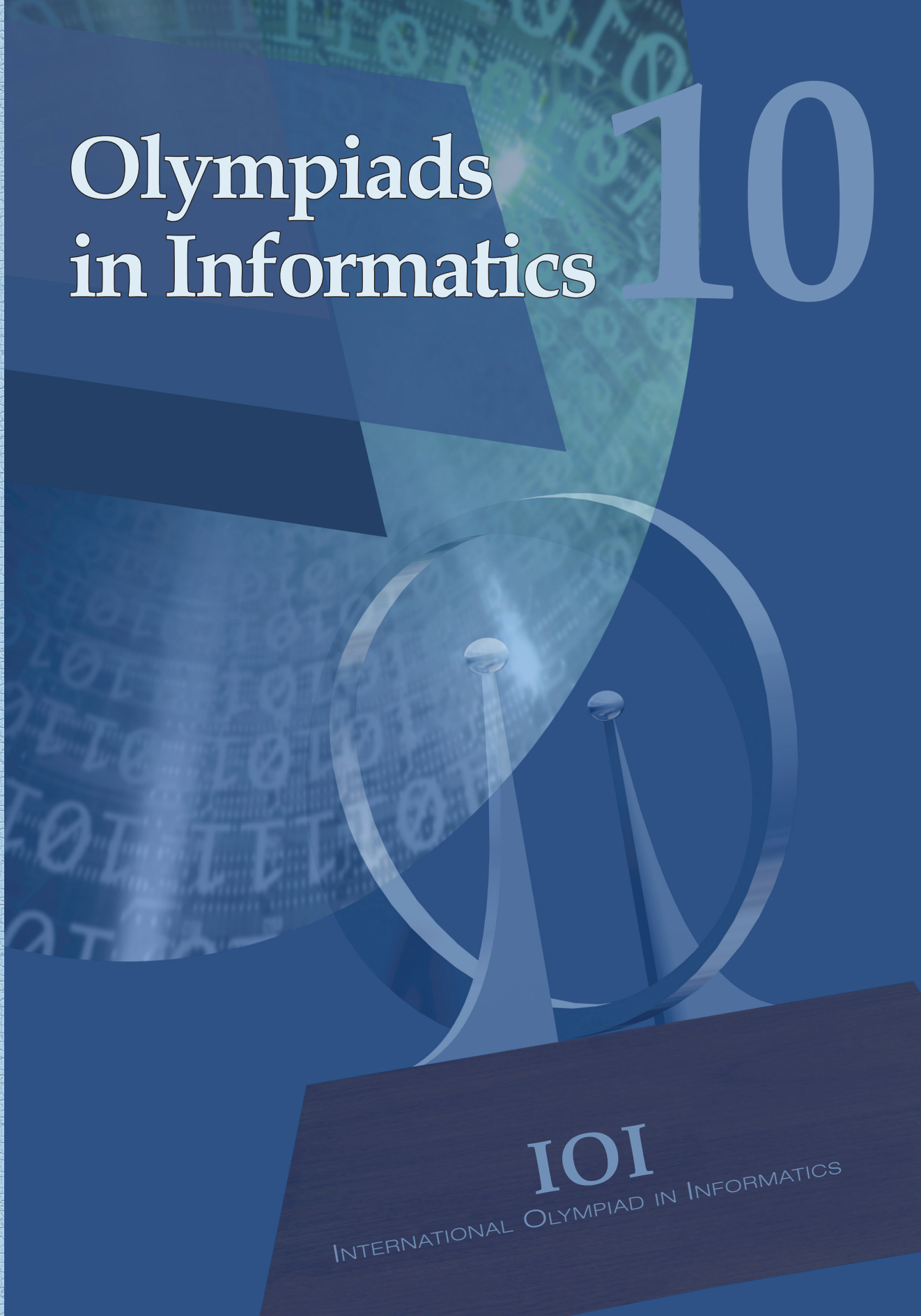
Olympiads in Informatics 10



IOI

International Olympiad in Informatics

ISSN 1822-7732

INTERNATIONAL OLYMPIAD IN INFORMATICS

VILNIUS UNIVERSITY
INSTITUTE OF MATHEMATICS AND INFORMATICS

OLYMPIADS IN INFORMATICS

Volume 10 2016

Selected papers of
the International Conference joint with
the XXVIII International Olympiad in Informatics
Kazan, Rusia, 12–19 August, 2016

OLYMPIADS IN INFORMATICS

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, 2016, Vol. 10, 1–2

Foreword

Time flies like an arrow

A popular example from the world of Artificial Intelligence (and Computational Linguistics), the phrase "Time flies like an arrow" is used to illustrate the ambiguity of language and the difficulty in comprehending structure. As a metaphor, it expresses how quickly time passes. Study the syntax and you can argue that it's an instruction to record the movement of flies (perhaps in the manner of an arrow, or only those that are similar to arrows), a description as to the preferences of a breed of fly, etc.… Some interpretations have meaning, others less so, but underlying this is the message that something apparently straightforward can have multiple meanings, nuances, interpretations. An unexpected depth of meaning within a simple form.

IO

IO – Informatics Olympiads. For those of us involved with national and international olympiads there are numerous reasons why we got involved. For some, it has been the challenge of competing: an opportunity to flex the intellectual muscles, a delight in problem solving or even the pleasure of pushing yourself against other similarly minded individuals. For others, it is the academic pursuit as the olympiads provide a mechanism for learning and teaching: a way to practice skills and infuse knowledge; a way to demonstrate that knowledge, skill and application to the wider world. Perhaps for others it is the sense of community: fellow students, fellow educators, competitors and friends. All of this before we even ask the question as why we stay involved.

`10`

There are, so says the old joke, `10` types of people in the world – those who understand binary and those who do not. In our modern world it is increasingly important to be amongst those who understand. If we go back to the first days of the IOI, computer science was still a specialist pursuit. We had had a decade of home computers, so computers were accessible to many, and computing was appearing school curricula, but unless you took an interest your exposure could be fleeting. We now exist in a world where people carry powerful computers in their pockets – ask yourself how many of the competitors at this year's IOI carry a phone but do not wear a watch. So many facets of daily life aided by computer programs, a many-headed beast that demands constantly to be

fed. It is not just teaching individuals how to program, most people will never write or need to write a line of code, but an understanding of what is happening inside the black box – how things work, what is feasible, knowing what is going wrong to know how to make things right – is an important skill.

10

This is the **10**th year for the *Olympiads in Informatics* journal. The first year when we have publishing two volumes, as we are delighted to publish a special volume celebrating the informatics education of our host – special thanks is given to the guest editor Marina Tsvetkova. We have published a total of 185 papers showcasing 200 authors from 50 countries. Technical papers and country reports. Algorithms and tasks. Reviews and opinions. The format of the journal is very similar to those early days; a forum for those in the community to write about specialised technical issues, an opportunity to share our experiences and knowledge, and to give freedom for those who are not academics to talk about their work. Looking back with the benefit of hindsight we might have developed the journal in a different direction but, with that same knowledge, we can say that we have fulfilled (in the most part) the vision and aims that we had when first establishing the conference.

Things are not always as they seem; it certainly does not seem like 10 years already. Time flies like an arrow indeed.

There are individuals without whose tireless work this volume of the journal – indeed every volume of the journal – would not exist. As always, thanks are due to all those who have assisted with the current volume – authors, reviewers and editors. A lot of work goes, not only to the writing of the papers, but to an extended period of review and correction and, in several cases, translation. Peer viewing all of the papers takes a significant amount of time and work and special thanks should be given to those otherwise unsung reviewing heroes: Benjamin Burton, Sébastien Combéfis, Walter Gander, Gintautas Grigas, Mathias Hiron, Ville Leppänen, Päivi Kinnunen, Jari Koivisto, Krassimir Manev, Martinš Opmanis, Rein Prank, Jūratė Skūpienė, Peter Taylor, Ahto Truu, Willem van der Vegt. Particular thanks are due to the organisation committee for IOI'2016 in Russia without whose assistance we would be unable to hold the conference. Their assistance, during what is an already busy period, is gratefully received.

Editors

Olympiads in Informatics, 2016, Vol. 10, 3–18

DOI: 10.15388/ioi.2016.01

eSeeCode: Creating a Computer Language from Teaching Experiences

Joan ALEMANY FLOS[1], Jacobo VILELLA VILAHUR[2]
[1]*Fundació Aula: 34 Av. Mare de Déu de Lorda, 08034 Barcelona Spain*
Explorium: Samalús 1 L3, 08530 La Garriga Spain
eSeeCode.com
[2]*Aula Escola Europea: 34 Av. Mare de Déu de Lorda, 08034 Barcelona Spain*
eSeeCode.com
e-mail: jalemany@eseecode.com, jvilella@eseecode.com

Abstract. It has been almost 50 years since the creation of Logo – one of the first educational programming languages. Although most people agreed that it was important to teach computational thinking to children, only recently have school and district leaders begun to adopt curricula to include it – mainly through Scratch. In many cases this adoption has failed to provide the right methodologies and tools to teachers.

In this paper, we analyse some of the different languages used in classrooms today and we propose an improved alternative that we have created – eSeeCode. We also share our experiences using this language in classrooms and how students can learn using this tool.

Keywords: informatics curriculum, promoting informatics, programming language.

1. Overview

Reading Papert's *Mindstorms: Children, Computers, and great ideas,* one can have the feeling that we have not advanced much in the past 35 years (Papert, 1980). Many countries are trying to include Coding as a required skill to learn in schools, either as a specific subject or as part of a technology course. However, in many schools, teachers do not have the resources, materials and/or knowledge to bring computer science and coding into the classroom. This is the case of Spain, among other countries, where computer science has been introduced into the curriculum, but has failed to provide the details on how to implement it properly, thus providing teachers the freedom and responsibility to decide how to teach some basic computer science concepts (Saez-Lopez *et al.*, 2016, Ackovska *et al.,* 2015, Duke *et al.*, 2000).

In this paper, we will analyse several computer languages and materials, and we will explain the difficulties students find when using them in class. As Edsger Dijkstra explains, the selection of the programming language will influence how the students will

understand computer science (Dijkstra, 1999). We will base our examples on some of the most popular computer languages used in Spain today, although this experience can be extrapolated to many other countries.

1.1. *Key Terms*

To begin, we will use the term "educational computer language" in a broad sense, as a programming language used to teach coding in classrooms, ranging from *professional* computer languages such as **C++** to puzzle-related languages as **Lightbot**. Because in many cases language and programming environment cannot be analysed separately, we will use them indistinguishably.

We will use the term **text-based coding** to describe languages that permit students to type their own code, giving them total freedom in the expressions they write. In contrast, we will use the term **visual block based programming** as the type of coding where you select your own blocks, and build the programs with a drag-and-drop interface. There is a fundamental difference when discussing the use of these languages in the classroom, as the former, generally speaking, cannot prevent students from making syntax errors, while the latter prevents these type of mistakes. Fig. 1 and Fig. 2 show different examples of text-based coding and visual block based programming.

We will differentiate between two kinds of paradigms when discussing different approaches to teaching. The first will be **Problem Solving,** where the teacher can present the student a short, self-contained problem that needs to be answered. In the particular case of this paradigm, we will also use the term **Puzzle Solving.** Similar in concept to problem solving, we consider puzzle solving to have more of a recreational focus, where there is a known set of rules that include multiple variations. For example, when solving a sudoku puzzle there is a specific set of rules, and by changing the numbers

```
to square :c
   repeat 4[
      forward :c
      left 90
   ]
end
```

Fig. 1. Text-based coding example with Logo to create a square.

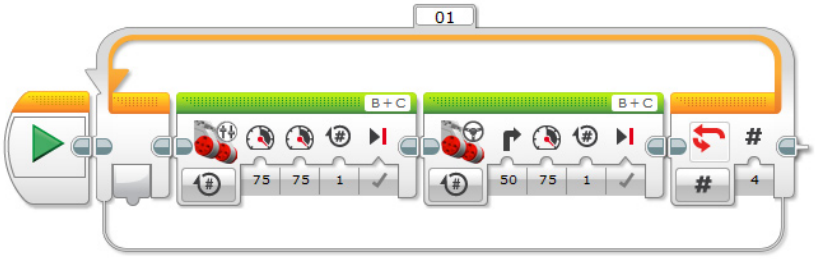


Fig. 2. Visual coding example with EV3 to make a Lego Robot move in a square pattern.

you change the problem. The teacher acts as a problem-setter or planifier, deciding which problems to be solved at each moment, such as when a student is asked to code the Eratosthenes sieve.

As an alternative to problem solving we find **Project-based learning**. In Spain, this strategy is gaining increased attention in the school setting. As part of project-based learning, students take a more active and creative role by pursuing an authentic and real-world project that addresses essential questions and works towards gaining an enduring understanding of a relevant issue. The teacher here is more of a guide or facilitator that helps the student when needed. In computer science this has the form of *software design*, *applied programming* and *modelling*. For example, a student could work on solving the problem of being able to identify a number as prime or not, with the project title being "Create a game that uses prime numbers".

1.2. *Languages*

To help keep the explanation focused, we have created a list of the main languages used in schools in Spain, and have classified them by similarity of characteristics.

The final result is shown below:

Under *text-based coding* we will find two separate groups: the pure educational languages such as Logo or Processing, and the professional languages such as C++, Java, Python and Javascript.

Under *visual block based coding* we find again two groups: educational languages such as Scratch, Alice, Kodu, Ev3, AppInventor, and the group of puzzle languages such as Lightbot and Beebot. Strictly speaking, Lightbot and Beebot are not complete languages, but they are used to teach basic structures to students. It is for this reason that we have decided to include them in our analysis.

Although each language has its own particular characteristic, we will analyse a representative of each group. The languages we have chosen are: Logo, C++, Scratch and Lightbot.

2. Main Characteristics

To be able to compare the different languages we have created a short introduction for each one.

2.1. *Logo*

Originally created by Wally Feurzeig and Seymour Papert in 1967, there are many versions of the language used in schools. Its main objective was to teach programming to students from ages five to thirteen (Papert, 1980). Due to the fact that it has been around

for so long, there has been more than 300 versions of the language, with a substantial amount of literature related to it. (Boytchev, 2013).

Main characteristics:

- Educational (ages 5–13).
- The majority of versions use text-code.
- The first activities proposed were problem-solving oriented, although there was some space for creativity. This will vary with versions.
- Uses a drawing area where you can move a pointer called a "Turtle".
- Main academic fields are basic algorithmics and 2D–3D geometry depending on the version.
- In general there were no debuggers, but a step-by-step execution was possible in many versions, which helped locate errors. Also syntax errors are difficult to correct.

2.2. *C++*

Designed by Bjarne Stroustrup in 1983, C++ is a *professional* computer language utilized heavily in many academic and professional circles. It is a compiled language, allowing the user to be able to select a programming environment. Its standard library makes it easier to use, as compared to its predecessor C (Stroustrup, 2007). It is important to note that in this category that there are very different approaches depending on the paradigms used (Duke *et al.*, 2000), but we will not take them into consideration for the current analysis and will common ground.

Main characteristics:

- Professional (ages 12+)
- C++ uses formal code.
- The language accepts both problem-solving activities and project-oriented learning.
- No drawingarea. General interface uses a console to write and read (input/output). This can be extended to use of files.
- Main academic fields are advanced algorithms, data structures, and numerical problems. Uses of classes helps teach system design.
- Depending on the Interface there is a good debugger, but syntax errors are hard to read.

2.3. *Scratch*

Scratch was created in 2005 by Lifelong Kindergarten research group at Massachusetts Institute of Technology Media Lab led by Mitchel Resnick. Although you can trace some of its origins to Logo, its different approach to teaching computer science places it in a different category (Resnick *et al.*, 2009).

Main characteristics:

- Educational (ages 8–16).
- It uses visual block based programming.
- The language is more focused on project-oriented programming.
- There is a "drawing area" used to place objects and move them around.
- Main academic fields cover basic algorithm, software design.
- Due to the visual programming interface with blocks, it is impossible to get syntax errors. There is no debugger.

2.4. *Lightbot*

There is an educational movement promoted by *code.org* to reach children in different regions of the world and give them tutorials to learn how to code. This movement started as the "Hour of code" and has had a big impact all around the world. Almost 200,000 events were carried out in more than 140 countries. (Code.org, 2013–2016)

Many schools around the world use this material as an introductory material for Computer Science. Among the different tutorials that you can find, we selected one that had a large acceptance rate in the teacher community in Spain – Lightbot.

Lightbot is a puzzle-based game where you need to light some squares on a grid. It was created in 2008 by Danny Yaroslavski, but it was with the hour of code that it became popular. (Gouws *et al.*, 2014)

Main characteristics:

- Educational (ages 4–8 and 9+).
- It uses visual block based programming.
- The language is focused only on puzzle solving.
- It has nice animations of a robot moving around.
- Main academic fields cover basic algorithms (no variables).
- Impossible to get syntax errors, the execution can be considered *step-by-step.*

3. Developing a Curriculum

It is clear that these languages are difficult to compare to one another because of their fundamental differences. To be able to do so we will analyse its uses in the classroom and their main drawbacks if used alone. Later we will study the combination of one or more language.

When considering the development of a curriculum we have to take into account many factors, but mainly the maturity of the students (their age) and the amount of time they will spend engaging with that curriculum. In our case we will consider students from primary – secondary schools, and a relatively long length of time of engagement (3–4 years). Although some consider this length insufficient (Winslow, 1996), it is a valid starting point to achieve competence in programming. Some main objectives can

be found in some studies (Duncan and Bell, 2015) and in some publications like the Computing Progression Pathways (Dorling, 2014).

3.1. *Teaching with Only One Language*

Using the *code.org* tutorials in the classroom may seem like a good idea because of the students' high level of initial engagement; however, over time this strategy can backfire if it is assigned for too long as the teacher has limited control over the content (s/he cannot put his own problems), and the tutorials are short and very specific.

One of the program's strong points is that the students get positive feedback and do not feel frustrated when they fail. This is probably due to the fact that they view Lightbot and other tutorials in *code.org* as a game, instead of a class problem.

If we look at the drawbacks from Lightbot we can see that the number of commands is very limited and do not include variables. For younger children this is good because the less options you give them the more focused and easy it will be to arrive at solutions. On the other hand it is not good if the students are mature enough to learn and understand it quickly. The use of loops is another drawback. To create a loop one must make a procedure that calls upon itself (as in recursion). This makes an infinite type of loop. Although the students can find it intuitive, they have a difficult time understanding conditionals and being able to predict this kind of behaviour. Fig. 3 shows the use of loops and functions. We have to keep in mind that although some teachers might use it as a tool to teach programming, it does not cover some basic algorithmic concepts that are important, and its programs cannot be generalised. (Lightbot 2016, Gouws *et al.*, 2014)

Currently in Spain the most popular educational programming language is Scratch, where it has become quite popular, evidenced by the fact that Barcelona hosted the Connecting Worlds Scratch Conference in 2013. With Scratch, students are engaged and motivated (Saez-Lopez, 2016), but after many years of use their interest wanes. This is a big drawback because it generates apathy towards programming and, in some cases, a dislike for programming altogether. There are also some technical drawbacks. For example, the lack of text coding makes it difficult to read and write long conditionals and programs. There are some projects to overcome this difficulty (Harvey and Mönig

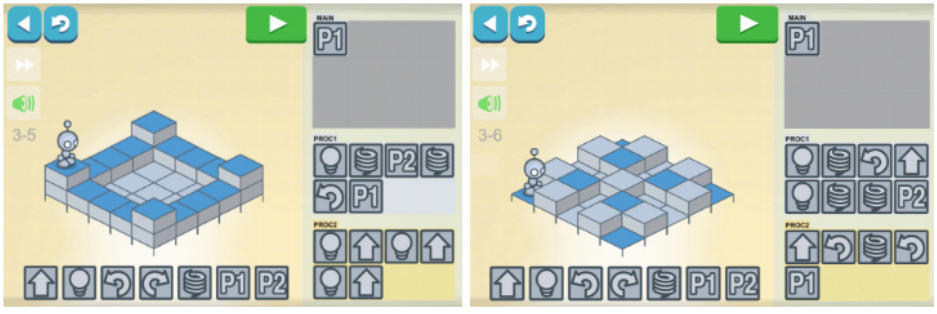


Fig. 3. Lightbot usage of recursive procedures instead of loops.

2010), but it is not included in the main tool. In addition, the high level language makes the student miss the opportunity to understand what is behind some of the instructions. Finally, as it can be seen in Fig. 4, Scratch allows some "parallel programming", which is actually not accurate as it is executed sequentially and depends on the order the procedures were created. This is a problem because the student cannot guess the results of a given program, which is one of the main objectives when teaching CS. (Dorling, 2014)

For a long time the most popular educational programming language in Spain was Logo. It was used in the 1980's through early 1990's, but its use faded away in the late 1990's. We can see this, for example, because it disappeared from the teacher majors in universities. There was not an immediate substitution by another language, but the government stopped promoting it as it was not news. This is reflected, for instance, in the use of it when training teachers at that time. (Simon, 1996) At the onset, students were motivated by this new approach to teaching and learning (Rubinstein, 1974), but today the look of a majority of the Logo platforms appear outdated and need a visual update. There are two major drawbacks to Logo. Its theoretical use was for students aged 5–13, with newer versions this could be extended until age 15, but text-based coding may bring syntax errors, and students need some maturity to be able to correct them. If not well attended students would get frustrated with programs that could not run. At the same time, similarly to Scratch, it has a limited life span in the student studies due to the apparent lack of practical utility.

