favourite pastime. And if you like what you are doing, you become motivated and are willing to spend a lot of time on it. As it was mentioned above, investments into yourself are the most beneficial. In general motivation is affected by: the work itself, a sense of achievement received from performing the work, recognition received for work performed; the possibility of advancement and growth; and a sense of trust and responsibility (Mallikarjuna, 2012). The fact that learners are the winners of several Olympiads proves that teenagers are very motivated (Table 1).

Despite the constant decrease of the population aged 15 to 19 (the main group involved in Olympiads) (see: Table 2), there is a rise in the number of prize-winners and winners of Olympiads, which speaks about the desire of young people to participate, win and get all the bonuses of participation.

The system of Olympiads is changing and evolving. Every year new types of Olympiads arise, e.g. Olympiads in Engineering, Olympiads in Computer or Cyber Security,

Table 1

The structure of winners of two and more "Olympiads", before enrolling Computer Technologies Chair of ITMO University, 2011-2016

| | 2011 | 2012 | 2013 | 2014 | 2015 | 2016 |
|---|---|---|---|---|---|---|
| Informatics, Mathematics | 21 | 6 | 16 | 17 | 19 | 22 |
| Mathematics, Physics | 2 | 2 | 4 | 6 | 7 | 7 |
| Informatics, Physics | 1 | - | 4 | 5 | 4 | 4 |
| Informatics, Informatics | 1 | - | 1 | 1 | 1 | 2 |
| Informatics, Mathematics, Physics | 1 | 4 | 5 | 3 | 5 | 5 |
| Informatics, Mathematics, Mathematics | - | - | - | 1 | 1 | 2 |
| Informatics, Physics, Russian language | - | - | - | 1 | 2 | 3 |
| Overall | 26 | 12 | 30 | 34 | 39 | 45 |
| Informatics + other types of Olympiads | 19 | 10 | 26 | 28 | 32 | 38 |

Table 2

The dynamics of total Russian population aged 15-19 years

| Indicators | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| The population aged 15–19 years, thousand | 12212 | 11852 | 11244 | 10485 | 9650 | 8389 | 8237 | 7631 | 7152 | 6956 | 6829 |

Table 3
The results of Russian teenagers' participation in International Olympiad in Informatics

| World Ranking | Gold | Silver | Bronze | Total |
|---|---|---|---|---|
| 2 | 49 | 31 | 12 | 92 |

Olympiads in Nanotechnology, Robotics and others. But in each set of Olympiads' problems there are tasks covering Computer Science. The data presented in Table 1 show that the most popular Olympiads with teenagers are still Olympiads in Informatics and the data of Table 3 show that International Olympiad in Informatics (IOI) is also very popular with Russian teenagers.

Thus the data of Tables 1, 3 show that the interest in Olympiads and ambition to win are increasing. It means that youth are seeking for knowledge, self-development and intellectual work.

**Development of cognitive (mental) abilities and creative capacities**. It is generally known that progress is driven by cognition and knowledge. Cognitive abilities of the person are properties of the brain to learn and analyse the surroundings, finding ways to use the information obtained in practice. The cognitive abilities of individual are almost limitless and infinite. The Olympiads require serious and thorough preparation. To get the title of a winner one has to learn a lot in Maths and Informatics, spend a lot of time on solving problems, studying algorithms and mastering programming skills. "Informatics offers an important opportunity for developing informatics knowledge, computational thinking and problem solving skills (Kabátová *et al.*, 2016). Also Olympiads' problems usually include creative tasks (at least one). And "unlike typical problems in programming competitions, creativity tasks usually do not have an optimal solution." (Grütter *et al.*, 2016). It implies such tasks require more efforts, thinking outside the box, devising new ways. Thus, Olympiads encourage to study hard, which promotes the development of mental and creative abilities. Consequently, it helps to achieve higher results.

**Olympiads make teenagers organize time wisely**. Teenagers have many tasks at hand and often have to deal with them quickly, so they become more disciplined, learn to make the most of their time and to balance the training for Olympiads with school-related tasks and family time. Being into many activities evolves very useful qualities, such as adaptability, flexibility, mobility and so on.

**Olympiads develop important "innovative qualities" – perseverance and persistence, diligence (being hard working).** Spending a lot of time on problem solving, perseverance, hard work and persistence in achieving goals are developed. Perseverance and persistence are very important qualities for research and development, innovative society. As noted by Thomas Edison insistence and persistence are expressed in a constructive, tolerant attitude to work, when to reach the goal you have to make dozens or even hundreds of attempts and experiments. With persistence and patient approach, failures are viewed as a step that brings you closer to achieving your aims and not as another setback, which pushes to abandon the goal.

**Olympiads involve into scientific and technological sphere and create the feelings of participation in real research activities.** Olympiads' problems, as a rule, are related to real life situations. Also it is noted that "computer science is a chance to introduce engineering as a highly creative, constructive activity (Hromkovič, 2016).

Teenagers get accustomed to constant learning that turns into a favourite pastime, or hobby, and at last, into a habit, which facilitates "life-long learning".

**Selection and devotion to the craft.** Olympiads help to choose the occupation from early age. Besides, they help to acquire one of the most popular & in-demand occupation in the nearest future because of rapid development of ICT, global automation and computerization. The selection of occupation is very important as individual is bound to follow the chosen professional course for the biggest part of his or her life. When one spends on education four or six years only to find out that one does not want to pursue chosen career any more, it is not considered rational. Olympiads guide and orient oneself, help to select one's way in the wide range of occupations and trends.

To figure out if bachelor degree students are going to continue chosen career in IT area, the survey was conducted at Computer Technology Chair (Fig. 1). The study shows that most of them have the desire to continue IT-career. Moreover, the wish to work by occupation proves that: a) learners made the right career choice; b) the occupation is popular with young people; c) learners are motivated and d) learners will improve their professional competences which will result in a more highly qualified labour resource.

Preparation to Olympiads may take various forms. Teens may have classes with enthusiastic school teachers in Informatics, visit special computer clubs, take courses held by universities or just visit summer computer school. Summer computer schools are like summer camps where teenagers both have an active rest and four or six hour classes on Informatics. In camp teenagers are offered various courses with pre-defined subjects, which they select according to their interests and levels of knowledge.

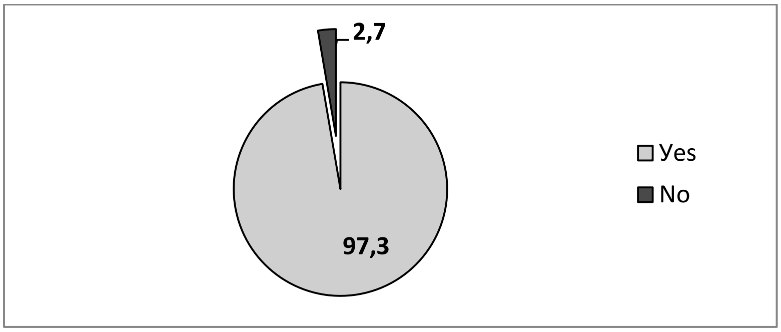


Fig. 1. The desire to continue chosen career in IT area, 2016.

Participating in Olympiads and summer computer schools allow teenagers to get acquainted with like-minded individuals, people with the common interest. Communication and companionship contribute to solving the issue of youth's isolation, standoffishness and insularity.

Another advantage is that command Olympiads develop teamwork skills. Ability to work in a team is quite important nowadays as most research projects are carried out by big or small teams of scientists and necessary for effective interaction team members and successful project completion. Also Olympiads facilitate the development of such skills and abilities which can be useful in life. Olympiads' contests help to find solutions in short time and in the most creative way, make the most of resources available. They teach to be concentrated while solving problems, to think critically, logically & analyse problems. Participants are taught to solve tasks in intense, stressful situations and help to acquire stress resilience.

Other advantages and useful qualities, which are associated with Olympiads, are revealed in the works of an expert on Olympiads, Dagienė V. (Dagienė and Stupurienė, 2016), which confirms our conclusions about Olympiads' significance.

**Disadvantages**. Now let us consider disadvantages of youth engagement into Olympiads, if there are any. Olympiads are extra activities. "Extra" means they are not obligatory and they are supposed to be in one's spare time. The only two disadvantages that catch the eye are the problems of over-scheduling and lessening of informal socializing with peers. Over-scheduling can be bad as it may cause stresses related to work overload and less informal communication can cause poor-socialization among peers. But according to the research undertaken by the society for research in child development (Mahoney *et al.*, 2006) for the society the greater concern than over-scheduling should be the fact that many youth do not participate in any activities at all. As the well-being of youth not participating in organized activities is reliably less positive compared to youth who do participate. It means over-scheduling is better than doing nothing. And as the old adage says: "Idle hands are the devil's tools, Angels hover about the busy" (Li *et al.*, 2008). Also their research shows that youth enrolled into extra activities are able to balance their organized activities effectively with school-related tasks, family time, informal socializing with peers, and relaxing.

Thus, academics, teachers and senior university management suppose that Olympiads' fruitfulness is obvious and such competitions are well worth taking part in as they influence positively youth development and bring more benefits than harm. To find out the opinion of learners on participating in Olympiads we asked students of Computer Technology Chair to write mini essay in a free form about pros and cons of Olympiads. The results of feedback are presented in Table 4.

Judging by the table results, learners themselves identify more positive aspects in Olympiads' participation than negative ones. Thus, involvement into Olympiads affects youth in a positive way and lessens the risk of negative youth adjustment (Mahoney *et al.*, 2006).

Table 4

Pros and cons of Olympiads' movement according to the participants of the competition (data based on the research conducted among 1st and 2d year students of Computer Technology Chair, 2016)

| | |
|---|---|
| Pros | • The Olympiad is a competition, and any competition creates the desire to be better and motivate you to study any subject more seriously.<br>• The Olympiad is an occasion to study hard and to prepare carefully, as a result, the participants possess a deeper knowledge of the subject.<br>• Olympiads open the chances to learn something more seriously, the sooner you start to participate in the Olympiads the better.<br>• Olympiads are a source of new ideas, problems and solutions, thanks to them, you can think more freely and more widely, overrunning school curricula and improving their education.<br>• Owing to Olympiads you can visit in a variety of interesting places and meet with peers who are addicted to the same things as you.<br>• Olympiads provide an opportunity to define the aim of the future, with the occupation.<br>• Olympiads develop self-discipline.<br>• Olympiads allow to spend time in a useful way rather than roam the streets or hang out.<br>• Excitement and responsibility during Olympiads develop patience and spirit.<br>• Olympiads provide an opportunity to win a lot of prizes (including scholarships).<br>• Olympiads provide admission to the University without taking into account the results of the exam (USE).<br>• The trips to Olympiads help to become independent. |
| Cons | • Numerous trips and preparation for Olympiads take a long time as a result you have little spare time.<br>• We learnt about Olympiads late, as a result we took up participating in them late. |

## 5. Education Satisfaction of Those Who Were Initially Involved in Olympiads and Afterwards Acquired the Qualification in Informatics

Since 2013 Computer Technology chair has been annually carrying out a series of surveys and questionnaires for monitoring study programs in Informatics and identifying their weaknesses and strengths. The feedback from learners helps to evaluate the current situation, make some changes in the curriculum and syllabus in order to improve the quality of educational process and learning outcomes.

The results of several surveys are represented below. The one of the studies was focused on identifying the level of education satisfaction among 4-th year BA students and 2-d year MA students (Fig. 2), i.e. those students who work on their final thesis and are about to finish the cycle of studies.

The results show that most graduates are satisfied with acquired education and it means it lived up to their expectations. In particular, 94% say they are satisfied and 6% claim they are almost satisfied mentioning that they need extra knowledge in some areas (e.g., finance or biology).

Another study examines if students have a wish to continue chosen career of IT-specialist. The results show that all the graduates (100%) are going to work in IT-area,

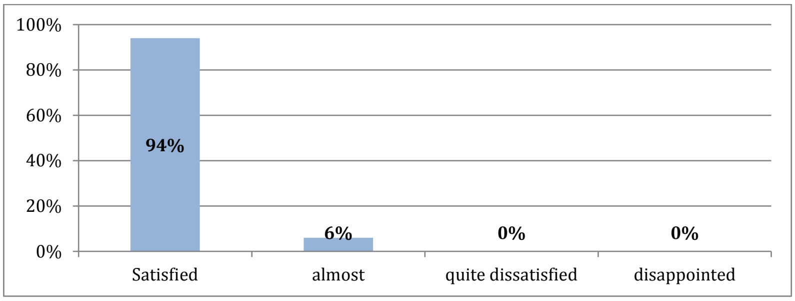


Fig. 2. The level of education satisfaction among 4-th year BA students and 2-d year MA students, 2016.

which means they are satisfied with their work, will continue their development and self-improvement, they will master their competences. It implies that Olympiads contribute to the concept "Life-long learning". The improving of abilities and skills is necessary for maintaining professional competences at a high level as IT area develops rapidly and without knowing cutting edge technologies, knowledge (qualification) may turn obsolete soon and efficiency may decrease. In its turn maintaining and perfecting professional competences will move specialists up the career ladder.

The next survey was devoted to identifying MA students' opinion on having sufficient knowledge for work by occupation. Since the majority of MA students have permanent jobs and for most of them MA program is the continuation of the first cycle of the program, they may evaluate the sufficiency of theoretical and practical base and the level of education. The data are in Fig. 3.

The research undertaken shows that the biggest share of respondents (96%) thinks they have sufficient level of theoretical knowledge and professional competences, which is proved by their employability with leading Russian and international IT-companies (Google, Facebook, Yandex, Devexports, Mail.group, "VKO" – social network). Such success (high level of employability) is achieved not only due to internships in progressive IT-companies as a part of a curriculum. Also it happens due to substantial theoretical base acquired before and after entering the university and due to big amount of hands-on training from early age (within preparation to Olympiads). High achievements

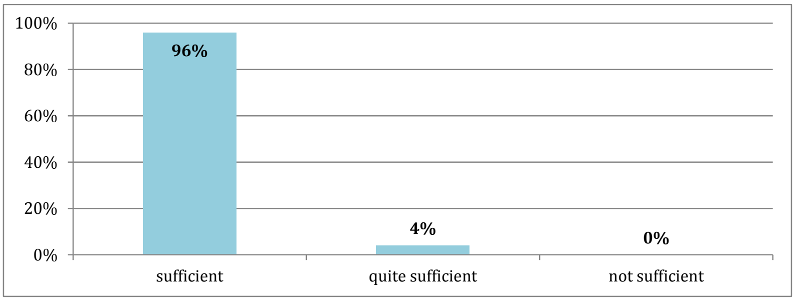


Fig. 3. Sufficiency of acquired knowledge for work by occupation, 2016.

and professional progress are only possible due to complex approach, consistent development during many years. This process starts from primary school and never finishes, that facilitates the high rate of employability.

Thus, the effectiveness of Olympiad's usage as a mechanism of training professionals in IT is noticeable and evident.

## 6. Popularization of the System of Olympiads and Its Benefits

The research proves that Olympiads are an effective mechanism of investments into youth's education and their future. It shows the interconnection between Olympiads and motivation of students in receiving the profession, proves that Olympiads contribute to the effective development of youth as an element of future intellectual and human capital and shows that Olympiads provide the opportunity to acquire a good education that will bring an interesting job, well-being and self-realization. Therefore, popularization of Olympiads and the development of Olympiads' system are important issues as Olympiads exert huge positive influence on youth development and everyone within the country benefits from the development of Olympiad's system. Country or region is provided with economic & social growth as new innovative start-ups are set up, they pay taxes, offer new services and goods and also place of employment. Companies get real IT professionals. Universities get access to best teenagers (prospective students) and become more competitive. Also university awareness arises in the minds of people. It means public learn about universities which realize competitive study programs and provide education satisfying needs of the market. And students' employability is the best advertising for universities and study programs.

So investments into children and teenagers are turned out to be justified investments as Olympiads make meaningful contributions to well-being, life satisfaction and positive outlook on life, and the list of benefits seems to be endless. What we should take into account that at present schools play the greatest role in engaging teenagers into Olympiads. The data are presented in Fig. 4.

The study shows that 66,7% respondents got to know about Olympiads from schools' staff, 16,2% – became aware of them by themselves, just surfing the Internet, 9,4%

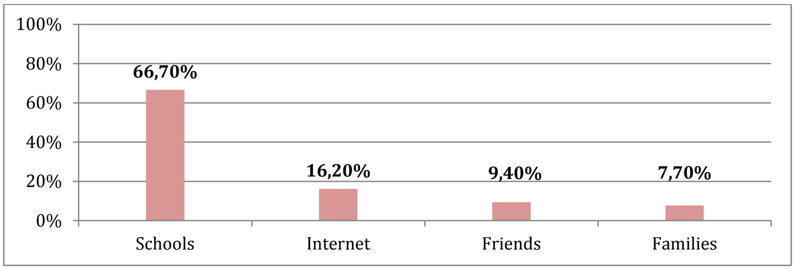


Fig. 4. How teenagers get to know about Olympiads, 2015.

– learnt from friends, 7,7% – from families. Thus, the greatest role in popularizing Olympiads belongs to school. But we cannot neglect other actors, such as universities, families and IT-companies. They have other, but also important roles. Families support teenagers, motivate them and create comfortable conditions for studying and participating. Universities take full responsibility for organizing safe and fair competitions and IT-companies maintain financially the organization of Olympiads' events and sponsor presents for participants and prize-winners, their prospective employees. The organization of Olympiads is quite a costly event, but these contests are important and the issue of raising funds emerges. Crowdfunding and endowment funds are other ways for higher education institutions to finance these events (Yanova *et al*., 2015). It is up to universities what to choose.

Engaging teenagers into Olympiads and training them, developing and popularizing the system of Olympiads, the state and the family invest in the development of mental abilities and creative capacities of young people, i.e. they invest into future intellectual and human capital of the country. Thus, the level of Olympiads' development is the indicator to what extent the government, higher education institutions and families care about the future of their nation.

## References

Bebras – International Contest on Informatics and Computer Fluency (2007–2011). `http://bebras.org`
Becker, G.S. (1993). *Theoretical and Empirical Analysis with Special Reference to Education*. New York.
Damon, W. (2004). What is positive youth development? *The ANNALS of the American Academy of Political and Social Science*, 591, 13–24. `http://ann.sagepub.com/content/591/1/13.full.pdf+html`
Dagienė, V., Stupurienė, G. (2016). Bebras – a sustainable community building model for the concept based learning of informatics and computational thinking. *Informatics in Education,* 15 (1), 25–44. DOI: 10.15388/infedu.2016.02
`https://www.mii.lt/informatics_in_education/htm/infedu.2016.02.htm`
*Global Human Capital Trends 2014: Engaging the 21st-century workforce.* (2014). Deloitte University Press. `http://dupress.com/wp-content /uploads/2014/04/Global HumanCapitalTrends_2014.pdf`
Grütter, S., Graf, D., Schmid, B. (2016). Watch them fight! Creativity task tournaments of the Swiss Olympiad in Informatics. *Olympiads in Informatics,* 10, 73–85. DOI: 10.15388/ioi.2016.05
Hromkovič, J. (2016). Homo Informaticus – why computer science fundamentals are an unavoidable part of human culture and how to teach them. *Olympiads in Informatics,* 10, 99–109. DOI: 10.15388/ioi.2016.07
Kabátová, M., Kalaš, I., Tomcsányiová, M. (2016). Programming in Slovak primary schools. *Olympiads in Informatics,* 10, 125 – 159. DOI: 10.15388/ioi.2016.09
Lerner, R.M. (2005). Promoting positive youth development: theoretical and empirical bases. In: *Workshop on the Science of Adolescent Health and Development*. National Research Council/ Institute of Medicine, Washington DC, 9 September 2005. `http://www.oalib.com/references/15356505`
Li, Y., Bebiroglu, N., Phelps, E., Lerner, R.M., Lerner, J.V. (2008). Out-of-school time activity participation, school engagement and positive youth development: Findings from the 4-h study of positive youth development. *Journal of Youth Development Bridging Research & Practice,* 3(3), 9–23.
`http://www.nae4ha.com/assets/documents/JYD_09080303_final.pdf`

Mahoney, J.L., Harris, A.L., Eccles, J.S. (2006). Organized activity participation, positive youth development, and the over-scheduling hypothesis. *Social Policy Report*, 20(4), 1–31.
`http://www.srcd.org/sites/default/files/documents/spr_20-4.pdf`

Mallikarjuna, N.L. (2012). Human resources responsibility on job satisfaction. *Journal of Business and Management,* 2(1), 11–14. `http://iosrjournals.org/iosr-jbm/papers/vol2-issue1/C0211114.pdf`

Murray, J. (2011). Cloud network architecture and ICT. *Information and Communication Technology.*
`http://itknowledgeexchange.techtarget.com/modern-network-architecture/cloud-network-architecture-and-ict/`

Pavlova, O., Kazin, Ph. (2016). Talent management in e-society: how investments in human capital work in the community of world class programmers. *Communications in Computer and Information Science*, 674, 322–332. DOI: 10.1007/978-3-319-49700-6_29

Pozdniakov, S.N., Kirynovich, I.F., Posov, I.A. (2016). Contest "Bebras" on Informatics in Russia and Belarus. *Olympiads in informatics*, 10 (special issue), 55–65. DOI: 10.15388/ioi.2016.special.07

Yanova, E.A., Bulatova, M.A., Mikhalevsky, D.A., Ufimtseva, A.Y. (2015). Crowdfunding at technical universities as a source of funding for scientific and technological development. *Journal Modern trends in science and technologies*, 8(8), 5–9.
`http://issledo.ru/wp-content/uploads/2015/12/Sb_k-8-8.pdf`

**O. Pavlova** is an academic of Computer Technology chair of ITMO University (St. Petersburg National Research University of Information Technologies, Mechanics and Optics). She has been teaching students having talent for mathematics, physics, computer science and programming since the year 2000. She holds a PhD in Economics & the main research interest is mechanisms & tools of higher education development and investments into school and university students, which facilitate their intellectual and professional development and bring various benefits. Her main professional focus is to create effective infrastructure of ITMO University, to engage students into hard studies, internship & active way of life. A lot of efforts are directed to bring up real experts or human capital, that will make great contributions to the development of the society and the world.

**E. Yanova** is an academic of Economy and Strategic Management chair of ITMO University (St. Petersburg National Research University of Information Technologies, Mechanics and Optics). She has been teaching students interested in micro- and macroeconomic since 1998. She holds a PhD in Economics & is a professor of the Russian Academy of Natural Sciences. The main research focus is management of the higher education, mechanisms of innovative development of economy, public administration of modern problems of institutional economy.

Olympiads in Informatics, 2017, Vol. 11, 123–142
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.10

# Learning Trajectory of Item Response Theory Course Using Multiple Softwares

Heri RETNAWATI
*Mathematics Department, Math ans Science Faculty, Universitas Negeri Yogyakarta*
*Yogyakarta, Indonesia*
*e-mail: heri_retnawati@uny.ac.id*

**Abstract**. In a learning process with regards to parameter estimation of item response theory (IRT), which has been related to quite intricate mathematical equations, software use should be an urgent matter. The study was to describe the learning trajectory of item response theory that had been integrated to the use of multiple software and obstacles of advanced IRT course. The study was validation study-type of design research with qualitative approach. The participants in the study were the graduate school students of Educational Research and Evaluation Study Program who concentrated on Educational Measurement. The learning trajectory that had been hypothesized was based on the researcher's experience in lecturing and in identifying the necessary requirements for each chapter in the course. The data were gathered by means of observation during learning, course activities documentation and graduate school students' learning results documentation. The data were analyzed qualitatively. By formulating and testing the hypotheses of learning trajectory, the researcher found that multiple software use in course might assist the students in understanding and improving the students' skills in terms of implementing IRT although they might encounter with several obstacles.

**Keywords**: item response theory (IRT), software, obstacles in courses.

## 1. Introduction

In the activities of research, education and development in many fields, good measurement instrument is an urgent matter. Such instrument will be more urgent if the components under measurement within these activities belong to latent variables, which has been assesses through observable indicators. For this interest, the instrument that will be implemented should be analyzed in order to attain well-qualified instrument. The quality of an instrument might be seen in its validity and reliability and it is also based on the characteristics of items that compose the instrument.

Item analysis is conducted to identify the quality of each item in the instrument. This analysis might be conducted using classical test theory approach and item response theory approach. The classical approach is implemented in estimating reliability, item difficulty and item discrimination index. Such approach is easy to implement in identifying the quality of measurement instrument; however, this approach has weakness. The weaknesses are that there is dependency between the participants' ability and the item parameters and that there is difficulty in implementing the concept of item response theory toward multiple measurement problems.