The last group of languages to analyse is the professional ones. This are the most common choice among high school students. The students see the real value of coding and can explore other areas such as physics and mathematics, but because it is not prepared for the classroom the learning curve is very steep. Winslow describes the five steps from novice to expert (Winslow, 1996), in small-to-medium classrooms, the steep learning curve produces a clear and early separation between those who understand the content (and move through Winslow learning steps) and those who do not and remain as Novices. Another drawback is that due to the lack of visual assistance (in general) and

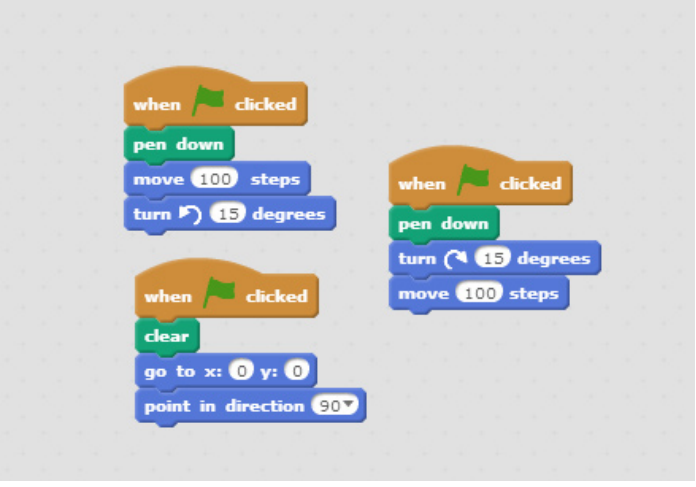


Fig. 4. *Scratch multiple parallel procedures.*

less attractive interface it is more difficult to motivate the students to use it. The syntax of some functions can also be a problem for some students. For instance, the syntax of the *for* loop is more complicated than in other languages, which makes basic algorithms sometimes hard to teach (Finkel *et al.,* 1994, Robins and Rountree, 2003). Depending on the programming environment used, the syntax errors are very difficult to read to the untrained eye, as can be seen in Fig. 5, where only a single character missing produces 20 lines of errors when compiling.

One big advantage is the amount of resources that can be found on the internet, in particular online judges like Codeforces, Timus, Topcoder, etc. With this sort of information it is really easy to prepare lists of problems for the students to engage.

It is worth pointing out that, in general, all of these languages and platforms are student focused. There are few tools for the teachers, and this makes using only one language more difficult. There are some positive exceptions, like the case of *Jutge.org***.** (Jutge 2008–2016, Giménez *et al.*, 2012) This online automated judge has been prepared as a tool for the classroom and not just for self-learning. Teachers can set up classes, have their own sets of problems and can view students' progress. This kind of platform contributes to the gamification of learning computer science, by giving achievements, keeping track of the number of problems solved, etc. When students perceive it as a game, they become more motivated and less frustrated.

4. Our Proposal: eSeeCode

After the analysis of the pros and cons of the different languages we decided that we should try to create a language that would eliminate almost all the weaknesses.

The result of our work is eSeeCode both a language and a programming environment. Main characteristics:

- Both visual code and formal code. You can transition from one to another.

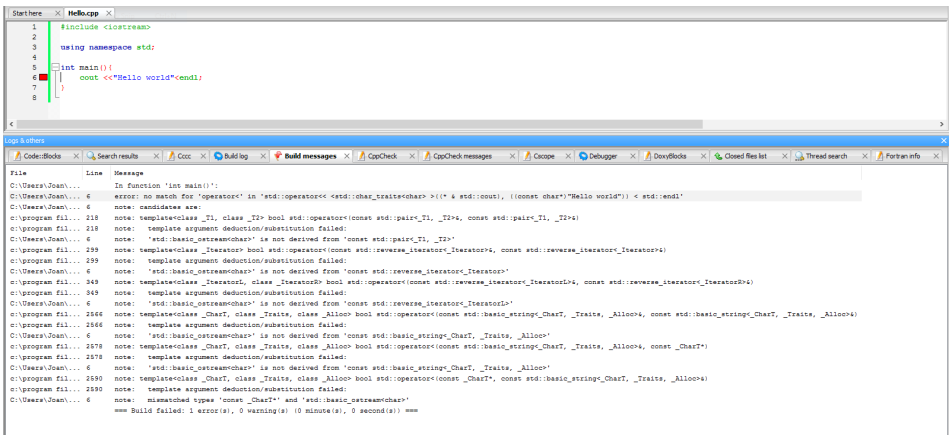


Fig. 5. Syntax error in C++. The programming environment is CodeBlocks.

- Both educational and professional language.
- It is problem solving oriented.
- Main academic areas cover basic and advanced algorithms, 2D geometry (turtle graphics) and numerical problems.
- Has a debugger, and includes simple syntax errors handling.

4.1. *Description of eSeeCode*

eSeeCode's environment offers four different programming levels: from a pure graphical click-and-run interface to a pure text syntax highlighting editor, with two middle interfaces. We call this levels *views*, because we want to show the students that code is just a representation that can have different forms. This can be seen in Fig. 6. This allows for a smooth progress in programming learning while keeping a common general interface, instruction set and platform. Time saved in this manner can be spent reinforcing other important objectives or learning a complementary second language.

The Touch view is our approach to the Puzzle Solving problems and is designed to work with students of ages 5–8. The set of instructions is represented by a set of icons (Fig. 7). The icons at this level have no text to maximize ease of use. These instructions

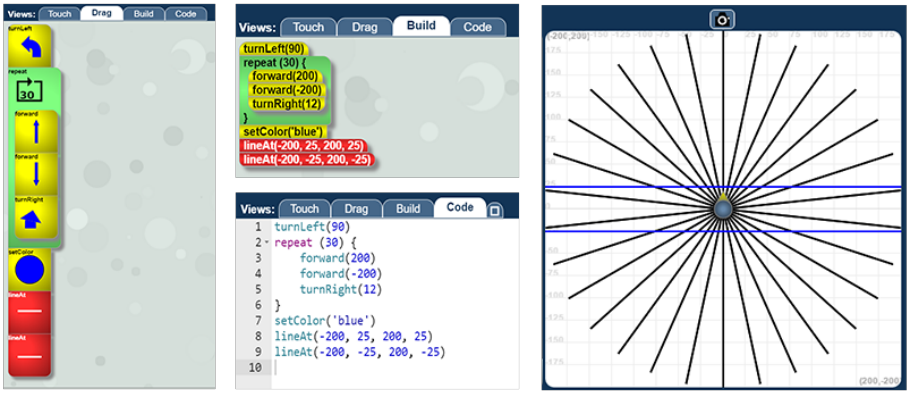


Fig. 6. Different views of the same code and its result.

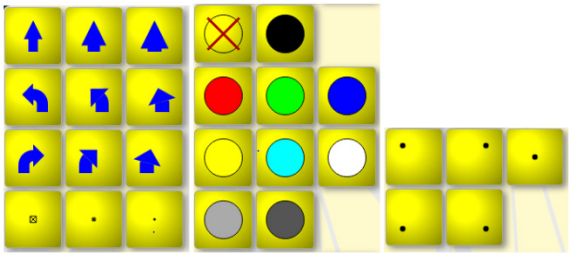


Fig. 7. Instruction set of the Touch view.

cover the basic movement of the guide, size of the drawing, colours and general positioning. Each time the student clicks one instruction the environment executes it directly.

The second view created is the Drag view. This view still uses icons as instructions but the student can move them freely once placed in the coding area. These instructions accept different arguments and the icons change to match the values of the parameters. Some examples of this behaviour can be seen in Fig. 8. Notice that the icons also include a name to help the student "read" the code. To execute a program the student needs to click the run button.

The next view is the Build view, which is similar to the Drag view as the blocks can be displaced around the code area freely. However these blocks don't include icons, just the name of the instructions and the arguments. The student can read the code fully, but to create it has to choose among a specific set available. This set is larger than the one from the drag view and contains a greater variety of instructions. This list of instructions allows the student to be able to explore eSeeCode by him/herselves, and provides the student with the familiarization of the names without having to memorize the commands and the arguments.

The last view is the Code view. In this level students can type their own code. We created eSeeCode based on JavaScript (although this base is well hidden), so after mastering in the programming in Code view the students can program freely using this well-known programming language. The platform accepts and executes any JavaScript program allowing for a deeper learning. A side advantage to the use of JavaScript is the fact that it is not required to be installed to function, as it will work with any browser.

In the Code view syntax errors are possible, but we try to give short errors that the student can correct. This can be seen in Fig. 9. The platform also has a debugger to be used in case the student's program does not execute as expected. When running a program, the editor will restyle the code to encourage students to use clean code.

In the context of the "low-floor, high-ceiling" proposed by Papert (Papert, 1980) and used by Resnick (Resnick *et al.*, 2009), eSeeCode has a lower-floor (easy-to-use) than Logo and Scratch, and a much higher ceiling (being able to hold complex programs) comparable to that of C++. The Touch view could be considered our low-floor while the Code view our high-ceiling.

Although you can try to create long programs, we have designed eSeeCode to be a problem-solving tool and have provided it with an optional easy-to-use Input/Output interface.

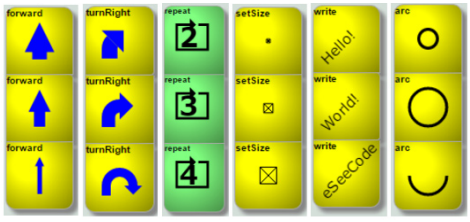


Fig. 8. The icons of the Drag view adapt depending on their parameters.

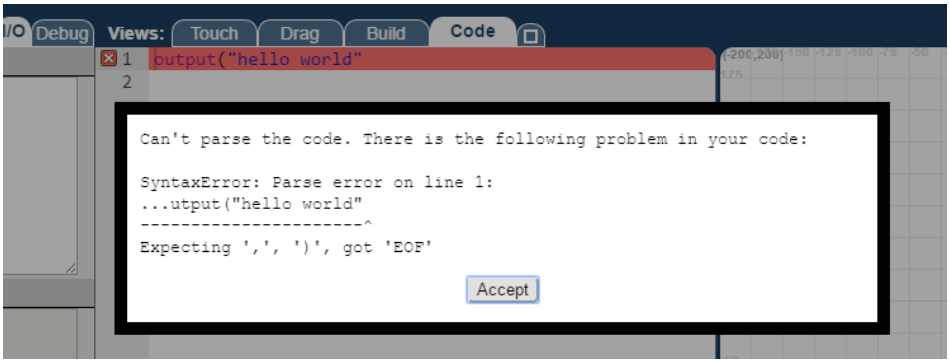


Fig. 9. Syntax error in eSeeCode.

4.2. *Teacher Point of View*

Programming languages, platforms, and materials are only oriented towards the student's use, obviating that one of the most influential parts on the learning of any process is the teacher. Providing teacher tools, and configurable platforms is key to an excellent use in class, given the uniqueness of each environment.

One of the main purposes of creating eSeeCode was also to be able to give the teacher a set of tools so s/he can create high quality exercises and materials for their courses. The interface is highly and easily configurable and it can be embedded in any webpage, allowing the teachers to configure it for each problem if needed. The tools already implemented are:

- A tutorial creation assistant, which creates dynamic tutorials
- A problem setter assistant, with which the teacher can restrict the views that can be used, decide which instructions are allowed (and how many times each can be used), preload code (hidden or not to the student) so that the student only needs to complete part of it, etc.
- Create step-by-step animations of the execution of programs.

Some of this tools are complemented by a Moodle module that allows the teacher to collect students problems and to set up specific exercises.

4.3. *Experiences with Students*

Many experiences have been carried out with students, both in an academic context and as an extracurricular activity.

eSeeCode has been used as a language to transition from Scratch to C++, and avoid the difficulties that appear from going from a visual block based language to a textual based language (Dorling *et al.,* 2015). This experience was done with 12 years old students that had previous knowledge of Computer Science since they had taken some

Scratch courses. This allowed the teachers to teach directly using Code view, but they allowed the students to go back to the Build view to avoid the empty page difficulty (Resnick *et al.*, 2009), where the student doesn't know how to start because he is used to having a set of blocks. With the introduction of eSeeCode the teachers had to review the basic concepts of variables, conditionals, loops, etc. The methodology that was used was based on Polya problem-solving principles, where the student should take the following steps (Polya, 1945):

1. Understanding the problem *i.e. Can you explain the problem with different words? Can you create your own "paper and pencil" examples?*
2. Devising a plan. *In our case this is clearly the Algorithm. Some things to consider when devising a plan are to try to find previous, similar problems.*
3. Carrying out the plan. *This is the implementation of the algorithm. We propose the "baby steps" methodology, where you write the code step-by-step and execute along to avoid syntax errors, while making sure everything goes accordingly to the plan.*
4. Looking Back. *Although no judge system has been created, the student should analyse if s/he obtained the desired result, going back to previous steps if s/he did not.*

Although this experience is different than the experiment done by Lewis in a study to compare Scratch vs Logo (Lewis, 2010), a similar survey was created and it was taken by 59 of the students in the course. The survey consisted of 16 questions each being a 4-level Likert scale.

As it can be seen in Fig. 10 it seems that Scratch is easier to program, but in fact the total number of students that have a positive feedback (Strongly agree and Agree)

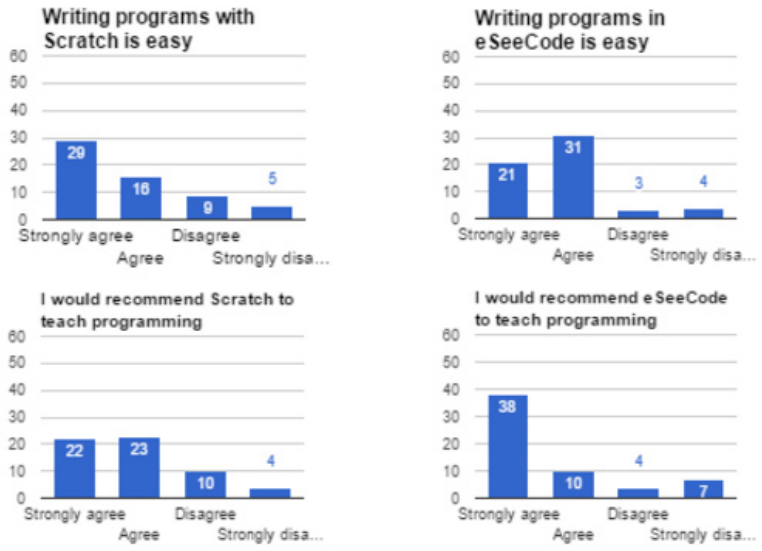


Fig. 10. Students responses to a 4-level likert test about the use of Scratch and eSeeCode in class.

is larger in the case of eSeeCode. One of the difficulties to analyse this question is the fact that the problems that the students had to solve in the two languages are different. What is interesting is that students would recommend in general to use eSeeCode to teach programming, this might have similar reasons to what Lewis describes (Lewis, 2010), as the students feel more self-secure when typing their own code, and might see Scratch as something different than programming. This would be interesting to analyze in a future study.

When asking questions about the difficulties when learning the language, we encounter different opinions depending on the topic. In Fig. 11 we can see this results. Similar to what happened when asking about *writing a program* the opinion of the students is less strong with eSeeCode than with Scratch, although the numbers of positive vs negative are similar. We have to take into account that the view students were using more is the Code view, which makes it difficult to give a precise analysis of the situation.

Fig. 11. Student responses to a 4-level likert test about the difficulties when learning with Scratch and eSeeCode.

What might be interesting is that the students opinion about the difficulties of the concept of *repeat* is statistically independent of the language (a Fisher exact test gives us a p-value of 0.3401 which tells us that the groups are not significantly different) This can be because the *repeat* concept is very easy to understand by the students.

Another experience we want to share is our own version of hour of code (Fig. 12) where students had to make programs to draw some typical optical illusions. Placing this activity in the context of students having an enjoyable time seemed to motivate the students to complete the tutorials and to try to draw their own images. Some of the activities were designed to be more difficult than the level of knowledge the students possessed and were accompanied with a solution code. The students would read the code and try to figure out the expected result. Although some students were not able to complete the activity, most of them enjoyed it. Another experiment we conducted was to give students the partial code for the program. In this activity we eliminated all the numbers from the code. The student was required to fill the gaps, until the right image would appear. Very few students would try numbers at random, the majority would first try to understand what the code did, and place the right numbers directly.

Three different sets of students tried the platform with this activity: Students that had never programmed before, students that had been introduced to Scratch and students that already knew Logo and C++. The difficulties found were similar in each group, concluding that it adapts to the student's needs.

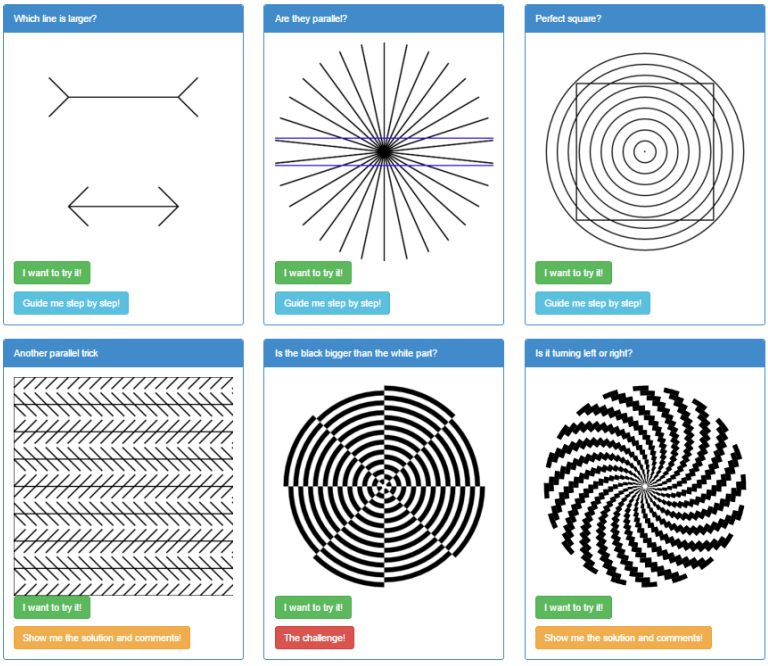


Fig. 12. Tutorials from our Hour of Code.

4.4. *Further Work*

We have released eSeeCode as an open source project, and as such it is a continuing development. We believe that further work in the platform should include:

- A new and adaptable design to make it feel more modern, and more user-friendly.
- Not allowing syntax errors in the Blocks and Build views.
- Teacher support material to be able to be used in class by non-programmer teachers.
- A formal study of the impact in the long run process and how it can be included in the regular curriculum. We believe this study should contain reports on Scratch, C++ and eSeeCode.

5. Conclusions

The time has come for teachers in Spain to take on the responsibility of curriculum development. This responsibility will come first by understanding the different options there exist, understanding the previous objectives, the ones we want to have in their place, and taking on a global vision. Right now one language cannot satisfy all the learning process. It is also valuable for the students to know more than one language, which would provide the option of overcoming the inherent weaknesses of each one.

eSeeCode tries to provide a new platform to overcome the main weaknesses found, but we believe it does not need to be a replacement but rather a complement to the learning process. The trials so far show that it is a viable language to take into the classroom, and that the students show a good overall satisfaction with it.

6. References

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

Boytchev, P. (2014). *Logo Tree Project*. `http://www.elica.net/download/papers/LogoTreeProject.pdf`

Code.org (2013). `http://www.code.org`

Dijkstra, E. (1999). Computing Science: achievements and challenges. *ACM SIGAPP Applied Computing*, 7(2), 2–9.

Dorling, M. (2014). Computer progression pathways. *Computing at Schools.* `http://www.computingatschool.org.uk`

Dorling, M., White, D. (2015). Scratch: a way to Logo and Python. In: *SIGCSE '15: Proceedings of the 46th ACM Technical Symposium on Computer Science Education.*

Duncan, C., Bell, T. (2015). A pilot computer science and programming course for primary schools. In: *Proceeding WiPSCE '15 Proceedings of the Workshop in Primary and Secondary Computing Education.* 39–48.

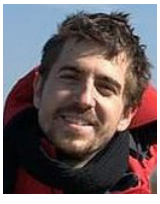
Duke, R., Salzman, E., Burmeister, J., Poon, J., Murray, L. (2000). Teaching programming to beginners – choosing the language is just the first step. In: *ACSE '00: Proceedings of the Australasian conference on Computing education.*

eSeeCode (2015–2016). `http://www.eseecode.com`

Finkel, D., Hooker C., Salvidio, S., Sullivan, M., Thomas C. (1994). Teaching C++ to high school students. In:

SIGCSE '94: Proceedings of the twenty-fifth SIGCSE symposium on Computer science education.
FMSLogo (2006–2016). `http://fmslogo.sourceforge.net/`
Giménez, O., Petit, J., Roura, S. (2012). Jutge.org: an educational programming judge. In: *Proc. of the 43rd ACM Technical Symposium on Computer Science Education* (SIGCSE-2012). 445–450.
Gouws, L.A., Bradshaw, K., Wentworth, P. (2013). Computational thinking in educational activities: an evaluation of the educational game Light-bot. In: *ITiCSE '13: Proceedings of the 18th ACM conference on Innovation and technology in computer science education.*
Harvey, B., Mönig, J., (2010) Bringing "No Ceiling" to Scratch: can one language serve kids and computer scientists? *Constructionism 2010, Paris.* `http://www.eecs.berkeley.edu/~bh/BYOB.pdf`
Jutge.org (2012). `http://www.jutge.org`
Lewis C. (2010). How programming environment shapes perception, learning and goals: logo vs. Scratch. In: *SIGCSE '10: Proceedings of the 41st ACM technical symposium on Computer science education.* 346–350.
Light-Bot (2016). `http://www.lightbot.com`
Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas.* Basic Books. Inc. USA.
Polya, G. (1945). *How to solve it.* Princeton University Press, USA.
Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, A., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11).
Robins, A., Rountree, N. (2003). Learning and teaching programming: a review and discussion. *Journal Computer Science Educations*, 13(2), 137–172.
Rubinstein, R. (1974). Using Logo in teaching. *ACM SIGCUE Outlook*, 9(SI).
Saez-Lopez, J.M., Roman-Gonzalez, M., Vazquez-Cano, E., (2016). Visual programming languages integrated across the curriculum in elementary school: a two year case study using "Scratch" in five schools. *Elsevier computers & Education,* 97, 129–141.
Scratch (2013–2016). `https://scratch.mit.edu`
Simon, J. (1996). *L'evolució de l'ensenyament del llenguatge logo a l'escola de mestres Blanquerna 10 anys.* VI Seminari Logo Terrassa, Spain.
Stroustrup, B. (2007). Evolving a language in and for the real world: C++ 1991–2006. In: *HOPL III: Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages.* ACM.
Winslow, L. (1996). Programming pedagogy – a psychological overview. *ACM SIGCSE Bulletin*, 28(3).

J. Alemany Flos holds a degree in mathematics and is currently completing a Master's Degree in Advanced Mathematics and Mathematical Engineering. He is a former high school teacher, who now designs and helps develop programming curricula in schools as a consultant. He has been involved with the National Informatics Olympiad for the last eight years, also attending the International Olympiad in Informatics as a team leader and deputy leader since 2010.



J. Vilella Vilahur holds a degree in Computer Science and has been teaching programming and robotics for the last ten years at Aula Escola Europea where he also provides the IT support. He has been involved in several open source projects and has been the lead programmer in eSeeCode.

Olympiads in Informatics, 2016, Vol. 10, 19–37

DOI: 10.15388/ioi.2016.02

Wavelet Trees for Competitive Programming

Robinson CASTRO[1], Nico LEHMANN[1], Jorge PÉREZ[1,2], Bernardo SUBERCASEAUX[1]

[1]*Department of Computer Science, Universidad de Chile*
Beauchef 851, Santiago, Chile
[2]*Chilean Center for Semantic Web Research*
email: {rocastro, nlehmann, jperez, bsuberca}@dcc.uchile.cl

Abstract. The wavelet tree is a data structure to succinctly represent sequences of elements over a fixed but potentially large alphabet. It is a very versatile data structure which exhibits interesting properties even when its compression capabilities are not considered, efficiently supporting several queries. Although the wavelet tree was proposed more than a decade ago, it has not yet been widely used by the competitive programming community. This paper tries to fill the gap by showing how this data structure can be used in classical competitive programming problems, discussing some implementation details, and presenting a performance analysis focused in a competitive programming setting.

Key words: wavelet tree, data structures, competitive programming, quantile query, range query.

1. Introduction

Let $S = (s_1, \ldots, s_N)$ be a sequence of integers and consider the following query over S.

Query 1. *Given a pair of indices (i, j) and a positive integer k, compute the value of the k-th smallest element in the sequence $(s_i, s_{i+1}, \ldots, s_j)$.*

Notice that Query 1 essentially asks for the value of the element that would occupy the k-th position when we sort the sequence $(s_i, s_{i+1}, \ldots, s_j)$. For example, for the sequence $S = (3, 7, 5, 2, 3, 2, 9, 3, 5)$ and the query having $(i, j) = (3, 7)$ and $k = 4$, the answer would be 5, as if we order sequence $(s_3, s_4, s_5, s_6, s_7) = (5, 2, 3, 2, 9)$ we would obtain $(2, 2, 3, 5, 9)$ and the fourth element in this sequence is 5. Consider now the following *update* query.

Query 2. *Given an index i, swap the elements at positions i and $i + 1$.*

That is, if $S = (3, 7, 5, 2, 3, 2, 9, 3, 5)$ and we apply Query 2 with index 5, we would obtain the sequence $S' = (3, 7, 5, 2, 2, 3, 9, 3, 5)$.

Consider now a competitive programming setting in which an initial sequence of 10^6 elements with integer values in the range $[-10^9, 10^9]$ is given as input. Assume that

a sequence of 10^5 queries, each query of either type 1 or type 2, is also given as input. The task is to report the answer of all the queries of type 1 considering the applications of all the update queries, every query in the same order in which they appear in the input. The wavelet tree (Grossi, 2015) is a data structure that can be used to trivially solve this task within typical time and memory limits encountered in programming competitions.

The wavelet tree was initially proposed to succinctly represent sequences while still being able to answer several different queries over this succinct representation (Grossi *et al.*, 2003; Navarro, 2014; Grossi, 2015). Even when its compression capabilities are not considered, the wavelet tree is a very versatile data structure. One of the main features is that it can handle sequences of elements over a fixed but potentially large alphabet; after an initial preprocessing, the most typical queries (as Query 1 above) can be answered in time $O(\log \sigma)$, where σ is the size of the underlying alphabet. The preprocessing phase usually constructs a structure of size $O(K \times n \log \sigma)$ for an input sequence of n elements, where K is a factor that will depend on what additional data structures we use over the classical wavelet tree construction when solving a specific task.

Although it was proposed more than a decade ago (Grossi *et al.*, 2003), the wavelet tree has not yet been widely used by the competitive programming community. We conducted a *social experiment* publishing a slightly modified version of Query 1 in a well known Online-Judge system. We received several submissions from experienced competitive programmers but none of them used a wavelet tree implementation to solve the task. This paper tries to fill the gap by showing how this structure can be used in classical (and no so classical) competitive programming tasks. As we will show, its good performance to handle big alphabets, the simplicity of its implementation, plus the fact that it can be smoothly *composed* with other typical data structures used in competitive programming, give the wavelet tree a considerable advantage over other structures.

Navarro (2014) presents an excellent survey of this data structure showing the most important practical and theoretical results in the literature plus applications in a myriad of cases, well beyond the one discussed in this paper. In contrast to Navarro's survey, our focus is less on the properties of the structure in general, and more on its practical applications, some adaptations, and also implementation targeting specifically the issues encountered in programming competitions. Nevertheless, we urge the reader wanting to master wavelet trees to carefully read the work by Navarro (2014).

2. The Wavelet Tree

The wavelet tree (Grossi, 2015) is a data structure that recursively partitions a sequence S into a tree-shaped structure according to the values that S contains. In this tree, every node is associated to a subsequence of S. To construct the tree we begin from the root, which is associated to the complete sequence S. Then, in every node, if there are two or more distinct values in its corresponding sequence, the set of values is split into two non-empty sets, L and R; all the elements of the sequence whose values belong to L

form the left-child subsequence; all the elements whose values belong to R form the right-child subsequence. The process continues recursively until a leaf is reached; a leaf corresponds to a subsequence in which all elements have the same value, and thus no partition can be performed.

Fig. 1 shows a wavelet tree constructed from the sequence

$$S = (3, 3, 9, 1, 2, 1, 7, 6, 4, 8, 9, 4, 3, 7, 5, 9, 2, 7, 3, 5, 1, 3).$$

We split values in the first level into sets $L = \{1, \ldots, 4\}$ and $R = \{5, \ldots, 9\}$. Thus the left-child of S is associated to $S' = (3, 3, 1, 2, 1, 4, 4, 3, 2, 3, 1, 3)$. If we continue with the process from this node, we can split the values into $L' = \{1, 2\}$ and $R' = \{3, 4\}$. In this case we obtain as right child a node associated with the sequence $S'' = (3, 3, 4, 4, 3, 3, 3)$. Continuing from S'', if we split the values again (into sets $\{3\}$ and $\{4\}$), we obtain the subsequence $(3, 3, 3, 3, 3)$ as left child and $(4, 4)$ as right child, and the process stops.

For simplicity in the exposition, given a wavelet tree T we will usually talk about *nodes* in T to denote, interchangeably, the actual nodes that form the tree and the subsequences associated to those nodes. Given a node S in T, we denote by $\texttt{Left}_T(S)$ its left-child and by $\texttt{Right}_T(S)$ its right-child in T. The *alphabet* of the tree is the set of different values that its root contains. We usually assume that the alphabet of a tree is a set $\Sigma = \{1, 2, \ldots, \sigma\}$. Without loss of generality, and in order to simplify the partition process, we will assume that every node S in T has an associated value $\texttt{m}_T(S)$ such that $\texttt{Left}_T(S)$ contains the subsequence of S composed of all elements of S with values $c \leq \texttt{m}_T(S)$, and $\texttt{Right}_T(S)$ the subsequence of S composed of all elements with values $c > \texttt{m}_T(S)$. (In Fig. 1 the value $\texttt{m}_T(S)$ is depicted under every node.) We can also associate to every node S in T, two values $\texttt{l}_T(S)$ and $\texttt{r}_T(S)$, such that S corresponds to the subsequence of the root of T containing all the elements whose values are in the range $[\texttt{l}_T(S), \texttt{r}_T(S)]$. Notice that a wavelet tree with alphabet $\{1, \ldots, \sigma\}$ has exactly σ leaves. Moreover, if the construction is done splitting the alphabet into halves in every node, the depth of the wavelet tree is $O(\log \sigma)$.

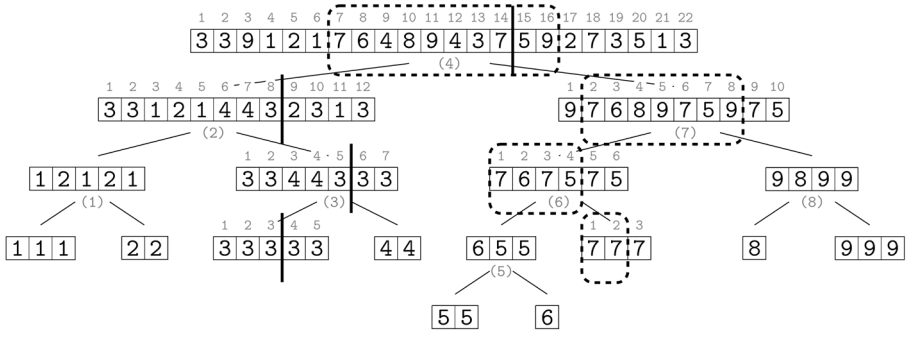


Fig. 1. Wavelet tree for the sequence $S = (3, 3, 9, 1, 2, 1, 7, 6, 4, 8, 9, 4, 3, 7, 5, 9, 2, 7, 3, 5, 1, 3)$. Solid lines illustrate the execution of $\texttt{rank}_3(S, 14)$. Dashed lines show the execution of $\texttt{quantile}_6(S, 7, 16)$.

As we will see in Section 4, when implementing a wavelet tree the complete information of the elements stored in each subsequence of the tree is not actually necessary. But before giving any details on how to efficiently implement the wavelet tree, we use the abstract description above to show the most important operations over this data structure.

Traversing the Wavelet Tree

The most important abstract operation to traverse the wavelet tree is to map an index in a node into the corresponding indexes in its left and right children. As an example, let S be the root node of wavelet tree T in Fig. 1, and $S' = \texttt{Left}_T(S)$. Index 14 in S (marked in the figure with a solid line) is mapped to index 8 in S' (also marked in the figure with a solid line). That is, the portion of sequence S from index 1 to index 14 that is mapped to its left child, corresponds to the portion of sequence S' from index 1 to 8. On the other hand, index 14 in root sequence S is mapped to index 6 in $\texttt{Right}_T(S)$.

We encapsulate the operations described above into two abstract functions, $\texttt{mapLeft}_T(S, i)$ and $\texttt{mapRight}_T(S, i)$, for an arbitrary non-leaf node S of T. In Fig. 1, if S is the root, $S' = \texttt{Left}_T(S)$ and $S'' = \texttt{Right}_T(S')$, then we have $\texttt{mapLeft}_T(S, 14) = 8$, $\texttt{mapRight}_T(S', 8) = 5$ and $\texttt{mapLeft}_T(S'', 5) = 3$ (all indexes marked with solid lines in the figure). Function $\texttt{mapLeft}_T(S, i)$ is essentially counting how many elements of S until index i are mapped to the left-child partition of S. Similarly $\texttt{mapRight}_T(S, i)$ counts how many elements of S until index i are mapped to the right-child partition of S.

As we will describe in Section 4, these two operations can be efficiently implemented (actually can be done in constant time). But before going into implementation details, we show how $\texttt{mapLeft}_T$ and $\texttt{mapRight}_T$ can be used to answer three different queries by traversing the wavelet tree, namely, *rank*, *range quantile*, and *range counting*.

2.1. *Rank*

The *rank* is an operation performed over a sequence S that counts the occurrences of value q until an index i of S. It is usually denoted by $\texttt{rank}_q(S, i)$. That is, if $S = (s_1, \ldots, s_n)$ then

$$\texttt{rank}_q(S, i) = |\{k \in \{1, \ldots, i\} \mid s_k = q\}|.$$

So for example, in sequence S in Fig. 1 we have that $\texttt{rank}_3(S, 14) = 3$.

Assume that T is a wavelet tree for S, then $\texttt{rank}_q(S, i)$ can be easily computed with the following strategy. If $q \leq \texttt{m}_T(S)$ then we know that all occurrences of q in S appear in the sequence $\texttt{Left}_T(S)$, and thus $\texttt{rank}_q(S, i) = \texttt{rank}_q(\texttt{Left}_T(S), \texttt{mapLeft}_T(S, i))$. Similarly, if $q > \texttt{m}_T(S)$ then $\texttt{rank}_q(S, i) = \texttt{rank}_q(\texttt{Right}_T(S), \texttt{mapRight}_T(S, i))$. We

repeat this process until we reach a leaf node; if we reach a leaf S with this process, we know that $\texttt{rank}_q(S, i) = i$.

In Fig. 1 the execution of $\texttt{rank}_3(S, 14)$ is depicted with solid lines. We map index 14 down the tree using either $\texttt{mapLeft}_T$ or $\texttt{mapRight}_T$ depending on the $\texttt{m}_T$ value of every node in the path. We first map 14 to 8 (to the left), then 8 to 5 (to the right) and finally 5 to 3 (to the left), reaching a leaf node. Thus, the answer to $\texttt{rank}_3(S, 14)$ is 3.

Rank is computed by performing $O(\log \sigma)$ calls to (either) $\texttt{mapLeft}_T$ or $\texttt{mapRight}_T$, thus the time complexity is $O(M \times \log \sigma)$ where M is the time needed to compute the map functions. Also notice that a rank operation that counts the occurrences of q between indexes i and j can be computed by $\texttt{rank}_q(S, j) - \texttt{rank}_q(S, i-1)$, and thus the time complexity is also $O(M \times \log \sigma)$.

2.2. *Range Quantile*

The range quantile operation is essentially Query 1 described in the introduction: given a sequence $S = (s_1, \ldots, s_n)$, $\texttt{quantile}_k(S, i, j)$ is the value of the k-th smallest element in the sequence $(s_i, s_{i+1}, \ldots, s_j)$. For instance in Fig. 1 for the root sequence S we have that $\texttt{quantile}_6(S, 7, 16) = 7$. It was shown by Gagie *et al.* (2009) that wavelet trees can efficiently solve this query.

To describe how the wavelet tree can solve quantile queries, lets begin with a simpler version. Assume that $i = 1$ and thus, we want to find the k-th smallest element among the first j elements in S. Then having a wavelet tree T for S, $\texttt{quantile}_k(S, 1, j)$ can be easily computed as follows. Let $c = \texttt{mapLeft}_T(S, j)$. Recall that $\texttt{mapLeft}_T(S, j)$ counts how many elements of S until index j are mapped to the left-child of S. Thus if $k \leq c$ then we know for sure that the element that we are searching for is in the left subtree, and can be computed as

$$\texttt{quantile}_k(\texttt{Left}_T(S), 1, \texttt{mapLeft}_T(S, j)).$$

On the other hand, if $k > c$ then the element that we are searching for is in the right subtree, but it will no longer be the k-th smallest in $\texttt{Right}_T(S)$ but the $(k-c)$-th smallest and thus can be computed as

$$\texttt{quantile}_{(k-c)}(\texttt{Right}_T(S), 1, \texttt{mapRight}(S, j)).$$

This process can be repeated until a leaf node is reached, in which case the answer is the (single) value stored in that leaf.

When answering $\texttt{quantile}_k(S, i, j)$ the strategy above generalizes as follows. We first compute $c = \texttt{mapLeft}_T(S, j) - \texttt{mapLeft}_T(S, i-1)$. Notice that c is the number of elements of S from index i to index j (both inclusive) that are mapped to the left. Thus, if $k \leq c$ then the element we are searching for is in the leftchild of S between the indexes $\texttt{mapLeft}_T(S, i-1) + 1$ and $\texttt{mapLeft}_T(S, j)$, and thus the answer is

$$\texttt{quantile}_k(\texttt{Left}_T(S), \texttt{mapLeft}_T(S, i-1)+1, \texttt{mapLeft}_T(S, j)).$$

If $k > c$ then the desired element is in the right and can be computed as

$$\texttt{quantile}_{(k-c)}(\texttt{Right}_T(S), \texttt{mapLeft}_R(S, i-1)+1, \texttt{mapLeft}_R(S, j)).$$

As before, the process is repeated until a leaf node is reached, in which case the answer is the value stored in that leaf. In Fig. 1 the complete execution of $\texttt{quantile}_6(S, 7, 16)$ is depicted with dashed boxes in every visited node.

As for the case of the rank operation, quantile can be computed in time $O(M \times \log\sigma)$ where M is the time needed to compute the map functions.

2.3. *Range Counting*

The range counting query $\texttt{range}_{[x,y]}(S, i, j)$ counts the number of elements with values between x and y in positions from index i to index j. That is, if $S = (s_1, \ldots, s_n)$ then

$$\texttt{range}_{[x,y]}(S, i, j) = |\{k \in \{i, \ldots, j\} \mid x \leq s_k \leq y\}|.$$

A sequence of size n can be understood as the representation of a grid with n points such that no two points share the same row. A general set of n points can be mapped to such a grid by storing real coordinates somewhere and breaking ties somehow. For this representation the range counting query corresponds to count the number of points in a given subrectangle (Navarro, 2014).

To answer a range counting query over a wavelet tree T we can use the following recursive strategy. Consider the interval $[\texttt{l}_T(S), \texttt{r}_T(S)]$ of possible elements of a sequence S. If $[\texttt{l}_T(S), \texttt{r}_T(S)]$ does not intersect $[x, y]$, then no element of the sequence is in $[x, y]$ and the answer is 0. Another case occurs when $[\texttt{l}_T(S), \texttt{r}_T(S)]$ is totally contained in $[x, y]$; in this case all the elements of the sequence between i and j are counted, so the answer is $|[i, j]| = j - i + 1$.

The last case (the recursive one) is when $[\texttt{l}_T(S), \texttt{r}_T(S)]$ intersects $[x, y]$ (but is not completely contained in $[x, y]$); in that case the answer is the sum of the range counting query evaluated in both children. The queries for children are called with the same $[x, y]$ as in the parent's call, but the indexes i and j are replaced by the mappings of these indexes. That is, the answer is

$$\texttt{range}_{[x,y]}(\texttt{Left}_T(S), \texttt{mapLeft}_T(S, i), \texttt{mapLeft}_T(S, j)) + \texttt{range}_{[x,y]}(\texttt{Right}_T(S), \texttt{mapRight}_T(S, i), \texttt{mapRight}_T(S, j)).$$

Note that if `range` is called on a leaf node S, then $\texttt{l}_T(S) = \texttt{r}_T(S) = z$, so the interval is either completely contained (if $z \in [x, y]$) or completely outside (if $z \notin [x, y]$). Both cases are already considered.

It is not difficult to show that for a range counting query, we have to make at most $O(\log \sigma)$ recursive calls (Gagie *et al.* (2012) show detailed proof), and thus the time complexity is, as for rank and quantile, $O(M \times \log \sigma)$ where M is the time needed to compute the map functions.

3. Simple Update Queries

We now discuss some simple update queries over wavelet trees. The idea is to shed light on the versatility of the structure to support less classical operations. We looked for inspiration in typical operations found in competitive programming problems to design update queries that preserve the global structure of the wavelet tree. We only describe the high level idea on how these queries can be adopted by the wavelet tree, and we later (in Section 4) discuss on how to efficiently implement them.

3.1. *Swapping Contiguous Positions*

Consider Query 2 in the introduction denoted by $\texttt{swap}(S,\, i)$. That is, a call to $\texttt{swap}(S,\, i)$ changes $S = (s_1,\, \ldots,\, s_n)$ into a sequence $(s_1,\, \ldots,\, s_{i+1},\, s_i,\, \ldots,\, s_n)$.

The operation $\texttt{swap}(S,\, i)$ can be easily supported by the wavelet tree as follows. Assume first that $s_i \leq \texttt{m}_T(S)$. Then we have two cases depending on the value of s_{i+1}. If $s_{i+1} > \texttt{m}_T(S)$, we know that s_i is mapped to the left subtree while s_{i+1} is mapped to the right subtree. This means that swapping these two elements does not modify any of the nodes of the tree that are descendants of S. In order to modify S, besides actually swapping the elements, we should update $\texttt{mapLeft}_T(S,\, i)$ and $\texttt{mapRight}_T(S,\, i)$; $\texttt{mapLeft}_T(S,\, i)$ should be decremented by 1 and $\texttt{mapRight}_T(S,\, i)$ should be incremented by 1 as the new element in position i is now mapped to the right subtree. Notice that these are the only two updates that need to be done to the map functions.

The other case is if $s_{i+1} \leq \texttt{m}_T(S)$. Notice that both s_i and s_{i+1} are mapped to $\texttt{Left}_T(S)$, and moreover, they are mapped to contiguous positions in that sequence. In this case, no update should be done to $\texttt{mapLeft}_T(S,\, i)$ or $\texttt{mapRight}_T(S,\, i)$. Thus, besides actually swapping the elements in S, we should only recursively perform the operation $\texttt{swap}(\texttt{Left}_T(S),\, \texttt{mapLeft}_T(S,\, i))$. The case in which $s_i > \texttt{m}_T(S)$ is symmetrical. The complete process is repeated until a leaf node is reached, in which case nothing should be done.

To perform the swap in the worst case we would need to traverse from top to bottom of the wavelet tree. Moreover, notice that the map functions $\texttt{mapLeft}_T$ and $\texttt{mapRight}_T$ are updated in at most one node. Thus the complexity of the process is $O(M \times \log \sigma + K)$ where K is the time needed to update $\texttt{mapLeft}_T$ and $\texttt{mapRight}_T$, and M is the time needed to compute the map functions.

3.2. *Toggling Elements*

Assume that every element in a sequence S has two possible states, *active* or *inactive*, and that an operation $\texttt{toggle}(S, i)$ is used to change the state of element i from *active* to *inactive*, or from *inactive* to *active* depending on its current state. Given this setting, we want to support all the queries mentioned in Section 2, but only considering active elements. For example, assume that $S = (1, 2, 1, 3, 1, 4)$ and only the non 1 elements are active. Then a query $\texttt{quantile}_2(S, 1, 6)$ would be 3.

A simple augmentation of the wavelet tree can be used to support this update. Besides $\texttt{mapLeft}_T$ and $\texttt{mapRight}_T$, we use two new mapping/counting functions $\texttt{activeLeft}_T$ and $\texttt{activeRight}_T$. For a node S and an index i, $\texttt{activeLeft}_T(S, i)$ is the number of active elements until index i that are mapped to the left child of S, and similarly $\texttt{activeRight}_T(S, i)$ is the number of active elements mapped to the right child. Besides this we can also have a count function for the leaves of the tree, $\texttt{activeLeaf}_T(S, i)$, that counts the number of active elements in a leaf S until position i. We next show how these new mapping functions should be updated when a toggle operation is performed. Then we describe how the queries in Section 2 should be adapted.