The approach of item response theory is implemented in order to overcome the weakness that that of classical test theory has. In this item response theory, the latent trait that has been measured is called as ability (Hambleton and Swaminathan, 1985; Hambleton, *et al.*, 1991). This ability is measured through the indicators of an instrument, both the test ones and the non-test ones. The relationship between ability and item parameters, such as item difficulty index, item discrimination index, pseudo guessing and alike, might be applied towards the dichotomous and polytomous data; certainly, the application should involve different equation model. The reason is that the mathematical model of this item response theory is not linear; therefore, the item parameter and ability estimations might not be directly conducted. However, through certain process, both by means of maximum likelihood and Bayes method, the item parameter and ability estimations might be directly conducted.

The skills of performing analysis to estimate item and ability parameters have been trained in universities especially in equipping university students as the researcher and psychometrical analyst candidates. These skills are trained through IRT course. In this course, the students learn multiple chapters related to item response theory or IRT, starting from classical test theory weakness, mathematical model in unidimensional IRT, parameter estimation, item characteristics curve, multidimensional IRT models and IRT implementation such as linking score, differential item functioning, cut of score and computerized adaptive testing. All IRT concepts and their implementation make use of mathematical models and several analyses, such as item parameter and ability estimations, are conducted by means of numeric approach; certainly, such approach demands software use.

Based on the course that has been taught in previous years, IRT has been conducted conventionally. In this course, the lecture has merely been explaining the theory and discussing the mathematical models without any practice of real data analysis in order to implement the IRT concepts. The software use has also been very minimum. The students' learning trajectory during the course has also been less systematic; as a result, the students have been less able to understand the inter-chapter relationship. This situation has caused the students to have less experience in analyzed the data related to instrument characteristics using IRT approach, to have difficulties when they should implement IRT in their undergraduate thesis and to have been less able in using software for assisting their analysis.

In IRT courses under measurement concentration within Educational Research and Evaluation Study Program, Graduate School of University Negeri Yogyakarta, the students come from multiple study programs (multi-entry). In one hand, this situation is beneficial because they come from both the exact study programs and the social science study programs. On the other hand, the lecturers who teach IRT become challenged because they should teach the students about multiple mathematical equations that lead to the implemented statistics. In order to assist the item analysis of IRT, there are multiple softwares that might be used. There are several mathematical models of IRT and the name of each model depends on the item parameter that the model contains, the data that the model uses and the dimension that the model measures. The software analysis depends on the mathematical models. For the IRT using Rasch Model, the students can run QUEST (Adams and Khoo, 1993) and CONQUEST (Wu, *et al.*, 1997). For unidimensional IRT with parameter model 1, 2 and 3 on graded response model-type dichotomous data or generalized partial credit model-type dichotomous data, the students can use PARSCALE (Muraki and Block, 1997). For multidimensional IRT with dichotomous data, the students can use TESFACT (Wilson, *et al.*, 1984). For the linking score analysis, the students can use IRTEQ (Han, 2009). In order that the students might understand and use these softwares, the course should implant the concept and the strategy of application if the measurement activities are to be integrated into the software use.

Along with the development of science and technology, the integration of course implementation and software use has been an urgent matter. The integration of information and communication technology (ICT) into learning process provides multiple benefits. One of such benefits has been proposed by Ranasinghe (2009), "Integrating technology into classroom is an approach to develop better understanding of basic concepts provided it is applied appropriately." His statement has also been supported by Ochkov and Bogomolova (2015). Several experts in the past had studied the integration of learning process and ICT use like Keirns (1992), Brekke and Hogstad (2010), Thomas (2006), Hudson and Porter (2010), Aydin (2009) and use of ICT in assessment (Retnawati, 2013).

The use of computer in the learning process is influenced by several factors. Norton and Cooper (2001) states that there has been a complex interaction among the factors of "belief, pedagogical knowledge, knowledge about using technology in teaching, cultural press and perceptions of assessment" and this complex interaction is related to syllabus reformation and teacher professional development. Hudson, *et al.* (2008) state the existence of certain burden for teachers in benefitting ICT with regards to lesson plans. Another problem will be insufficient of learning plans (Hudson, *et al.*, 2008), insufficient equipment (Unal and Ozturk, 2012) and insufficient time (Salehi and Salehi, 2012; Unal and Ozturk, 2012). These factors cause the computer use in learning process has not been optimum. Budinski (2013) also states that the level of software integration in the classroom has not been satisfying.

Several strategies might help integrating ICT into learning process so that the learning process might achieve the expected results. These strategies have been proposed by Fu (2013) as follows: "more induction, orientation and training for students; an increased emphasis on the importance of instructor access and effective administration; and the expansion of podcasting and online conferencing tools." Similarly, Goktas *et al.* (2009) state that "… strategies could provide a generic approach towards enhancing this ICT integration: technology plan, in-service training, strong infrastructures, technical support and role models." In addition, professional development (Afshari *et al.*, 2009) also holds an important role in determining the successful integration of ICT into learning process.

One of the parts in lesson plan is learning trajectory design. According to Simon (1995), learning trajectory includes learning goals, learning activities and hypothetical learning process. With the presence of learning trajectory, learning becomes social aspect, continuous iterative cycle and activity that enable teachers to attain learning theories that might be adjusted to university students' ability and reasoning level (Gravemeijer *et al.,* 2003). In line with the efforts that the integration of ICT into IRT course become successful, the study is to hypothesize the learning trajectory of item response theory and its implementation by means of ICT application toward the course. The learning trajectory then will be put into experiment in order to develop the activities in IRT course.

## 2. Method

The study was a design research because it was to design/develop an intervention that took the form of strategy and learning material for solving the problems in education (Lidinillah, n.d.). The design research model that the researcher deployed in the study was the study validation model by Nieven et al (2006, p.152). Validation study consisted of three phases namely preparing for the experiment/preparation and design phases, design experiment and retrospective analysis (Greivemeijer and Cobb, 2006). Preparing for the experiment/preparation and design phase was conducted in order to formulate the local instructional theory by performing an analysis toward the objectives that would be achieved, such as learning objectives, and by designing the conjecture from the local instructional theory that would be developed in the form of learning trajectory. Then, design experiment had been a stage of experimental design stage that aimed to test whether the learning trajectory that had been developed would be working. Next, retrospective analysis had been a stage of analyzing the data that had been gathered in order to identify whether these data supported the learning trajectory that had been developed.

The study was conducted by designing the learning trajectory of IRT first; then, the learning trajectory would be tested in the course. The learning trajectory was designed by developing the concepts of necessary prerequisites, by analyzing the software that might be used and by planning the data that would be used in the integration of ICT toward learning process. The experiment of the learning trajectory that would be hypothesized was conducted in 2015 and 2016 toward IRT course. The participants in the course were the students from Educational Research and Evaluation Study Program who had been concentrated in measurement. In 2015 there were six students and in 2016 and there were seven students who attended the course.

The data were gathered using documentation toward learning trajectory design and observation and documentation toward learning trajectory implementation. The results of learning trajectory documentations were the results of the students' work from the practice in IRT course. These data then would be analyzed by means of Miles and Huberman (1984) model through the steps of reduction, display and verification. The results of the analysis then would be described in order to attain the learning trajectory that would be in accordance with the design and the experiment.

## 3. Findings

In this section, the researcher would like to present the results of learning trajectory design activities and the results of the experiment along with the analysis.

### 3.1. *Designing the Learning Trajectory*

The preliminary part of the study was designing the learning trajectory. The learning trajectory was designed based on the literature review toward multiple references with regards to theory and by considering the materials that the students had studied. Such consideration was taken in relation to the prerequisite courses for IRT in the concentration of measurement under Educational Research and Evaluation Study Program.

#### 3.1.1. *Investigating the Weakness of Classical Test Theory*

Based on the study plan of all students who concentrated on measurement, the course that they attended has been Classical Test Theory. From the information provided by these students, the weakness of this theory had been studied but it had not been investigated by using empirical data. In order that students had in-depth understanding toward the weakness of classical test theory, empirical data were necessary in order to proof the dependency of item parameters and latent ability parameters. The empirical data that had been used in the first place were the ones from participants' responses toward a test using of dichotomous scoring, the ones from participants' responses toward a test by means of polytomous scoring and the ones from participants' responses toward a questionnaire by means of polytomous scoring.

#### 3.1.2. *Understanding the Mathematical Models in Item Response Theory with Characteristic Curve*

In item response theory, the probability of test taker to answer item correctly stated by a functional equation from the participants' ability and item parameters. This relationship might be understood by drawing a characteristic curve by implementing Rasch Model known as logistic 1 parameter model, or implementing 2 parameters logistic model, or implementing 3 parameters logistic model. The Microsoft Excel can be used to ease the calculation and in order that the students might observe the relationship pattern between the item parameters and the ability parameters. This program might also be used in drawing one characteristic curve or more.

#### 3.1.3. *Understanding the Parameter Estimation with Excel Assistance*

After having understood the relationship between the item parameters and the ability parameters, the next step that should be taken would be estimating the ability parameters based on the already identified item parameters. The estimation might be performed using maximum likelihood approach and Microsoft Excel so that the students might understand the process step by step. Then, by deploying the raw data in the form of participants' responses toward the test with dichotomous scoring the students might estimate the preliminary value for the item and ability parameters. With this preliminary score parameters and maximum likelihood approach, the students might estimate the item and ability parameters altogether. The competence of performing manual estimation provided a description toward the students in designing another program with regards to the application of IRT.

### 3.1.4. *Estimating the Item Parameters and the Ability Parameters by Implementing Rasch Model with QUEST and 2PL and 3PL Model with BILOGMG*

After completing the challenge of manual estimation, the students then might practice the estimation using computer softwares. For Rasch Model, the estimation might be performed by QUEST program. For analysis in 2 and 3 parameters logistic model, the estimation might be performed by BILOGMG or PARSCALE. The students might be guided in interpreting the output of this software.

### 3.1.5. *Testing the IRT Assumption in Terms of Unidimension, Local Independent and Parameter Invariance*

This assumption might be studied early before the item parameter estimation and the ability parameter estimation. However, due to the fact that the assumption testing had made use of the item parameters and the ability parameters that had been estimated, this step was put after the item and ability parameters estimation. The results of the estimation that had been used were the item parameters estimation and the ability parameters estimation from the data that had been split into two parts randomly based on the items and the test participants. Then, each data would be estimated.

The first assumption was the unidimension assumption. The assumption was applied altogether in order to prove the second assumption, namely the local independence. The assumption was applied in order to proof that the instrument only measure one dominant dimension. This analysis principally calculated the eigenvalues of the inter-item correlation matrix and it might be performed using SPSS software. The third assumption was the invariance parameters that made use of the results of item and ability parameters estimation.

### 3.1.6. *Understanding the Relationship of Item Parameters and Ability Parameters for Polytomous Data*

In the dichotomous data, the test participants' responses in a test or in a questionnaire might be scored with 1 and 0. In the polytomous data, the test participants' responses were scored 0, 1, 2, 3 and alike. Similarly in the dichotomous data, each item in a test or in a questionnaire had item parameters and each participant had ability parameters. The relationship between the item parameters and the ability parameters was stated in a complex mathematical relationship and was dependent on the selected model like Rasch Model, Partial Credit Model (PCM), and Generalized Partial Credit Model (GPCM) and alike. The item parameters estimation might also be conducted with the assistance of PASCALE software; the output of this software might be used in order to get an in-depth understanding toward the relationship between the item parameters and the ability parameters along with their characteristics.

### 3.1.7. *Understanding the Relationship Between the Item Parameters and the Ability Parameters for the Multidimension Dichotomous Data with TESTFACT*

Similarly in the unidimensional items, the relationship between the item and ability parameters for the dichotomous data in multidimensional IRT was studied. The analysis for estimating item and ability parameters was conducted by means of TESTFACT assistance and the item characteristics in this case was named as item characteristic surface. This item characteristics surface was visualized by means of MAPLE software.

### 3.1.8. *Implementation of IRT in Equating*

After the basic concept of IRT had been understood by the students, the next step was implementing the concept in the equaing of test set. This equating might be conducted altogether with the common items or without the common items by implementing multiple methods such as mean and mean, mean and sigma and characteristic curve. At the beginning, the students used Excel in order to understand the concept of equalization. Then, they can used IRTEQ software.

### 3.1.9. *Implementation of IRT to Detect DIF*

Another implementation of IRT that had been frequently used was the detection of differential item functioning (DIF). This detection was conducted by calculating the different probability in responding appropriately toward the test items of two groups under study, namely focal group and reference group. This step was taken after the item parameters and the ability parameters of each group had been estimated. The analysis might be directly conducted by means of software such as BILOGMG, PARSCALE or TESTFACT. The pattern of inter-group difference might be identified both by drawing the characteristic curve and by drawing the item characteristic surface for the items that measured multidimension.

### 3.1.10. *Determining the Cut of Score in a Test by Means of Bookmark Method that Had been An Implementation of IRT*

Another implementation of item response theory was the determination on the cut of score, especially bookmark method. The steps in determining the cut of score were studied and practiced by the students to gain their understanding.

### 3.1.11. *Estimating the Ability Parameters in CAT by Means of Simulation through Excel*

In line with the development of science and technology, the implementation of computer-based testing and computer adaptive testing (CAT) that had been adapted to the participants' ability had been an urgent matter; as a result, the estimation of participants' ability might be conducted efficiently. By combining the ability parameters and the algorithm of item determination, the students might conduct a simulation of ability estimation in IRT. This process might be conducted with the assistance of Excel that had become a bridge before the students developed the actual CAT.

The learning trajectory that had been hypothesized then would be tested into Item Response Theory course. The results displayed in each stage would be analyzed and the students would also report the obstacles that they encountered during the learning process.

## 3.2. *The Results of Experiment and Analysis*

The learning trajectory that had been hypothesized would be tested in Item Response Theory course. The results of each stage in the test were as follows.

### 3.2.1. *Identifying the Weakness of Classical Test Theory*

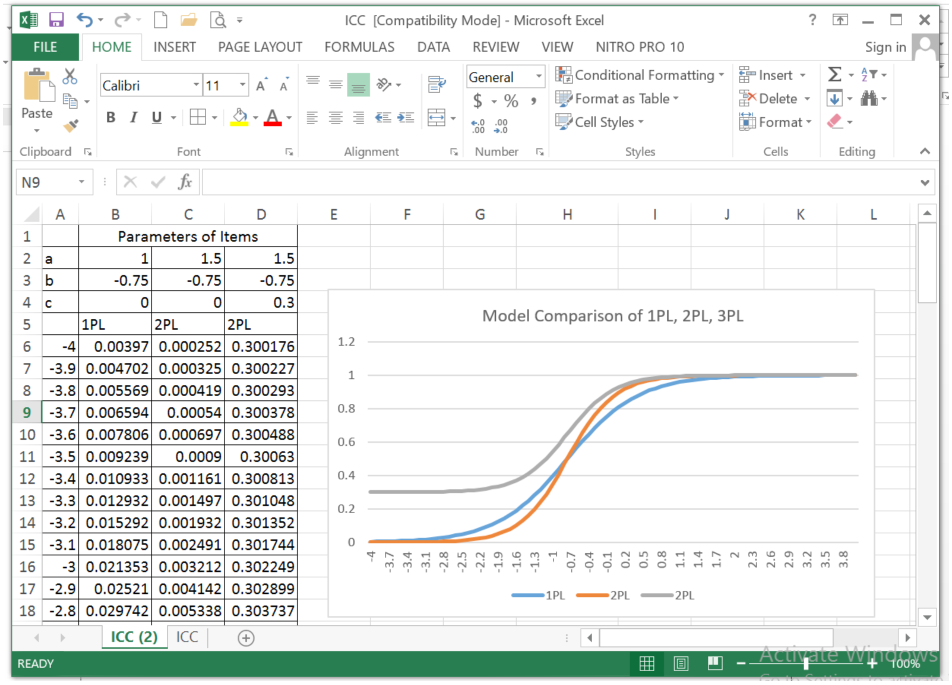
In order to identify the weakness of classical test theory, the students made use of the data from the national examination taken by the school that had high achievement and by the school that had low achievement. In the same time, the students also made use of the data from the documentation of Junior High School Mathematics National Examination in Indonesia. By comparing the item parameters in classical test theory between both school categories, the weakness of classical test theory regarding the dependence of item parameters, both the item difficulty index and the item discrimination index, toward the test participants' ability might be identified. Since the data that had been gathered were still raw, the students should change these data into the dichotomous data or the polytomous data.

In this step, most of the students had difficulties because they had not been used to managing huge data size. In order to change these data into dichotomous and polytomous data, the students used Excel while learning about several expressions of function that they had not understood.

### 3.2.2. *Understanding the Mathematical Models in IRT by Means of Characteristic Curve*

The relationship between the item parameters and the ability parameters in several models within unidimensional item response theory was investigated by means of graphic. An example of the graphic that had been drawn by the students would be displayed in Fig. 1.

The obstacle that had been encountered was that the students had not been used to the exponential function because there were about 25% of the students who had used to study and to draw function graphics. In order to draw these graphics, the lecturer's guideline was necessary for the students. Then, another obstacle that had been found was that there were 4 students who had less understanding toward the nonlinear function because the concept of linear function had been strongly embedded into their mind. By



| | A | B | C | D |
|---|---|---|---|---|
| 1 | | Parameters of Items | | |
| 2 | a | 1 | 1.5 | 1.5 |
| 3 | b | -0.75 | -0.75 | -0.75 |
| 4 | c | 0 | 0 | 0.3 |
| 5 | | 1PL | 2PL | 2PL |
| 6 | -4 | 0.00397 | 0.000252 | 0.300176 |
| 7 | -3.9 | 0.004702 | 0.000325 | 0.300227 |
| 8 | -3.8 | 0.005569 | 0.000419 | 0.300293 |
| 9 | -3.7 | 0.006594 | 0.00054 | 0.300378 |
| 10 | -3.6 | 0.007806 | 0.000697 | 0.300488 |
| 11 | -3.5 | 0.009239 | 0.0009 | 0.30063 |
| 12 | -3.4 | 0.010933 | 0.001161 | 0.300813 |
| 13 | -3.3 | 0.012932 | 0.001497 | 0.301048 |
| 14 | -3.2 | 0.015292 | 0.001932 | 0.301352 |
| 15 | -3.1 | 0.018075 | 0.002491 | 0.301744 |
| 16 | -3 | 0.021353 | 0.003212 | 0.302249 |
| 17 | -2.9 | 0.02521 | 0.004142 | 0.302899 |
| 18 | -2.8 | 0.029742 | 0.005338 | 0.303737 |

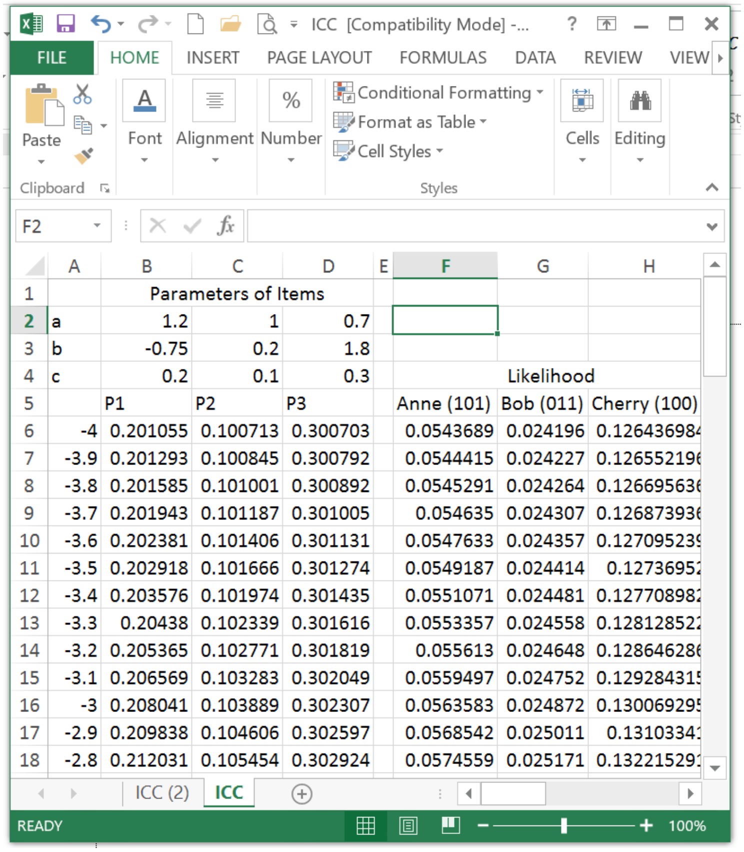
Fig. 1. Model Comparison in Unidimensional Item Response Theory.

providing the examples of nonlinear function case in the real life and the examples of other model such as the quadratic one or the cubical one, the 4 students gained better understanding. This obstacle was presented in the following statements.

> *"I have not drawn any graphics previously. Since my major is language, I have several difficulties in drawing graphics."*
>
> *"I just found that there is a nonlinear function. I just realized it. As far as I know, ability is always directly proportional to anything; so, my assumption is that the graphic will be straight."*

### 3.2.3. *Understanding the Parameter Estimation by Means of Excel Assistance*

After identifying the relationship between the item parameter and the ability parameter, the next step would be understanding the process of estimating the ability parameter based on the item parameter that had been found. The estimation method that would be implemented was maximum likelihood. At the beginning the students had difficulties in applying the likelihood equation in order to perform their estimation. In this case, the lecturer should guide the students so that they would be able to identify the parameters, both the item ones and the ability ones, that caused the likelihood to achieve maximum values. In order to ease the process, the estimation might be conducted by means of



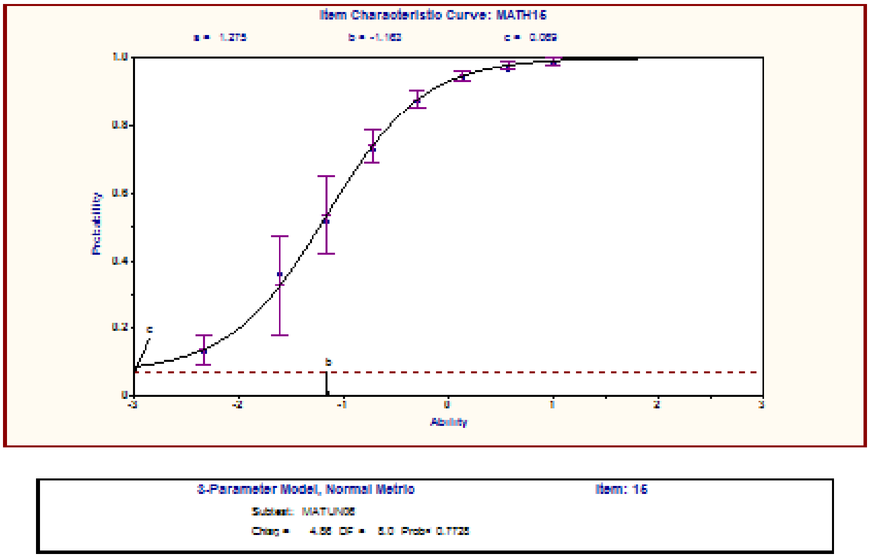
|  | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 |  | Parameters of Items |  |  |  |  |  |  |
| 2 | a | 1.2 | 1 | 0.7 |  |  |  |  |
| 3 | b | -0.75 | 0.2 | 1.8 |  |  |  |  |
| 4 | c | 0.2 | 0.1 | 0.3 |  |  | Likelihood |  |
| 5 |  | P1 | P2 | P3 |  | Anne (101) | Bob (011) | Cherry (100) |
| 6 | -4 | 0.201055 | 0.100713 | 0.300703 |  | 0.0543689 | 0.024196 | 0.126436984 |
| 7 | -3.9 | 0.201293 | 0.100845 | 0.300792 |  | 0.0544415 | 0.024227 | 0.126552196 |
| 8 | -3.8 | 0.201585 | 0.101001 | 0.300892 |  | 0.0545291 | 0.024264 | 0.126695636 |
| 9 | -3.7 | 0.201943 | 0.101187 | 0.301005 |  | 0.054635 | 0.024307 | 0.126873936 |
| 10 | -3.6 | 0.202381 | 0.101406 | 0.301131 |  | 0.0547633 | 0.024357 | 0.127095239 |
| 11 | -3.5 | 0.202918 | 0.101666 | 0.301274 |  | 0.0549187 | 0.024414 | 0.12736952 |
| 12 | -3.4 | 0.203576 | 0.101974 | 0.301435 |  | 0.0551071 | 0.024481 | 0.12770898 |
| 13 | -3.3 | 0.20438 | 0.102339 | 0.301616 |  | 0.0553357 | 0.024558 | 0.12812852 |
| 14 | -3.2 | 0.205365 | 0.102771 | 0.301819 |  | 0.055613 | 0.024648 | 0.128646286 |
| 15 | -3.1 | 0.206569 | 0.103283 | 0.302049 |  | 0.0559497 | 0.024752 | 0.129284315 |
| 16 | -3 | 0.208041 | 0.103889 | 0.302307 |  | 0.0563583 | 0.024872 | 0.130069295 |
| 17 | -2.9 | 0.209838 | 0.104606 | 0.302597 |  | 0.0568542 | 0.025011 | 0.13103341 |
| 18 | -2.8 | 0.212031 | 0.105454 | 0.302924 |  | 0.0574559 | 0.025171 | 0.13221529 |

Fig. 2. An Example of Worksheet of Ability Parameters Estimation with Maximum Likelihood.

Microsoft Excel software assistance. An example of estimation process would be presented in Fig. 2.

### 3.2.4. *Performing Item Parameter Estimation and Ability Parameter Estimation by Means of Software*

After performing semi-manual estimation, the students then were guided in performing the data analysis toward the test participants' responses in order to estimate the item parameters and the ability parameters by means of software. The software that would be used was dependent on the parameter models that might be used. For the analysis using Rasch Model, the students might use QUEST, 1PL model, 2PL model and 3PL model with BILOGMG. In using BILOGMG, in addition to performing item parameters estimation and ability parameters estimation, the students might also perform an analysis of model fitness both the one that used chi-square analysis and the one that used graphics. An example of output resulted the analysis toward model fitness would be presented in Fig. 3.



```
3-Parameter Model, Normal Metric                    Item: 15
        Subtest:  MATUN05
        Chisq =     4.88  DF =    8.0  Prob< 0.7725
```

Fig. 3. An Example of Output Results from the Analysis Using 3PL Model with BILOGMG

In this analysis, the students had difficulties in conducting analysis on several stages namely preparing the data, preparing the syntax and interpreting the analysis results. In order to anticipate these problems, the lecturer should provide step-by-step tutorial toward all students with repetition. For preparing the data, the lecturer might ease this step by editing the data in the Excel's worksheet and saving these data in the *.prn format. The students' difficulties might be found in the following statement.

*"I have difficulties in inputting the data, understanding the data and compiling the syntax analysis."*

*"Well, this is it: reading the output results and interpreting them become my problems."*

### 3.2.5. *Testing the IRT Assumptions in Terms of Unidimension, Local Independence and Parameter Invariance*

The IRT assumptions in terms of unidimension were analyzed by means of exploratory factor analysis; the exploratory factor analysis was conducted by measuring the scree-plot of eigenvalues. Based on the scree-plot, the students might identify the content of a dimension that had been measured by a measurement instrument and in the same time the students might prove the local independent. The parameter invariance was proven by splitting an instrument set into two parts based on the ability; then, the students analyzed each part in order to estimate the ability parameters. Next, the students would pair the ability parameters to each participant in order to create the scatterplot. Similarly, the test participants' responses were split in order to estimate the item parameters. The results of estimation were paired for each item in order to create the scatterplot. Proximity between the points in the scatterplot and the x = y line was used in order to prove the parameter invariance.

The assumption test was conducted by multiple software. As an example, the exploratory factor analysis might be conducted with the assistance of SPSS software and item and ability parameters estimation might be conducted with the assistance of BILOGMG software; the graphics of these analyses might be drawn with the assistance of Microsoft Excel software. Multiple software that might be implemented provided peculiar challenges for the students because they demanded adjustment and the adjustment took a relatively long time. An example of the analysis results would be presented in Fig. 4 and Fig. 5.

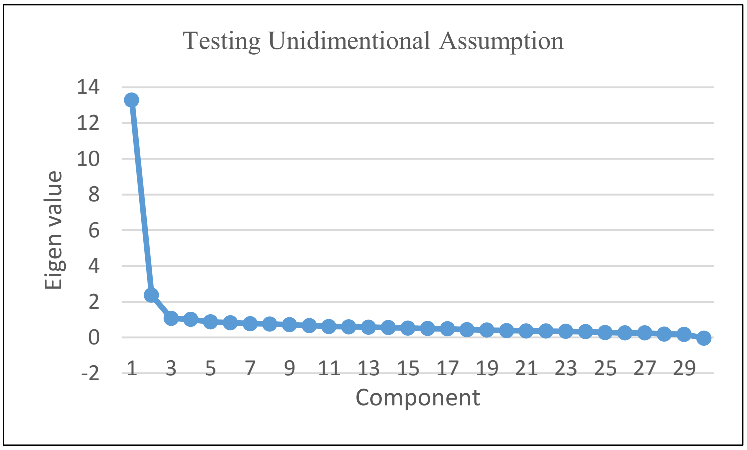


Fig. 4. An Example of Scree-Plot for Proving the Unidimension

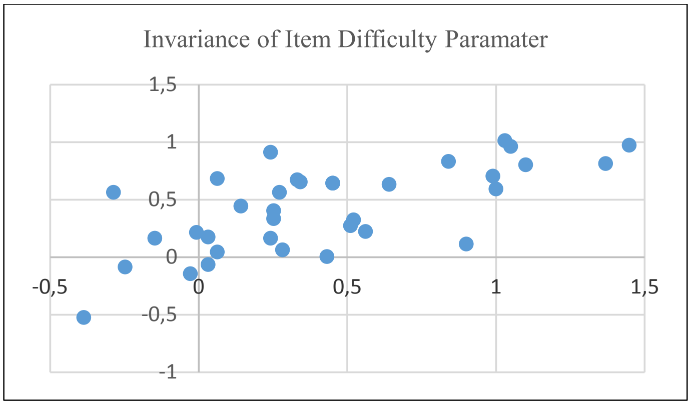
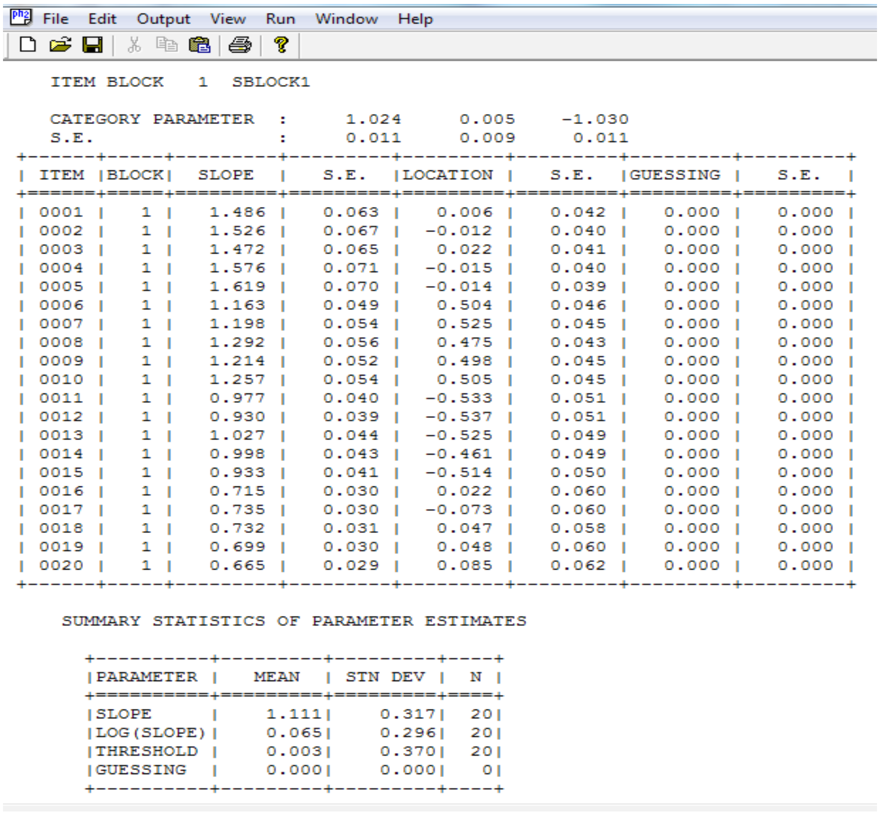


Fig. 5. An Example of Scatterplot for Proving the Parameter Invariance

### 3.2.6. *Understanding the Relationship between the Item and Ability Parameters for the Polytomous Data*

The students also used the similar stages in the dichotomous data for the analysis toward the polytomous data. The stages are understanding the relationship between the item parameters and the ability parameters, drawing the item characteristic curve (also known as the category response function) and estimating parameters). The parameter estimation was conducted using QUEST software for the partial credit model (PCM) and PARSCALE for the partial credit model (PCM), graded response model (GRM) and generalized partial credit model (GPCM). An example of analysis output for the item parameters would be presented in Fig. 6, for the category response function would be presented in Fig. 7 and for the value of information function and measurement errors would be presented in Fig. 8.

The difficulties that the students had in conducted the polytomous data analysis were related to the complex mathematical model and the interpretation on the output of analysis results. However, because they had experiences in analyzing the dichotomous data, it had been easier for the students to arrange the analysis syntax. The students' opinions about this stage would be presented as follows.

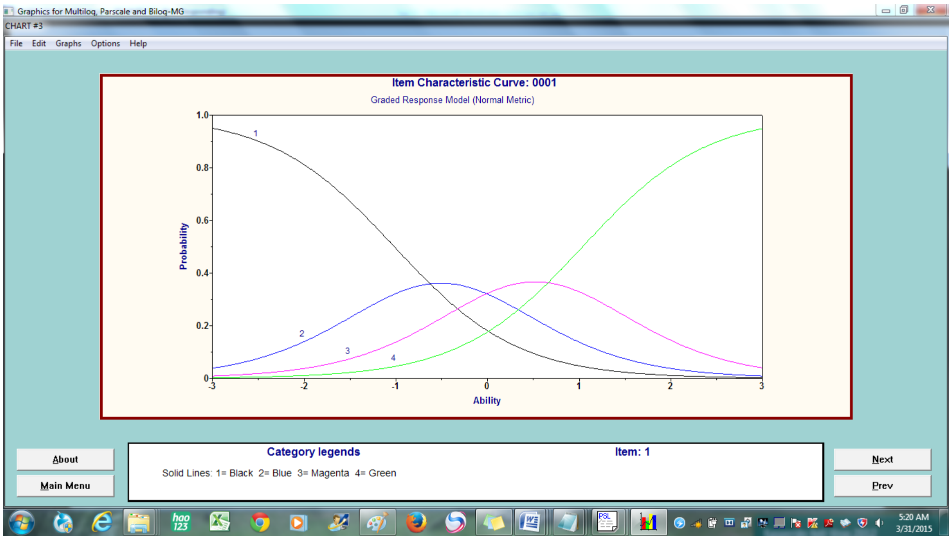


```
   ITEM BLOCK    1   SBLOCK1

   CATEGORY PARAMETER   :      1.024      0.005     -1.030
   S.E.                 :      0.011      0.009      0.011
 +------+-----+---------+---------+---------+---------+---------+---------+
 | ITEM |BLOCK|  SLOPE  |   S.E.  |LOCATION |   S.E.  |GUESSING |   S.E.  |
 +======+=====+=========+=========+=========+=========+=========+=========+
 | 0001 |   1 |   1.486 |   0.063 |   0.006 |   0.042 |   0.000 |   0.000 |
 | 0002 |   1 |   1.526 |   0.067 |  -0.012 |   0.040 |   0.000 |   0.000 |
 | 0003 |   1 |   1.472 |   0.065 |   0.022 |   0.041 |   0.000 |   0.000 |
 | 0004 |   1 |   1.576 |   0.071 |  -0.015 |   0.040 |   0.000 |   0.000 |
 | 0005 |   1 |   1.619 |   0.070 |  -0.014 |   0.039 |   0.000 |   0.000 |
 | 0006 |   1 |   1.163 |   0.049 |   0.504 |   0.046 |   0.000 |   0.000 |
 | 0007 |   1 |   1.198 |   0.054 |   0.525 |   0.045 |   0.000 |   0.000 |
 | 0008 |   1 |   1.292 |   0.056 |   0.475 |   0.043 |   0.000 |   0.000 |
 | 0009 |   1 |   1.214 |   0.052 |   0.498 |   0.045 |   0.000 |   0.000 |
 | 0010 |   1 |   1.257 |   0.054 |   0.505 |   0.045 |   0.000 |   0.000 |
 | 0011 |   1 |   0.977 |   0.040 |  -0.533 |   0.051 |   0.000 |   0.000 |
 | 0012 |   1 |   0.930 |   0.039 |  -0.537 |   0.051 |   0.000 |   0.000 |
 | 0013 |   1 |   1.027 |   0.044 |  -0.525 |   0.049 |   0.000 |   0.000 |
 | 0014 |   1 |   0.998 |   0.043 |  -0.461 |   0.049 |   0.000 |   0.000 |
 | 0015 |   1 |   0.933 |   0.041 |  -0.514 |   0.050 |   0.000 |   0.000 |
 | 0016 |   1 |   0.715 |   0.030 |   0.022 |   0.060 |   0.000 |   0.000 |
 | 0017 |   1 |   0.735 |   0.030 |  -0.073 |   0.060 |   0.000 |   0.000 |
 | 0018 |   1 |   0.732 |   0.031 |   0.047 |   0.058 |   0.000 |   0.000 |
 | 0019 |   1 |   0.699 |   0.030 |   0.048 |   0.060 |   0.000 |   0.000 |
 | 0020 |   1 |   0.665 |   0.029 |   0.085 |   0.062 |   0.000 |   0.000 |
 +------+-----+---------+---------+---------+---------+---------+---------+

    SUMMARY STATISTICS OF PARAMETER ESTIMATES

      +----------+---------+---------+----+
      |PARAMETER |   MEAN  | STN DEV | N  |
      +==========+=========+=========+====+
      |SLOPE     |    1.111|    0.317|  20|
      |LOG(SLOPE)|    0.065|    0.296|  20|
      |THRESHOLD |    0.003|    0.370|  20|
      |GUESSING  |    0.000|    0.000|   0|
      +----------+---------+---------+----+
```

Fig. 6. An Example of Output from the Results of Item Parameters Analysis Using PARSCALE.

*"The data analysis toward the polytomous scoring turns out to be more complicated than the one toward the dichotomous scoring, whereas the data analysis toward the dichotomous scoring itself has been difficult."*

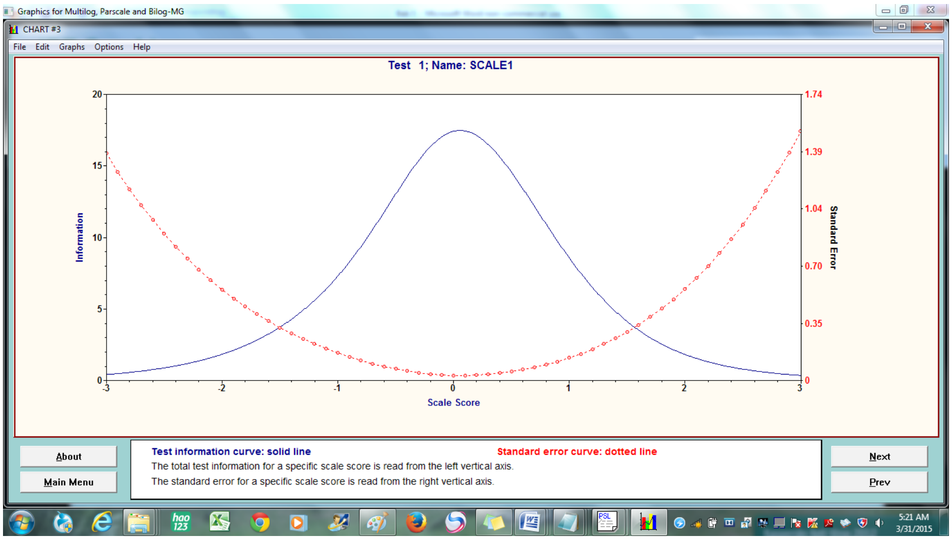
*"The analysis syntax of the polytomous data is similar to that of the dichotomous data; this is pretty helpful."*


Graphics for Multilog, Parscale and Bilog-MG
CHART #3
File Edit Graphs Options Help
Item Characteristic Curve: 0001
Graded Response Model (Normal Metric)
Probability
Ability
1
2
3
4
About
Main Menu
Next
Prev
5:20 AM
3/31/2015

Category legends Item: 1

Solid Lines: 1= Black 2= Blue 3= Magenta 4= Green

Fig. 7. An Example of Output from the Item Category Response Function with GRM-Type.



Test information curve: solid line Standard error curve: dotted line

The total test information for a specific scale score is read from the left vertical axis.

The standard error for a specific scale score is read from the right vertical axis.

Fig. 8. An Example of Output from the Value of Item Information Function and Measurement Errors.

### 3.2.7. *Understanding the Relationship between the Item Parameters and the Ability Parameters for the Multidimensional Dichotomous Data*

If the assumption of unidimension for IRT in the dichotomous data analysis was not prevailing, then the students would conduct an alternative analysis using multidimensional IRT. First, the researcher introduced the mathematical models of multidimensional IRT; then, the researcher would introduce the relationship between the item parameters and ability parameters. The item characteristics surface would be described with the assistance of MAPLE software. The analysis for estimating the item parameters and ability parameters would be conducted by means of TESTFACT software. An example of item characteristics surface resulted from TESTFACT software would be presented in Fig. 9.

Although at the beginning the students had difficulties in understanding the mathematical relationship in the multidimensional item response theory, they found an interesting matter. Their difficulties were caused by the complex mathematical models but the students were happy with the graphic that they made especially the items that measured 2 dimensions. The students had been accustomed to reading output from the analysis results and interpreting these results; therefore, they did not have any difficulties at all. These findings were supported by the following statements.

> *"The mathematical models of multidimensional IRT have been more complex than the last course."*

> *"Looking at the graphics, I am happy. It is easier to understand the relationship between the ability parameters and the probability of students' to respond item correctly."*

> *"The more I read the output, the easier I interpret the output."*

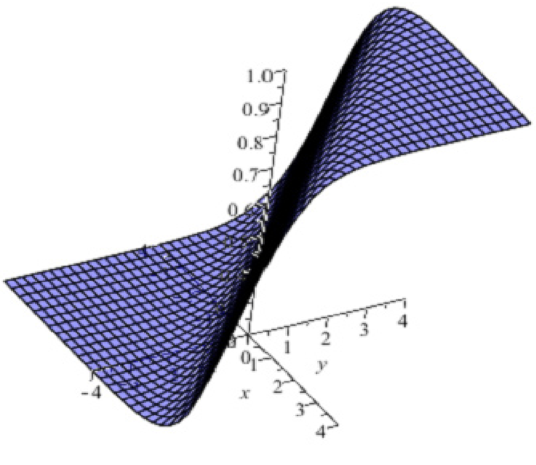


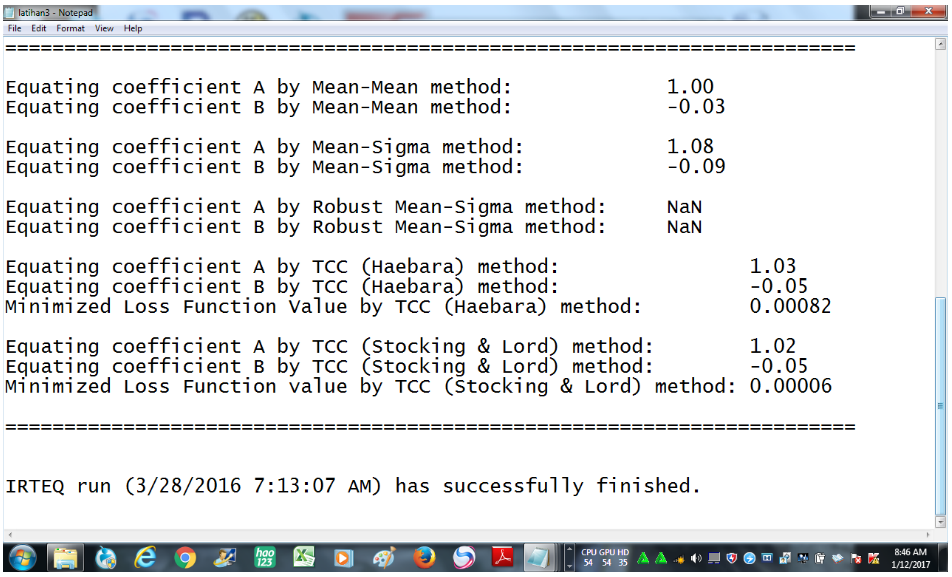
Fig. 9. An Example of Item Characteristic Surface that Measured 2 Dimensions

### 3.2.8. I*mplementing IRT toward Equating*

By using the data of participants' responses toward several equal test sets, the students were invited to implement IRT for the linking score, both in equating manner and in concordance manner. The practice of implementing the concept of linking score was conducted semi-manually by means of Microsoft Excel software, mean and mean method, mean and sigma method and characteristic curve method (Stocking and Lord method and Haebara method). After estimating the item parameters and the ability parameters, the steps in this stage were simple yet the students had difficulties because they needed to understand the regression analysis in order to generate the prediction scores. However, after using IRTEQ program it was easier for the students to implement the concept of linking score. An example of analysis resulted by IRTEQ program would be presented in Fig. 10.

### 3.2.9. *Detecting Differential Item Functioning*

To detect DIF, the students categorized first the responses of focal group participants and reference group participants; then, they analyzed the item and ability parameters of both groups. The item parameter number-n for the focal group and the reference group was used in order to draw the characteristic curve so that the students would identify at which ability scale interval an item would benefit certain group. An example of results from the response data generated by the polytomous scoring would be presented in Fig. 11 and for the dichotomous scoring would be presented in Fig. 12. In this stage, the students' difficulties were more related to the technical aspects in terms of mathematical expressions in drawing graphics.



```
========================================================================

Equating coefficient A by Mean-Mean method:              1.00
Equating coefficient B by Mean-Mean method:              -0.03

Equating coefficient A by Mean-Sigma method:             1.08
Equating coefficient B by Mean-Sigma method:             -0.09

Equating coefficient A by Robust Mean-Sigma method:      NaN
Equating coefficient B by Robust Mean-Sigma method:      NaN

Equating coefficient A by TCC (Haebara) method:                 1.03
Equating coefficient B by TCC (Haebara) method:                 -0.05
Minimized Loss Function Value by TCC (Haebara) method:          0.00082

Equating coefficient A by TCC (Stocking & Lord) method:         1.02
Equating coefficient B by TCC (Stocking & Lord) method:         -0.05
Minimized Loss Function value by TCC (Stocking & Lord) method: 0.00006

========================================================================


IRTEQ run (3/28/2016 7:13:07 AM) has successfully finished.
```

Fig. 10. An Example of Analysis Resulted by IRTEQ Program for Generating the Prediction Scores.

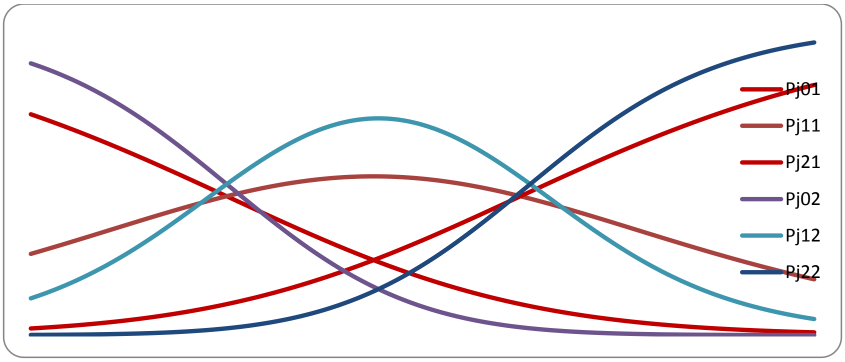


Fig. 11. An Example of Category Response Function for the Focal Group and the Reference Group.