Upon an update operation $\texttt{toggle}(S, i)$ we proceed as follows. If $s_i \leq \texttt{m}_T(S)$ then we should update the values of $\texttt{activeLeft}_T(S, j)$ for all $j \geq i$ adding 1 to $\texttt{activeLeft}_T(S, j)$ if s_i was previously inactive, or substracting 1 in case s_i was previously active. Now, given that s_i is mapped to the left child of S, we proceed recursively with $\texttt{toggle}(\texttt{Left}_T(S), \texttt{mapLeft}_T(S, i))$. If $s_i > \texttt{m}_T(S)$, we proceed symmetrically updating $\texttt{activeRight}_T(S, j)$ for $j \geq i$, and recursively calling $\texttt{toggle}(\texttt{Right}_T(S), \texttt{mapRight}_T(S, i))$. We repeat the process until a leaf is reached, in which case $\texttt{activeLeaf}_T$ should also be updated (similarly as for $\texttt{activeLeft}_T$). The complexity of the toggle operation is then $O((A + M) \times \log \sigma)$, where A is the time needed to update $\texttt{activeLeft}_T$ and $\texttt{activeRight}_T$ in every level (plus $\texttt{activeLeaf}_T$ in the last level), and M is the time needed to compute the map functions $\texttt{mapLeft}_T$ and $\texttt{mapRight}_T$.

Consider now the $\texttt{quantile}_k(S, i, j)$ query. Recall that for this query we first computed a value c representing the number of elements of S from index i to index j that are mapped to the left. If $k \leq c$ we proceeded searching for $\texttt{quantile}_k$ in the left subtree, and if $k \geq c$ we proceeded searching for $\texttt{quantile}_{(k-c)}$ in the right subtree (mapping indexes i and j accordingly in both cases). In order to consider the active/inactive state of each element, we only need to change how c is computed; we need to consider now how many active elements from index i to index j are mapped to the left, and thus c is computed as

$$c = \texttt{activeLeft}_T(S, i-1) - \texttt{activeLeft}_T(S, j).$$

Then, we proceed exactly as before: if $k \leq c$ we search for $\texttt{quantile}_k$ in the left subtree, otherwise, we search for $\texttt{quantile}_{(k-c)}$ in the right subtree. Notice that we always assume that when executing $\texttt{quantile}_k(S, i, j)$ the number of active elements

between i and j in S is not less than k (which can be easily checked using $\texttt{activeLeft}_T$ and $\texttt{activeRight}_T$).

Queries $\texttt{rank}_q$ and $\texttt{range}_{[x,y]}$ are even simpler. In the case of $\texttt{rank}_q$ we only need to consider the active elements when we reach a leaf; in the last query $\texttt{rank}_q(S,\, i)$ in a leaf S, we just answer $\texttt{activeLeaf}_T(S,\, i)$. In the case of $\texttt{range}_{[x,y]}(S,\, i,\, j)$, we almost keep the recursive strategy as before but now when $[\texttt{l}_T(S),\, \texttt{r}_T(S)]$ is totally contained in $[x,\, y]$ we only have to consider the number of active elements between index i and index j, which is computed as

$$(\texttt{activeLeft}_T(S,\, j) + \texttt{activeRight}_T(S,\, j)) - (\texttt{activeLeft}_T(S,\, i-1) + \texttt{activeRight}_T(S,\, i-1)).$$

In the case in which S is a leaf, this value is computed as $\texttt{activeLeaf}_T(S,\, j) - \texttt{activeLeaf}_T(S,\, i-1)$.

The complexity of these new queries is $O((L + M) \times \log\sigma)$ where L is the time needed to compute the $\texttt{activeLeft}_T$ and $\texttt{activeRight}_T$ functions and M is the time needed to compute the map functions.

3.3. *Adding and Deleting Elements from the Beginning or End of the Sequence*

Consider the operations $\texttt{pushBack}(S,\, a)$, $\texttt{popBack}(S)$, $\texttt{pushFront}(S,\, a)$ and $\texttt{popFront}(S)$, with their typical meaning of adding/deleting elements to/from the beginning or ending of sequence S.

First notice that when adding or deleting elements we might be changing the alphabet of the tree. To cop with this, we assume that the underlying alphabet Σ is fixed and that the tree is constructed initially from a sequence mentioning all values in Σ. Thus, initially there is a leaf in the tree for every possible value. We also assume that in every moment there is an *active alphabet*, which is a subset of Σ, containing the values actually mentioned in the tree. To support this we just allow some sequences in the tree to be empty; if there is some value k of Σ not present in the tree at some point, then the sequence corresponding to the leaf node associated with k is the empty sequence. It is straightforward to adapt all the previous queries to this new setting.

Consider now $\texttt{pushBack}(S,\, a)$ and assume that before the update we have $|S| = n$. Then, besides adding a to the end of sequence S, we should update (or more precisely, create) $\texttt{mapLeft}_T(S,\, n{+}1)$ and $\texttt{mapRight}_T(S,\, n{+}1)$. If $a \leq \texttt{m}_T(S)$ then we let $\texttt{mapLeft}_T(S,\, n+1) = \texttt{mapLeft}_T(S,\, n) + 1$ and $\texttt{mapRight}_T(S,\, n+1) = \texttt{mapRight}_T(S,\, n)$, and then perform $\texttt{pushBack}(\texttt{Left}_T(S),\, a)$. If $a > \texttt{m}_T(S)$ then we let $\texttt{mapLeft}_T(S,\, n+1) = \texttt{mapLeft}_T(S,\, n)$ and $\texttt{mapRight}_T(S,\, n+1) = \texttt{mapRight}_T(S,\, n){+}1$, and then perform $\texttt{pushBack}(\texttt{Right}_T(S),\, a)$. Finally when we reach a leaf node, we just add a to the corresponding sequence.

The $\texttt{popBack}(S)$ operation is similar. Assume that $|S| = n$, then besides deleting the last element in S, we should only delete that element from the corresponding sub-

tree. Thus, if $s_n \leq \mathtt{m}_T(S)$ then we do $\mathtt{popBack}(\mathtt{Left}_T(S))$, and if $s_n > \mathtt{m}_T(S)$ we do $\mathtt{popBack}(\mathtt{Right}_T(S))$. When we reach a leaf node we just delete any element from it. Notice that in this case no $\mathtt{mapLeft}_T$ or $\mathtt{mapRight}_T$ needed to be updated.

The `pushFront` and `popFront` are a bit more complicated. When we do $\mathtt{pushFront}(S,\, a)$ we should do a complete remapping: if $a \leq \mathtt{m}_T(S)$ then for every $i \in \{1,\, \ldots,\, n\}$ we should do

$$\mathtt{mapLeft}_T(S,\, i+1) = \mathtt{mapLeft}_T(S,\, i) + 1$$
$$\mathtt{mapRight}_T(S,\, i+1) = \mathtt{mapRight}_T(S,\, i)$$

and finally set $\mathtt{mapLeft}_T(S,\, 1) = 1$ and $\mathtt{mapRight}_T(S,\, 1) = 0$ and perform the call $\mathtt{pushFront}(\mathtt{Left}_T(S),\, a)$. If $a > \mathtt{m}_T(S)$ then we should do

$$\mathtt{mapLeft}_T(S,\, i+1) = \mathtt{mapLeft}_T(S,\, i)$$
$$\mathtt{mapRight}_T(S,\, i+1) = \mathtt{mapRight}_T(S,\, i) + 1$$

and finally set $\mathtt{mapLeft}_T(S,\, 1) = 0$ and $\mathtt{mapRight}_T(S,\, 1) = 1$ and perform the call $\mathtt{pushFront}(\mathtt{Right}_T(S),\, a)$. When a leaf node is reached we just add a at the beginning of the corresponding sequence. The $\mathtt{popFront}(S)$ operation is similar. Let $|S| = n$. If $s_1 \leq \mathtt{m}_T(S)$ then we should update $\mathtt{mapLeft}_T\ (S,\, i)$ to $\mathtt{mapLeft}_T\ (S,\, i+1) - 1$, and $\mathtt{mapRight}_T(S,\, i)$ to $\mathtt{mapRight}_T(S,\, i+1)$ for all i from 1 to $n-1$, and then do $\mathtt{popFront}(\mathtt{Left}_T(S))$. Symmetrically if $s_1 > \mathtt{m}_T(S)$ then we should update $\mathtt{mapLeft}_T(S,\, i)$ to $\mathtt{mapLeft}_T(S,\, i+1)$, and $\mathtt{mapRight}_T(S,\, i)$ to $\mathtt{mapRight}_T(S,\, i+1) - 1$ for all i from 1 to $n-1$, and then do $\mathtt{popFront}(\mathtt{Right}_T(S))$. Upon reaching a leaf node, we just delete the value from the front.

The complexity of all the operations above is $O((A+M) \times \log\sigma)$ where A is the time needed to update `mapLeft` or `mapRight` in every level, and M is the time needed to compute the map functions. Just notice that for the cases of the `pushFront` and `popFront` we have to update several values of `mapLeft` and `mapRight` per level.

4. Implementation

In this section we explain how to build a wavelet tree and how to construct the auxiliary structures to support the mapping operations efficiently. Based on this construction we also discuss how to implement queries explained in the previous section. Additionally, we present an implementation strategy alternative to the direct pointer based one. We implemented both approaches in `C++` and the code is available in github[1].

[1] `https://github.com/nilehmann/wavelet-tree`

4.1. *Construction*

A wavelet tree implementation represents an array $A[0,\ n-1]$ of integers in the interval $[0,\ \sigma-1]$. Our construction is based on a node structure and pointers between those nodes. Every node v will be identified by two elements ℓ_v and r_v (which essentially correspond to $\mathtt{l}_T(v)$ and $\mathtt{r}_T(v)$ in Section 2), and an associated sequence A_v which is the subsequence of A formed by selecting elements in the range $[\ell_v,\ r_v]$. As we will see, values ℓ_v and r_v and the sequence A_v do not need to be explicitly stored and can be computed when traversing the tree if needed.

The construction of a wavelet tree starts creating the root node associated to the original array A and the interval $[0,\ \sigma-1]$. We then proceed recursively as follows. In each node v we found the middle of the interval $m_v = (r_v+\ell_v)/2$ (which corresponds to the value $\mathtt{m}_T(v)$ described in Section 2). We create two new nodes u and w as left-child and right-child of v, respectively. Then, we perform a stable partition of the array A_v into two arrays A_u and A_w, such that A_u contains all values less than or equal to m_v and A_w those greater than m_v. The construction continues recursively for the left node with the array A_u and the interval $[\ell_v,m_v]$, and for the right node with A_w and the interval $[m_v+1,\ r_v]$. The base case is reached when the interval represented by the node contains only one element, i.e., $r_v = \ell_v$. It is not necessary to store arrays A_v corresponding to each node v. They are only materialized at building time to construct the auxiliary structures to support the mapping operations required to traverse the tree as described below.

As previously discussed, the fundamental operations `mapLeft` and `mapRight` correspond to count how many symbols until position i belong to the left and right node respectively. To support these operations, when building a node v we precompute for every position i how many elements in the array A_v belong to the right node – they are greater than m_v – and store the results in an array C_v. We could store a similar array C'_v to store how many elements belong to the left node, but it is easy to note that values of both arrays are related as follows: $C'_v[i] = i - C_v[i] + 1$.

To understand how C_v is computed, it turns out useful to associate a bitvector B_v that marks with 0's elements less than or equal to m_v and with 1's those greater than m_v. This bitvector must support the operation of counting how many bits are set to 1 until a position i, which is commonly referred as a rank operation. Our array C_v is computed on build time as the partial sum of B_v by the recurrence $C_v[0] = B_v[0]$, $C_v[i] = C_v[i-1] + B_v[i]$, thus supporting the rank operation in constant time. The compression characteristics of the wavelet tree arise mainly because it is possible to represent these bitvectors succinctly while maintaining constant-time rank queries (Clark, 1998; Okanohara and Sadakane, 2007; Raman *et al.*, 2002). However, in a competitive programming setting memory constraints are less restrictive and our representation shows off to be sufficient. In case the memory is an issue, a practical and succinct implementation is presented by González *et al.* (2005).

4.2. *Implementing Queries and Updates*

We now briey discuss how every operation in Section 2 can be efficiently implemented.

`mapLeft` and `mapRight`. These two operations can be easily implemented with the array C_v; in a node v the number of elements until position i that go to the left is $i - C_v[i] + 1$. Since we are indexing from 0, position i is mapped to the left to position $i - C_v[i]$. Analogously, a position i is mapped to the right to position $C_v[i] - 1$. Notice that both mapping functions can thus be computed in constant time, which implies that `rank`, `quantile` and `range` operations can be implemented in $O(\log \sigma)$ time.

`swap`. The swap operation first map the position i down the tree until we reach a node v where the update needs to be performed. At this point the (virtual) bitvector B_v is such that $B_v[i] \neq B_v[i+1]$. Swapping both bits can only change the count of 1's until position i, and thus, only $C_v[i]$ should be updated. If $B_v[i] = 0$ we do $C_v[i] = C_v[i] + 1$, and if $B_v[i] = 1$ we do $C_v[i] = C_v[i] - 1$. This shows that the map functions can be updated in constant time after a swap operation, which implies that the complexity of swap is also $O(\log \sigma)$.

`toggle`. In this case we only need to implement `activeLeft`, `activeRight` and `activeLeaf`. To mark which positions are active we can use any data structure representing sequences of 0's and 1's that efficiently supports partial sums and point updates. For example we can use a *binary indexed tree* (BIT) (Fenwick, 1994) which is a standard data structure used in competitive programming that supports both operations in $O(\log n)$ time. Thus with a BIT we are adding a logarithmic factor for each query and now `rank`, `quantile` and `range` operations as well as `toggle` can be implemented in $O(\log n \times \log \sigma)$. In terms of construction, when using a BIT in every level we are only paying a constant factor in the size of the wavelet tree.

`pushBack` and `popBack`. These operations only modify the array C_v in some nodes. Pushing an element at the end updates the (virtual) bitvector B_v appending a new 0 or 1 (depending on the comparison between the new element and m_v), so C_v being a partial sum of B_v of size n_v only needs a $C_v[n_v] = C_v[n_v - 1] + B_v[n_v - 1]$ update. Popping an element from the end updates B_v and C_v doing the inverse operation, so if C_v is of size n_v we only need to delete $C_v[n_v - 1]$ from memory. Both operations can be done in amortized constant time using a dynamic array, thus the complexity of all queries plus `pushBack` and `popBack` is $O(\log \sigma)$ time.

`pushFront` and `popFront`. These are similar to `pushBack` and `popBack`, but act at the beginning of the bitvector B_v. To prepend a bit b to a bitvector B_v we must prepend its value to C_v. If the value of b is equal to 1 we must also increment by 1 every value in C_v. Because it is too slow to update every position of C_v, we define a counter δ_v that starts at 0 and is incremented by 1 every time a bit equal to 1 is prepended. We then just prepend $b - \delta_v$ to C_v, in which case the real count of ones until position i is obtained by $C_v[i] + \delta_v$. Popping an element is as easy as deleting the first element of C_v from

memory and decrementing δ_v by 1 if the value of $C_v[0] + \delta_v$ was equal to 1. If we want to mix front and back operations, we could use a structure such as a dequeue (Knuth, 1997), which allows amortized constant time insertions at the beginning and end of an array while maintaining constant random access time. Thus the complexity of all queries is still $O(\log \sigma)$ time.

4.3. *Big Alphabets and the Wavelet Matrix*

In a competitive programming setting the size of the array A will depend on time restrictions, but typically it will not exceed 10^6. However the number of possible values that A can store could be without any problems around 10^9. Thus the number of values actually appearing in A is much smaller than the range of possible values. For this reason one usually have to map the values that appear in the sequence to a range $[0,\ \sigma - 1]$. Commonly, this will require a fairly fast operation to translate from one alphabet to the other with a typical implementation using, for example, a binary search tree or a sorted array combined with binary search.

To avoid having this map operation, the wavelet tree could be constructed directly over the range of all possible values allowing the subsequences of some nodes to be empty. A naive pointer-based construction will require $O(\sigma)$ words which might be excessive for $\sigma = 10^9$. Because many nodes will represent empty subsequences, one can save some space explicitly tracking when some subsequences become empty in the tree.

There is an alternative implementation of the wavelet tree called *wavelet matrix* (Claude *et al.*, 2015) that was specifically proposed in the literature to account for big alphabets. Given an alphabet its size can be extended to match the next power of two, yielding a complete binary tree for the wavelet tree representation. For each level, we could then concatenate the bitvectors of each node in that level and represent the structure with a single bitvector by level. The border between each node is lost, but it can be computed on the fly when traversing. This means extra queries yielding worse performance. Instead, the wavelet matrix breaks the restriction that in each level siblings must be represented in contiguous positions in the bitvector. When partitioning a node at some level ℓ the wavelet matrix sends all zeroes to the left section of level $\ell + 1$ and all ones to the right. The left and the right child of some node at level ℓ do not occupy contiguous positions in the bitvector at level $\ell + 1$, but the left (resp. right) child is represented in contiguous positions in the left (resp. right) section of the level $\ell + 1$. Additionally, a value z_ℓ is maintained at each level to mark how many elements were mapped to the left.

With this structure the traversing operations can be directly implemented by performing rank operations on bitvectors at each level. Specifically, instead of maintaining an array C_v for every node, we maintain an array C_ℓ for each level. Array C_ℓ store the cumulative number of 1's in level ℓ. Then, a position i at level ℓ is mapped to the left to position $i - C_\ell[i]$ at level $\ell + 1$. The same position i is mapped to the right to position $z_\ell + C_\ell[i] - 1$.

The wavelet matrix has the advantage of being implementable using only $O(\log \sigma)$ extra words of memory instead of the $O(\sigma)$ used to store the tree structure in the pointer based alternative while maintaining fast operations. This $O(\log \sigma)$ words are insignificant even for $\sigma = 10^9$, which means that the structure could be constructed directly over the original alphabet. On the other hand the wavelet matrix is somehow less adaptable, because it does not support directly the `pop` and `push` updates. However, it can support `swap` and `toggle` in a similar way as the one described for the wavelet tree.

5. Wavelet Trees in Current Competitive Programming

We conducted a *social experiment* uploading 3 different problems to the *Sphere Online Judge* (SPOJ)[2]. All the problems can be solved with the techniques shown in the previous sections. We analyze the solutions to these problems submitted by SPOJ users. Our analysis reveal two main conclusions: (1) experienced programmers do not consider the use of wavelet trees, even in the case that its application is straightforward, and (2) for the most complex cases when they succeed, they use fairly involved techniques producing solutions that are dangerously close to time and memory limits. We have found, however, some incipient references of wavelet trees in the competitive programming community[3], as well as more detailed explanations in Japanese[4], which obviously establish an idiomatic barrier for many programmers.

We identify the three mentioned problems as ILKQ1, ILKQ2 and ILKQ3 and they are described as follows.

ILKQ1 considers a slightly modified version of the quantile query. The size of the initial sequences is 10^5, the range of possible integer values in the sequence is $[-10^9, 10^9]$, and the number of queries is 10^5. The time limit is 1s.

Link: `http://www.spoj.com/problems/ILKQUERY`

ILKQ2 considers rank queries plus toggling the state of arbitrary elements. The size of the initial sequence is 10^5, the range of possible values is $[-10^9, 10^9]$, and the number of rank plus toggle queries is 10^5. The time limit is 0.4s.

Link: `http://www.spoj.com/problems/ILKQUERY2`

ILKQ3 considers the quantile query of ILKQ1 plus swaps of arbitrary contiguous positions. The size of the initial sequences is 10^6, the range of possible values is $[-10^9, 10^9]$, and the number of quantile plus swap queries is 10^5. The time limit is 1s.

Link: `http://www.spoj.com/problems/ILKQUERYIII/`

Notice that ILKQ3 although involves the same query as ILKQ1, it is considerable harder as it can mix updates (in the form of swaps) and the input sequence can be 10 times bigger than for ILKQ1. Table 1 shows an analysis of the submissions received[5].

[2] `http://www.spoj.com/`

[3] `http://codeforces.com/blog/entry/17787`

[4] `http://d.hatena.ne.jp/sune2/20131216/1387197255`

[5] This data considers only until late March 2016.

Table 1
General submission statistics

	Submitted	Accepted	Non-accepted		
			WA	TLE	RTE
ILKQ1	49	9	19	18	3
ILKQ2	32	6	15	8	3
ILKQ3	35	2	12	15	6

5.1. *Analysis of Users Submitting Solutions*

We received submissions from several type of users, several of them can be considered as experienced programmers. From them, even expert coders (rank 100 or better on SPOJ) got lot of Wrong Answers (WA) or Time Limit Exceeded (TLE) verdicts which shows the intrinsic difficulty of the problems. Considering the three problems, 5 out of the 10 distinct users who got an Accepted (AC) veredict have rank of 60 or better on SPOJ, and 8 are well-known ACM-ICPC World finalists. For problem ILKQ3 we received only two AC. Both users solved the problem after several WA or TLE verdicts. For ILKQ1 and ILKQ2 the best ranked submitter was the top 1 user in SPOJ who obtained AC in both problems. For ILKQ3, the best ranked submitter was among the top 5 in SPOJ and obtained only TLE veredicts.

5.2. *Analysis of the Submitted Solutions*

As we have told before, we received 0 submissions implementing a wavelet tree solution. We now briefly analyze the strategies of the submissions received. For the sake of the space, we cannot deeply analyze every strategy but we provide some pointers for the interested reader.

The most common approach for ILKQ1 was sorting queries (as the problem is offline) plus the use of a tree data structure. One of the mainly used in this case was *mergesort tree*. In a mergesort tree, nodes represent sequences of contiguous elements of the original array and contains a sorted version of those sequences. Leaves represent one element of the array, and the tree is built recursively by merging pairs of nodes starting from the leaves. The construction can be done in $O(n \log n)$ time and space. Quantile queries can be answered by identifying the, at most $O(\log n)$, nodes that define a query range, and then doing two (nested) binary searches, one for counting elements less than or equal than a value X, and the second over X to find the k-th minimum element. The total strategy gives $O(\log^3 n)$ time which can be optimized up to $O(\log^2 n)$ using *fractional cascading*. This was enough given the time constraints.

For ILKQ2 and ILKQ3 sorting of queries or any offline approach is not directly useful as queries are mixed with updates. For ILKQ2 we received some submissions implementing a *square root decomposition* strategy, and run extremely close to the time limit. The most successful strategy in both problems was the use of ideas coming from *persistent data structures*, in particular *persistent segment trees*[6]. As in any persistent structure, the main idea is to efficiently store different states of it. Exploiting the fact that consecutive states do not differ in more than $O(\log n)$ nodes, it is possible to keep n different segment trees in $O(n \log n)$ space. Persistent segment trees can be used to answer quantile queries but need some more work to adapt them for updates like swaps as in ILKQ3. The two correct solutions that we received for ILKQ3 make use of this structure. It's relevant to notice that given the input size and the updates, implementing a persistent segment tree for this problem can use a considerable amount of memory. In particular, one of the AC submissions used 500MB and the other 980MB. Our wavelet tree solution uses only 4MB of memory.

6. Performance Tests

Existing experimental analyses about wavelet trees focus mostly on compression characteristics (Claude *et al.*, 2015). Moreover, they do not consider the time required to build the structure because from the compression point of view the preprocessing time is not the most relevant parameter. Thus, we conducted a series of experiments focusing on a competitive-programming setting where the building time is important and restrictions on the input are driven by typical tight time constraints. The idea is to shed some light on how far the input size can be pushed. We expect these results to be useful for competitors as well as for problem setters.

We performed experimental tests for our wavelet tree and wavelet matrix implementations comparing construction time and the performance of rank, quantile and range counting queries. We consider only alphabets of size less than the size of the sequence. To analyze the impact of the alphabet size, we performed tests over sequences of different *profiles*. A profile is characterized by the ratio between the size of the alphabet and the size of the sequence. For example, a sequence of size 10^3 and profile 0.5 has an alphabet of size 500.

Measurements. To measure construction time we generated random sequences of increasing size for different profiles. For each size and profile we generated 1,000 sequences and we report the average time. For queries rank, quantile and range counting, we generated 100,000 queries uniformly distributed and averaged their execution time. The machine used is an Intel® Core™ i7-2600K running at 3.40GHz with 8GB of RAM memory. The operating system is Arch-Linux running kernel 4.4.4. All our code are single-threaded and implemented in `C++`. The compiler used is `gcc` version 5.3.0, with optimization flag `-O2` as customary in many programming contests.

[6] `bit.ly/persistent-segment-tree`

Results. No much variance was found in the performance between different profiles, but as may be expected sequences of profile 1 – i.e., permutations – reported higher time in construction and queries. Thus, we focus on the analysis of permutations to test performance on the most stressing setting. For the range of input tested we did not observe big differences between the wavelet tree and the wavelet matrix, both for construction and query time. Though there are little differences, they can be attributed to tiny implementation decision and not to the implementation strategy itself.

Regarding the size of the input (Fig. 2), construction time stays within the order of 250 milliseconds for sequences of size less than or equal to 10^6, but scales up to 2 seconds for sequences of size 10^7, which can be prohibitive for typical time constraints. For the case of queries rank, quantile and range counting we report in Fig. 3 the number of queries that can be performed in 1 second for different sizes of the input sequence. For rank and quantile, around 10^6 queries can be performed in 1 second for an input of size 10^6. In contrast for range counting, only 10^5 queries can be performed in the same setting (Fig. 3).

It would be interesting as future work to perform a deep comparison between the wavelet tree and competing structures for similar purposes such as mergesort trees and persistent segment trees, testing time and memory usage. From our simple analysis in the previous section one can infer that wavelet trees at least scales better in terms of memory usage, but more experimentation should be done to draw stronger conclusions.

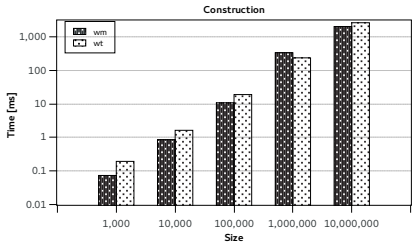


Fig. 2. Construction time in milliseconds.

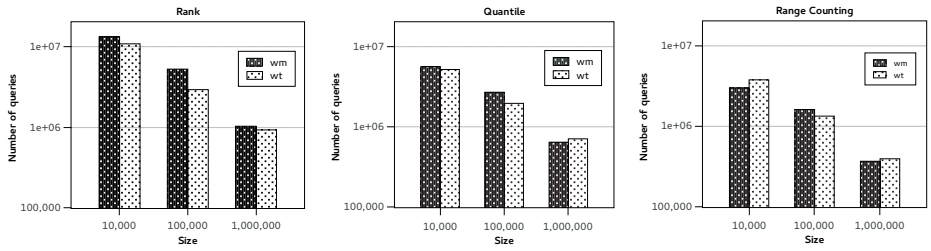


Fig. 3. Number of queries that can be performed in one second.

7. Concluding Remarks

Problems involving advanced data structures are appearing increasingly often in world-wide programming competitions. In this scenario, competitive programmers often prefer versatile structures that can be used for a wide range of problems without making a lot of changes. Structures such as binary indexed trees or (persistent) segment trees, to name a few, conform part of the lower bound for competitors, and must be in the toolbox of any programmer. The wavelet tree has proven to be a really versatile structure but, as we have evidenced, not widely used at the moment. However, we have noted that some programmers have already perceived the virtues of the wavelet tree. We believe that the wavelet tree, being quite easy to implement, and having such amount of applications, is probably becoming a structure that every competitive programmer should learn. With this paper we try to fill the gap and make wavelet trees widely available for the competitive programming community.

Acknowledgments

J. Pérez is supported by the Millennium Nucleus Center for Semantic Web Research, Grant NC120004, and Fondecyt grant 1140790.

References

Clark, D. (1998). *Compact Pat Trees*. PhD thesis, University of Waterloo.

Claude, F., Navarro, G., Ordóñez, A. (2015). The wavelet matrix: an efficient wavelet tree for large alphabets. *Inf. Syst.*, 47, 15–32.

Fenwick, P. M. (1994). A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24(3), 327–336.

Gagie, T., Navarro, G., and Puglisi, S.J. (2012). New algorithms on wavelet trees and applications to information retrieval. *Theor. Comput. Sci.*, 426, 25–41.

Gagie, T., Puglisi, S.J., Turpin, A. (2009). Range quantile queries: another virtue of wavelet trees. In: *SPIRE*. 1–6.

González, R., Grabowski, S., Mäkinen, V., Navarro, G. (2005). Practical implementation of rank and select queries. In: *WEA*. 27–38.

Grossi, R. (2015). Wavelet trees. In: *Encyclopedia of Algorithms*. Springer.

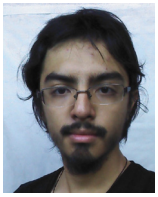
Grossi, R., Gupta, A., Vitter, J.S. (2003). High-order entropy-compressed text indexes. In: *SODA'03*. 841–850.

Knuth, D. (1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Third Edition. Addison-Wesley. 238–243.

Navarro, G. (2014). Wavelet trees for all. *J. Discrete Algorithms*, 25, 2–20.

Okanohara, D., Sadakane, K. (2007). Practical entropy-compressed rank/select dictionary. In: *ALENEX*. 60–70.

Raman, R., Raman, V., Rao, S. (2002). Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In: *SODA*. 233–242.



R. Castro is a computer engineering student at Department of Computer Science, Universidad de Chile, bronze medalist of IOI'2013 in Brisbane, and problem setter for OCI (Chilean Olympiad in Informatics).

N. Lehmann is master's student in computer science at Department of Computer Science, Universidad de Chile. His research interests are semantics, design and implementation of programming languages and type systems. He is the Scientific Committee Director of OCI (Chilean Olympiad in Informatics). Deputy Leader at IOI 2014 and 2015. Chilean Judge of ACM ICPC 2014 and 2015.

J. Pérez is Associate Professor at the Department of Computer Science, Universidad de Chile, and an Associate Researcher of the Chilean Center for Semantic Web Research. His research interests are database theory, data exchange and integration, graph databases, and the application of database technologies to the Semantic Web and the Web of Data. He is one of the directors of OCI (Chilean Olympiad in Informatics), and the Chilean team leader at IOI since 2013.

B. Subercaseaux is an undergraduate student of computer science at Department of Computer Science, Universidad de Chile, and problem setter for OCI (Chilean Olympiad in Informatics).

Olympiads in Informatics, 2016, Vol. 10, 39–60

DOI: 10.15388/ioi.2016.03

Learning Programming through Games and Contests: Overview, Characterisation and Discussion

Sébastien COMBÉFIS[1,2], Gytautas BERESNEVIČIUS[3]
Valentina DAGIENĖ[3]

[1] *Electronics and IT Unit, École Centrale des Arts et Métiers (ECAM)*
Promenade de l'Alma 50, 1200 Woluwé-Saint-Lambert, Belgium
[2] *Computer Science and IT in Education ASBL, Belgium*
[3] *Vilnius University Institute of Mathematics and Informatics*
4 Akademijos Street, Vilnius LT-08663, Lithuania
E-mails: s.combefis@ecam.be; gytber@gmail.com; valentina.dagiene@mii.vu.lt

Abstract. Learning of programming and, more generally, of computer science concepts is now reaching the public at large. It is not only reserved for people who studied informatics (computer science) or programming anymore. Teaching programming to schoolchildren presents many challenges: the big diversity in ability and aptitude levels; the big amount of different tools; the time-consuming nature of programming; and of course the difficulty to motivate schoolchildren to keep them busy with hard work. There are various platforms that offer to learn coding and programming, in particular game-based platforms, which are more and more popular. These latter exploits of the gamification process focused on increase in motivation and engagement of the learners. This paper reviews the main kinds of online platforms to learn programming and more general computer science concepts, and illustrates the review with concrete platforms examples.

Keywords: game-based learning, gamification, learning programming, online programming platform, programming contest.

1. Introduction

Informatics (computer science or computing) as a science discipline, and in particular programming as an important part of that, is gaining a lot of popularity these years in education. As core subjects in a computer science (CS) major, programming subjects play an important role in a successful CS education. One of the greatest challenges faced by most students is the understanding of programming basics, especially for novices or for those who are in their first year of studies. Students develop algorithmic

thinking or computational thinking at secondary and even at primary school. Using simple programming languages such as Scratch became accessible to young children (Maloney *et al.*, 2004).

Games have been used for educational purposes for decades. The popularity of games has led to the idea of using them in the learning of programming, taking advantage of the engaging features of games. In a very broad sense, two different approaches are used when using games or game-like elements in educational contexts: 1) gamification and 2) serious games. The term *gamification* is commonly defined as the use of game design elements in non-game contexts (Deterding *et al.*, 2011). Serious games are associated with a standalone game or platform, as well as indicated as a computer game (Djaout *et al.*, 2011). In this paper, we are going to use the general term to refer to game-based learning with main focus on teaching programming.

Game-based learning is concerned with using games not for entertainment, but for educational purposes. Those who work within the field of gamification focus on identifying the context and conditions that support the integration of digital games within informal and formal learning environments. Educational scientists have pointed up several features of games that allow them to be used as learning tools. For example, games are engaging (Dickey, 2005) and motivating (Prensky, 2003). They also provide a lot of experiences (Arena and Schwartz, 2013) and an excellent feedback on performances (Shute, 2011). Finally, games support very well the learner centred education (Gee, 2005).

A contest can be seen as a part of game-based learning. Contests make the teaching of programming more attractive for students. Furthermore, programming with computer is one of the appropriate and effective ways to develop problem solving skills and computational thinking. During contests, students have the possibility to compare their abilities and learn from others. There have been many contests in programming throughout all over the world; most of them focus on algorithmic problem solving.

Many teaching environments already contain game-like elements such as points, instant feedback, and goals. However, there are engaging aspects in games that could be used in educational settings more widely. In well-designed games, even a failure can be a reward and triggers positive emotions (Ravaja *et al.*, 2005). There is no off-the-shelf formula for designing successful games, nor is there one perfect learning environment. However, by deepening our understanding of the effects of games and game-like environments in educational settings, we can design and support more effective learning activities, and ultimately improve the learning of computer science.

The move from classical learning platforms to online contests and games can be explained as a way to ensure the best motivation as possible for their users. The online platforms can provide tools such as rankings, duels, discussion rooms, etc. to motivate their users to participate regularly. Finally, research on gamification in all its forms has proven that educational games improve the engagement of learners if the game is designed properly (Barata *et al.*, 2013; Nah *et al.*, 2014). Online platforms where students can learn programming are being developed all over the world, ranging from simple direct learning platforms to platforms of games that indirectly teach programming (Combéfis and Wautelet, 2014).

This paper reviews literature of the last decade on programming education for school students (high, secondary and primary levels) using games and online platforms that support games and contests which goal is to teach programming. The literature review of programming games and an overview of programming learning platforms were structured to accomplish the following three research objectives:

1. To overview the last decade literature on programming education using games.
2. To identify the important online platforms for learning programming through games and contests.
3. To identify suitable tools for programming novices.

The goal of the paper is twofold: from one side, to provide an overview of the work that has been done in this area, and from the other side, to discuss the important tools (online platforms and contests) and to improve programming education. After this introductory part, Section 2 makes a short presentation of the understanding of gamification and overviews the literature on teaching programming using games. Section 3 presents three kinds of online platforms, which allow students to learn programming. Then Section 4 tours several existing contest platforms. Finally, the last section concludes the paper with some thoughts about what could be the future evolution of these online platforms and contests and makes some suggestions for teaching programming by educational games or by other ways.

2. Short Overview of Literature

The application of game design elements of non-game scenarios is known as gamification. Some authors stated that gamification uses elements of experiences (Deterding *et al.*, 2011). It must also impart some rules or structure to the experience to influence participants' action. Gamification adds game design elements as well as promoting the psychological benefits and motivational ability of games but in different contexts.

Programming games are an important part of educational games. A programming game is a computer game where the player has little or no direct influence on the course of the game. Instead, a computer program or script is written in some domain-specific programming language in order to control the actions of the characters or other entities.

There are several definitions of serious games. For example, Alvarez and Michaud (2008) state that a serious game must include a genuine entertainment element combined seemingly with a practical dimension. Some researchers argue that all games have a serious purpose, such as gambling. That is why a majority of simulators would be considered as serious games (Sawyer and Rejeski, 2002).

Game design may be defined as *"the action of making sense of things related to a game"* (Mora *et al.*, 2015, p. 3). Another description is *"the act of deciding what a game should be"* (Schell, 2008, p. xxxviii). Game design is related to enjoyment, while gamification points towards a business objective. Similarity of game design and gamification design is that both rely on the principles of game design theory.

2.1. *Related Works*

The ACM and IEEE computer society state that *"computer science is becoming one of the core disciplines of a 21st century university education, that is, something that any educated individual must possess some level some level of proficiency and understanding of proficiency and understanding"* (ACM/IEEE…, 2013, p. 48). However, there are a lot of students who are becoming less interested in computer science. Colleges and universities routinely report that almost half of those students who initially choose computer science study soon decide to abandon it. There are high dropout rates as well as high failure rates of high school students in programming courses. Decades of effort have been put into decreasing these rates.

There were several approaches to motivate students to learn programming. Programming fundamentals may be hard skills to learn for students, especially for novices. In order to help students, there are several attempts to teach programming by using educational or serious games.

Programming courses are an important part in computer science subjects. Object-oriented programming is difficult for students, especially first year students. The embedding of gamification in programming courses could maximize student participation and have a positive impact on learning (Azmi *et al.*, 2015). The gamified social activity is designed to promote interaction among students and they may assist and give feedback to each other online. Online collaborative learning and the participation of students should be emphasised as well. Moreover, aesthetics is also a very important part in a game, because it makes it *"fun"*, which contributes to engage students in the learning process. The embedding of gamification in programming courses were analysed and showed that gamification in online collaborative learning can enhance participation among first year programming students. The participation is an important part in each computer science study due to motivation and lower dropout rates.

Vihavainen *et al.* (2014) present another approach in a literature overview. Their paper reports about 60 pre-interventions and post-interventions and analyses their influence on students' pass rates in programming courses. Statistically, pass rates after these interventions increased nearly by one third compared to traditional teaching. A more detailed discussion is presented about seven courses on gamification intervention after which pass rates increased by 10.8% on average.

Tillmann *et al.* (2013) deals with an alternative platform called *"Pex4Fun"*. On this platform, grading of traditional MOOCs (*Massive Open Online Course*) assignments has been changed to an automated grading assignment system based on the symbolic execution, designed for students and teachers. This platform is a game-based learning and teaching environment, where teachers may create interactive games for students. Students can learn programming by playing programming games in *"Pex4Fun"* as well as having programming duels between each other.

There are several important components of game elements. As Hunicke *et al.* (2004) have discovered the game mechanics represents data and algorithms. Game dynamics is the lifecycle of the mechanics that act to engage a player's input and other's output. Also, game aesthetics plays important role in the emotional part of the player in order to keep them engaged with the game.

Several researchers worked on more detailed classification of games. For example, Azmi *et al.* (2015) classified game elements into three categories 1) game mechanics (badges, leaderboard, points, levels...), 2) dynamics (personal dynamics – desire for reward, personal promotion; social dynamics – altruism; achievements, peer collaboration), and 3) aesthetics (challenges). The gamified design needs to embed collaborative elements for collaboration among the students. Aesthetics is recommended element in educational games, because it makes the game *"fun"* and engages with the learning process. Online educational game should have the online support as well as collaborative learning.

While there is still scepticism that something called a game could be anything more than a leisure activity, serious organizations are getting serious results with serious games. Serious games may be defined as an application with three main components: experience, entertainment, and multimedia (Laamarti *et al.*, 2014). Especially entertainment dimension in serious games has the potential to enhance the user's experience through multimodal interaction. Additionally, digital serious games contain different media.

The paper of Laamarti *et al.* (2014) overviews studies on serious games in different application areas including education, well-being, advertisement, cultural heritage, interpersonal communication, and health care. Also, many platforms were analysed and discussed, what the necessary components of the game to be successful are. A survey on digital serious games in general is conducted and a taxonomy of serious games is created.

Several components of serious games were distinguished and categories of criteria were suggested (Laamarti *et al.*, 2014). These criteria may be useful for any programming games as well. So, providing guidance to players within the game is important. This will provide them with the necessary knowledge and prevents them from feeling *"lost"* or disturbed. Another thing is avoiding negative consequences in the game, because otherwise it may result in the player's low performance. Music element makes players play the game longer, because of more motivation and engagement will be as an effect of this. Multiplayer collaborative exercise games are more motivating and engaging than single-player appropriate games. Collaborative environment increases players' motivation. It is significantly important to offer challenges in the game for children, but the challenges must be at the right level (neither too high, nor too low). It is the key in keeping the players' interest in the game. Educational games need to take into consideration sound instructional models to be successful.

More detailed recommendations are provided for educational games designed for using in a classroom. School teachers are concerned about the curriculum and they also have limited time resources. It is recommended that educational games would be based on the curriculum due to the acceptance of many teachers and integration in the class.

Additional recommendations for necessary components of the games to be successfully used can be as follows (Laamarti *et al.,* 2014):

a. **User-centred software engineering**: an important element is the perspective that the designers contribute to the development teams and the experience that the player will obtain.
b. **Multimodal games**: multiple modalities should be incorporated (e. g. visual, auditory and haptic combination).

c. **Social well-being**: stimulating a feeling of virtual presence or connectedness of social well-being in real life.
d. **Adaptive gaming**: a serious game should adapt or personalize a particular player's capabilities, needs, and interests.
e. **Standardization of evaluation**: heuristic evaluation standards must turn into a reality. That is useful for acquiring higher credibility.
f. **Sensory-based simulations**: serious games can be created based on real world sensory data so that real world scene would be accurately reconstructed.

Many commercial games demonstrate the properties of sound instructional design, even in games not intended for educational purposes (Becker, 2008). The author has recommended the following success factors for educational games: 1) Gain attention; 2) Inform learners of the objective; 3) Stimulate recall of prior learning; 4) Present stimulus material; 5) Provide learning guidance; 6) Elicit performance; 7) Provide feedback; 8) Assess performance; 9) Enhance retention and transfer. These factors were based on well-known learning and instructional theories.

One suggestion for recreating games is to modify an existing non-educational games into programming games (Muratet *et al.*, 2010). The authors have used existing online non-educational games ideas to develop their own serious games. For example, there is a game "*Kernel Panic*", having following features: it has no resource management except for time and space; all units are free to create; it has a small technology improvement tree with less than ten units; and it uses low-end vectorial graphics, which match the universe. In this game a player gives orders to the units to carry out operations (i.e. moving, building, and so forth). So the authors (Muratet *et al.*, 2010) changed the game design that the player has to write a program code in order to make the actions of his units and play. The authors modified some games into serious games and students, e.g. players may play these games using different programming languages: C, C++, Java, OCaml, Ada and an interpreted language called *"Compalgo"*.

Another paper (Cuba-Ricardo *et al.*, 2015) describes some characteristics and regularities from the behaviour of three students as they solved a programming problem as well as reveals the methodology stages using several methods, techniques and tools. The paper is limited by the time of contestants, who have between 60 and 90 minutes in contest of final states.

Successful computer programmers must have several cognitive skills as listed by Surakka and Malmi (2004). Contestants' opinions must be taken into account in order to have full view of their outputs. So, it was decided to use computer programs that take screenshots in short periods of time. The methodology of characterizing the cognitive process of solving programming problems was organized into five stages:

a. Preparation of the process.
b. Recording the process of exercise application.
c. Analysis, processing and assessing of the partial reports (observation of pictures).
d. An interview session.
e. Final assessment.

The implementation of the methodology revealed that programmers' took notes in the worksheets and the reasoning followed when determining the solution algorithm. So, monitoring programming students' actions is a useful method to discover not only final outputs, but intermediate results of the learning process as well. We think it may be one of the essential issues to be taken into consideration of teaching programming curriculum, especially for novices. Monitoring process could be done by computer automated programs and be used as a general method (e.g. it may be a live survey after your programming contest).