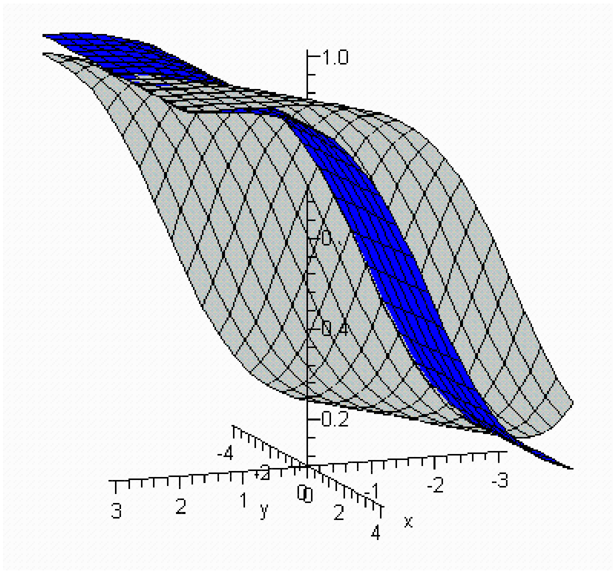


Fig. 12. An Example of Item Characteristics Surface for the Focal Group and the Reference Group Using MAPLE.

### 3.2.10. *Determining the Cut of Score in a Test by Means of Bookmark Method which Had been an IRT Implementation*

By using the real data as well, the students benefitted the results of item parameters estimation in order to design the booklet and they designed the booklet by ordering the necessary minimum practice for attaining 67% chance of responding an item correctly. The booklet then was tested in order to determine the cut of score for certain lessons; in the study, the booklet was used in mathematics. In the practice, the students did not experience any significant difficulties because they also implement the insight that they gained from the course. They also had ideas to conduct studies related to the IRT implementation in the future.

### 3.2.11. *Estimating the Item Parameters on CAT by means of Simulation through Excel Program*

Using a set data such as item parameters, the students were invited to do simulation in CAT. They pretended as if they were arranging an item bank and then they calculated the algorithm of simple item selection like tree algorithm. Using simulation data in the form of students' response pattern, the students stimulated the test participants' ability estimation by means of one method such as maximum likelihood. In addition, the students also practiced the arrangement of stopping rule regulations within the CAT implementation.

There were several obstacles that the students encountered in studying CAT. In relation to these obstacles, there should be an in-depth understanding toward the concept of item bank and its arrangement by means of certain method like the tree algorithm. Although it had been simple, the tree algorithm should be explained by the lecturer with a real example and by involving examples from certain lessons that the students might understand. The real description regarding the example of CAT presentation should also be a topic of question for the students. However, these multiple questions had been the trigger for the students to conduct more studies in the future. This finding was supported by the following statements.

> *"After learning CAT, I have many questions in my mind. Anyway, the positive aspect is that I would like to conduct research about CAT."*

> *"It turns out that, there have been so many ideas of research that we have, especially in instrument item characteristics and its implementation such as equating, cut of score and CAT."*

## 4. Conclusions and Discussions

Paying attention to the results of the analysis toward each experimental stage in the learning trajectory, the researcher found that the learning trajectory of item response theory through the software use that had been hypothesized had provided the expected results. In the study, the students might interpret their understanding through the output that had been resulted from the practice, especially their understanding toward relationship between item and ability parameters. The students had also been able to use multiple software in performing real data analysis and simulated data analysis by implementing multiple concepts in item response theory. This had been a trigger for the students to perform their studies by implementing the theory.

This success was supported by several matters. First, by hypothesizing the learning trajectory, the lecturer actually had already planned and had already prepared the course, including the learning materials, the software and the supporting data. With sufficient preparation, the course might run well. This situation had been in accordance with the suggestions by Hudson *et al.* (2008) that the successful learning should be supported by a good learning plan. Second, the learning trajectory that had been planned had given attention toward the necessary prerequisite materials. The materials that should be studied were supported by the materials that had previously been mastered by the students. Third, the instructor had been very encouraging; the instructor had trained the integration of ICT into the learning process step by step along with its application toward the related problems and the use of existing software; as a result, it would be easier for the students to understand the materials of item response theory especially the materials that had been related to the mathematical equations and the courses that made use of multiple complex software. The third aspect here had been in accordance with the results of a study by Ranasinghe (2009).

The students encountered multiple obstacles during while they were integrating the technology into the learning process. Their obstacles within the learning process were coming from multiple study programs (multi entries), having not been accustomed to using Microsoft Excel, having not been accustomed to arranging analysis syntax, having to deal with complex mathematical equations and having to use multiple software. The strategies that had been taken in order to overcome these obstacles were designing handouts for the students and training the students to have step-by-step understanding toward the concept in details. It took a long time to implement these strategies. These findings, by the way, had been in accordance to the results of several studies (Salehi and Salehi, 2012; Unal and Ozturk, 2012). The students also had difficulties in using multiple software. Therefore, using single software that might cover all types of data analysis in IRT course is an urgent matter that should be pursued.

## References

Adams, R.J., Khoo, S.T. (1993). *Quest: The Interactive Test Analysis System [Computer Program]*. Hawthorn: Australian Council for Educational Research.

Afshari, M., Abu Bakar, K., Luan, W., S., Abu Samah, B., Fooi, F.,S. (2009). Factors affecting teachers' use of information and communication technology**.** *International Journal of Instruction,* 2(1), 77–104.

Aydin, E. (2005). The use of computers in mathematics education: a paradigm shift from "computer assisted instruction" towards "student programming". *The Turkish Online Journal of Educational Technology – TOJET*, 4(2), 27–34.

Brekke, M., Hogstad, P.H. (2010). New teaching methods – Using computer technology in physics, mathematics and computer science. *International Journal of Digital Society* (*IJDS*), 1(1), 17–24.

Budinski, N. (2013). A survey on use of computers in mathematical education in Serbia. *The Teaching of Mathematics*, 16(1), 42–46.

Du Toit, M. (2003). *IRT from SSi: BILOG-MG, MULTILOG, PARSCALE, TESTFACT*. Lincolnwood: SSi.

Fu, J.S. (2013). ICT in education: A critical literature review and its implications. *International Journal of Education and Development using Information and Communication Technology (IJEDICT),* 9(1), 112–125.

Ghavifekr, S., Rosdy, W.A.W. (2015). Teaching and learning with technology: Effectiveness of ICT integration in schools. *International Journal of Research in Edution and Science (IJRES)*, 1(2), 175–191.

Goktas, Y., Yildirim, S., Yildirim, Z. (2009). Main barriers and possible enablers of ICTs integration into preservice teacher education programs. *Educational Technology & Society*, 12(1), 193–204.

Gravemeijer, K., Cobb, P. (2006). Design research from a learning perspective. In: van den Akker, J., Gravemeijer, K., McKenney., S., Nieveen, N. (Eds.), *Educational Design Research.* New York: Routledge.

Gravemeijer, K., Bowers, J., Stephan, M. (2003). A hypothetical learning trajectory on measurement and flexible arithmetic. *Journal for Research in Mathematics Education. Monograph,* 12, 51–66.

Hambleton, R.K., Swaminathan, H. (1985). *Item Response Theory*. Boston, MA: Kluwer Inc.

Hambleton, R.K., Swaminathan, H., Rogers, H.J. (1991). *Fundamental of Item Response Theory*. Newbury Park, CA: Sage Publication Inc.

Han, K.T. (2009). IRTEQ: Windows application that implements IRT scaling and equating [Computer program]. *Applied Psychological Measurement*, 33(6), 491–493.

Hudson, R., Porter, A. (2010). ICT use to improve mathematics learning in secondary schools. *Paper in ACEC2010: Digital Diversity Conference*, 6–9 April 2010, Melbourne, Australia.

Hudson, R., Porter, A.L., Nelson, M.I. (2008). Barriers to using ICT in mathematics teaching: issues in methodology. In: Luca, J., Weipl, E. (Eds.), *ed-Media 2008 World Conference on Educational Multimedia, Hypermedia and Telecommunications*. Chesapeake, VA: Association for the Advancement of Computing in Education, 5765–5776.

Keirns, J.L. (1992). Does computer coursework transfer into teaching practice? A follow-up study of teachers in a computer course. *Journal of Computing in Teacher Education*, 8(4).

Lidinillah, D.A.M. (n.d.). *Educational Design Research: a Theoretical Framework for Action*. Available at `www.file.upi.edu/Direktori/`

Miles, M.B., Hubberman, A.M. (1994). *Qualitative Data Anaysis*. California: SAGE Publications.

Mislevy, R.J., Bock, R.D. (1990). *BILOG 3: Item Analysis & Test Scoring with Binary Logistic Models [Computer Program]*. Moorseville: Scientific Sofware Inc.

Muraki, E., Bock, R.D. (1997). *Parscale 3: IRT Based Test Scoring and Item Analysis for Graded Items and Rating Scales [Computer Program]*. Chicago: Scintific Software Inc.

Nieveen, N., McKenney, S., van den Akker (2006). Educational design research. In: van den Akker, J., Gravemeijer, K., McKenney., S., Nieveen, N. (Eds.), *Educational Design Research.* New York: Routledge.

Norton, S., Cooper, T. (2001). Factors influencing computer use in mathematics teaching in secondary schools. *Paper in 24th Annual MERGA Conference*, July 2001, Sydney, Australia*,* 386–393.

Ochkov, V.F., Bogomolova, E.P. (2015). Teaching mathematics with mathematical software, *Journal of Humanistic Mathematics*, 5(1), 265–285. DOI: 10.5642/jhummath.201501.15. Available at: `http://scholarship.claremont.edu/jhm/ vol5/iss1/15`

Ranasinghe, A.I., Leisher, D. (2009). The benefit of integrating technology into the classroom. *International Mathematical Forum*, 4(40), 1955–1961.

Retnawati, H. (2015). The comparison of accuracy scores on the paper and pencil testing vs. computer based testing. *The Turkish Online Journal of Educational Technology* (TOJET) 14(4), 135–142.

Salehi, H., Salehi, Z. (2012). Challenges for using ICT in education: Teachers' insights. *International Journal of e-Education, e-Business, e-Management and e-Learning,* 2(1), 40–43.

Simon, M.A. (1995). Reconstructing mathematics pedagogy from a constructivist perspective. *Journal for Research in Mathematics Education,* 26, 114–145.

Stultz, S.,L. (2013). The effectiveness of computer-assisted instruction for teaching mathematics to students with specific learning disability. *The Journal of Special Education Apprenticeship*, 2(2), Article 7. Available at: `http://scholarworks.lib.csusb.edu/josea/vol2/iss2/7`

Thomas, O.J. (2006). Teachers using computers in mathematics: A longitudinal study. In: Novotná, J., Moraová, H., Krátká, M., Stehlíková, N. (Eds.), *Proceedings 30 Conference of the International Group for the Psychology of Mathematics Education*, 5, 265–272. Prague: PME5 – 265.

Unal, S., Ozturk, I.H. (2012). Barriers to ITC integration into teachers' classroom practices: Lessons from a case study on social studies teachers in Turkey. *World Applied Sciences Journal*, 18(7), 939–944, DOI: 10.5829/idosi.wasj.2012.18.07.1243

Wilson, D., Wood, R., Gibbons, R. (1984). *TESTFACT: Test Scoring and Fullinformation Item Factor Analysis* [*Computer Program*]. Mooresville, IL: SSi.

Wu**,** *M.L., Adams, R.J., Wilson, M.R. (1997). ConQuest: Multi-Aspect Test Software [Computer program].* Camberwell: Australian Council for Educational Research.

**H. Retnawati** is Associate Professor in Mathematics Department, Mathematics and Science Faculty, Universitas Negeri Yogyakarta, Indonesia, as researcher and lecturer. Her research interest are teaching methods and assessment in mathematics education, measurement in education, and psychometric.

Olympiads in Informatics, 2017, Vol. 11, 143–150
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.11

REPORTS

# Informatics and Programming Education at Primary and Secondary Schools in Japan

Susumu KANEMUNE[1], Shizuka SHIRAI[2], Seiichi TANI[3]
[1]*Department of Electro-Mechanical Engineering, Osaka Electro-Communication University*
[2]*Department of Human Environmental Sciences, Mukogawa Women's University*
[3]*Department of Information Science, Nihon University*
*e-mail: kanemune@gmail.com, shirai@mukogawa-u.ac.jp, tani.seiichi@nihon-u.ac.jp*

**Abstract.** In this paper, we introduce the present situation of informatics and programming education at primary and secondary schools in Japan. Furthermore, we also explain post-2020 new informatics education. Previously, the importance of informatics education has not been recognized in Japan. However, with this educational reform, it is expected to start to change the attitude toward informatics education. In the New Course of Study, all elementary school students will experience programming and all high school students will learn informatics.

**Keywords:** informatics education, programming education, information study.

## 1. Introduction

In recent years, there is increasing interest in informatics and programming education at the elementary and secondary level around the world. In Japan, the Prime Minister Shinzo Abe declared at Industrial Competitiveness Council in April 2016 that Japan will make programming education compulsory from primary and middle school (Prime Minister of Japan and His Cabinet, 2016). This declaration had a great impact on the whole society in Japan including industries and local governments as well as parents, teachers and schools. In Japan, school curriculum guidelines are revised every ten years.

MEXT (Ministry of Education, Culture, Sports, Science and Technology) prepares to reflect this policy in the new Courses of Study which will be published from 2017 to 2018 and implemented beyond 2020.

In this paper, we introduce the present situation of informatics and programming education at primary and secondary schools in Japan. Furthermore, we also explain post-2020 new informatics education.

## 2. Current Japanese Education System

The education system in Japan consists of 6 grades of primary education, 3 grades of lower secondary education, 3 grades of upper secondary education and higher education. The academic year starts from April and ends in March. Fig. 1 shows the education system in Japan. Compulsory education is nine years: six years in primary school and three years in junior high school. After compulsory education, about 98% of students enter high school and about 50% of students enter university. In Japan, in

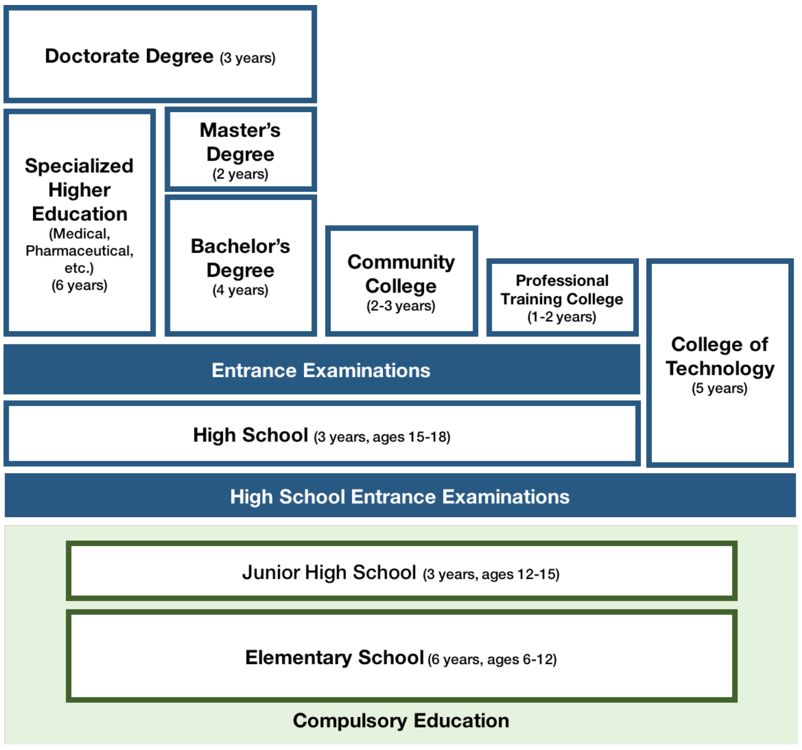

Doctorate Degree (3 years)
Specialized Higher Education (Medical, Pharmaceutical, etc.) (6 years)
Master's Degree (2 years)
Bachelor's Degree (4 years)
Community College (2-3 years)
Professional Training College (1-2 years)
College of Technology (5 years)
Entrance Examinations
High School (3 years, ages 15-18)
High School Entrance Examinations
Junior High School (3 years, ages 12-15)
Elementary School (6 years, ages 6-12)
Compulsory Education

Fig. 1. Education System in Japan.

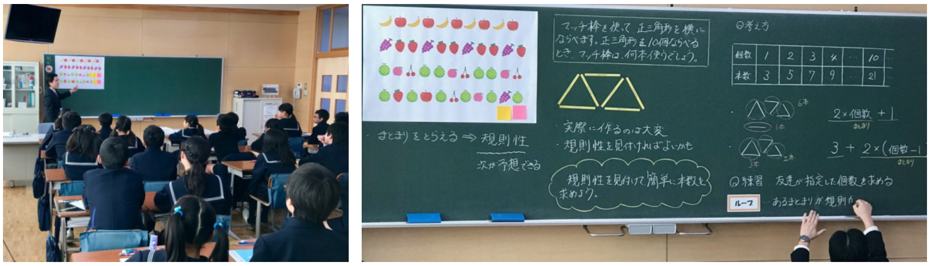


Fig. 2. Classroom in a primary school.

2015, there were about 20,601 primary schools, 10,484 junior high schools and 4,939 high schools (MEXT, 2017). There are several types of high schools including general course, specialized course (commercial studies course, technical studies course, and others) and integrated courses. Here, we report the informatics education in general high schools.

In elementary school, students learn mainly Japanese language, arithmetic, science, social studies, music, arts and handicrafts, homemaking, and physical education. The existing course of study does not include informatics education. Therefore, most teachers do not have training in teaching informatics. Each class is assigned a homeroom teacher. He or she teaches almost all subjects. In recent years, electronic blackboards were introduced to almost schools. Furthermore, some schools began to use tablets in the classroom. Fig. 2 shows a classroom in a primary school. In this class, the teacher lessons mathematics using informatics such as iteration.

In junior high school, students learn mainly Japanese language, mathematics, science, social studies and English language. Junior high schools have specialize teachers who teach their subjects. Students have different teachers for different subjects. All students also learn computer literacy and basic robot programming in "Technology", which is a branch of the subject "Technology and Home Economics". To study "measurement and control" section in "Technology" that covers basic robot programming, students purchase learning materials such as robots dedicated to line tracing. Fig. 3 shows some robots and sample program using in junior high schools. There are several kinds of robots. For example, Robot cars for line tracing such as the left picture in Fig. 3 have two

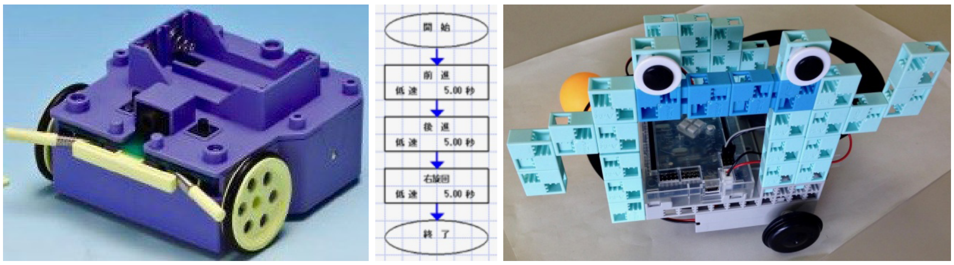


Fig. 3. Some robots and sample program using junior high school.

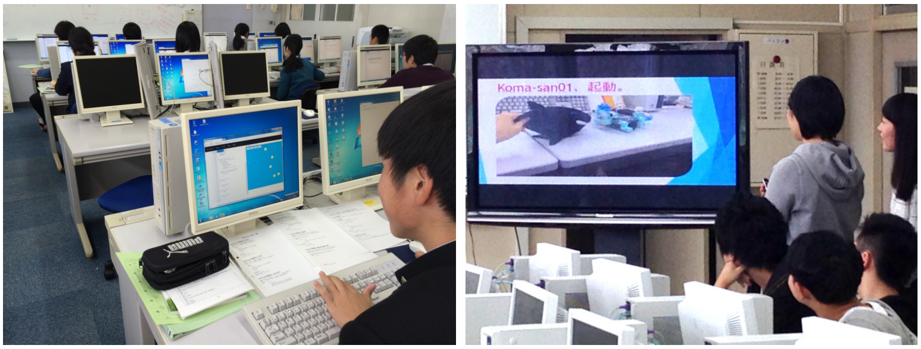


Fig. 4. Classroom in a high school.

motors, an infrared sensor and touch sensors. Robots such as the right picture in Fig. 3 are constructed by some blocks and CPU board (Arduino compatible). Students write programs using flowcharts, visual programming languages such as Scratch, and text-based programming languages such as Dolittle (Kanemune, 2005).

In high school, all students learn the subject "Information" from 2003. "Information" consists of two optional subjects named "Information Study for Participating Community (Society and Information)" and "Information Study by Scientific Approach (Information Science)". Students learn about programming in "Information Science". Schools are able to select either of the subjects to teach. About 80% of high schools teach "Society and Information" and about 20% of high schools teach "Information Science" (Kano, 2016). In other words, only 20% of students have an opportunity to get the education about "Information Science" and programming in high school. Fig. 4 shows a classroom in a high school.

## 3. Informatics Eeducation in New Curriculum

### 3.1. *Elementary School*

The New Course of Study is to be fully implemented at elementary schools from 2020. The course of study includes compulsory programming education. Students will learn logical thinking through programming experiences. However, there is no subject to teach informatics. Therefore, students will learn programming in the subject such as arithmetic and science or the Period for Integrated Studies, which is a period that has been allocated for cross-curricula study.

Teacher education is a major challenge for elementary school programming education. Elementary school teachers in Japan teach all subjects. In other words, all teachers have the possibility for teaching programming classes. Conducting programming education to 400,000 teachers until 2020 is difficult.

Consequently, it is crucial that schools, communities, local governments and other organizations such as academic societies and educational institutions join together in resolving these challenges. It is also important that textbook publishers develop useful textbooks for teaching programming and the governments publish case studies books. Additionally, local governments need to conduct the training for teaching programming and send ICT support staffs to schools for helping teachers.

### 3.2. *Junior High School*

The New Course of Study is to be fully implemented at junior high schools from 2021. In the subject "Technology", students will learn two types of programming, "measurement and control" and "network communication".

The challenge of junior high school is class hours to study informatics. In 1976, the lesson of the subject "Technology" was specified that 315 class hours in 3 years should be spent on the subject including "drawing", "woodworking", "metalworking","machinery", "electricity", "cultivation" and "total practice". However, "Technology" in the present Course of Study has been specified that 88 class hours should be spent on the subject including "craft", "energy", "cultivation" and "information". Table 1 shows the proposed content of "Technology". It is not enough for learning informatics deeply.

Table 1
Proposed contents of the subject "Technology" in the new course

| Technology |
|---|
| (1) Materials and Processing Technology<br>• Materials and Processing Technology as a Foundation for Society<br>• Problem-Solving using Materials and Processing Technology<br>• Social Development and Materials and Processing Technology |
| (2) Cultivation Technology<br>• Cultivation Technology as a Foundation for Society<br>• Problem-Solving using Cultivation Technology<br>• Social Development and Cultivation Technology |
| (3) Energy conversion Technology<br>• Energy conversion Technology as a Foundation for Society<br>• Problem-Solving using Energy conversion Technology<br>• Social Development and Energy conversion Technology |
| (4) Information Technology<br>• Information Technology as a Foundation for Society<br>• Problem-Solving using Computer Network<br>• Problem-Solving using Equipment Automation Technology<br>• Social Development and Information Technology |

### 3.3. *High School*

The new high school curriculum starts from 2022; however, the relevant year will apply only to the first grade. After that, it will be followed by yearly progress. Two selective courses of subject "Informatics" will be integrated into compulsory subject "Information I". A new advanced subject "Information II" will be prepared. Each subject has been specified that 70 class hours.

Table 2 shows the proposed content of "Information I" and "Information II". Both subjects have similar structures as follows: (1) is an introduction, and students learn about utilization of information technology in society. (2) Students learn to use information in their lives and work. (3) and (4) Students learn information science and technology including programming such as data processing, statistical processing and network programming, which are fundamental technologies of AI and IoT. Students learn a basic knowledge in "Information I" and learn an advanced knowledge of each fields in "Information II".