Researchers focus on investigating successful components of educational games that could help to reduce dropout rates of students in computer science. The embedding of gamification in programming courses can be one of solutions for that: it can help to maximize student participation and learning, motivate and reduce dropout rates, especially for novices in programming (Azmi *et al.*, 2015). Some platforms, e.g. *Prog&Play* may help even to recruit students in computer science (Muratet *et al.*, 2010). Intervention of educational games in programming courses raises pass rates by 10.8% on average (Vihavainen *et al.*, 2014).

A feedback is an important and necessary part in education and educational games (Azmi *et al.*, 2015; Cuba-Ricardo *et al.*, 2015; Tillmann *et al.*, 2013). Good feedback can be implemented by providing guidance to players within the game so that they do not feel *"lost"* (Becker, 2008) or by avoiding negative consequences in the game. Another solution is to make the game a multiplayer one to motivate players to play the game longer.

The most successful factors are multiple modalities of games, players' collaboration, adaptive or personalized game components based on real world sensory data (Laamarti *et al.*, 2014). Non-educational games may be gamified and used for teaching programming courses as well (Muratet *et al.*, 2010). Contests and duels in programming games are engaging and may be used to raise programming skills (Cuba-Ricardo *et al.*, 2015).

All the authors that are mentioned in this section have stated that educational games motivate students to learn. In general, you may use all these researched factors when your own create programming games.

2.2. *Taxonomy of Educational Games*

As there is growth of researches in the educational games field and it is a necessity to improve the value of teaching or learning programming by educational game, we were working on the taxonomy of educational games. We adapted the framework of serious games' taxonomy based on the paper (Laamarti *et al.*, 2014). Firstly, we will provide a short introduction to the mentioned taxonomy. After that we will discuss our novelties and provide a classification of several online programming games.

Five categories for serious games classification are presented by (Laamarti *et al.*, 2014, p. 6):

1. **Application Area** refers to different application domains (education, well-being, training, advertisement, interpersonal communication, health care, others).

2. **Activity.** It depends on the activity of a player to play a game (physical exertion, psychological, mental).
3. **Modality.** Here it is the channel by which information is sent from the computer to the human(s) participating in the game (visual, auditory, haptic, smell, others).
4. **Interaction Style.** The interaction style defines whether the interaction of the player with the game is done using keyboard, mouse, or Joystick or using some intelligent interfaces, e. g. a brain interface, eye gaze, movement tracking, and tangible interfaces. The research showed that choosing the right interface may have an impact on the success of the game.
5. **Environment.** This criterion defines the environment of the digital game (social presence, mixed reality, virtual environment, 2D/3D, local awareness, mobility and online).

We investigated the classification and found limitations, especially when applying for programming education by using games. For educational online programming games an extension and adaptation of the proposed taxonomy are needed (Fig. 1).

Application area. So, an application area of programming games is only one – education to program educational games. A field of the programming educational games activity usually is a mental process. We will have in mind that application area of educational programming games is education, activity field is mental and we will omit these fields as a result.

Modality. Usual modality fields of educational programming games are visual and auditory, but sometimes it may be haptic.

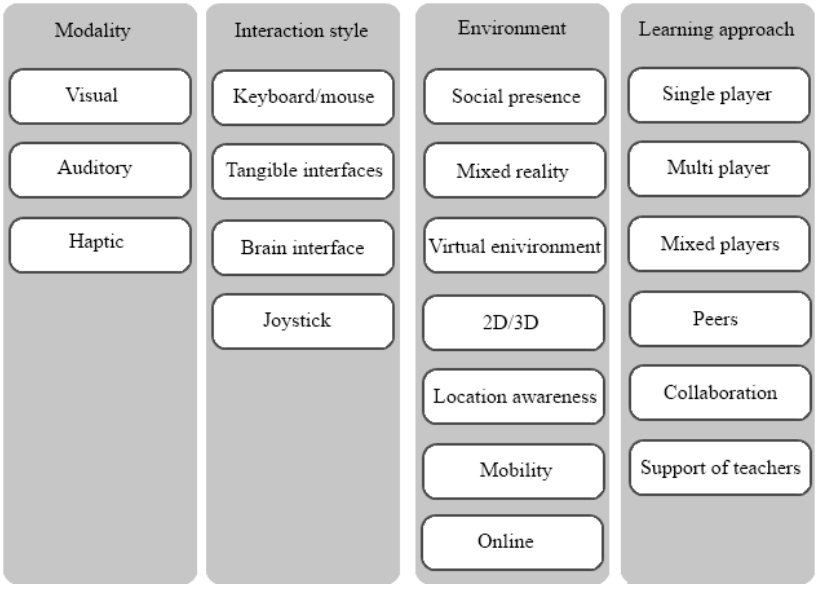


Fig.1. The classification of the educational programming games (adapted and extended schema of Laamarti *et al.*, 2014).

Interaction style. Interaction style in our taxonomy is keyboard/mouse, tangible interface, brain interface and joystick, because educational games usually may be classified just in these fields. But we remain open to discussion about this question.

Environment. As environments of programming games may be any of Laamarti *et al.* suggested environments, we include all of the areas: social presence, mixed reality, virtual environment, 2D/3D, local awareness, mobility and online.

Learning approach. We include this new field to classification of educational games. This is because analysed research papers on educational games pays a lot of attention to successful methods of teaching or learning. We propose that the learning approach of educational games may be classified into single player, multi player, mixed players (part of the game may be single-player game and part of the game may be multi-player game), peers (it is important to pay attention to communication between peers), collaboration (some educational games may teach team work), support of teachers (students easier learn while they are playing with a help of teachers/lecturers).

3. Online Platforms for Learning Programming

Several different kinds of online platforms to learn programming do exist. The most classical platforms are able to automatically execute code and provide feedback. They provide lessons that the learner can follow and propose interactive coding exercises with immediate correction. Combéfis and le Clément de Saint-Marcq (2012) proposes a short review of such platforms. Swacha and Baszuro (2013) propose such a classic platform, but they included game elements in it. For example, the courses in their platform are characterised by the set of completed areas, the number of earned points and the attained level. Moreover, it is possible to challenge other students and take part in contests.

In addition to the specific platforms to learn coding and programming, online courses are also proposed in the form of MOOCs (Combéfis *et al.*, 2014). Another example is Khan Academy, which provides interactive lessons and videos with coding exercises (Morrison and DiSalvo, 2014). Both these kinds of platforms propose a full course with videos for the theory and then practical coding exercises with more or less detailed direct automated feedback. They are designed with the learning process as the main objective.

This section presents three main categories of online game platforms with different goals: learn to code, learn algorithmic thinking and learn to create games. Proposed examples are discussed according to the background presented in the previous section, for the particular case of learning computer science skills. The last subsection, then discusses how and what game elements can be added to an online platform in order to gamify it.

3.1. *Learn to Code*

The first category of game platforms contains those whose goal is to make their users learning and training to code. Coding games require the learner to understand and to be

able to write code to solve challenges. Different coding activities are possible, the main ones being:

- **Bugfix**: the user is given a program that contains a bug to be fixed.
- **Recovery**: the user is given a program where some parts are missing that have to be filled.
- **Code writing**: the user has to write an instruction, a function or a program from scratch.
- **Agent**: the user has to write a program that represents the behaviour of an intelligent agent.

To help the learner to identify the bug, additional information is provided in the statement of the challenge. For example, a precise specification of the program or the results of the execution of test sets can be provided. Once the learner has written/fixed the code, he submits it and then gets a feedback. As summarised by Combéfis and Wautelet (2014), feedback is very important for the learning process. Those feedbacks can take several forms such as:

- A simple succeeded/failed status.
- The result of the execution of test sets.
- A textual feedback providing hints regarding the failure.

Codecademy is an example of an online game platform to learn how to code (Fig. 2). The left part of the window provides the explanations of the new concept to learn. Instructions about the exercise are then provided at the very bottom of this left pane. The learner can directly code in the editor located in the right part and then submit his/her code for correction. If the code is wrong, a simple feedback trying to explain why is provided and if it is right the learner earns a badge, increases his/her progress and can go to the next lesson.

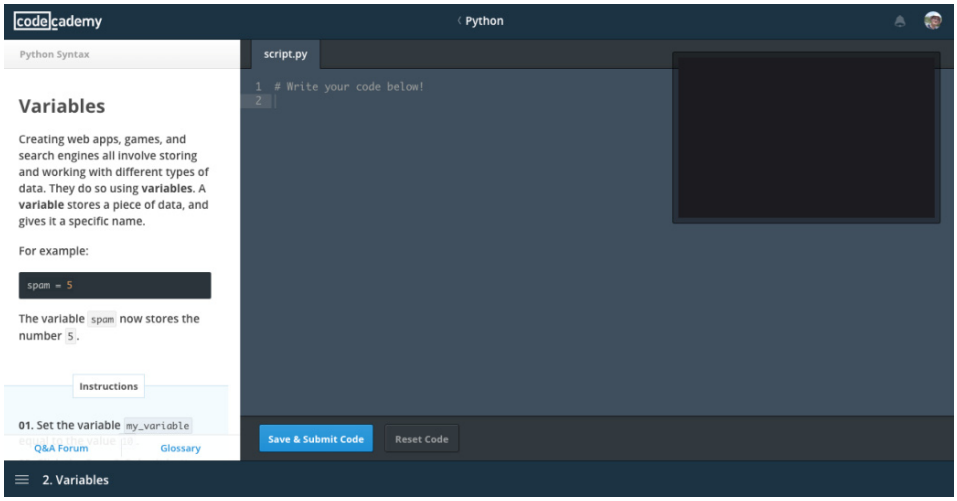


Fig. 2. *Codecademy* proposes a course composed of lessons in which you have to write code and for which you will earn badges and make progress whenever you succeed.

3.2. *Learn Algorithmic Thinking*

Coding is not the only field of computer science that can be taught through online platforms. As described by Combéfis *et al.* (2013), it is also possible to grow algorithmic thinking through interactive problems, in particular to learn programming concepts.

For example, *LightBot* (`https://lightbot.com`) is used to learn the notion of programming and in particular recursion through a simple game where the user has to solve puzzles. Each puzzle requires the learner to write a program composed of visual blocks to drive a robot to a goal. Fig. 3 shows the main window of the game where the robot and its environment are shown on the left and the program representing the behaviour of the robot is shown on the right. *LightBot* is not a platform to learn how to code, but to teach programming concepts. In the more advanced levels, the notions of function and therefore recursion are introduced.

Another example is *Initial Conditions* (`https://reheated.org/games/initial`), shown in Fig. 4, where the player has to find a solution to a search problem. In this game, the player faces to a grid with rivers and has to place a certain number of villages so that a river passes through them and so that no two cities are adjacent (horizontally or vertically).

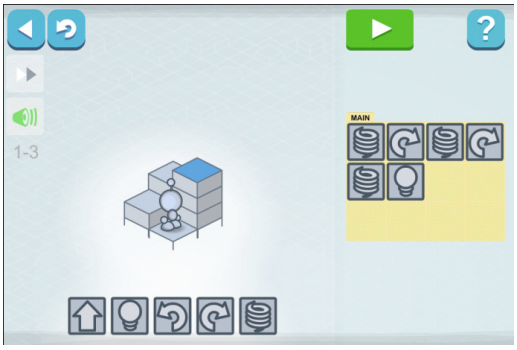


Fig. 3. In *LightBot*, the player has to find a correct sequence of actions to perform so that the robot reaches the blue cell and lighten it.

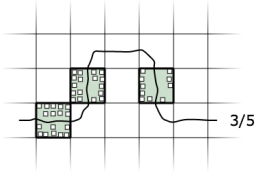


Fig. 4. In *Initial Conditions*, the player has to place cities on a map so that a river is passing through them and so that two cities cannot be adjacent. This particular map has one river and the player has to place five cities.

This problem is a typical an artificial intelligence (AI) search problem that could for example be solved using any exploration algorithms, or with a constraint programming solver, for example. Again, it is not explicitly explained to the learner who just sees a game with puzzles to solve. Playing the game will challenge the learner and should develop algorithmic thinking in learner's mind.

Again, if the purpose is educational, the provided feedback is very important. For *LightBot*, the feedback is visual since the learners can visualise the execution of their programs and therefore visually debug them if the robot failed to reach the goal. For Initial Conditions, the only feedback is a success/failed status indicating which cities are violating the constraints or which river is missing villages.

3.3. *Learn to Create Games*

Finally, another kind of online programming learning platforms offers the possibility for the users to create their own games. On these platforms, the learner has to program a game, typically with a visual programming language. For example, *Scratch* (`https://scratch.mit.edu`) uses a visual block programming language to program the behaviour of sprites.

The main window of the application where the game is shown on the left part and the code is shown on the right (Fig. 5). The behaviour of each sprite can be programmed and they can also communicate together.

Another similar platform is *Flowlab* (`http://flowlab.io`) where the game is represented by flowchart diagrams (Fig. 6). As in *Scratch*, the game is built with sprites whose individual behaviours can be programmed.

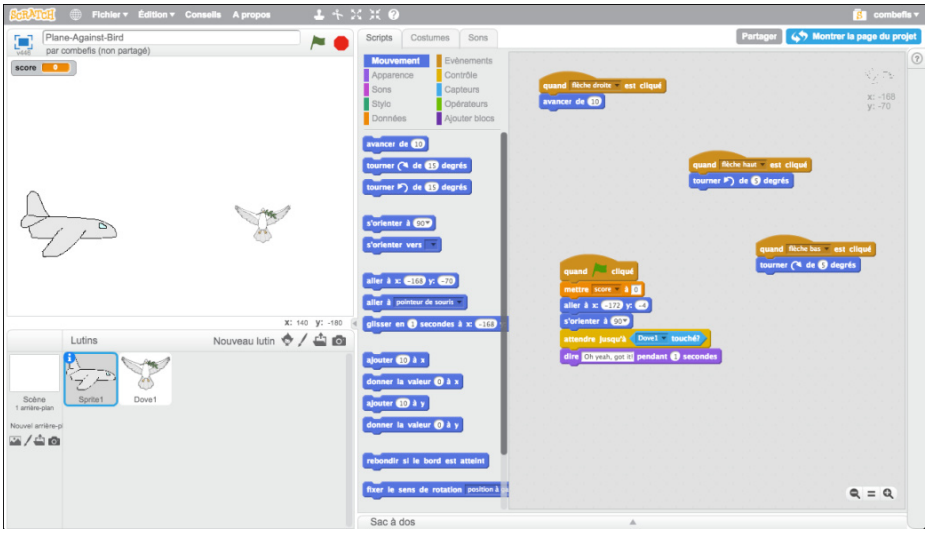


Fig. 5. The main window of the *Scratch* program is split into two parts: the left part shows the sprites of the game and the right part shows the programs representing the behaviour of the different sprites.

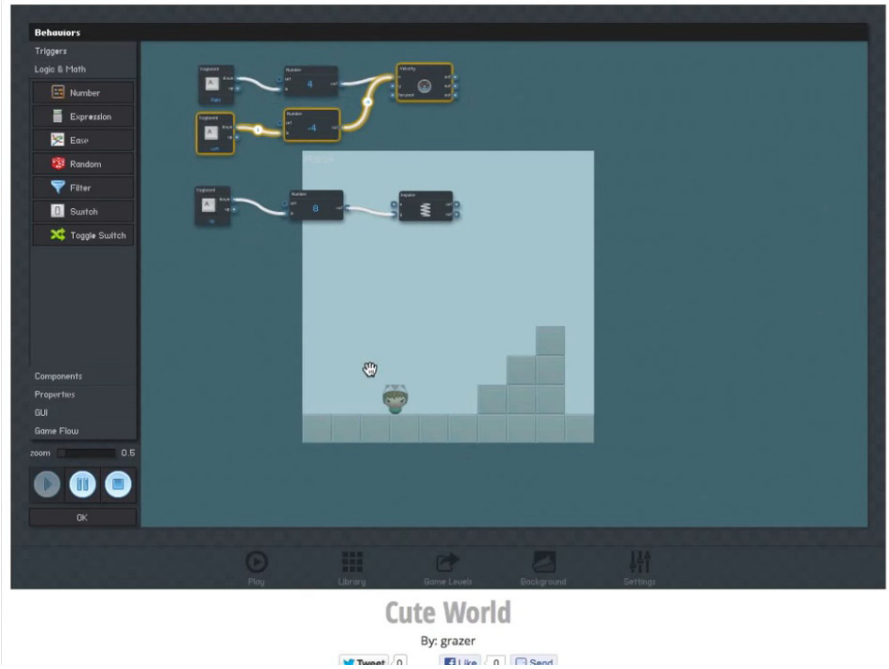


Fig. 6. In *Flowlab*, the user programs the game using flowchart diagrams representing the behaviours of the different sprites it included in the game.

This last category requires the learner to be able to program, but not necessarily to write code. The focus is put on the creativity and on the skills needed to design and architecture an application.

3.4. *Gamification of an Online Learning Platform*

Proposing an online platform is useful to make the material available to every learner. But only relying on the online aspect is not enough to motivate them to learn. As detailed in the second section, gamification can help to foster learners to make progress, increasing their engagement and motivation. Several kinds of elements can be introduced to build online game platforms:

- Points and rankings amongst players.
- Levels, grades, badges that are related to the progression.
- Live fights/contests between players or against bots.

One other important element that can be taken into account is the contextualisation of the statements of the challenges and exercises. There are indeed two main strategies that can be used to gamify an online platform to learn programming. Either game elements can be added to an already existing platform (points, rankings, badges…) or the platform can be rethought and redesigned so that to transform it completely into

a game. As highlighted by Morrison and DiSalvo (2014) who relate the gamification of the Khan Academy, one critical motivational element to have is to place the user at the centre by adding elements of pure play, to make the gaming more "playful". Examples of such approaches include *CodeSpells* (Esper *et al.*, 2013) and *Gidget* (Lee *et al.*, 2013).

Ibáñez *et al.* (2014) made an experiment to explore the impact of gamification techniques on the engagement and learning about the C programming language. The authors conclude that the gamification they performed has a positive impact on the students' engagement. Most students continued to work even after having earned the maximum amount of points. O'Donovan *et al.* (2013) present a case study where they used gamification in a university-level course about game development. In their paper, the authors provide insights about gamification mechanics and explain how the design of their game has been created and which game elements have been used. Their research allows them to conclude that the gamification they performed in a university setting was effective, increasing the engagement and understanding of the students. Again, it is a situation where the platform has been thought as a game from the starting point.

4. Online Programming Contests

This section presents several online game platforms, so that to cover the different tendencies that exist today and the characteristics presented in the previous section. The focus has been put on platforms that available for free (at least a standard version).

4.1. *Leek Wars*

Leek Wars (`http://leekwars.com/`) is a game where the player has to program the behaviour of a leek. The goal of the leek is to defeat another leek during a fight that takes place in a garden. Depending on the level of the player, his/her leek can be equipped with more weapons, health-recovering objects, etc.

The leeks are programmed in JavaScript and an API provides functions to manipulate the leak and to get information about the environment. This API is very well documented, which also trains learners to manipulate such documentation.

The platform offers the possibility to develop several AIs and to choose which one to use at any time. The player can experiment his/her AIs against bots, before jumping into the garden to fight with other real players. To motivate the player, a level system has been put in place. For each fight that is won, the player gains experience points, which allow him/her to level up. A higher level grants access to more objects such as weapons, magic spells, etc. It also improves the ranking of the player.

The game also proposes a cooperative mode where teams composed of leeks belonging to different players can fight against other teams. Such cooperative mode is another technique used in games to foster people to play regularly.

Fig. 7. Two leeks are fighting in a war, playing alternatively one after the other. The garden is divided into cells between which the leeks are moving and has some impassable obstacles.

4.2. *CodinGame*

CodinGame (`https://www.codingame.com`) proposes coding challenges to solve. Basically, the user is asked to code agents that have to interact in a given environment in order to achieve a given goal. In the example shown in Fig. 8, the user has to code the spaceship so that to destroy all the targets on the ground. The spaceship is moving from

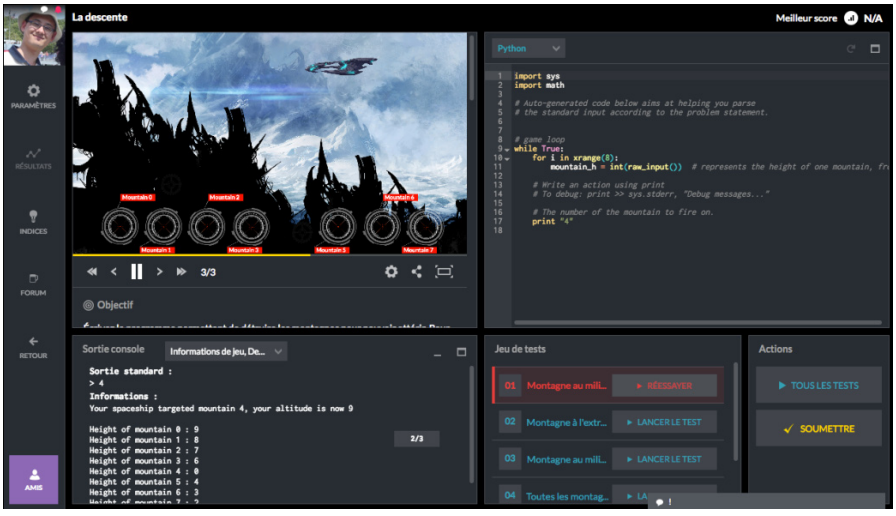


Fig. 8. In this mission, the agent is a spaceship that must destroy targets on the ground. These targets have different heights so that the spaceship must destroy them in the right order if it does not want to enter in collision with any of them.

left to right then back from right to left while descending when it reaches the rightmost part of the world. The difficulty is that targets have different heights and the spaceship must avoid entering in a collision with any of them.

Several test sets are provided and to succeed, the code of the player must pass them all (some being public and others being hidden). The execution of the code also results in an animation, which makes it possible for the learner to directly visualise the behaviour he/she wrote. This latter point helps to increase the motivation and engagement of the learner, better than a simple textual test execution report. Finally, the platform offers the possibility to code the agents with a lot of different programming languages.

Several challenges/missions are available for the user to tackle. They are organised into categories such as tutoring, advanced, etc. The users can follow their progress and that motivates to solve more challenges.