## 4. Conclusion

In this paper, we introduced the situation of informatics education and future educational policy in Japan. Previously, the importance of informatics education has not been recognized in Japan. However, with this educational reform, it starts to change the attitude

Table 2
Proposed contents of the subject "Informatics" in the new course

| Information I (compulsory subject) | Information II |
|---|---|
| (1) Problem-Solving in Information Society<br>• Utilizing computers in society<br>• Information security | (1) Development of Information Society and Information Technology<br>• Influence of information technology on society<br>• Use of information technology to solve problems |
| (2) Communication and Information Design<br>• Information and media<br>• Information design | (2) Communication and Information content<br>• Various communication using video, etc.<br>• Appropriate use of information contents |
| (3) Computer and Programming<br>• Data representation inside the computer<br>• Basics of programming<br>• Modeling and simulation | (3) Information and Data Science<br>• Statistics basics<br>• Processing according to characteristics of data |
| (4) Information Network and Data processing<br>• How the network works<br>• Server<br>• Database | (4) Information System and Programming<br>• Information system design<br>• Communication of computers<br>• Project management |

toward informatics education. In the New Course of Study, all elementary school students will experience programming and all high school students will learn informatics. Adoption of 'informatics' at college entrance examination is also being considered. The Science Council of Japan's Committee on Informatics defined a reference standard in informatics (Hagiya, 2015).

The Japanese Committee for the IOI (JCIOI) conducts the Bebras challenge in Japan. It also develops computer science unplugged (CSU) activities and holds a CSU event for primary school students every year. We consider that these activities help primary and secondary school teachers to teach informatics. For example, some schools has been started to adopt the Bebras challenge. Bebras is a challenge-contest on Informatics and computational thinking (Dagienė *et. al.*, 2015). At a school, students make Bebras tasks to deepen understanding related to information science after the challenge-contest. Several tasks which are designed by students were suggested via JCIOI to Bebras task workshop. Furthermore, they are selected as good Bebras tasks and used in some countries. Actually, It was reported that a problem made by a student was one of the most interesting tasks among students in Lithuania (Dagienė *et. al.*, 2016). We would like to spread such practice nationwide and to keep preparing for new education from 2020.

## Reference

Prime Minister of Japan and His Cabinet (2016). *Industrial Competitiveness*. [Accessed 2 May. 2017] Available at: `http://japan.kantei.go.jp/97_abe/actions/201604/19article6.html`

MEXT (2017). *Statistical Abstract 2016 edition*. [Accessed 2 May. 2017] Available at: `http://www.mext.go.jp/en/publication/statistics/title02/detail02/1379369.htm`

Kano, T. (2016). The present state and future prospects of informatics education (in Japanese). *Documents for Informatics Education*, 42, 3–7.

Kanemune, S., and Kuno, Y. (2005). Dolittle: an object-oriented language for K12 education. *EuroLogo2005*. Warszawa, Poland, pp.144–153.

Hagiya, M. (2015). Defining informatics across Bun-kei and Ri-kei. *Journal of Information Processing*, 23(4), 525–530. [Accessed 2 May. 2017] Available at: `https://www.jstage.jst.go.jp/article/ipsjjip/23/4/23_525/_article`

Dagienė, V., Pelikis, E., Stupurienė, G. (2015). Introducing computational thinking through a contest on informatics: problem-solving and gender issues, *Informacijos Mokslai*, 73, 55–63.

Dagienė, V., Stupurienė, G. (2016). Informatics concepts and computational thinking in K-12 education: a Lithuanian perspective. *IPSJ Transactions on Computers and Education*. 2(1), 1–8.

**S. Kanemune** is a professor at Department of Electro-Mechanical Engineering, Osaka Electro-Communication University. He received Ph.D. in Systems Management from Tsukuba University. He is a director of JCIOI since 2016. His has been working on database, information system, programming education and robotics education.

**S. Shirai** is an Assistant Instructor in the Department of Human Environmental Sciences at Mukogawa Women's University. She received her Ph.D. in Informatics and Mediology from Mukogawa Women's University in Japan. Her research focuses on human-computer-interaction, user experience design, e-Learning system, and informatics education.

**S. Tani** has been an executive director of the Japanese Committee for IOI (JCIOI) since 2009, and before this was a director of JCIOI since 2005. He received the BSc, MSc, and Ph.D. degrees from Waseda University, Tokyo, Japan, in 1987, 1990 and 1996, respectively. He is currently a professor at Department of Information Science, Nihon University. His research interests include computational complexity theory, computational topology, and complex network analysis.

Olympiads in Informatics, 2017, Vol. 11, 151–158
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.12

# Bulgarian Olympiad in Informatics: Excellence over a Long Period of Time

Emil KELEVEDJIEV[1], Rusko SHIKOV[2], Zornitsa DZHENKOVA[3]
[1]*Institute of Mathematics and Informatics, Bulgarian Academy of Sciences*
[2]*OpCo Sys Ltd, Yambol and Mathematics High School, Yambol*
[3]*Ivan Vazov High School, Gabrovo*
*e-mail: keleved@math.bas.bg, rusko@opcosys.com, zornica.dzhenkova@gmail.com*

**Abstract.** Competitions in informatics for secondary school students in Bulgaria have long traditions. The National Olympiad in Informatics started in 1985. Bulgaria is the founder and the first host of the International Olympiad in Informatics (IOI) in 1989. Our participation in IOI has brought us 24 gold, 41 silver and 31 bronze medals, with which Bulgaria ranks 5-th place in the all-time overall standings by medals up to the last IOI 2016. The paper presents the current situation and challenges in the area of Informatics competitions in Bulgaria. The structure of the competition system, including the Bulgarian National Olympiad in Informatics is outlined.

**Keywords:** competition in informatics, informatics education, Olympiad in informatics, training.

## 1. Introduction

### *Starting from 80's*

The programming contests for school students in Bulgaria started in early 80's of the past century (Manev et al., 2007). In the schedule of traditional Winter Mathematical Competitions, organized by the Union of the Bulgarian Mathematicians, a Programming tournament was included. The participants had to write a program in one of the languages FORTRAN or PL/1, that solve a given algorithmic task, to punch source on cards, compile it (computers was IBM/360 compatible machines) and to try to debug the program for 3–4 runs (no more runs were possible in limited to four hours contest).

In 1982 Bulgaria started to produce the Apple II-compatible machine Pravetz 8. Very soon many Bulgarian schools had at least one computer lab. So, each participant in Winter tournament had the possibility to work on an individual computer. The languages BASIC and Pascal replaced FORTRAN and PL/1. Evaluation of solutions in those years was pure manual and some quantity of marks was assigned for the style of programming.

Four years expertise from Winter tournament was enough for the Team of the Union of the Bulgarian Mathematicians to be able to organize a National Olympiad in Informatics (NOI). In 1985 the First NOI took place. This was a two day contest. In the first day contestants had to solve some theoretical task – concerning the algorithmic knowledge and knowledge of the programming language. The second day was similar to the Winter tournament – the contestants had to write and debug a program. Starting from the Second NOI, in each of two days contestants had to solve one task by writing a corresponding program. There was no special qualification for participating in the Final round of the first few NOI. Many schools organized its own contests to decide which students would be sent to the Final.

The NOI started with one age group in 1985. Now the contestants are divided in 5 age groups – E (4th–5th grade), D (6th–7th grade), C (8th grade), B (9th–10th grade) and A (master group). Contests in different groups have different duration – 3 hours for E, 4 hours for D and C, and 5 hours for B and A. Contests are purely conforming to the format of IOI. Does not matter in which group participates, each contestant has an own workplace and must solve three task of algorithmic type, writing the corresponding programs in the official language of IOI: C++.

## *Today's Structure of National Olympiad in Informatics (NOI)*

For more than 30 years the NOI of Bulgaria totally changed. Nowadays we have 3 rounds – Local/School (in February), Regional (in March) and Final/National (in April). The teachers of schools are free to prepare their own tasks for the Local round. For helping schools (especially in small villages) that are not able to prepare tasks, the National Committee (NC) proposes a set of sample tasks for this round. The round is not a formal qualification. The teachers, that evaluate the contestants, decide who is ready to participate in the Regional round.

Tasks for Regional and Final round are prepared by the NC. Regional round is organized in schools in one day with common start. Solutions of the pupils are sent immediately to the work groups of NC that evaluate and grade them with common set of test data. The round is a qualification for the Finals of the NOI, Where 120 students from all age groups participate.

The Final round is organized in a different town each year. There are two contests (in two consecutive days). The first 12 students from the Final round in group A form the long list of the Senior National Team for Balkan Olympiad in Informatics (BOI) and IOI, and similarly the first 12 students from the Final round in group C (aged less than 15.5 years) form the long list of the Junior National Team for Junior Balkan Olympiad in Informatics (jBOI).

### *National Tournaments*

In parallel with the NOI, 3 National Tournaments (NT) are organized by the Union of the Bulgarian Mathematicians and with the support of the Ministry of Education. The season starts with the Fall tournament in November. The traditional Winter tournament is in January or February and the final for the season is the Spring tournament in the beginning of June. The NTs are open, but to the level of certain quotas. Format of the contests in the NTs is the same as those of the NOI. But, in principle, tasks are more difficult, and frequently, during these contests, some experiments are made. Some new topics or types of tasks usually first appear in the NTs before to be given in the NOI.

## 2. Preparation and Selection of Students for the International Competitions

### *Control Contests for Qualifications*

After forming of the two extended national teams (for junior and senior age) as a result of the National round of the National Olympiad, to continue with the selection of the teams that will represent Bulgaria in the international competitions, we conduct several control competitions. The number of these competitions and their dates depend on the registration deadlines of the corresponding International Competitions.

The senior team has presented our country on the following competitions:

- International Olympiad of Informatics.
- Balkan Olympiad in Informatics.
- Central European Olympiad.
- Romanian Masters of Informatics.
- International Tournament in Shumen.

The junior team has presented our country on the following competitions:

- Junior Balkan Olympiad in Informatics.
- International Tournament in Shumen.

Moreover, the junior team is preparing to participate for the first time this year on:

- European Junior Olympiad in Informatics.

### *Training*

In about ten major cities in Bulgaria there are developed networks of out-of-school training through various forms of group organization. These groups are closely connected with mathematical high schools, where students are taught and are led by teachers, university lecturers and people from companies. It is very important to note the role of students' own motivation and self-preparation, which has become particularly possible in the last years due to the contribution of the Internet to giving access to numerous foreign websites, related to competitions in informatics, including many online competitions.

The full publicity of all materials related to Bulgarian competitions in informatics is a great encouragement. The Competitive Informatics Committee of the Union of the Bulgarian Mathematicians maintains the "Infos" website with the full information on all national and international competitions (Infos, 2017).

We conduct two summer training camps to prepare students from the extended national teams for juniors and seniors, and one national academy where all competitors ranked in the top 10 places in the overall rankings for each school grade are invited.

## *Achievements of the Bulgarian Competitors at the IOI*

Traditionally and for the entire period of time of IOI, the Bulgarian team shows a stable performance with a large number of medals. Fig. 1 illustrates the distribution of the medals of the Bulgarian participants from the First IOI up IOI 2016. Individual achievements with ranges of competitors are shown in Fig. 2, where our best performance so far is the second place for in the "all time" standings. The statistic source is (International…, 2017).

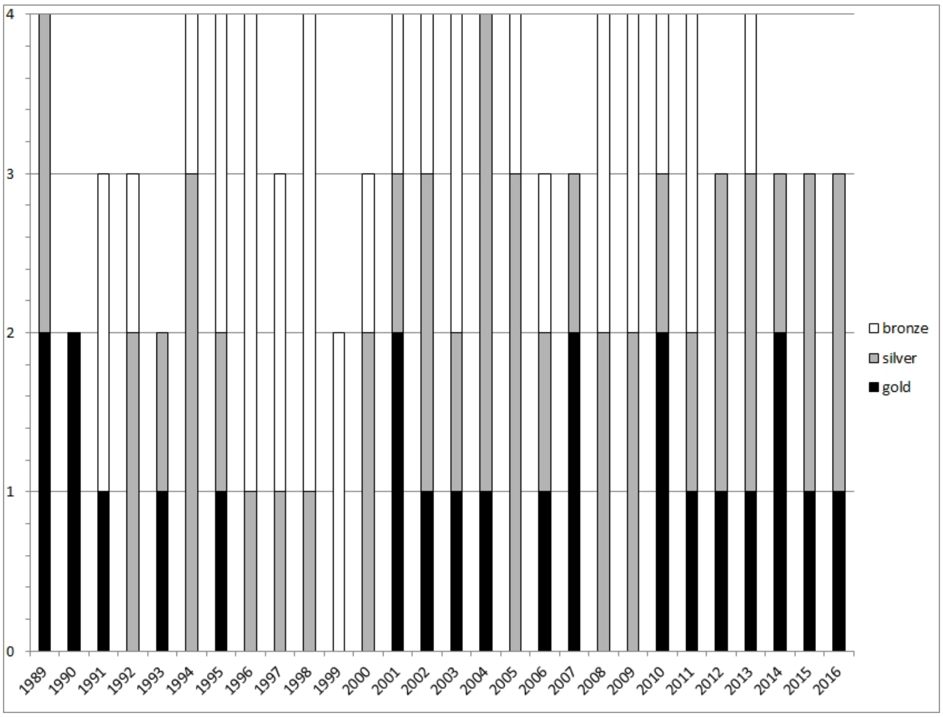


Fig. 1. Bulgarian participants medal achievement at IOI.

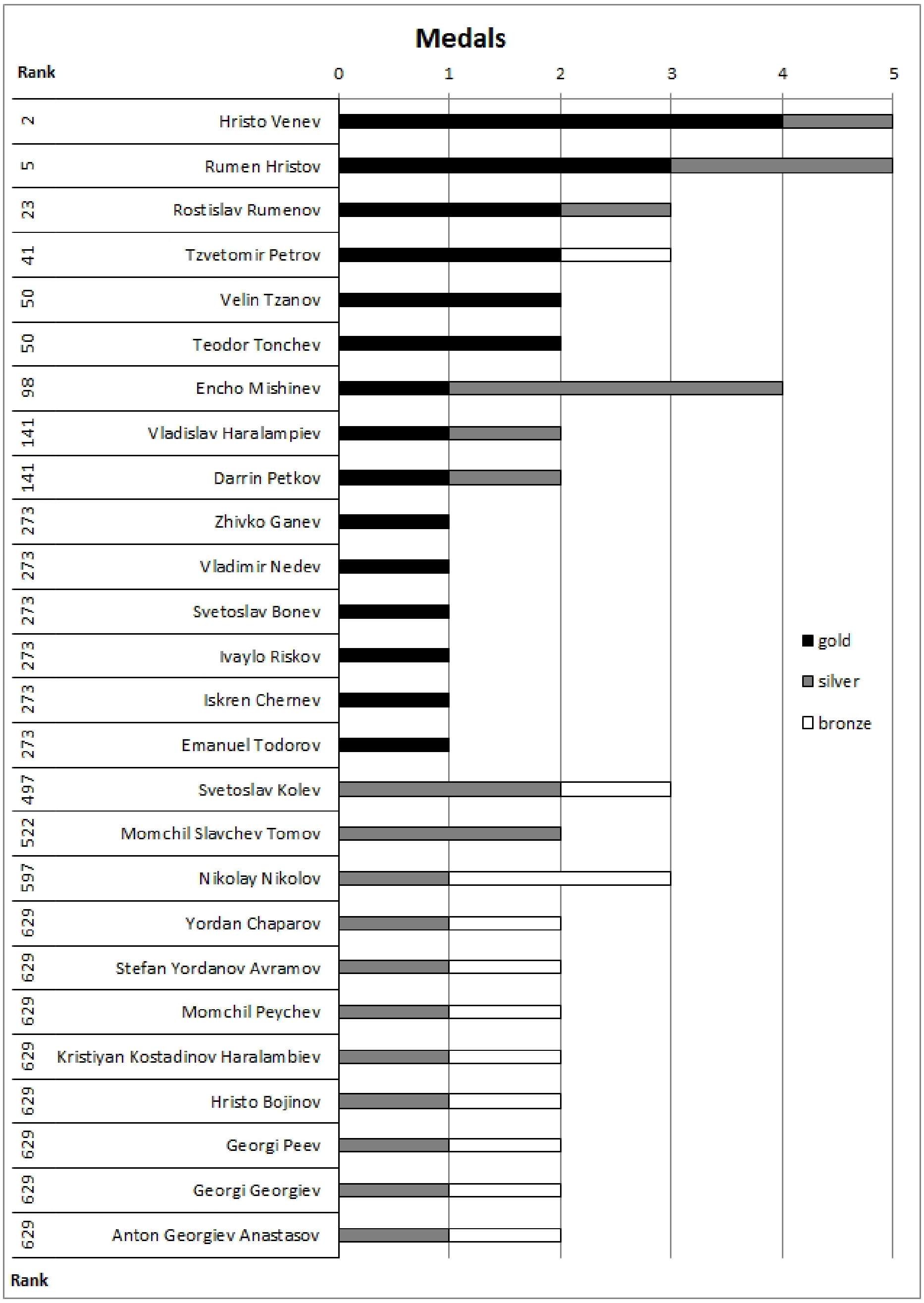


Fig. 2. Bulgarian achievements in the individual ranking for all IOIs up to IOI 2016.

*Educational Content*

Regarding the curriculum, which we use in preparing and defining the tasks given to the national competitions, it is aligned with the IOI contest materials. Fundamentals is our contribution in defining the material for beginners training starting at 11–12 age children, because this material does not exist at IOI.

**Group E (4–5th Grade, 11–12 Year Old)**

1. C++ Programming Environment. Simple data types. Input and Output.
2. Control structures and operations in the C++ . Conditional statements.
3. Cycles. Embedded cycles. Functions in C++. Strings.
4. Tasks related to dates and time.
5. Concept of a one-dimensional array.

**Group D (6–7th Grade, 13–14 Year Old)**

1. One-dimensional arrays and basic tasks with them. Introduction to Sorting Algorithms. Tools for strings and searching.
2. Divisibility of integers. Euclid's algorithm and its applications. Prime numbers. Sieve of Eratosthenes. Numeral systems.
3. Implementation for long integers. Random numbers.
4. Two-dimensional arrays and table processing. Structures in C. Arrays of structures.
5. Initial knowledge in computer geometry. Rectangles with sides, parallel to the coordinate axes. Square meshes, labyrinths and domains.
6. Standard library (including introduction to STL) and STL sorting tools.
7. Data structure: stack and queue.
8. Concept of recursion. Backtracking.
9. Fast algorithms for searching.
10. Introduction to Dynamic Programming.

**Group C (8th Grade, 15 Year Old)**

1. Extended Euclid's algorithm and applications.
2. Games with strategies, concerning parity and symmetry. Combinatorial games. Nim. Board games.
3. Bitwise operations and applications.
4. Dynamic programming: one-dimensional and two-dimensional tasks. Longest common subsequence. Shortest super-sequence.
5. Graphs: presentation and traversal (DFS, BFS). Directed graphs. Shortest path in graphs. Binary trees and trees for search. Pyramid data structure.
6. Algorithmic geoetry: oriented triplet of points and applications.
7. Combinatorial configurations and counting.
8. Arithmetic expressions: representation, computation and transformation.

**Group B (9–10th Grade, 16–17 Year Old)**

1. STL Library: Containers and iterators, basic algorithms. Hashing.

2. Permutations: basic properties. Combinatorial configurations: encoding and decoding. Numbers of Catalan. Structures for representation of sets. Gray's codes. Decomposing of sets and numbers.
3. Algorithmic geometry: mutual position of points and straight lines. Polygons. Convex hull. Closest and most distant points. Diagrams of Voronoy.
4. Graphs: Bi-connectivity, strong connectivity, Euler tours and Hamilton cycles, Minimum spanning trees, matching in graph, critical path method, Maximum flow. Coloring. Planar graphs. Geometric graphs. Complex tree structures: Fenwick tree, segment trees.
5. Dynamic Programming: Profiles. Recurrent relations and recursion. Conversation of recursive programs.
6. Strings: search by pattern, distances. Effective structures and algorithms for strings. Data Compression: Huffman Codes. Formal grammars and automata.
7. Games: minimal strategies, alpha-beta pruning. Reactive games.
8. Systems of linear equations and integer solutions.

**Group A (Master Group, 11–12th Grade, 18–19 Year Old)**

1. All materials from the previous groups combined in complex tasks of the level of the IOI.

## 3. Conclusion

The institutions that organize and support the overall running of the Bulgarian national competitions and our participation in the international events are:

- Ministry of Education.
- Union of Bulgarian Mathematicians.
- Institute of Mathematics and Informatics at Bulgarian Academy of Science.
- National Sciences Olympic Team Association.

### *Alumni Involvement*

With the increasing number of competitors from year to year, stronger and more solid alumni were built, especially since the establishment of National Sciences Olympic Team Association. Since most of them are still active participating in many programming competitions at the university level, they are up to date and exposed to the various new problems and materials that are suitable for training. Some of the alumni have worked and contributed at some prestigious IT related companies and some others have successfully established their own startups which are recognized both nationally and internationally.

Although we are supported by alumni, we are facing to a problem of shortage of young people, which are to be team leaders, coaches and task writers. Another problem that is appropriate to mention is the insufficient public attitude for recognition of the status of contestants and coaches in competitive informatics.

*Research on the Work of the Participants*

After having accumulated enough tasks previously given in competitions, it becomes possible to start an attempt for classification and introducing measure of difficulty as is considered in (Kelevedjiev, Dzhenkova, 2008, 2009, 2012)]. The authors of tasks for the Bulgarian competitions could find useful information about the history of tasks from the previous competitions in order not to duplicate or sometimes intentionally repeat some kinds of problems. In more broad sense, the study of the keywords might be applied for initializing appropriate changes and improvements in the national curriculum which is used now as a recommendable list of themes in all the set of local out-of-class forms for young student preparation in Bulgaria.

## References

Infos. *Website for Competitions in Informatics. Union of Mathematicians in Bulgaria* (in Bulgarian). Visited on May 16, 2017: `http://www.math.bas.bg/infos`

*International Olympiad in Informatics – Statistics* (2017). Visited on May 16, 2017: `http://stats.ioinformatics.org`

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

Kelevedjiev, E., Dzhenkova Z. (2009) Tasks and training the intermediate age students for informatics competitions. *Olympiads in Informatics*, 3, 26–37

Kelevedjiev, E., Dzhenkova Z. (2012) Competitions' tasks in informatics for the "Pre-Master" group of school students. *Olympiads in Informatics*, 6, 178–191.

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

**E. Kelevedjiev** is a researcher in the Institute of Mathematics and Informatics at the Bulgarian Academy of Sciences. His field of interests includes algorithms in computer science, operation research, digitization techniques, etc. He is chairman of the Bulgarian National Committee for Olympiads in Informatics and team leader of the Bulgarian teams for many international competitions.

**R. Shikov** is CEO of OpCo Sys Ltd, Yambol and head of a school for out-of-class work for the preparation of students for informatics competitions at the mathematical high school in Yambol. He is author of tasks for the Bulgarian national competitions in informatics and is involving in organizing competitions and training camps.

**Z. Dzhenkova** is a teacher. She is author of tasks for the Bulgarian national competitions in informatics and is involving in organizing competitions and training camps. She is coauthor of a manual for beginner's training in competitions and Olympiads in Informatics.

Olympiads in Informatics, 2017, Vol. 11, 159–166
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.13

# Kyrgyzstan Olympiad in Informatics: Training Students, Conducting the Olympiad and Using Contest Management System

Zamira MAKIEVA[1], F. KHALIKOV[2], Ruslan ALIMBAEV[3]
[1]*Software Engineering Department, Kyrgyz State Technical University, Bishkek*
[2]*Software Engineer at IT-Attractor Resource Centre, Bishkek*
[3]*Software Engineer at Booking.com, Amsterdam*
*e-mail: z.makieva@gmail.com, thesomeq@gmail.com, olymp96@gmail.com*

**Abstract.** The Article analyzes current state, problems, challenges, and opportunities of the Olympiad in Informatics activities in Kyrgyzstan. From the practical point of view, authors suggest mechanisms and instruments with a purpose of improving several main criteria of successful performance by Kyrgyz students at international informatics contests, including the most prestigious and respected one - International Olympiad in Informatics (IOI). One of the significant practical tools described and analyzed in the paper is the Contest Management System (CMS). CMS is an automated platform that allows not only to check and verify the solutions proposed by the contestants, but also optimize work and organizational processes of the olympiads at local and national levels. The authors suggest that further development of CMS will allow not only to increase the quality and efficiency of evaluating proposed solutions at the contests, but also improve overall performance of the Kyrgyz students at local and international olympiads in informatics.

**Keywords:** informatics, olympiad, training, grading system, contest management system.

## 1. Introduction

Olympiad in informatics of school students are conducted annually in Kyrgyzstan as well as in many other countries. Currently olympiads in informatics are organized on a four-round basis: 1) school, 2) district, 3) region (the cities of Bishkek and Osh have a special distinct status), and 4) national levels. The first and second rounds are carried out by local school teacher and are an open-call competition for all students, whereas the third and fourth rounds are prepared and organized by a National Commission.

A quota for certain number of participants is allocated for every round of the competition. However, there has been a common criticism that such an approach leads to having promising students dismissed at the early rounds before they make it to the regional level. Certain schools or districts have traditionally had more talented participants than allowed by the quotas, which practically lead to having them replaced by weaker participants from other schools and districts.

In 2017 the third round of the olympiad in informatics hosted 158 students from 9 regions, whereas in the fourth round only 19 students remained (2 students from every region and a last year winner that participates directly in the finals skipping preliminary rounds). The map below illustrates 7 regions, the cities of Bishkek and Osh owning a regional status, and the number of participants from each region (Fig 1.)

Kyrgyzstan high school students are allowed to code using C++, Pascal and QBasic at olympiads in informatics. These programming languages are taught at Kyrgyz schools. Usually participants using C++ and Pascal succeed to make it to the final round. Unfortunately, it is yet impossible to exclude Pascal and QBasic because students in the regions currently cannot learn C++ and other contemporary programming languages at schools due to lack of qualified teachers.

For many years in Kyrgyzstan assessing olympiad tasks was carried out by a simple launch method. The programs were run with a manual input of data or test files, and the results were compared with initial data afterwards. Obviously this method is inadequate for manual verification results in a poor quality of assessment due to the following:

- Number of possible tests is limited.
- Size of input and output data is limited.
- Tests are time-consuming.
- Programs can be tested only at the end of the olympiads (lack of interactivity).
- Assessing programs can be compromised by inputting wrong initial data during program tests.

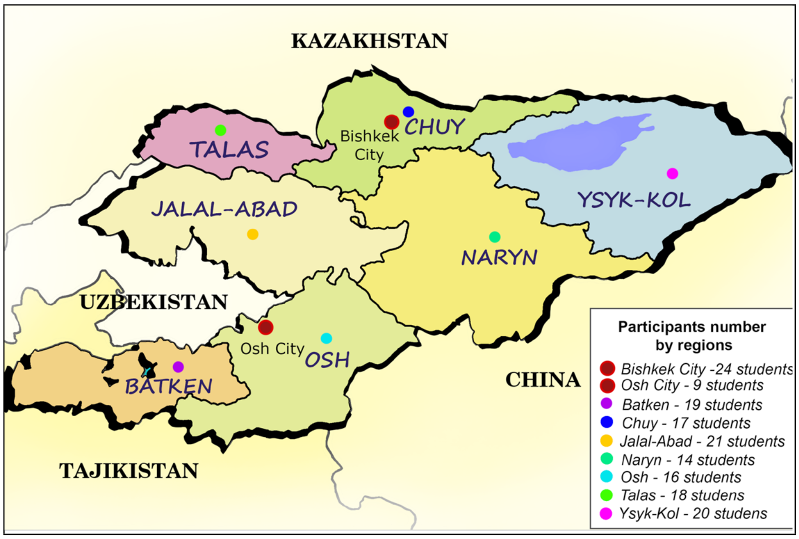


Fig 1. The number of a third round participants from each region of Kyrgyzstan in 2016–2017.

- Optimality of a proposed task solution (in terms of a program's performance duration and memory usage) is hard to evaluate.
- Task solutions are evaluated individually and subjectively (which may lead to corruption).

## 2. Problems and Improvement

Kyrgyzstan has participated in IOI since 2000. Kyrgyzstan's national teams haven't often demonstrated outstanding performances. The best result was achieved in 2005 (Table 1), whereas in other years Kyrgyzstan has always been at the bottom of the tournament table of IOI (International…, 2017).

One of the reasons of Kyrgyzstan's poor performance at the IOI contests is a decline in quality of education and economy that was partly triggered by 2 consecutive revolutions in 2005 and 2010.

The attitude to olympiad movement started to change from 2015 on, and Kyrgyzstan began to cooperate with neighboring Kazakhstan and Russia in preparing students for the IOI contest. Kyrgyz students attended summer schools: in Almaty, Kazakhstan, in 2015 and in Innopolis (Tatarstan, Russia) in 2016. This year they have applied to the Summer Computer School in Kostroma, Russia.

Kyrgyz students have started demonstrating better results in different contests in informatics. In APIO (Asia Pacific Informatics Olympiad) they received honorable mentions, whereas in IZho (International Zhautykov Olympiad) Kyrgyzstan participants usually get gold, silver, and bronze medals. In All-Russian olympiads in Informatics for school students Kyrgyzstan's team got diplomas of 2nd and 3rd degrees.

Development and growth of information technologies and internet, which allow learning and training for any contest based on various online grade systems, create ground for assisting the olympiad movement. Russian websites like acm.timus.ru, codeforces.com, acmp.ru, and e-maxx.ru, Ukrainian website like e-olymp.com as well as international sources like csacademy.com and usaco.org are very popular resources among Kyrgyzstan students. A similar Kyrgyz website olymp.krsu.edu.kg was made for conducting college student olympiads according to ACM ICPC rules and regulations. Currently this website is being used for conducting regular local informatics contests for high school students as well as for learning and training.

Table 1
The best results of IOI (period 2000-2015)

| Year | Contestant | Score | | Rank | |
|---|---|---|---|---|---|
| | | Abs. | Rel. | Abs. | Rel. |
| 2005 | Igor Goroshko | 319 | 53.17% | 110/276 | 60.51% |
| 2004 | Aleksey Baryshnikov | 265 | 44.17% | 143/291 | 51.2% |
| 2000 | Andrey Mokhov | 250 | 35.71% | 129/278 | 53.96% |

Moreover, improvement in selecting participants for IOI has appeared. An automated Contest Management System has been used at the 4th stage of the olympiad since 2015. The system had been developed and was successfully applied in Kyrgyzstan at high school olympiad that had a more flexible IOI-standard task evaluation system compared to that of college student olympiads.

Overall, an approach to olympiad in informatics has changed in following direction:

1) Integrating an automated Contest Management System.
2) Having students to attend Preparatory Schools for Informatics Olympiads.
3) Intensive self-study by enthusiastic students.

Abovementioned techniques resulted in better performance of Kyrgyz national teams on IOI and Kyrgyz students rose from the bottom of the result table. Ranks of Kyrgyz students on IOI from year 2011 to 2014 demonstrated negative dynamics, whereas performances in years 2015 and 2016 progressed prominently (Fig. 2). On IOI-2016 (Ka-

Table 2
Rank of Kyrgyz students on IOI (2011-2016)

| Year | 1 student | 2 student | 3 student | 4 student |
|---|---|---|---|---|
| 2011 | 12,87% | 14,85% | 16,83% | 22,11% |
| 2012 | 13,23% | 17,74% | 21,61% | 26,13% |
| 2013 | 8,36% | 14,05% | 14,38% | 28,43% |
| 2014 | 2,89% | 9,00% | 10,61% | 15,43% |
| 2015 | 13,04% | 25,78% | 35,09% | 44,10% |
| 2016 | 21,43% | 25,32% | 42,53% | 51,95% |

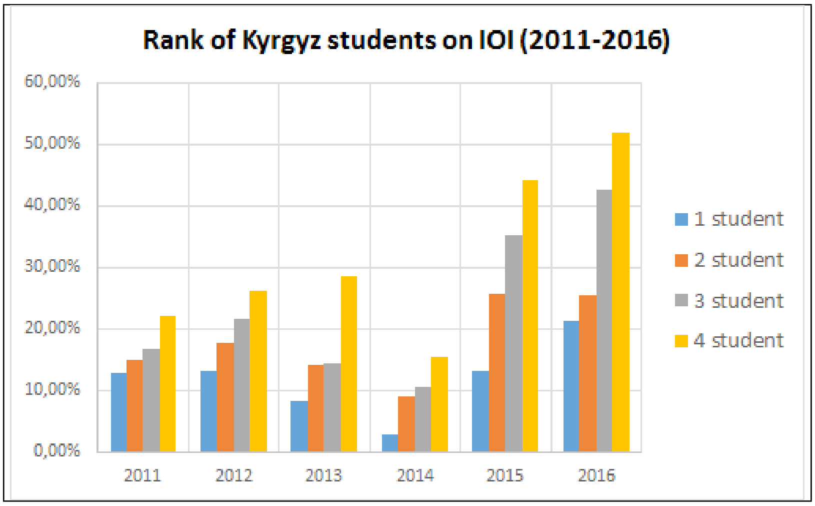


Fig. 2. Rank of Kyrgyz students on IOI (2011–2016).

zan, Russian Federation) Azret Kenzhaliev won a bronze medal to be the first winner from Kyrgyzstan in an 11-year-old break.

Alumni of our university and former participants of IOI are providing great assistance in developing and improving an automated Contest Management System, as well as preparing tasks for student contests.

## 3. About Contest Management System

Development of the Contest Management System began in 2014 at the Software Engineering Department of Kyrgyz State Technical University. Due to an urgent need of introducing an automated task evaluation system, the CMS was piloted in the 3rd round of the high school Olympiad (on a region level) in Bishkek in February 2015. The pilot launch of the automated system was then completed in the fourth round of the olympiad in national finals in March 2015. Updated versions of the Contest Management System were successfully applied in all rounds of Olympiads in 2016 and 2017.

CMS is written in Python with use of Django framework. MySQL is used as DBMS.

CMS used in Olympiads in Kyrgyzstan was then implemented into a web application. The application has a 3-level architecture. User's web browser - whether it's on PC, mobile or tablet - is used as a client. uWSGI functions as an application server (`https://uwsgi-docs.readthedocs.io/en/latest/`) and manages the application in Python/Django. Nginx (`https://www.nginx.com/resources/wiki/`) is used as a web server.

Direct interaction between the client and the database server is obviously unacceptable. Therefore, any interaction is possible only via application server. Detailed description of the system architecture is given in (Makieva, 2016; Makieva, 2016a).

Task description is provided in two languages because Kyrgyz is the state language of Kyrgyzstan, whereas Russian is official. All the design, management element captions and the system is menu are given in 3 languages: Kyrgyz, Russian, and English.

The system has 2 modes of task evaluation: IOI standard for managing high school student Olympiads and ACM standard for managing college student olympiads.

Testing programs developed by the Olympiad participants is automated. Therefore, the programs have to match the formats of input and output data described in every given task. Time and memory limits are set for the programs in every given task. As the participants finalize their programs, they can send their solutions for check. Once the system has checked the solution, the participant sees the testing result table on the screen with standard messages: Compiling, Proceeding, Compilation Error, Runtime Error, Time limit exceeded, Wrong Answer, and Accepted.

The system supports various types of task evaluation and checking. The process is implemented with program checker. There are 4 types of checkers:

1) **Exact answer**: there can be any amount of data in the answer, but their order and value have to be an absolute match.

2) **Answer with given precision** is used to check tasks whose answer is one or more real numbers. The system checks whether an absolute or relative error in the participant's answer does not exceed an allowed error margin.
3) **Multiple correct answers** are usually used for tasks that may have more than one correct answers. A well-known "8 Queens" chess problem or a search of a maze exit are examples of a task with multiple correct answers.The checker writes down the solution verification results into a special file, which the testing system then uses to check the solution.
4) **Interactive problems:** The interactor program can process conclusion of the participant's solution to the standard input stream and can also submit its data to the entrance of participant's solution using program's standard stream. Once communication is completed, the interactor writes the result to a special file that the system uses to display the result.

The system supports such programming languages as C++, Pascal and QBasic. Other languages can be added to the system if necessary.

Automated verification system provides following opportunities:

- To use large amount of tests - up to dozens of test per each problem, which increases reliability of problem checking.
- To process large amounts of input and output data, which allows to check performance of the tested algorithm with large sets of numbers (up to $10^9$ elements) and long strings.
- Checking and verifying a task is processed within several seconds depending on the number of tests. It allows performing an interactive check: a participant sends the program for check and can see the evaluation result till the end of the Olympiad, thus, having an opportunity to make changes to the proposed solution.
- Chances for errors at input or check are eliminated.
- The algorithm can be checked for optimality by analyzing time and memory it requires to work.
- All the results - including source code of the solution, time the program is sent, number of generated tests, etc - are saved to and stored in database on a single server.
- Reports with contest results are generated in a shorter periods of time.
- A large archive with test problems and solutions can be stored and used for learning and practicing.

## 4. Conclusion and Future Work

Application of an automated Contest Management System resulted in improving the quality of program check and, thus, resulting in a better preparation of high school students, as well as selecting the best contest participants by objective criteria.

From 2018 on it is planned to apply the CMS in the 3rd round of the high school contests, which will allow to select best contestants from the regions of Kyrgyzstan and see the an overall rank of all the contestants on a national level. Also this will allow to increase the possible number of contestants in the 3rd round throughout Kyrgyzstan.

Moreover, for the purpose of developing the olympiad movement and improving preparation of high school students for the informatics contests State Methodical Commission of the Kyrgyz olympiad in informatics has recommended to do the following:

- Oblige secondary school teachers to attend refresher courses.
- Teach secondary school students programming languages of C++ and Pascal and exclude QBasic and Pascal ABC from the curriculum, to be able to participate in international informatics contests.
- Set the 5th grade at schools as the latest year to introduce informatics into the curricula and to continue teaching till 11th grade inclusively.
- Remove any age and year of study limitation at the national informatics contest in order to maintain a maximum coverage of school students.
- Use the automated Contest Management System at the informatics Olympiads in 3rd and 4th rounds throughout the country.
- Create an official website of national high school Olympiad with an automated solution checking system for students to be able to practice before the Olympiad.

## References

International olympiad in Informatics – Statistics (2017). URL:
`http://stats.ioinformatics.org/results/KGZ`

Makieva, Z. (2016). Development of automated verification system for programming olympiad tasks. *News of Kyrgyz State Technical University named I. Razzakov*, 2(38), 54–61. (In Russian).

Makieva, Z. (2016a). The Automatic Testing System for Checking Tasks of Programming. Fundamental and applied problems of science. *Volume 2. – Materials of the XI International symposium of the Kyrgyz section.* Moscow: Russian Academy of Sciences, 18–24. (In Russian).

**Z. Makieva,** Associate Professor of the Kyrgyz State Technical University, Contest Management System development team leader. She has been a member of the State Methodical Commission of the National Olympiad in Informatics for high school students since 2014. Kyrgyzstan Informatics Olympiad team leader and deputy team leader since 2014. Coach and advisor to Kyrgyz State Technical University's team for ACM-ICPC International Collegiate Programming Contest since 2005.

**F. Khalikov,** developer of the Contest Management System. Developed first version of the system based on ACM-ICPC rules. Studied Software Engineering at Kyrgyz State Technical University and participated in ACM-ICPC multiple times. Currently, a web-developer at IT-Attractor Resource Centre.

**R. Alimbaev,** developer of the Contest Management System. Significantly refactored the system and made it suitable for IOI contests. Studied Software Engineering at Kyrgyz State Technical University and participated in ACM-ICPC multiple times. Currently works as Software Engineer at Booking.com, Amsterdam.

Olympiads in Informatics, 2017, Vol. 11, 167–170
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.14

REVIEWS, COMMENTS

# A New Book on Competitive Programming

Antti LAAKSONEN
*Department of Computer Science*
*University of Helsinki*
*e-mail: ahslaaks@cs.helsinki.fi*

**Abstract:** This paper presents a new book on competitive programming: *Competitive Programmer's Handbook*. The purpose of the book is to provide a modern introduction to competitive programming, and the book is especially intended for future IOI and ICPC participants.

**Keywords:** competitive programming, book

## 1 Introduction

While the popularity of competitive programming is growing every year, there are not many books devoted to the topic. Thus, in 2013, I started the project of writing a new book on competitive programming. The purpose of the book is to give a thorough introduction to modern competitive programming and to be accessible to readers without background in programming contests. Initially I wrote the book in Finnish, and I had to rewrote the entire book several times before the result was satisfactory. Then, in 2016, I decided to translate the book into English, because most people in the competitive programming community can't read Finnish. The title of the book became *Competitive Programmer's Handbook* (in Finnish: *Kisakoodarin käsikirja*).

Of course, before writing the book, I carefully studied the existing books on competitive programming. In 2003, Skiena and Revilla wrote the pioneering book in the field, *Programming Challenges*. Then, between 2010 and 2013, the Halim brothers published three books called *Competitive Programming 1–3*. My book would resemble those books, but it would also contain topics that are not discussed in them. There is also the book *Looking For a Challenge?* that was distributed during the IOI 2012. This book contains a collection of difficult problems from Polish contests, so it was targeted to a different audience.

Actually, the book is still under construction, but it is almost ready and the current version of the book is available online at:

`http://cses.fi/book.html`

I have already received a large amount of feedback from the competitive programming community that have greatly improved the quality of the book, and I appreciate all comments and suggestions regarding the book.

## 2. Book Contents

The book is divided into three parts, each of which contains ten chapters. The book covers almost all topics in the IOI syllabus, and also many topics outside the syllabus that may appear, for example, in ICPC contests.

Here is an overview of the contents of the book:

*Part 1: Basic techniques*

- the concept of time complexity
- sorting algorithms
- C++ data structures
- algorithm design techniques (complete search, greedy algorithms, dynamic programming)
- amortized analysis
- processing range queries on arrays
- using bit operations in algorithms

*Part 2: Graph algorithms*

- graph traversal (depth-first and breadth-first searches)
- shortest paths (Bellman–Ford, Dijkstra, Floyd–Warshall)
- tree algorithms (diameter, minimum spanning tree, queries on trees)
- topological sorting and strongly connected components
- Eulerian and Hamiltonian paths
- maximum flow/minimum cut theorem and its applications

*Part 3: Advanced topics*

- mathematical topics (number theory, combinatorics, matrices, probability, number theory)

- string algorithms (trie, string hashing, Z-algorithm)
- square root algorithms
- advanced segment trees
- geometric and sweep line algorithms

The primary purpose of the book is to *teach* the above topics to a reader. However, the book also attempts to give proper references to the techniques in the scientific literature.

## 3. Future Plans

My plan is to release the final version of the book this year (2017). Since many people have requested for practice problems, there will also be a collection of such problems available online.

It is clear that there are still a large number of more advanced topics that the book doesn't cover. Examples of such topics are as follows:

- finding bridges and articulation points in graphs
- tree decompositions
- dynamic programming optimization
- advanced string algorithms (e.g. suffix arrays)
- minimum-cost maximum flows
- mathematical topics (Gaussian elimination, advanced number theory, linear programming)

Thus, there would be material for even *another* book on competitive programming that may appear someday.

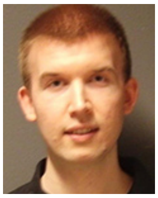
## References

Diks, K. *et al.* (2012). *Looking for a Challenge? The Ultimate Problem Set from the University of Warsaw Programming Competitions*.

Fori ek, M. *et al. The International Olympiad in Informatics Syllabus*. `https://people.ksp.sk/~misof/ioi-syllabus/`

Halim, S., Halim, F. (2013). *Competitive Programming 3: The New Lower Bound of Programming Contests*. Lulu.

Skiena, S., Revilla, M. (2003). *Programming Challenges: The Programming Contest Training Manual*. Springer.



**A. Laaksonen** received his PhD in Computer Science from the University of Helsinki. He is one of the organizers of the Finnish Olympiad in Informatics, and a coach and a team leader of Finnish BOI and IOI teams. He is the author of the book *Competitive Programmer's Handbook* and one of the developers of the CSES contest system.

Olympiads in Informatics, 2017, Vol. 11, 171–173
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.15

# First European Junior Olympiad in Informatics

Krassimir MANEV[1], Biserka YOVCHEVA[2]
[1]*Dept. of Informatics, New Bulgarian University*
*21, Montevideo str., 1618 Sofia, Bulgaria*
[2]*Konstantin Preslavsky University of Shumen*
*115 Universitetska str., 9700 Shumen, Bulgaria*
*e-mail: kmanev@nbu.bg, bissy_y@yahoo.com*

It is out of doubt that computer programming is "the fundamental activity of CS" (Ben-Ari, 2015) and a core of the preparation of the software engineers of all levels and branches. Unfortunately, there is no much place for teaching Informatics and Information Technologies in the secondary school curriculum in most of the countries of the world (Ackovska *et al.*, 2015). Competitions in programming, including International Olympiad in Informatics as well as different regional contests for secondary school students, are one of the possibility to start preparation of students from secondary schools for carrier in the domain earlier.

It is quite reasonable the question: When to start teaching of programing for young people with corresponding interests? Almost half of the top 100 contestants of IOI have 3 or more participations. So in age 15 or less these students were prepared to win an IOI medal, i.e. they started to learn programming and algorithms in age about 11–12. In some countries students of this age compete in programming contests. For most of them it is difficult to enter even the long lists of their national IOI teams and to participate in some of the international contests.

In 2007 Serbia organized the First Junior Balkan Olympiad in Informatics (JBOI) for school students of age less than 15.5 years (at July 1 of 2007) with Regulations similar to that of the Balkan Olympiad of Informatics (2 days, 3 tasks for 4 hours each day). In 2008 Bulgaria made the second JBOI. Then the contest was organized not each year because of lack of a host country. In such years an alternative 1 day contest of junior teams of some Balkan countries was organized inside the International Tournament in Informatics "John Atanasov" held each November in Shumen, Bulgaria. The effect of JBOI's was resumed and the results published in (Yovcheva *et al.*, 2009). It becomes apparent that organizing of competitions for students of age less than 15.5 years leads to their better training in programming, and attracts more students of this age to activities in the domain. Medalists of JBOI are very successful in IOI and later in ICPC.

Because the number of participating countries in JBOI is small the idea for extending JBOI to an European contest in programming for juniors was promoted. And this year, finally, the idea will be implemented.

The first **European Junior Olympiad in Informatics**, EJOI 2017, will be held in Sofia, Bulgaria, from September 7th to September 13th, 2017. In order to participate in EJOI 2017, students should be born after **December 31st 2001**. EJOI is an individual programming contest for young programmers from **Council of Europe countries**. The Olympiad will be similar to the International Olympiad in Informatics. Each country can participate with a team of 4 students at most, elected by the procedure adopted in their country. There will be 2 contest days with 3 tasks each day that have to be solved for 4 hours. The programming languages will be C/C++ and Java, the operating systems – Windows and Linux. Grading system of EJOI will be CMS. Translation of the task will be performed with the translation system of IOI.

The President of Republic of Bulgaria, Mr. Rumen Radev, as well as Mr. Tibor Navracsics, the Commissioner of European Union for Education, Culture, Youth and Sport, will be Patrons of the Olympiad.

EJOI will provides a great opportunity for youngest programmer to demonstrate their abilities in Informatics, to exchange knowledge and to enhance cross-cultural contacts in school education. Coming to EJOI 2017, they will have the chance to make new friends, to visit a friendly country, and to discover the culture of Bulgaria.

At `www.ejoi.org` it is possible to find: the proposal for Rules of the Olympiad, links to sites of the previous JBOI and tournament with used tasks (that could be used as an indication of kind of task that have to be expected till a Curriculum of EJOI is developed), list of the countries that would like to participate and other information about the contest.

## References:

Ackovska, N, Németh, Á.E., Stankov, E., Jovanov, M. (2015). Creating an international informatics curriculum for primary and high school education. *Olympiads in Informatics*, 9, 205–212.

Ben-Ari, M. (2015). In defense of programming. In: *Proc. of the ACM Conference on Innovation and Technology in Computer Science Education*, Vilnus, 2. `http://dx.doi.org/10.1145/2729094.2742581`

Yovcheva, B., Momcheva, G., Petrov, P. (2009). jBOI – one more possibility for increasing the number of competitors in informatics. *Olympiads in Informatics*, 3, 167–173.

**Kr. Manev** is a professor of Discrete mathematics and Algorithms in New Bulgarian University, PhD in Computer Science. He has published 75 scientific papers and more then 30 textbooks in the field of Informatics and Information technologies. Member of Bulgarian National Committee for Olympiads in Informatics since 1982 and President of NC from 1998 to 2002; member of the organizing team of IOI'1989 and IOI'1990, Chairmen of IOI'2009 and Leader/Deputy Leader of Bulgarian team for IOI in 1989, 1998, 1999, 2000, 2005 and 20014. From 2001 to 2003 and from 2011 to 2013 he was elected member of IC of IOI, since 2005 to 2010 represented in IC the Host country of IOI'2009. Now he is a President of IOI for the period 2010–2013.