4.3. *Code Hunt*

CodeHunt (`https://www.codehunt.com`) is developed by Microsoft Research and propose coding challenges driven by tests. The goal of the game is to guess what the code must do, and then fix it until it passes all the tests. Fig. 9 shows the main view of Code Hunt where the code is on the left and the results of test execution are shown on the right. Pressing the "*Capture code*" button triggers the execution of a bunch of tests, shown on the right.

When the user solves a yet unsolved puzzle, he/she gains points so that to increase his/her position in the ranking. Moreover, some puzzles are initially locked and will only unlock when the user has solved enough puzzles. This possibility to unlock puzzles is an incentive for the user that motivates him/her to progress.

Fig. 9. The main window of Code Hunt is split into two parts: the left part shows the code being run and the right part shows the result of the execution of the public tests.

4.4. *Code Fights*

Code Fights (`https://codefights.com`) is a platform where users can participate in fights against bots or other real players. A fight typically consists in three puzzles to solve. One fight typically consists of three challenges. As shown in Fig. 10, the main window shows the statement of the challenge on the right and the code with the results of public tests on the right.

The learner can train him/herself by fighting bots or can defy any other player for a fight. There are two kinds of bots: simple ones and others developed by companies. If you manage to beat a company bot, you got a chance to be contacted by the company to possibly get a job. Another interesting feature proposed by the platform is tourney, where you compete with several other real players on the same challenges. To win the tourney you have to solve all the challenges as fast as possible.

4.5. *Discussion and Comparison*

The presented online platforms have some common points and differences. First of all, they are all in the *"learn algorithmic thinking"* category described in the previous section. Then, the level of gamification of the different platforms is quite different. A platform that has been completely designed with games in mind is one that completely agrees with the definition of gamification, as detailed in the previous section. The first one, namely *Leek Wars*, is a full online game with cooperative aspects where the ultimate goal is for the player to level up to become the best leek, the one with the most

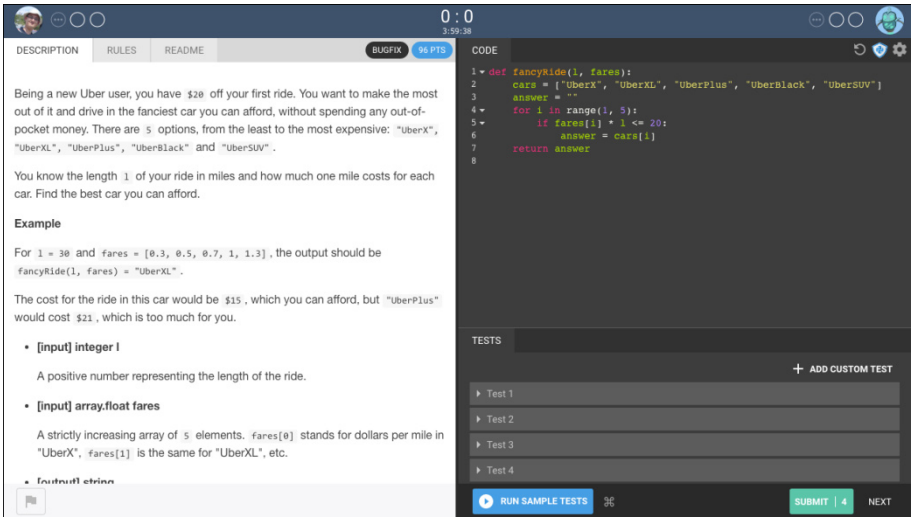


Fig. 10. The main window of *Code Fights* is split into two parts: the left part shows the statement of the challenge and the right part shows the code and the result of the execution of the public tests.

intelligent behaviours. The three other platforms do contain some game element, but they are not designed completely as full games. Table 1 summarises the game elements that appear in the presented platforms.

All the platforms presented in this paper can also be classified according to the adapted and extended schema we proposed to classify educational programming games. Table 2 shows how that classification can be done. All the games presented in this paper

Table 1

The four presented online game platforms to learn different skills of coding training and contains different game elements

Platform	Trained skills	Game elements
Leek Wars	Coding agents' AI Inventing algorithms	Points and rankings Challenges against bots and other players Cooperative mode
CodinGame	Understanding specifications Coding objects' behaviours	Points, trophies, badges, rankings Levels and achievements Matches against other players
Code Hunt	Fixing bugs Code recovery Understanding tests sets results	Points and rankings Levels with challenges to unlock
Code Fights	Understanding specifications Coding algorithms Fixing bugs Code recovery	Points, levels and rankings Matches against other players or bots

Table2

Classification of the platforms

Game	Modality	Interaction style	Environment	Learning approach
Codecademy	Visual	Keyboard/Mouse	Social Presence 2D Online	Single Player
Code Fights	Visual	Keyboard/Mouse	Social Presence 2D Online	Mixed Player
Code Hunt	Visual	Keyboard/Mouse	2D Online	Single Player
CodinGame	Visual	Keyboard/Mouse	Social Presence 2D Online	Single Player
Initial Conditions	Visual	Keyboard/Mouse	2D Online	Single Player
Leek Wars	Visual	Keyboard/Mouse	2D Online	Mixed Player Collaboration
LightBot	Visual	Keyboard/Mouse	2D Online	Single Player

are visual ones with which the players interact with the keyboard and mouse. Concerning the environment, all the games are of course online ones. They are also all 2D games and some of them will require social presence, in particular for those where duels are available. Finally, the learning approaches of those selected games can be either single or mixed player. One of the game, namely *Leek Wars*, supports collaboration through the creation of teams of Leeks.

5. Recommendations and Conclusions

To conclude this paper, the learning of programming is accessible not only to higher education students, but it is accessible to others as well. A lot of online platforms have been developed and made accessible on the Internet. In addition to the classical learning platform or MOOCs, a broad range of game-based online platforms have emerged. The main advantage of such platforms is that they foster their users to learn and keep progressing, making programming fun.

Tasks and challenges proposed by these games-based platforms could inspire tasks proposed in the IOI for two main reasons. First, contestants to the IOI are probably using those platforms so that they may be expecting similar tasks at the IOI. Second, the motivational aspect of those tasks could be transferred to the IOI contest, also the making it more motivational and interesting for its contestants.

We referred and analysed papers on gamification: contests; online platforms; online, educational, programming and serious games. These are our main findings from these papers for educational or serious programming games to be successful, motivating to learn programming and so on:

a. Feedback and assessment is very important in any educational games. Assessment may be automated or not, but it raises efforts of students. Feedback provides information about the learning process.
b. Games must have aesthetics – game must be fun to engage and improve intrinsic motivation to learn computer science or programming.
c. Collaborative educational games raise the participation of students.
d. Participation is necessary for motivation to learn as well as for reducing high dropout rates of programming students.
e. Multiplayer collaborative games are more motivating and engaging than single-player appropriate games.
f. Guidance in educational game helps players not to feel confused while they are playing.
g. Aesthetics make any educational game "*fun*". It involves players into the game.
h. Avoid negative consequences, because otherwise players may perform low.
i. Involve music in game design and the players will probably play the game longer.
j. The level of challenge neither ought to be not too low nor too high (it keeps the interest of players).
k. Games with multiple modalities, adaptive or personalized, based on real world sensory data may be more successful.

l. Contests and duels in programming games may be used to improve the programming skills.

We proposed taxonomy of the programming educational games and classified several online programming games / platforms. All of these games are visual; interactive with keyboard/mouse, 2D and online. The majority of them are designed for a single player, however, two of them are designed for mixed players (game may be played as single player or multiplayer) and one of them involves collaboration aspect. As we already mentioned, multiplayer and collaborating games are more motivating. 3D-games sometimes have more advantages than 2D, but it depends more on a combination of many other mentioned or not mentioned factors.

Our findings may be useful in creating educational programming games on your own, or recreating non-educational games and gamifying them to teach programming, it is especially useful for novices in programming. This research does not propose to use all these factors, because some of them may be successful in one kind of games, but may not be successful for other type of games. However, it is recommended to use at least some of these factors due to our research analyse.

References

ACM/IEEE Computer Society (2013). *Computer Science Curricula 2013.Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM Press, New York, N. Y. USA. Available on the internet: `https://www.acm.org/education/CS2013-final-report.pdf`

Alvarez, J., Michaud, L. (2008). Serious games: advergaming, edugaming, training, and more, *IDATE, BP 4167, 34092 Montpellier Cedex 5*. France, 3–6, 11–12.

Arena, D.A., Schwartz, D.L. (2013). Experience and explanation: using videogames to prepare students for formal instruction in statistics. *Journal of Science Education and Technology*, 1–11.

Azmi, S., Iahad,N.A., Ahmad, N. (2015). Gamification in online collaborative learning for programming courses: a literature review. *ARPN Journal of Engineering and Applied Sciences*, 10(23), 1–3.

Barata, G., Gama, S., Jorge, J., Goncalves, D. (2013). Engaging engineering students with gamification. In: *Proceedings of the 5th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES 2013)*. 1–8.

Becker, K. (2008). Video game pedagogy: good games = good pedagogy. In: T.C. Miller (Ed.), *Dames: Purpose and Potential in Education*. Springer, NewYork, NY, 73–125.

Bishop, J., Horspool, R., Xie, T., Tillmann, N., de Halleux, J. (2015). Code hunt: experience with coding contests at scale. In: *Proceedings of the 37th International Conference on Software Engineering (ICSE 2015)*. 398–407.

Combéfis, S., Bibal, A., Van Roy, P. (2014). Recasting a traditional course into a MOOC by means of a SPOC. In: *Proceedings of the European MOOCs Stakeholders Summit 2014 (EMOOCs 2014)*. 205–208.

Combéfis, S., Saint-Marcq, D. (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.

Cuba-Ricardo, G., Serrano-Rodríguez, M.T.,Leyva-Figueredo, P.A., Medoza-Tauler, L.L. (2015). Methodology for characterization of cognitive activities when solving programming problems of an algorithmic nature. *Olympiads in Informatics*, 9, 27–30.

Deterding, S., Dixon, D., Khaled, R., Nacke, L. (2011). From game design elements to gamefulness: Defining "gamification". MindTrek, 2.

Dickey, M.D. (2005). Engaging by design: how engagement strategies in popular computer and video games can inform instructional design. *Educational Technology Research and Development*, 53(2), 67–83.

O'Donovan, S., Gain, J., Marais, P. (2013). A case study in the gamification of a university-level games development course. In: *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference (SAICSIT 2013)*. 242–251.

Esper, S., Foster, S., Griswold, W. (2013). CodeSpells: embodying the metaphor of wizardry for programming. In: *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2013)*. 249–254.

Gee, J.P. (2003). What video games have to teach us about learning and literacy. *Computers in Entertainment* (CIE), 1(1), 20.

Gee, J.P. (2005). Good video games and good learning. *Phi Kappa Phi Forum,* 85(2), 33–37.

Hunicke, R., Leblanc, M. Zubek, R. (2004). MDA: A formal approach to game design and game research. In *Proc. AAAI workshop on Challenges in Game*. AAAI Press, 1–5.

Ibáñez, M.-B., Di-Serio, Á., Delgado-Kloos, C. (2014). Gamification for engaging computer science students in learning activities: a case study. *IEEE Transactions on Learning Technologies*, 7(3), 291–301.

Laamarti, F., Eid, M., El Saddik, A. (2014). An overview of serious games. *International Journal of Computer Games Technology*. Article ID 358152, vol. 2014, 1–15. Available on the internet: `http://dx.doi.org/10.1155/2014/358152`

Lee, M., Ko, A., Kwan, I. (2013). In-game assessments increase novice programmers' engagement and level completion speed. In: *Proceedings of the 9th Annual International ACM Conference on International Computing Education Research (ICER 2013)*. 153–160.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., Resnick, M. (2004). Scratch: a sneak review. In: *Proceedings of the 5th International Conference on Informatics in Schools: Situation, Evolution and Perspectives (ISSEP 2011)*. 155–164.

Mora, A., Riera, D., González, C, Arnedo-Moreno, J. (2015). A literature review of gamification design frameworks. In: *7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games)*, *At Skövde*. IEEE, 1–8.

Morrison, B., DiSalvo, B. (2014). Khan Academy Gamifies Computer Science. In: *Proceedings of the 45th Technical Symposium on Computer Science Education (SIGCSE 2014)*. 39–44.

Muratet, M., Torguet, P., Viallet, F., Jessel, J.-P. (2010). Experimental feedback on Prog&Play: a serious game for programming practice. In: L. Kjelldahl and G. Baronosk (Eds.), *EUROGRAPHICS.* 1–8.

Nah, F., Zeng, Q., Telaprolu, V., Ayyappa, A., Eschenbrenner, B. (2014). Gamification of education: a review of literature. In: *Proceedings of the First International Conference on HCI in Business (HCIB 2014)*. 401–409.

Prensky, M. (2003). Digital game-based learning. *Computers in Entertainment* (CIE), 1(1), 21.

Ravaja, N., Saari, T., Laarni, J., Kallinen, K., Salminen, M., Holopainen, J., Järvinen, A. (2005). The psychophysiology of video gaming: phasic emotional responses to game events. In: *Proceedings of DiGRA 2005 Conference: Changing Views – Worlds in Play*. Vancouver, Canada, 3, 1–13.

Sawyer, B., Rejeski, D. (2002). *Serious Games: Improving Public Policy through Game-Based Learning and Simulation*. Woodrow Wilson International Center for Scholars, Washington, DC, USA.

Schell, J. (2008). *The art of Game Design: A Book of Lenses*. Carnegie Mellon University and Schell Games, Pittsburgh, Pennsylvania, USA.

Shute, V.J. (2011). Stealth assessment in computer-based games to support learning. In: S. Tobias and D. Fletcher (Eds.).*Computer Games and Instruction*. Information Age: Charlotte N. C., 503–524.

Squire, K. (2008). Open-ended video games: a model for developing learning for the interactive age. In: K. Salen (Ed.), *The John D. and Catherine T. MacArthur Foundation Series on Digital Media and Learning*. MA: The MIT Press, Cambridge, 167–198.

Surakka, S., Malmi, L. (2004). Cognitive skills of experienced software developer: Delphi study. In: Korhonen A., Malmi, L. (Eds), *Kolin Kolistelut–Koli Calling 2004*. In: *Proceedings of the Fourth Finnish/Baltic Sea Conference on Computer Science Education*. Finland, Koli, 37–46.

Swacha, J., Baszuro, P. (2013). Gamification-based e-learning platform for computer programming education. In: *Proceedings of the X World Conference on Computers in Education (WCCE 2015)*. 122–130.

Tillmann, N., de Halleux, J., Xie, T., Bishop, J. (2014). Code hunt: gamifying teaching and learning of computer science at scale. In: *Proceedings of the First ACM Conference on Learning @ Scale Conference (L@S 2014)*. 221–222.

Tillmann, N., De Halleux, J., Xie, T., SumitGulwani, Bishop, J. (2013). Teaching and learning programming and software engineering via interactive gaming. In: *35th International Conference on Software Engineering (ICSE)*. IEEE, San Francisco, CA, 1117–1126.

Vihavainen, A., Airaksinen, J., Watson, Ch. (2014). A systematic review of approaches for teaching introductory programming and their influence on success. In: *ICER '14Proceedings of the tenth annual conference on International computing education research*. ACM, New York, NY, 19–26.

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), where he is mainly teaching informatics. He also got an advanced master in pedagogy in higher education in June 2014. He founded the Belgian Olympiad in Informatics (be-OI) with Damien Leroy in 2010. In 2012, he introduced the Bebras contest in Belgium and at the same time he founded the CSITEd non-profit organisation that aims at promoting computer science in secondary schools.

G. Beresnevičius is a doctoral student of informatics engineering at the Institute of Mathematics and Informatics of Vilnius University. He obtained his master degree in Mathematics and Informatics Didactics with Cum Laude diploma in Vilnius University. His research focuses on teaching informatics, especially programming through games, as well as the theories of serious games and gamification. He participated in several international conferences and presented his research. He likes programming websites using HTML, CSS, JavaScript, PHP and MySQL languages.

V. Dagienė is head of the Informatics Methodology Department at the Institute of Mathematics and Informatics of Vilnius University. Her current research is in Computer Science Education, focusing on cognitive aspects of algorithmic thinking as well as computational thinking. She has published over 200 scientific papers and methodological works, has written more than 50 textbooks in the field of informatics and information technology for primary and secondary education. She works with various expert groups all over the world. In 2004, she established the Bebras Challenge on Informatics and Computational Thinking, which is successfully implemented in more than 50 countries. She has participated in several EU-funded R&D projects, as well as in a number of national research studies connected with technology and education.

Olympiads in Informatics, 2016, Vol. 10, 61–72

DOI: 10.15388/ioi.2016.04

The Place of the Dynamic Programming Concept in the Progression of Contestants' Thinking

Ágnes ERDŐSNÉ NÉMETH[1,2], László ZSAKÓ[3]

[1]*Batthyány High School, Nagykanizsa, Hungary*
[2]*Doctoral School, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary*
[3]*Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary*
e-mail: erdosne@blg.hu, zsako@caesar.elte.hu

Abstract. The special problem-solving strategies have been receiving a lot of attention lately, whether it is teaching computational thinking for all or computer science for competitors. A didactically interesting question is how problem solving can be developed in children's minds, what steps and tasks lead through from understanding the idea to its professional usage. In this paper we present and explain how and in what forms the given problem-solving strategies, especially the dynamic programming concept, appear in children's informatics studies: from CS unplugged activities through Bebras tasks and national CS competitions to efficient coding at the IOI.

Keywords: dynamic programming concept, teaching informatics in primary and secondary schools, preparing for contests.

1. Overview

There are a lot of problem-solving strategies which every contestant has to be familiar with. In this paper, we want to focus on one of the most important ones, specifically dynamic programming, because "[Until] you understand dynamic programming, it seems like magic" (Skiena, 2008).

The dynamic programming name came from Bellmann in the 1950's (Dreyfus, 2002). He wanted to find a name for the multistage decision making process, in which the general case is very hard to solve thus one must use the divide & conquer concept. It leads back to special cases, when the solution is expressed more easily – not in terms of the unknown function, but in terms of an action or decision.

If we want to speak about dynamic programming, we also have to examine other corresponding concepts, like recursion, memoization and divide & conquer.

The concept of recursion consists of a recursive function and a recursive implementation; it is a top-down approach. Usually the runtime of the algorithm is exponential and this technique usually fails already in a small sample size. The memoization is a top-down approach as well, it is more effective than recursion. The core idea is the

same: recursive function and recursive implementation, but storing the calculated results, thus without calling the recursive step again when the actual state previously appeared.

When we are experienced enough to implement a recursive approach correctly, there is a special technique for dramatically reducing the runtimes of certain algorithms from exponential to polynomial or from factorial to exponential, at the expense of higher memory usage. This is done by solving sub-problems and storing the results using a dynamic programming concept. The concept is about a recursive function, but the implementation is to build the values from bottom up without calling the recursive step repeatedly.

The simplest type of dynamic programming is when an array is filled – namely a table – cell-by-cell in a predetermined order, implementing the recursive function from bottom up. We will call it a *basic DP*.

Some people use the term dynamic programming only for those recursive problems, which involve optimization, but the technique of completing a table to solve any other type of problem is almost the same when it comes to implementing the solution. We will call it *classic DP*.

The dynamic programming as problem-solving strategy is the implementation of the following five steps (Horváth, 2004):

1. Analysing the (optimal) solution's structure.
2. Dividing it into subproblems and components:
   a. Dependence of components must be acyclic.
   b. Every subproblem (optimal) solution should be a (recursive) expression of components' (optimal) solution.
3. Expressing each subproblem's (optimal) solution as a (recursive) function of components' (optimal) solution.

These three steps are the planning of recursive algorithm. DP comes from the next two steps:

4. Calculating subproblems' (optimal) solutions from bottom up (completing a table).
5. Calculating an (optimal) solution from the previously calculated and stored information.

If you are familiar with the basic DP techniques then you could continue with some advanced techniques (Steinhardt, 2008).

The first advanced type is to keep track of all possible transition states. In this case DP means filling out a table row-by-row, with the values depending on the row number and values in the previous row.

Second advanced type is the dynamic greedy type.

The third advanced type is a steady state convergence, only for more experienced students. In this case the recursive equation must be repeatedly applied, then values will converge exponentially to the correct values.

All of the DP types have a place in different stages and in different ages of contestants' studies.

2. Place of Dynamic Programming in Algorithms Textbooks

There are a lot of textbooks about algorithms. They discuss all algorithms sequentially and directly as they are written for university students. The structure of these books and the place of dynamic programming in them is different.

In the textbook of Skiena (2008) the DP takes place after data structures, sorting and selecting algorithms, graphs, combinatorics, backtrack and parallel programming. In the chapter of dynamic programming, he compares algorithms using cache vs. computation at first. After this, he discusses the problem of string matching, longest increasing subsequence and partition problem. The limitation of DP is presented via travelling salesperson's problem and there are some words about correctness and efficiency of DP. Just after the DP concept comes the recursion and memoization, the concepts of the top-down structures.

In the textbook of Kleinberg and Tardos (2006) there are basic algorithms, graphs and greedy algorithms first. To process sets which can be divided into independent parts they recommend divide & conquer concept. Just after this method, they speak about DP: the recursion, the memoization and iteration over sub-problems are the parts of this chapter.

In the textbook of Dasgupta *et al.* (2006), dynamic programming is presented after arithmetics, primes, cryptography, hashing, divide & conquer, graphs and greedy algorithms via examples: travelling salesman's problem, longest increasing subsequence and knapsack.

In the textbook of Sedgewick and Wayne (2011) there is no chapter dedicated to DP, but the concept appears in some points of the book.

In the textbook of Gupta (2009) DP comes after introduction to algorithms, divide & conquer method and greedy method.

Bebis (2007) has lot of chapters about sorting and searching methods, DP comes after that.

There are not two textbooks, in which the place of DP is the same in the sequence of algorithms. Sometimes it comes before graphs (Gupta, 2009), often it comes after graphs. Occasionally it is before recursion (Kleinberg and Tardos, 2006), but in the most common order its place is after recursion and memoization. Sometimes it is placed before the greedy algorithms, sometimes after. Textbooks do not usually stress the differences between dynamic programming and greedy approach, nor warn they about the danger of accidentally using one instead of the other.