**B. Yovcheva** is a senior lecturer in informatics in The Konstantin Preslavski University of Shumen and a director of A&B School of Shumen, PhD in Pedagogy of computer science education. She is Leader/Deputy Leader of the national team of informatics for 15.5 years of Bulgaria, 2007, 2008 and 2009. She published over 50 scientific papers and many methodical works, wrote 20 textbooks in informatics and IT for secondary and high education. She was a member of Scientific committee of JBOI 2016. She is a member of Bulgarian National Committee for Olympiads in Informatics.

Olympiads in Informatics, 2017, Vol. 11, 175–192
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.16

# IOI Host Guidelines: General Aspects

Bakhyt T. MATKARIMOV[1], Greg C. LEE[2],
Margot PHILLIPPS[3], Eljakim SCHRIJVERS[4]

[1]*Nazarbayev University, 53 Kabanbay batyr Ave, Astana, Kazakhstan*
[2]*National Taiwan Normal University, Taipei, Taiwan*
[3]*ACG Sunderland College, 6 Waipareira Ave, Auckland, New Zealand*
[4]*Nederlandse Informatica Olympiade, Winthontlaan 200, 3526 KV Utrecht, The Netherlands*
*e-mail: leeg@csie.ntnu.edu.tw, bakhyt.matkarimov@gmail.com,*
*margot.phillipps@gmail.com, eljakim@gmail.com*

**Abstract.** In this series of papers we accumulate recommendations on various aspects of event preparation, organization and management for the International Olympiad in Informatics (IOI) Host country. Although the IOI has some very specific events, like quarantine, the contest and translation sessions, our recommendations are not limited to only an IOI style event. However the main goal of this series is to distribute best practices for an IOI. Part I of this publication covers general aspects of an IOI and parts II and III cover scientific and technical aspects.

**Keywords:** IOI, programming contest, international event organization and management.

## 1. Introduction

The International Olympiad in Informatics (IOI) is an annual international informatics competition for individual contestants from various invited countries, accompanied by social and cultural programmes [1:S1.1]. Starting in Bulgaria, 1989, IOI is one of the world's families of scientific Olympiads for secondary schools students, which include the International Olympiads of: Mathematics (since 1959), Physics (since 1967), Chemistry (since 1968) and Biology (since 1990). The IOI has many specific features which distinguish it from other international events:

- The contestants are not adults, which mean that the Host has important specific requirements regarding responsibility, hospitality, security, etc.
- Contestants' scores are evaluated by an automated computer system based on each problem's test data; The IOI problem set and accompanying test data preparation is very time consuming.
- The competition consists of two practical (i.e. not on paper) contests as well as a practice session and requires the use of large and expensive facilities.

The IOI's official site is `http://ioinformatics.org` [1:N3.8.2]. For general information on the IOI we refer readers to the website. We suggest special attention is paid to the following documents:

1. IOI Regulations [1].
2. IOI syllabus [2].
3. International Technical Committee (was ITWG) guidelines [3].

The IOI regulation document is approved by the General Assembly of the IOI. The IOI Regulation [1:S1.2] covers the main principles of the IOI organization. (Note: this document should be considered as best practices guidelines and are not parts of the regulations.) We refer to appropriate sections of the IOI Regulations in the scope of our recommendations. We also use abbreviations from the IOI Regulations, such as GA (General Assembly), IC (International Committee), ISC and HSC (International and Host Scientific Committees), ITC (International Technical Committee; referred to previously as ITWG) and HSTC (Host scientific and technical committee). The Host steering and organization committees are referred to as the Host. The IOI syllabus [2] has been written by the ISC. The ITC/ITWG guidelines [3] have been developed by the ITC/ITWG.

This paper consists of three parts which are not completely independent, as the parts discuss similar subjects. While part I focuses on general aspects, parts II and III specifically outline the scientific and technical elements. Unfortunately, there are very few publications based on the IOI Host's experience. For that we refer you to the excellent online paper of Mirka Jeskanen [4].

We hope that this open access document will help the IOI community and it will be useful for future Hosts to organize future exciting and successful IOIs!

## 2. Host & IC/ISC/ITC

The IC/ISC/ITC should maintain oversight of the IOI development processes, ensure a consistent quality of the IOI every year, and intervene when issues arise. The Host should maintain some level of autonomy, since the Host puts a lot of effort into preparing an IOI and his/her reputation is at stake. Guidelines should be considered as advice but the Host is free to deviate from them. However, it is highly recommended that the Host informs and consults with the IC/ISC/ITC.

## 3. Timeline

The following steps, listed chronologically, are crucial in the organization of an IOI:

1. A country applies to the IC to become a Host (generally 3 to 4 years in advance) and is selected as the Candidate future Host [1:S4].
2. After agreeing to abide by the IOI regulations the country is confirmed as the Future Host [1:S5].

3. Beginning preparations, the creation of the Host's official website, the publication of important information and reporting regularly to the IC on progress.
4. The IOI Flag handover ceremony, the selected candidate becomes the Present Host.
5. Sending out the IOI Country Invitations.
6. The organization of the IC/ISC/ITC "February" meetings.
7. Scientific and Technical preparations of the IOI.
8. Opening of the IOI Registration system.
9. Offering visa support where applicable.
10. Publication of the IOI main event and the IOI Conference programmes.
11. Publication of the Competition Rules / Judging Procedures.
12. Publication and distribution of Technical preparation materials.
13. Publication of the practice session tasks.
14. Meeting teams at the airport and other arrival ports.
15. The IOI programme.
16. Transferring teams to the airport.
17. Post-IOI activities.

## 4. IOI Host Application

Read [1:S4]. The IOI Host team should be both experienced in organizing international events, and have access to qualified specialists for the IOI task and technical preparations. The number of IOI participants from foreign countries should be estimated using growth trends of previous IOIs. At the moment we expect up to 90 countries, about 28 IOI committee members, and up to 100 guests. In total, about 700 participants of which 340 would be contestants. Based on our previous experience, the IOI budget is estimated to be 1 to 2 million US dollars. This figure does not include facilities, which are estimated at about 0.5 to 1 million US dollars. For example, the IOI'14 budget was about $1.1M US and the facilities were completely covered as a result of sponsorships. The IOI'15 budget was about $1.6M US, including the purchased facilities. To be selected as the IOI Future Host, the Host Country needs to provide guarantees in terms of the budget allocation, which is usually supported by a Governmental letter. The IOI objectives include that member countries should be willing to host an IOI at some future time [S1.7]. If the application is unsuccessful, it is worthwhile re-submitting for a different year.

## 5. IOI Future Host Selected

Congratulations! Analyze the regulations and guidelines [1-3]. Although the IOI Host is selected at least 3 years before its respective IOI, preparations should start immediately. Decide on the general model of the IOI Venue, cultural programme, scientific and technical preparations, paperless or not, etc. Prepare a plan, and a backup plan taking into account all potential risks. Think as a programmer: do not ignore any possible conditions and constraints and be flexible.

- Confirm and sign all the necessary agreements for the allocation of the IOI budget, venues, halls, facilities, etc.
- Find sponsors and decide on the guest fee.
- Design the IOI logo in vector graphics format, IOI badges, banners, certificates/ diplomas, goodie bags and contents and medals and trophies. Always produce extra medals, because the exact numbers are unpredictable. The IOI trophy is presented to the overall winner(s) and possibly a Distinguished Service Awardee. Do ensure enough trophies are in hand in case there is a tie at the top.
- Create the IOI Host official website.
- Distribute related information on previous IOIs and urgent information on the Host website by duplicating it from the official IOI website.
- Think about the IOI newsletter and mailing lists.

Consideration may also be given to how you may contribute to the IOI development. This may include new initiatives, new solutions or setting new trends. (Previous examples include the adoption of a 4th task and a live scoreboard.)

## 6. IOI is an International Event

The IOI is an international event with the overall goal of bringing the world together. To facilitate a great experience for all participants, try to avoid the strict enforcement of Host Country laws that are not commonly enforced on the international level. Inform all relevant security organizations about the event in order to prevent troublesome situations. Prior to the IOI, inform the IOI Community of any possible issue that might occur. If issues do occur, try to resolve them in a friendly manner. Please note that age restrictive laws are different among the IOI Countries and that some contestants might believe that certain laws are not applicable to them. Note that the team leaders are responsible for the contestants, especially for those contestants that are of a valid age to sign documents. In particular, pay special attention to laws and regulations regarding smoking and alcohol consumption.

## 7. Host Team

Ensure that the organizers have clearly defined roles and responsibilities. Make sure to train the Host team and send a large team to observe other IOIs well in advance to ensure they experience an IOI first hand. [1:S2.6]. Establish good relations with the IOI community and do not hesitate to ask questions. All members of the Host team who are handling communications with foreign participants should be able to communicate fluently in English.

## 7.1. *Organisational Team*

An IOI Country delegation normally consists of two adults and four contestants. In most host countries there will be a mix of people with many years of experience in the IOI, but also key specialists in organizational work who do not specifically have actual experience with an IOI. Involve and consult the people who have previously participated an IOI. Create Steering and Organizing Committees and an IOI secretariat. Create volunteer selection procedures and select volunteers.

## 7.2. *HSTC (Host Scientific and Technical Committee)*

It is rare to have experience with an onsite programming competition with 350 contestants, thus it is advised that an IOI is not completely new to members of the HSTC. The HSTC is responsible for the IOI contest which includes specialists in IOI contest preparation as well as engineers of electrical power management, light, ventilation and climate control, etc. Make sure that technical specialists will be on duty during the IOI events, which includes the translation (each of the 2 evenings prior to the 2 competition days), practice, the actual competition days and HSTC competition preparation time. We recommend HSTC members have at least 3 years of experience in the scientific and technical preparations for international programming contests. This experience can be gained through the organization of world or regional programming contests. They should have experience of having run at least one contest before the IOI with more than 300 contestants. Select at least two people for every critical position. Risk management implies not allowing a single technical administrator for any important part of the system. In many previous IOIs less than 10 people of the core HSTC team undertook most of the work. It is preferable to divide the workload equally among more members. The IOI task authors are guest members of the HSTC and may be available to assist with eg: test case preparation.

## 7.3. *Volunteers*

The IOI Volunteers are also crucial to a successful IOI. They act as team guides, office clerks, persons on duty, event's organizers, technical support staff, etc. Ensure that there are dedicated groups of volunteers for every kind of logistical activity, such as arranging accommodation, transportation, excursions, registration, guest programme, activities, technical support, opening and closing ceremonies, GA meetings, translation sessions, the competition and such. Deploy a coordinator for each of the roles and events. Develop an "internal communication" infrastructure, for pushing information out to guides and volunteers and for sending information back to coordinators as required. Investigate the Host country volunteer organizations and consult and involve them. Distribute the IOI related publications and create a detailed manual for volunteers with contact information. Decide on the volunteer training program and when and how the training will take place.

## 8. IOI Host Official Web Site

Create the IOI'n website and publish important information as required by the IOI Regulation [1] and recommended in various guidelines. Decide on the information by publishing a workflow; this includes visa information, the daily program of the IOI, accommodation information and tourist information for early arrivals. Involve native English copywriters or editors. Plan the long term maintenance of the IOI Host website. Connect to and, if necessary, duplicate urgent information on the IOI official site.

## 9. IOI Registration System

The IOI Registration system must capture IOI participants' personal data, including names, roles, passport data and arrival/departure information, as well as a photo. Registration data is very important in order to communicate with team leaders and to produce personal badges/certificates/diplomas. The IOI Registration system may generate standard invitation letters, and various reports: list of participants with relevant information, such as arrival/departure information, information on married couples, etc. The IOI Community should be familiar with the registration system, which can be found at `https://ioiregistration.org/` and is managed by Eljakim Schrijvers. Although [1:S3.5] the IOI committee members are not representative of countries, they should be asked to register as part of their country teams. It is good practice to register the complete Host team in the system.

## 10. Personal Identification at IOI

The IOI participants are usually identified by badges. Using passports for identification is not recommended, as contestants may lose their passports. Design the IOI badge to display participants' country and role as well as their photo. Ensure reasonably quality with regards to the photo when the badge is prepared. If the photo in the registration system needs to be changed, capture a new photo on arrival. Different roles may be differentiated by using different colors. Avoid using country flags and other symbols that may cause political tensions [1:S5.14].

## 11. IOI Backpack

The IOI backpack should be given to all persons involved in the IOI (e.g. participants, staff and volunteers). Arrange and compile the IOI backpack and contents. Include all necessary (personal) contents in accordance with the planned excursions and events. Usually the IOI backpack contains a personal badge, the IOI programme, t-shirts, a head cap, a pencil and notebook, national souvenirs and possibly sponsors' promotional items. Assign different t-shirt colors to different roles. This helps to easily separate staff, volunteers, leaders, contestants, etc.

## 12. Invitation and Visa Support

Analyze IOI Regulation [1:N2.7]. Full time staff should be selected to help with invitations and visa support. Carefully study Host Country laws and procedures on migration and visa control, and liaise with related governmental organizations. Prepare all the necessary forms and instructions on how to get a visa and publish concrete information about it on the website. Contact the IOI Community, distribute materials and ask for their requirements. For every IOI Country that need visas, find the relevant consulate and contact the consulate whenever this is asked for by the invited team. Pay special attention to the invitation of new country observers. A standard electronic format of an invitation letter should be generated automatically in the IOI registration system. Usually this is enough for visa free countries. Some countries may request a special format for the invitation letter (e.g. from government authorities) or to extend the dates if the team intends arriving before the official arrival day. The preliminary IOI programme needs to be attached to the invitation letters. For the purchase of visas on arrival, make sure that each delegation knows what relevant forms and the amount of cash (and the currency) they need to bring. If possible, organize that visas are free of charge for the IOI participants.

## 13. Early Arrival and Late Departure

Decide on the policy of the hosting with regards to early arriving or late departing teams and inform the IOI community as soon as possible. Consider providing an invitation letter for the teams with the listed period of the stay in the Host country. Be ready to follow Host country procedures on migration control and to help the IOI teams on any related issue. Help teams with local accommodation and provide all the necessary information and contacts. Ensure early arriving teams have instructions for being collected from local accommodation as they are not arriving at the expected airports or train stations on Arrivals day.

## 14. Programme

Analyze the IOI Regulations, as they place some restraints on the IOI programme [1]. The IOI includes two Host events: 1) the IOI committees meetings [1:S5] about half a year before the main event (February meeting); 2) the IOI week. The IOI February meeting usually takes 5 days and the IOI main event takes 8 days, both of which include the arrival and departure days. Although a number of the IOI days are not regulated, we believe that the number of days cannot be reduced. The main purposes of the February meeting is to inspect the Host' preparations and to provide various recommendations to the Host. In short: the IC inspects the organization, the ISC selects the tasks and the ITC inspects the technical preparations. It is good practice to select Sunday to Sunday dates for the IOI week. Below we have listed the main activities for each day of the IOI week:

Day 1: Teams arrive and register.

Day 2: Practice session, opening ceremony, two GA meetings, quarantine and translation.
Day 3: Competition day 1. Clarifications, IOI Conference, analysis/appeals, GA meeting.
Day 4: Excursion day 1. GA meeting, quarantine, and translation.
Day 5: Competition day 2. Clarifications, IOI Conference, analysis/appeals, GA meeting.
Day 6: Excursion day 2.
Day 7: GA meeting, closing ceremony, farewell party.
Day 8: Teams depart.

The competition part of the IOI programme includes 1) A practice session (~2 hours) and two official (5 hours) contests; 2) two translation nights before contests; 3) two question/answer sessions during the official contests; 4) two analysis/appeals sessions after the official contests; 5) the HSTC/ISC/ITC report at the first GA meeting on competition rules, reporting on the practice session at the second GA meeting and two reports at the GA meeting at the end of each of the competition days. The GA report and its approval by the GA on contest day 2 ends the competition part of the IOI.

For the planning of the IOI programme & schedule, take into account:

1. Those different events are not serial and may overlap in terms of time.
2. Early breakfast, late dinner and a cultural programme on the arrivals day. As people may arrive from midnight to midnight and have crossed time zones, providing adequate food that day will set the tone for the IOI.
3. Team leaders and their contestants must be able to meet in order to discuss the competition rules after the post-practice GA and before quarantine [1], and traditionally at the end of competition days behind the competition hall.
4. Quarantine: team leaders and contestants should be physically and virtually separated, so that communications are controlled.
5. That there is enough time allocated for contestants to consume breakfast and to take their seats at the competition days, Allow about half-an-hour for people to be seated.
6. That there is enough time for getting from place to place. Measure the walking times within the IOI venue.
7. Request information on the planning of the IOI Committees meetings, select appropriate times and venues.
8. Free time: plan/recommend events for free time (e.g. short city excursions and open lectures).

## 15. Competition Rules/Judging Procedures

Analyze IOI Regulations:S5.3,E3.4,E3.15 and S3.2 [1]. The HSTC, ISC and ITC create detailed competition rules and judging procedures to publish on the IOI Host official site. The GA approves them at the first GA meeting. It is good practice to outline the differences, if any, between the current proposal and the rules of previous IOIs.

## 16. IOI Venue

There are two general models for the IOI venue: a University/Educational campus or a City/Hotel based model. Determine the location and walking distances for the venue including accommodation, competition halls, meeting rooms, ceremony theatres and so on. Ensure that the delegations know the accommodation address(es) and contact information before arrival. Ensure that the organizational and technical services will be continuous during planned IOI events. Analyze recent crime and emergency reports for every IOI proposed venue and switch to a backup plan in case of unacceptable risks.

## 17. IOI Palaces/Halls/Rooms: General Consideration

The exact number of IOI halls, meeting rooms and theatres depends on the programme. It is better to allocate more rooms in order to avoid unnecessary relocations. For every place create and publish emergency evacuation plans. Moreover, train volunteers to handle emergency situations. Special requirements for halls and rooms are written below in the corresponding sections of this document, and also in other parts of this publication. The general needs include Wi-Fi (wireless Internet), writing paper and pens or pencils, water and cups, food and drinks for breaks, presentation facilities such as screens, an audio system and microphones for large halls, facilities for printing, and technical staff on duty. Every need should be catered for in advance of the planned event and take into account the number of participants and the duration.

### 17.1. *Opening and Closing Ceremonies Theatre(s)*

The IOI opening/closing ceremonies involve all the IOI participants plus the invited representatives of IOI Countries, those living in the Host Country (e.g. diplomats), people and officials involved who are from the Host country and guests who come to participate in the public event. For ceremonies the theatre requires at least 1000 seats. Create a seating plan and instruct volunteers on the process for entering and exiting. The theatre must have Wi-Fi, presentation materials, large spaces, large monitors and the supporting technology.

### 17.2. *Registration Hall*

The registration hall is used on the Arrivals day. The registration upon arrival needs a large space, catering for at least 100 people. The Host should estimate the peak times based on the arrival of buses from the airport (look at the arrival information). The Host should plan optimal registration pathways for the IOI teams and prepare all the necessary materials (e.g. forms and backpacks). Wi-Fi and either a buffet or open restaurants should be provided.

### 17.3. *Competition/Contest and Practice Hall*

Either one large hall or a cluster of smaller halls should comfortably fit all the contestants at individual computers, taking into account that contestants should not easily view other contestant's screens. The HSTC needs to be accommodated very nearby. The peak load is all the IOI contestants plus leaders, who will be present during the practice and they will also visit after the contest. During the contest day's water and some food should be provided as well as the technology to run the contest. (This is detailed in another paper.)

### 17.4. *GA Meeting Hall*

This hall must accommodate all team leaders and committees. Guests registered by individual countries may also wish to attend so allow for some extra seating. The hall should provide Wi-Fi and presentation technology.

### 17.5. *Translation Hall/Rooms*

Seats and working facilities should be allocated for every IOI team. In practice, almost all the IOI adults participate in the translation session although English speaking countries often don't participate except to assist in some capacity. Translation occurs twice - each of the evenings before the 2 competition days. Translation begins at the end of the GA meeting where tasks are accepted/approved and continues often until breakfast. Wi-Fi, presentation technology, a buffet, printers and computers should be made available.

### 17.6. *IOI Conference & Question/Answer Session Hall*

All the IOI adults plus guests from the Host Country need to be seated in a hall. There must be good Wi-Fi (leaders are likely to be watching the scoreboard), presentation technology, coffee-break facilities, printers and technical support.

### 17.7. *Three IOI Committees Rooms (for the IC, ISC and ITC)*

Organize separate rooms for each committee, with sufficient space based on the committee sizes and their technical needs. There should be sufficient electrical power sockets for each member. Station the ISC and ITC near to the competition hall. Allocate additional meeting rooms nearby or at the accommodation as they may decide to meet at any time. Wi-Fi, presentation technology, water and a printer. Note: the Executive Director often needs a printer for small volumes but relatively urgently for election(s) material.

### 17.8. *IOI Office*

The IOI office is the Host staff office used to communicate with the IOI participants. Find an optimal location for the IOI office so that it is easily accessible by all attendees, organize information desks and create, if necessary, sub-offices. Inform the IOI participants of the room's location. This office needs Wi-Fi, water and printers.

### 17.9. *Medical Office*

The medical office should be staffed or another form of medical service needs to be available permanently at the IOI venue and during the excursions. Doctors on duty should be in the same building as the competition hall during competition days.

### 17.10. *Server Room*

Allocate a server room for the IOI grading system that meets the technical requirements for a relatively large computing system work (e.g. stable power, climate control and secure access).

### 17.11. *Miscellaneous Spare Breakout Rooms*

Rooms for Regional meetings (APIO, CEOI, etc.) and other events that are planned with the IOI community.

### 17.12. *IOI TV & Life Stream Room*

Allocate near the competition hall.

### 17.13. *Press/Media Room*

A daily newsletter is a feature of an IOI. Volunteers will photograph and interview teams. They will need computers and printers able to produce about 700 full color copies daily. Local press may also want space in which to interview visiting teams or individuals.

### 17.14. *Organizing Staff Administration Rooms*

Allow sufficient space and communication technology for all key functions. (Accommodation, transport etc.)

17.15. *Working Area and Storage Rooms*

A single hall may be allocated for the GA meetings, the IOI conference including a question and answer session, and possibly even for the translation sessions.

## 18. Opening/Closing Ceremonies

Create detailed programmes for the opening/closing ceremonies, plan for dealing with dignitaries and also consider providing printed programmes. Choose appropriate presenters, particularly for the awards section and try to ensure presenters understand their audience, thus avoiding long "boring" talks.

Plan the presentation for the "parade of teams" that takes place during the opening ceremony. Ceremonies may have different sections for leaders at the closing ceremony but generally teams are seated together. Occasionally hosts have seated medalists closer to the stage and not with the rest of their team. Plan to show video and photographs of the week while people enter the closing ceremony. The closing ceremony should culminate with the IOI flag handover ceremony, to the next year's host. [1:S5.1].

## 19. GA meetings

Select the chair for the GA meetings [1:A3.2]. The chair should be a confident English speaker, be used to managing potentially challenging meetings and have some understanding of an Informatics competition. Design and prepare the IOI Country signs/sticks used for voting. Design and arrange the pigeonholes for every participating IOI Country and also for each member of the IOI committees. Before the GA meetings, print and distribute various materials provided by the IC/ITC/ITS/HSTC (e.g. contest rules, objection forms for task selection and forms for voting). Arrange a printer near the GA meetings hall for "emergency printing" during meetings.

## 20. IOI Conference

The IOI Conference is run in parallel with the contest and question/answer sessions. Contestants do not attend this event. However, the participating team leaders should be able to get and translate contestants' questions during the conference. The IOI Conference is a great opportunity for the Host country teachers and scientists to come in contact with the IOI team leaders. It is common practice to invite Host country teachers to the IOI Conference and to arrange special sessions/seminars for them. Information on the IOI Conference (i.e. call for papers and programmes) should be published on the Host web site. The Host should prepare the participants' diplomas, signed by the IOI conference Chair.

## 21. Question/Answer Session

During the contest contestants may ask questions directly to the HSTC/ISC in English or some other languages using the Contest Management System (CMS). However, the need for paper clarification forms cannot be eliminated and questions may need translation by team leaders [1:S6.8]. The workflow is: a contestant hands a clarification form to the contest staff, it is then sent to the ISC and delivered to the team leader for translation. After that it is delivered back to the ISC. If the ISC answer needs translation, the response is delivered to the team leader for translation. Then it is delivered to the ISC and finally it is delivered to the contestant. In recent IOIs all questions were open to all leaders and any leader who happened to know the language was allowed to translate.

## 22. Translation Sessions

One the contest tasks have been approved by the GA they are made available to team leaders for translation. The HSC and ISC will announce when the final version of a problem is available but translation will often begin prior to that announcement. Develop protocols for printing draft and final versions, providing envelopes with printed statements and other materials, and signing off on printed final versions. Inform all the teams of the protocol. Create a secure and safe plan to transfer task translations from the translation hall to the contest hall. If printed, distribute them on tables and for online access use the CMS frontend. The number of different translation languages is usually fewer than the number of participating IOI Countries because of a number of English speaking countries, and often leaders from different teams will work on a single translation in a commonly used language. In practice, translation sessions are night events. The last team leaders may leave the translation hall at breakfast time. Other leaders will require transportation to their accommodation throughout the night. All translations of task statements should be ready for contestants before the competition. The Host does not now allocate computers for all teams for translation; instead, laptops are allocated on request. In the IOIs 2014 & 2015, less than 20 laptops were requested. We recommend contacting the leaders before the IOI and asking them to bring their own laptops, but some should be available for those that don't bring their own or possible loss or breakage of personal laptops.

## 23. Quarantine

Quarantine is designed to prevent any communication between contestants and the leaders/guests who know about the tasks, from the moment the competition tasks are presented to the GA until the end of the contest [1].

Excellent correlation of contestant results with previous and former achievements (e.g. at ACM ICPC and other programming competitions) shows that the IOI Community is ethically healthy. The Host should develop sound quarantine procedures. Decide how internet access will be managed during the quarantine period for leaders and contestants. In IOI practice Internet access is not blocked during the translation sessions, as the leaders often use online translation services. However, it is recommended that Internet access and mobile communications are blocked for contestants. Bear in mind though that, the organization staff on duty must be able to make calls in case of emergency during this time.

## 24. Contest Hall

Create and publish emergency evacuation plans and train volunteers to handle emergency situations. Provide enough water and snacks for contestants during the contests and possibly the practice session. Organize papers and pencils for people to work with. Enforce the contest rules during the contest, which includes any place the contestants may be (eg: no communications in refreshment rooms). There should be a functioning climate control and ventilation system. The light should be comfortable, both for working on a computer as well as on paper. Avoid having direct sunlight in the room. Backup power should be enough to support all the systems, including light and ventilation. Carefully look at table sizes and tables with network facilities. The contestants should not be able to see monitors and printed materials of other contestants. They should also not be able to touch other contestants whenever they move with their chairs, stand up or go elsewhere. Passageways in the contest hall should be wide enough for two people to pass walking in opposite directions and in order to reach any contestant or any technical facility. The space around the contestant table should be sufficient that other contestants are not disturbed whenever technical support is given. Power/network cables and common facilities should be properly installed and protected in order to prevent damage and accidents. An audio system for general announcements must be available. Ensure there will be sufficient toilets with secure the pathways to them. Plan a "holding area" for students to wait in before the competition. Create a secure space in order to keep contestants' keyboards and mascots. Prepare bags and identification paper for them and distribute them on the tables before the contest. Ensure that every contestant will get or has access to a complete set of task statements, including official statements and the relevant translations he/she needs. Experiments with paperless contests have shown that contestants prefer printed task statements and that they will print them, creating long printing queues at the beginning of the contest. Print and distribute contestant materials on the tables, provided by the HSTC (e.g. task statements and clarifications forms).

## 25. Accommodation

Accommodation, like the competition, should follow principles of equality of standards for different teams. If using hotels, find out the check-in/check-out times and the foreign citizen registration policy. Decide on your policy for early arrivals and late departures of teams and inform the IOI participants before the IOI about any possible supplementary expenses. For hotels it is good practice to lock the minibar by default. Check that everything that is expected to be at the accommodation is indeed available, such as washing facilities, window blinds, beds, cupboards, chairs, tables, enough supplies (e.g. toilet paper, towels of sufficient size, sheets, pillows, pillow slips and such). Check that the bed linen is changed frequently. Check the number of showers/toilets/bathrooms. We recommend having at least one shower for four participants. Ensure that rooms have separate beds for different persons. Identify families within the IOI participants and allocate family rooms to them. Also, make sure to take gender into account during the allocation of rooms. Avoid mixing participants from different countries in a single room. When there is no other option, allow participants to choose or change. Create information desks at the hotels. Install additional wireless Internet capacity in the accommodation, as the Internet may not be sufficient. Shops for everyday needs should be near to the accommodation and instruct volunteers to help participants find them.

### 25.1. *Adults Accommodation*

The team leader and deputy leader may want to stay together in a single room, with the exception of different genders or any other given reason. Traditionally, the IOI committee members stay in single rooms.

### 25.2. *Contestants Accommodation*

Contestants may be placed in rooms for four people, a single team room, unless the team has mixed gender. It is important that rooms also have a separate shower and toilet.

## 26. Meals

Analyze the IOI Regulation guidelines [1:A5.3]. Decide exactly when and where each meal will be consumed each day, for all the IOI participants including the Host country staff (e.g. contestants, leaders, guests, IOI Committee members, Host staff and volunteers). Arrange snacks and (non-alcoholic) drinks for the IOI conference breaks, translation nights and the competition. Ensure that the logistics will allow for all the participants to eat in the allocated time slots, especially the breakfasts on the competition days. Ensure that dietary requirements such as vegetarian, halal, allergies etc. are catered for. If the dining space allocated does not allow for everyone to be seated at once then a longer time should be allocated for meal times. It is good practice to organize VIP meals with the Host Country officials.

## 27. Social and Cultural Programme

The organizational staff and a reasonable number of volunteers should participate in every event of the social and cultural programme. Inform all participants about the places they will visit and develop corresponding actions plan. Carefully count the participants before leaving and inform them what to do if they become lost.

### 27.1. *Guest Excursions*

The Host may plan guest excursions for every IOI day. Traditionally, an excursion is open for all participants on the day of arrival. The Host may organize daily guest excursions with lunch on site. Note that guest excursions are not mandatory. Many of the IOI adults are in fact teachers/coaches of contestants and plan to participate in team leader activities. On the other hand, some team leaders may prefer to participate to the guest excursions. Typically two busses, taking up to 100 people, are enough for a guest excursion. An excursion programme which highlights the beauties of your country may convince the participants to visit your country again!

### 27.2. *Big Excursions (Two)*

The IOI full day excursions (between the first and second competition days and after the second competition day) are excellent chances to showcase your country's attractions. Volunteers should participate in the big excursions with their teams. It is possible to develop different programmes for adults and contestants. If you schedule long trips, make sure to plan regular stops at places with restrooms and, if possible, offer comfortable buses with air conditioning, toilets and wireless Internet. It is preferable that the first of the two excursions will return to the accommodation before dinner time as contestants need to be well rested for the next day and the leaders may have a full night of translation ahead of them.

### 27.3. *Invited Lectures/Sport Events/Entertainment*

Traditionally the IOI programme includes many special events for contestants, leaders and guests. The Host is free to organize programmes to show the spirit of the Host Country. It is good practice to organize special events for the contestants during the quarantine evening time - obviously, in accordance with quarantine requirements.

## 28. Transportation

Determine all the needs for transportation. Based on arrival and departure information, create transportation plans for arrival and departure days. Decide on how to handle transfers of early arrivals and late departures. Based on a detailed programme, request schedules of the planned meetings from the international committees and consider other events that may need transportation. Analyze the traffic data, choose optimal routes, prepare backup plans and measure transportation time. It is better to allow a larger tolerance for transportation times than you might for individual travel. In practice, people boarding a bus may take up to half-an-hour. Whenever walking times are greater than 15 minutes, organize shuttles between accommodation and other venues. Allocate small busses for every day organization and for transportation for the international committees (e.g. the HSTC/ISC/ITC usually prepare a contest day report at competition site and may, therefore, need transportation to the GA meeting site). Plan action plans in relation to lost luggage at the arrival days. Pay special attention to the transportation for the translation nights. Create plans that are flexible and can cope with unexpected changes (e.g. late arrivals). We recommend transferring a whole IOI Country team in a single bus on the day of departure. Select appropriate busses for transportation needs (e.g. for long way excursions).

## 29. Live stream, Press & IOI Newsletter

The IOI Community is a lot bigger than merely the IOI Participants. We know that worldwide many people are highly interested in what is happening at the IOI. Therefore, ensure real time photos and videos (with translations) of the IOI events are provided and distribute any related materials on the IOI Host website. Plan the production and distribution of the daily IOI newsletters. Live stream has, to date, been provided by a volunteer, Eljakim Schrijvers.

## 30. After IOI Activities

Prepare the IOI report. This will be presented to the International Committee the following year and should be an honest reflection of the IOI. It is of great benefit to future hosts. Refresh the video and photo archive and publish selected material. Maintain and do not close the official Host website after the respective IOI has finished.

## Acknowledgements

An IOI is forged by the collective efforts of many professionals and we are very thankful to all those whose experiences we have tried to present in this paper. Special thanks to Ricardo Anido and Ben Burton.

## References

1. IOI (2016). *Official Regulation*. `http://ioinformatics.org/rules/index.shtml`
2. Verhoeff, T., Horváth, G., Diks, K., Cormack, G. (2017). *The International Olympiad in Informatics Syllabus.* `http://www.ioinformatics.org/a_d_m/isc/iscdocuments/ioi-syllabus.pdf`
3. IOI (2016). *ITC Guidelines*, Last modified on 3 April 2016. `http://wiki.ioinformatics.org/wiki/HostingAnIOI`
4. Jeskanen, M. (2002). *Handbook for Arranging the International Olympiad In Informatics, Scholarly Thesis*. `http://olympiads.win.tue.nl/ioi/guides/ioi-handbook.pdf`

**B.T. Matkarimov.** IOI Team leader of Kazakhstan from 2005. IC member, Host Country 2013-2016. Chair of IOI 2015. Initiator and Jury chairman of Kazakhstan subregion of the Northeastern European Regional Contest of the ACM International Collegiate Programming Contest.

**G.C. Lee.** IOI Team leader of Taiwan since 1994. IC member 2012-2015. Host and Chair of IOI 2014. Professor and Chair of Taiwan Olympiad in Informatics.

**M. Phillipps.** IOI Team leader for New Zealand 2008 to 2013, Deputy 2014 and Executive Director of the IOI in 2015 and 2016. Chair of New Zealand Olympiad in Informatics.

**E. Schrijvers.** IOI Team leader of The Netherlands since 1995. IC member (2007-2012), Organizer of IOI Workshop, Treasurer of the IOI. Chairman of the Dutch (the Netherlands) Olympiad in Informatics.

Olympiads in Informatics, 2017, Vol. 11, 193–198
© 2017 *IOI, Vilnius University*
DOI: 10.15388/ioi.2017.17

# Casual Programming: A Channel for Widespread Computational Education

Shaya ZARKESH
*Polyup Inc., USA*
*email: shaya@polyup.com*

## 1. Introduction

Over the last ten years, casual gaming has seen an impressive rise to the mass market. The smartphone industry currently poses the fastest-growing sector of videogames, with an incredible 23.7% year-over-year growth (Newzoo 2017). Smartphone games present the biggest opportunity to reach a mass market for casual gaming, so an increase in smartphone game popularity is indicative of a greater general shift towards casual gaming.

The question thus becomes the following: how do we harness the megatrend of casual gaming for good? How do we turn smartphone games into useful, educational tools?

At *Polyup*, we aim to answer these questions to empower a global community of creative problem solvers. Taking into account the wild popularity of casual gaming, we set out to find the best path to help primarily older children and teens, but also adults, with their "computational thinking" skills. Computational thinking lies at the heart of new topics becoming indispensable in the information age – subjects like data science, cryptography, informatics, and artificial intelligence. Computational thinking is a way of approaching and analyzing computational problems by honing the skills of pattern recognition, decomposition, abstraction, and algorithm design (Wing 06). In a sense, computational thinking is like critical thinking for STEM – whereas critical thinking focuses on finding relationships between ideas in written text, computational thinking is devoted to finding patterns in numerical contexts. Although the concept is immensely important to many STEM fields like mathematics and computer science, it often goes unaddressed directly in schools, and students are thus left without the ability and creativity to solve more difficult computational problems on their own. By creating an attractive casual game where users can lead their own learning, *Polyup* can help fill in the 21st-century skills that most of today's population lacks.

So, enough introduction. What is this mysterious environment that *Polyup* has created? In short, it is the world's first mobile, casual programming environment. While block-based environments have been available on tablets and computers for years, there has yet to be a major environment optimized for mobile phones. Because of massively constrained screen real-estate, it seems a programming environment on a phone would require a major simplification and reduction of features. However, in the sections that follow, we detail how *Polyup's* gamified programming environment still retains all possible computational tasks and is thus Turing complete.

## 2. Learning by Teaching

One of the most effective and motivating ways to learn is by teaching others. The story behind *Polyup* capitalizes on just that – it revolves around teaching an AI sidekick named Poly. At first, Poly gloats about all his ability in the computational programming environment; he knows how to do everything from solving quadratic equations to proving Fermat's last theorem! However, Poly's hubris soon leads to his demise, as he attempts a program too computationally complex for his own good, and his internals break, leaving his memory banks empty. From here on, it is the player's job to (re)teach Poly all that he has forgotten, starting from basic tasks like adding numbers to more complicated ones like calculating factorials and building a GCD (greatest-common-denominator) algorithm.

Learning by Teaching is not a novel idea – it is embedded in modern pedagogy, especially given new opportunities posed by digital learning (Biswas 05). However, *Polyup* takes a novel approach to Learning by Teaching by combining it with dynamic scaffolding, the adjustment of difficulty mid-game based on player performance.

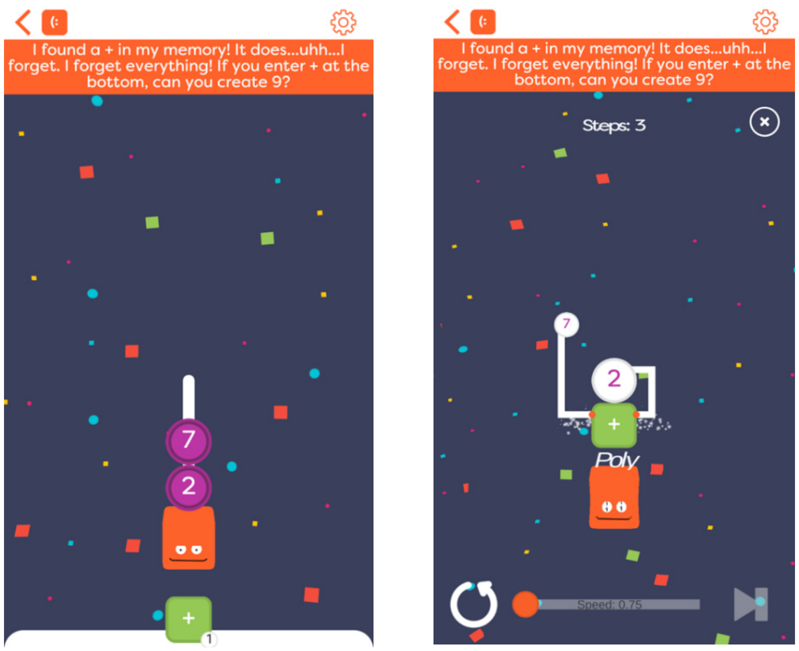
*Polyup* is a level-based system: the player has to solve puzzles by creating a program to achieve a stated output. The levels are of increasing difficulty, but no player will traverse every level. Levels are designed such that multiple levels teach a similar concept; when it is clear the user has mastered a programming concept, they move on to a more difficult one. Otherwise, if the user is unable to pass a level quickly and efficiently, it seems they have not mastered the concept, so further levels on the same concept are given. Such is the backbone of dynamic scaffolding. By implementing dynamic scaffolding in a level-based system, *Polyup* is adaptable to any skill level in programming. Furthermore, by using a non-classical programming paradigm, *Polyup* levels the playing field between experienced programmers familiar with programmatically syntax and those unfamiliar with any programming languages, thus making the educational experience universal.

## 3. The Environment

One of the most difficult aspects of creating a mobile programming environment is retaining functionality while simplifying the interface to manage a small amount of screen space available. In particular, typing on a phone is cumbersome and frustrating, so we avoided typing altogether. Like many existing educational programming environments, we thus resorted to a gamified form of block-based programming. However, the similarities to existing environments end there: we have completely rethought the programming paradigm and implemented a novel, simpler user interface.

The first major deviation *Polyup* has taken from most existing programming environments is the use of a functional programming environment with Reverse Polish Notation (RPN). The functional programming part means that the environment evaluates expressions immediately instead of executing statements. RPN is a style of programming in which the operator follows the operands (e.g.: 5 2 +), instead of putting the operator between operands like in classical mathematics (e.g.: 5 + 2). Such a notation has a dual purpose: first, it removes the need for parentheses, thus greatly simplifying the programming experience, and second, it exposes the user to a new programming paradigm, making them think of operators as functions. Unlike the typical step-based environments, *Polyup*'s environment truly requires the player to think about the order of operations and how operators and operands will interact.

*Polyup*'s user interface is also unlike most step-based programming languages in that it allows for manipulation of multiple functions on the same page. At the base level, the user interface of Polyscript involves dragging blocks onto a main stack of blocks, called "Poly," and any number of additional stacks. In Fig. 1 is a level in *Polyscript*; note the general layout of a puzzle with a target text on top, a function (stack) in the middle, and a set of draggable blocks available on the bottom.


I found a + in my memory! It does...uhh...I forget. I forget everything! If you enter + at the bottom, can you create 9?

I found a + in my memory! It does...uhh...I forget. I forget everything! If you enter + at the bottom, can you create 9?

Fig. 1. (left) A simple level in Polyscript (right) the level running.

In Fig. 2 the custom workspace is shown; here, the user has full creative control over the program, with draggable blocks organized into panes.

The panes are laid out as follows:

**Number Pane**
The number pane contains a system for typing any number or decimal. It contains blocks for the digits 0 through 9, as well as a decimal point; the user can either tap a sequence of these numbers to make a longer number, or drag any block directly to the stack for a number from 0 through 9.

**Math Pane**
The math pane contains the four basic operations +, –, ×, ÷ as well as more advanced functions like sin and mod.

**Variable Pane**
The variable pane contains the variable names x, y, and z, as well as the set blocks ->x, ->y, ->z. A set block is an operator that takes in the value above it and sets the respective variable to that value. The variable blocks are operands that are simply references to the values contained in them.

**Boolean Pane**
The Boolean pane contains the Boolean values, True and False, and the inequality symbols >, <, ≥, and ≤. It also contains the “if”, block, an operator that takes in a Boolean value and two functions: one for the “True” case and one for the “False” case.

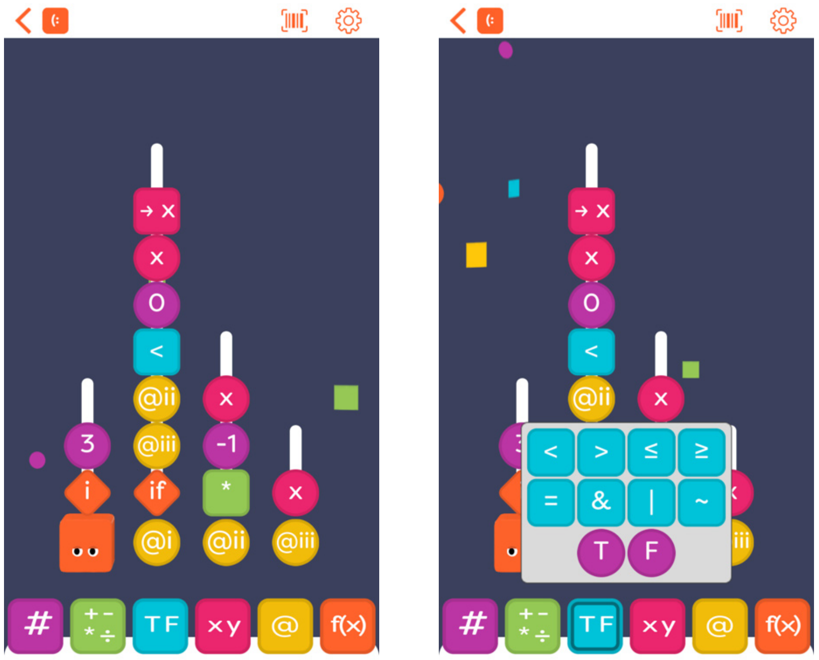


Fig. 2. A more complicated program written in Polyscript.

The magic of *Polyup* lies in its extremely simple compiler. A pointer reference starts at the top of Poly's stack and continually moves down. If it hits an operand, nothing occurs. If it hits an operator, the operator "eats" a number of blocks above it, corresponding to the number of values it takes in (e.g. the + operator takes in 2 values, so the 2 blocks above the "+" are eaten). If the pointer hits a reference to a "file", it adds the file to the stack and continues moving down. The program ends when the pointer hits the bottom of the stack.

## 4. Expanding the Scope: Socializing the Platform

Any gamified environment will have a hard time attracting users if it does not implement interaction with real humans, in order to motivate users to be the best at the game. *Polyup* thus implements multiple avenues for social interaction in-game. First, there is a puzzle challenge system where a player can create a level and challenge his or her friends to tackle it. Second, we have a live co-editing mode where two players can collaborate on the same puzzle workspace. Lastly, leaderboards for puzzle solving allow players to see their relative rank in problem solving ability, as determined by the dynamic scaffolding algorithm of the app. These avenues for social interaction can make the app far more attractive and create a viral effect, whereby players willingly share the app with friends in order to challenge or collaborate with them.

## 5. Results and Conclusions

*Polyup* is currently undergoing rapid prototyping of its application and has visited many middle and high schools to receive feedback, much of which has been implemented in its current conception. Responses are overwhelmingly positive. Of 39 students surveyed in the latest major version of the app, the average rating is 7.46. Of the 8 who came from an underprivileged high school, the average rating is 8.13! These results are staggering, and are indicative of the fact that *Polyup* has built an extremely attractive educational programming environment.

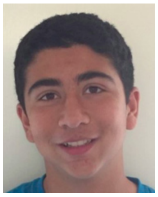
We have introduced and reviewed a novel educational programming environment set forth by *Polyup*. By gamifying the programming environment and translating programming to mobile, *Polyup* has created a more attractive and approachable environment to gain computational skills. By partnering with existing educational resources and competitions, *Polyup* has the potential to gain widespread use as a go-to computational learning application.

## References

Biswas, G., Leelawong, K. (2005). Learning by teaching: a new agent paradigm for educational software. *Applied Artificial Intelligence*, 19(3), 363–392.

The Global Games Market Reaches $99.6 Billion in 2016, Mobile Generating 37%. Newzoo, Jan. 2017.

Wing, J.M. Computational thinking. (2006). *Communications of the ACM*, 9(3).



**S. Zarkesh** is a rising senior at the Harker high school in California's bay area. He is an avid coder and math student and has won many regional and national awards in these fields, including national champion at TSA TEAMS (Tests of Engineering Aptitude, Mathematics, and Science) 2016 and the Science Bowl Regional Champion of 2017. Shaya is a co-founder of *Polyup* Inc., a high-tech startup in Silicon Valley founded in 2015.

# About Journal and 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)
- ERIC
- INSPEC
- SCOPUS – Elsevier Bibliographic Databases

**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:

- concise and informative title
- full names and affiliations of all authors, including e-mail addresses
- informative abstract of 70–150 words

- list of relevant keywords
- full text of the paper
- list of references
- 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.

The references cited in the text should be indicated in brackets:

- for one author – (Johnson, 1999)
- for two authors – (Johnson and Peterson, 2002)
- for three or more authors – (Johnson *et al.*, 2002)
- the page number can be indicated as (Hubwieser, 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:*

Hubwieser, P. (2001). *Didaktik der Informatik*. 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* (2008).
`http://www.IOInformatics.org/`

Hassinen, P., Elomaa, J., Ronkko, J., Halme, J., Hodju, P. (1999). *Neural Networks Tool – Nenet (Version 1.1)*.
`http://koti.mbnet.fi/~phodju/nenet/Nenet/General.html`

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 computerprocessed 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 str. 4, LT-08663 Vilnius, Lithuania
Phone: +370 5 2109 732
Fax: +370 52 729 209
E-mail: valentina.dagiene@mii.vu.lt

**Internet Address**

All the information about the journal can be found at:

`http://ioinformatics.org/oi_index.shtml`

Publisher office: Vilnius University Institute of Mathematics and Informatics
Akademijos str. 4, LT-08663 Vilnius, Lithuania
July, 2017

# Olympiads in Informatics

Volume 11, 2017

ISSN 1822-7732

# Olympiads in Informatics

Volume 11, 2017

Olympiads in Informatics Volume 11, 2017

ISSN 1822-7732

9 771822 773007