These textbooks are almost useless for primary and secondary school pupils because of their advanced mathematics and informatics contents. Some parts of them may be useful in upper secondary, just for contestants preparing for IOI.

3. New Way for Contestants Learning DP in Upper Secondary School

Last year there was a paper in IOIJournal about the critical analysis of textbooks and new way of teaching DP (Forišek, 2015) for upper secondary school students preparing for national olympiads and IOI. He started with the Fibonacci sequence – something is

well-known by children from math studies – implementing it with a recursive function and making this function more efficient with memoization. He compares iterative and dynamic solution, then introduces DP bottom-up. He demonstrates that the exponential solution longest common subsequence problem with a top-down approach turns polynomial ($o(n^2)$) with a bottom-up approach. It is a very good structure if students want to prepare for olympiads all at once.

There is a book about competitive programming, which was written for contestants preparing for ACM ICPC and IOI (Halim and Halim, 2014). It is not a real textbook, it teaches the effective type detection of tasks and the correct, error-free coding, not the concepts behind the algorithms. According to it contestants' main goal 'should be to honing [their] ability to recognize a problem as DP, finding the recursive formula for such a problem, coding the problem, and doing all of this quickly'.

After data structures and problem-solving strategies like searching – iterative & recursive – divide & conquer concept and greedy algorithms comes dynamic programming through an example: UVA 11450 wedding shopping. The chapter begins with the repetition of recursion, backtrack, optimization and counting problems. It shows, that this task's solution: greedy method failed with wrong answer (WA), divide & conquer method failed with WA (because of non-independent parts), complete search failed with time limit exceeded (TLE). The dynamic programming method with top-down (memoization) and with bottom-up approach is working. After this very detailed analysis he gives six more example with analysis and many task recommendation to become familiar with this method.

We think these methods work effectively for students in upper-secondary-school-age (grade 11–13) preparing purposefully for national and international olympiads.

4. Teaching Dynamic Programming in Primary and Secondary School

If the students know structured programming concept (Floyd), they are familiar with the top-down concept too, because of stepwise refinement; and they are familiar with the concept of bottom-up, because concrete objects and functions. So the pupils knows top-down and bottom-up paradigms as soon as they begin to implement a computer program.

We think teaching dynamic programming ideally begins in upper primary school in mathematics and informatics lessons. Implementation of DP on computers is possible when children are familiar with the basic data structures (integer, boolean, array), basic algorithms (sequence, iteration, selection, searching, procedures and functions) and the concept of recursion.

4.1. *CSUnlugged*

There is not any activity yet that covers dynamic programming in the CSUnplugged repository, but there is an intention to make a good one from the change-making problem.

4.2. *BEBRAS*

In the Bebras competitions there are tasks about DP every year. There are other tasks, in which the recursion is the best solution. There are many in which the greedy gives wrong answer and the dynamic programming concept must be used for the right solution.

On Bebras competitions the DP problems appear in a wide range of age groups and difficulties also. It proves that the concept of DP comes much earlier than at the end of secondary school and it is understandable for everybody, not only for contestants: it is part of computational thinking skills.

The easiest task of DP is from 2013 for grade 5–8: the Pairs without Crossing (Kreuzungsfreie Pärchen). The brute force algorithm is working, but takes long time to try every possible connection pairs, the dynamic concept make it easy. In 2011 the problem Earn coins (Münzen verdienen) is a special case of the classic knapsack problem. This one was one of the hard problems for the grade 5–6, middle for grade 7–8 and easy for grade 9–10. In 2014 another classic DP problem appeared, the Expensive Bridges (Teure Brücken) which one was hard problem for the grade 7–8, middle for grade 9–10 and easy for grade 11–13.

There are more problems connected to DP for secondary-school-age students: the Jumping Puddles (Pützenspringen), the game ROOK, in 2010 the task Pinecone (Tannenzapfen), in 2014 the Best translation (Beste übersetzung) and in 2015 the Fireworks2 (das Feuerwerk2).

4.3. *Tasks for Contestants of Upper-Primary-School-Age*

In the grade 5–6 we could start with LOGO programming. It provides a strong foundation in the basic programming structures, like sequence, iteration and selection. Through the drawings, it also visualizes the concept of recursion well. They can imagine and implement binary tree (Fig. 1), the Sierpinski-triangle (Fig. 2), the Koch-curve (Fig. 3).

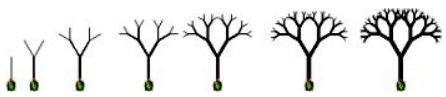


Fig. 1. Binary tree.

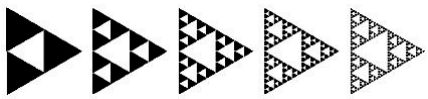


Fig. 2. Sierpinski-triangle.

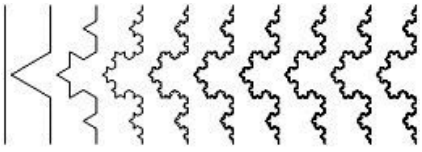


Fig. 3. Koch-curve.

In Hungary and many other countries, they meet table filling and the thought of recursion in mathematics. They calculate total number of possible paths in a grid of characters, from the top-left corner to the right-bottom corner to spell a given word (Fig. 4). They calculate total number of possible paths, even if there are empty squares in the grid (Fig. 5).

D	Y	N	A	M	I	C	P
Y	N	A	M	I	C	P	R
N	A	M	I	C	P	R	O
A	M	I	C	P	R	O	G
M	I	C	P	R	O	G	R
I	C	P	R	O	G	R	A
C	P	R	O	G	R	A	M

1	1	1	1	1	1	1	1
1	2	3	4	5	6	7	8
1	3	6	+				

Fig. 4. Spelling a given word.

D	Y	N	A	M	I	C	P
Y	N	A	M	I	C	P	R
N	A		I	C	P	R	O
A	M	I	C	P	R	O	G
M	I	C	P	R	O	G	R
I	C	P	R	O	G	R	A
C	P	R	O	G	R	A	M

1	1	1	1	1	1	1	1
1	2	3	4	5	6	7	8
1	3	0	+				
1	4	4					

Fig. 5. Spelling a given word with a pit.

Pupils of grade 7–8 meet simple recursive sequences and functions, like the Fibonacci sequence. In mathematics lessons, they meet combinatorial problems (permutation, variation and combination without repetition) without naming them. They come across problems, like longest/shortest path in a directed/undirected graphs and coloring problems on very small graphs, coin changing problem for small numbers, shaking hands/sitting in a row/around a table, they calculate extreme values in Diophantine problems. These problems are solved with table filling or with recursive expressions. The formulas, like n! or $\binom{n}{k}$ are not formulated.

They can solve problems like these:

How many different, 10 cm high towers can build from 2 cm high blue, 2 cm high yellow and 1 cm high red building blocks?

Fig. 6. $a_1 = 1$, $a_2 = 3$, $a_3 = 5$, $a_4 = 11$ – Counting with drawing systematically.

If they draw it systematically (seeing the blocks from bottom), they can guess the recursive expression $a_n = a_{n-1} + 2*a_{n-2}$ *and they check it empirically. The solutions are: 1, 3, 5, 11, 21, 43, 85, 171, 341, 683.*

*How many different covering exist on a 2*8 table with 1*2 size dominos?*

Fig. 7. $a_1 = 1$, $a_2 = 2$, $a_3 = 3$, $a_4 = 5$ – Counting with drawing systematically.

If they draw it systematically (seeing the blocks from the right), they can guess the recursive expression $a_n = a_{n-1} + a_{n-2}$ *and they check it empirically. The solutions are:1, 2, 3, 5, 8, 13, 21, 34. The solution can be built for any given N with a table filling.*

In primary schools the children meet problems with large numbers also, like 1000 points, 2016 numbered cards, 10 000 people. In such cases, the obvious idea is – instead of the original task – to examine a simplest problem. What happens if the number of points is 2, 3, 4, ..., the number of cards is 5, 6, 7,…, the number of people is 2, 3, 4,…? If any regularity is noticed, they try to verify it empirically and apply it on the original problem with large numbers. On primary-school-level, formulating and using the hypotheses is enough to solve such problems.

On the other hand, these problems can be implemented on the computer, as a recursive expression or with table filling for larger numbers. The next classical problems are the first appearance of dynamic programming approach for them:

Robot – *A robot starts from the top left corner (1,1) of a MxN grid. At each step the robot can make one of the two choices: move one cell to the right or move one cell down. How many possible paths are there for the robot to reach the right-bottom corner of the grid?*

The robot problem can occur in many variations at grade 7 and 8:

- *The question is the same, but there are cells in a grid, on which the robot can't step on (traps).*
- *The question is the same, but there are cells in a grid, on which the robot have to step on (mandatory fields).*
- *In every cell there are given number of pearls and the question is, what is the maximum number of pearls the robot can collect on its way.*
- *Mix of traps and pearls.*

Staircase – *You are standing in front of a staircase, which has N stairs. Your goal is to reach the top. If you are standing on the* i^{th} *step, you can hop to* $(i+1)^{st}$ *or* $(i+2)^{nd}$ *or* $(i+3)^{rd}$ *step. Given N, calculate the count of total possible paths for you to reach* N^{th} *stair!*

Coin Change – *You want to make change for given N cents and you have infinite supply of each of* S_1, S_2, .. , S_m *valued coins. How many ways can you make the change?*

Subset sum – *Detect, if any of the subset from a given set of N non-negative integers sums up to a given value S!*

Dice Throw Problem – *Given N dice, each with m faces, numbered from 1 to m. Find the number of ways to get sum X! (X is the summation of values shown by the dices.)*

Flooring – *How many different coverings exist on a 1*N floor with 1*1 and 1*2 parquet pieces? How many different covering exist on 2*N floor with 1*2 parquet pieces?*

Towers *– How many different, N meters high towers can be built from 2 meters high blue, 2 meters high yellow and 1 meter high red blocks?*

The previous dynamic programming problems should be solved at the primary-school-level as the analogous math problems: children formulate and use the hypothetical recursive expressions and implement them with table filling, without extensive argumentation. Mostly they cannot calculate directly the answer, but they can give a recursive formula and the direction of filling the table, and this way they can solve the problem. They can solve basic DP problems, without optimization.

On informatics contest the 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 in various wording. Choosing, selecting, counting, searching, summarizing, selecting maximum/minimum, sorting, separating into two groups, prime testing are these basic algorithms. Stages of solving tasks are understanding the problem, choosing the right data structure, selecting right algorithm, implementing and testing it. In addition to the conservative tasks there are ad-hoc problems where children can apply basic algorithms creatively. The basic DP problems appear in the regional and national rounds of competitions.

4.4. *Tasks in Lower-Secondary-School-Age*

In Hungary and many other countries the children continue to learn combinatorial problems in grades 9–10 in mathematics lessons, they group and formalise these problems. During these years, they also meet the idea of mathematical induction, so they can prove the previously discovered recursive formulae. They learn about sequences and sometimes they give the explicit formulae for recursive expressions. They know the formula of n! and $\binom{n}{k}$, they also learn Pascal-triangle. But they do not necessarily know the relationship between the binomial coeffitients and the Pascal-triangle.

They can discover and solve more complex recursive expressions, sometimes these functions call each other, like:

*How many different covering exist on a 2*N table with 1*2 and 1*1 size dominos?*

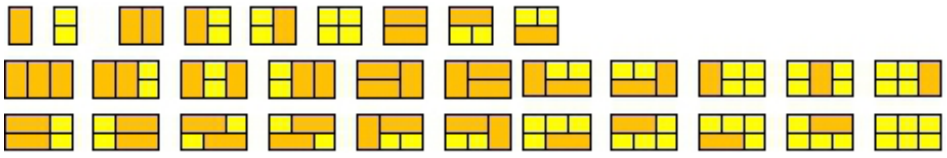


Fig. 8. $a_1 = 2$, $a_2 = 7$, $a_3 = 22$, $a_4 = 71$ – Counting with drawing systematically.

Drawing it systematically is not enough to formulate the recursive expression. One must analyze possible cases. The result comes with two expressions in simultaneous recursion: $d_n = a_{n-1} + f_{n-1}$, $f_n = a_{n-1} + d_{n-1}$, $a_n = 2*a_{n-1} + a_{n-2} + f_{n-1} + d_{n-1}$. *After simplification:* $a_n = 3*a_{n-1} + a_{n-2} - a_{n-3}$.

In this age they learn the basics of graph theory in mathematics, they learn types of graphs (trees, binary trees, relational matrix), storage of graphs (vertex matrix, edge matrix, edge list) and algorithms of graphs (breadth first traversal, depth first traversal, relations) in informatics. They meet recursion again, divide and conquer, backtrack, and greedy algorithms too, on basic level.

They learn all classic dynamic programming problems, as follows:

Partition problem – *Divide the set of numbers into two groups, where sum of each group is same!*

Longest Increasing Subsequence – *Find the length of the longest subsequence of a given sequence, such that all the elements are sorted in increasing order.*

Knapsack Problem – *A thief robbing a store can carry a maximal weight of W in his knapsack. There are N items and* i^{th} *item weighs* w_i *and is worth* v_i *dollars. What items should the thief take?*

Contiguous subsequence with maximum value – *Find the contiguous array with the maximum sum in a given an array, containing both positive and negative integers!*

Minimal number of coins for change – *What is the minimal number of coins, to make change for a given amount T only coins of values* $v_1, v_2, \ldots, v_n$ *can be used?*

These examples can be solved by using recursion with optimization, the right order of filling the table must be given usually it is not evident. There are a number of variations of these problems, sometimes the difference is merely wording.

Next type of DP problems is game strategy in various formats, like:

Optimal Strategy – *Consider a row of N coins of values* $v_1, \ldots, v_N$, *where N is even. We play a game against an opponent by alternating turns. In each turn, a player removes either the first or last coin from the row and receives the value of the coin. Determine the maximum possible amount of money we can definitely win if we move first?*

Basic data structures may be supplemented with string and real. There are a number of dynamic programming problems with strings, too:

Longest Common Subsequence – *Find the longest common subsequence of two strings, where the elements are letters from the two strings and they should be in the same order!*

Longest Common Substring – *Find the longest common substring of two strings!*

Edit Distance – *Given two strings and a set of operations Change (C), insert (I) and delete (D). Find minimum number of operations required to transform one string into another!*

On informatics contest classic dynamic tasks can be in every round, so the contestants have to be ready to solve it.

4.5. *National Olympiads*

In grades 11–12 contestants prepare for national Olympiads. They know a lot of algorithms, in these years they learn to use advanced data structures, like set, priority queue, stack, and they implement previously learned algorithms with these data types. While in previous years they were asked for the existence of the solution or the number of steps in the solution, now they also have to decrypt the entire path traversal in dynamic programming problems.

Sometimes there are strict memory limits and there is no space for the whole table. In this case, only the values of those cells have to be stored in memory, which are essential to the calculation of the next row. During the decryption process the rebuilding of the table may be as hard and interesting to implement as the original problem.

They learn about combinatorial and geometrical problems.

Within the concept of divide and conquer, recursion or dynamic programming more complicated expressions should be optimized.

There are advanced DP problems: all possible transitions and dynamic greedy type.

Example of complicated problems:

> **Balanced Partition** – *There is a set of N integers each in the range 0 ... K. Partition these integers into two subsets such that you minimize* $|S1 - S2|$*, where* $S1$ *and* $S2$ *denote the sums of the elements in each of the two subsets!*

The dynamic programming concept among strings also leads complicated problems, like:

> **Shortest Palindrome** – *Form a shortest palindrome by appending characters at the start of the given string.*
>
> **Palindrome Min Cut** – *Find the minimum number of cuts required to separate the given string into a set of palindromes.*
>
> **Longest Palindromic Substring** – *Find the longest palindromic substring of a given string!*
>
> **Longest Palindromic Subsequence** – *Find the longest palindromic subsequence of a given string!*

The contestants in the upper secondary age want to be computer scientists or engineers, they do not only know the basic algorithms but they can cope with complex tasks.

4.6. *Regional and International Olympiads*

When you want to prepare for Regional or International Olympiads you have to know everything about dynamic programming which is included in the textbooks. You have to solve a huge number of tasks. On Olympiads the solution of tasks are some kind of creative mixture of known algorithms.

Some examples:

> **Interval-Scheduling Problem** *(Greedy and DP Approach) N k-tuples processes are given with start & end times. Select as many processes as possible such that*

(I) no two selected processes intersect and (II) at most one process is selected from each k-tuple!

intersec Complete all tasks given the deadline, so that no task overlap!

Box Stacking – *You are given a set of N types of rectangular 3-D boxes, where the i^th box has height h(i), width w(i) and depth d(i). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. You can rotate a box, any of the side can be its base. It is also allowable to use multiple instances of the same type of box.*

Counting Boolean Parenthesizations – *You are given a boolean expression consisting of a string of the symbols 'true', 'false', 'and', 'or' and 'xor'. Count the number of ways to parenthesize the expression such that it will evaluate to true. For example, there are 2 ways to parenthesize 'true and false xor true' such that it evaluates to true. The order is defined only by parentheses.*

There are many tasks for practice on the online preparing and contest sites, like usaco.org, codeforces.com, codechef.com, uva.onlinejudge.org, spoj.pl. If you have met each type of concept detailed above, the textbook of Halim is a good choice for preparing for IOI.

5. Conclusions

Some antecedents of the dynamic programming concept for example the concept of recursion, might come up in earlier mathematics and informatics studies. If you are aware of this, introducing DP as a new problem-solving strategy is much easier.

We think, if you want to teach the technique of DP you have to start from a simple recursion then through memoization and table filling you could end with a real DP for optimization problems. You could start the whole process in the primary school age and circularly, returning to it in higher and higher levels your students would be familiar with this hard concept.

In the upper primary school, the basic DP comes up: a recursive expression implemented with table filling. At the beginning of secondary school, the classic DP programs continue the sequence: recursive expressions with optimization and at implementation the right order of table filling need to be thought of. In upper secondary school, the children can be familiar used with advanced types of DP: all of the previous problems with the retrieval of the way, how the optimal solution is built up, dynamic greedy type and the type, when you have to keep track of all possible transition states. Preparing for Olympiads, the students need everything from textbooks and the combination of other types of approaches.

Finally, it would not be magic for the contestants, just a useful problem-solving strategy.

References

Bebras–International Contest on Informatics and Computer Fluency (2007–2015). http://bebras.org http://www.beaver-comp.org.uk/; http://informatik-biber.de/archiv/

Bebis, G. (2007). *CS477/677 Analysis of Algorithms*. http://www.cse.unr.edu/~bebis/CS477/

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.

Dreyfus, S. (2002). Richard Bellman on the birth of dynamic programming. *Operations Research, INFORM.* 50(1), 48–51.

Erdős, G. (2010). A rekurzív módszer. In: *Magas Szintű Matematikai Tehetséggondozás Konferencia, ZALAMAT.* 20–32.

Forišek, M. (2015). Towards a better way to teach dynamic programming. *Olympiads in Informatics,* 9, 45–55.

Gupta, N. (2009). *Introduction to Algorithms*. http://www.curriki.org/oer/Introduction-to-Algorithms/

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

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

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

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

Steinhardt, J. (2008). *Advanced Dynamic Programming Techniques*. https://activities.tjhsst.edu/sct/lectures/0708/dpadvanced.pdf

Á. Erdősné Németh teaches mathematics and informatics at Batthyány Lajos High School in Nagykanizsa. A lot of her students are in the final rounds of the national programming competitions, some on CEOI and IOI. She is a Ph.D. 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.

L. Zsakó Dr. is a professor at Department of Media & Educational Informatics, Faculty of Informatics, Eötvös Loránd University in Hungary. Since 1990 he has been involved in organizing of programming competitions in Hungary, including the CEOI. He has been a deputy leader for the Hungarian team at International Olympiads in Informatics since 1989. His research interest includes teaching algorithms and data structures; didactics of informatics; methodology of programming in education; teaching programming languages; talent management. He has authored more than 68 vocational and textbooks, some 200 technical papers and conference presentations.

Olympiads in Informatics, 2016, Vol. 10, 73–85

DOI: 10.15388/ioi.2016.05

Watch them Fight! Creativity Task Tournaments of the Swiss Olympiad in Informatics

Samuel GRÜTTER, Daniel GRAF, Benjamin SCHMID
Swiss Olympiad in Informatics
e-mail: {samuel,daniel,benjamin}@soi.ch

Abstract. As part of the qualification process for the Swiss Olympiad in Informatics, the contestants are each year confronted with one "Creativity Task". Unlike typical problems in programming competitions, creativity tasks usually do not have an optimal solution, and are often adaptations of popular board or computer games. After receiving all submissions, a tournament is organized, where the students can watch how their programs play interactively against each other, and points are awarded to the authors according to the tournament ranking.

We present and discuss this task format in general, as well as the specific creativity tasks of the past 10 years, accompanied by an online archive of the task descriptions, sample solutions and game servers.

Moreover, we describe our task selection process and criteria, the task creation process, and the experience gained and best practices established in the past years.

Finally, we present the many advantages of this task format, showing why we think it is a refreshing alternative to the common IOI-style tasks given in most national selection rounds.

Keywords: creativity tasks, interactive tasks, heuristics, programming competition, board games, artificial intelligence contest, tournaments, task visualization.

1. Introduction

During the first round of the Swiss Olympiad in Informatics (SOI), the students have two months to solve a set of tasks at home and submit them on the SOI website. There are practical and theoretical tasks, and one creativity task. This paper is about the creativity tasks.

Contrary to typical tasks in programming competitions, the creativity tasks are usually designed in such a way that there is no optimal solution, or that the optimal solutions are not efficient enough. The creativity tasks are often adaptations of popular board or computer games, and the game can be played with two or more players. The participants' task is to write a program able to play the game. We will refer to such programs as *bots*.

An interesting aspect of this task format is that the participants' bots can play and compete against each other.

At the start of each year's first round, the following material for the creativity task is published on the SOI website:

- The **task description**, describing the rules of the game and its parameters. Depending on the game, these could include, for instance, width and height of the playing field, or a map of the playing field, the number of players, the initial amount of money/food of each player, etc. The task description also specifies the *communication protocol* between the bots and the game server (see below), which defines how the bots have to communicate their actions and how they are informed about the other bots' actions. All communication is done via standard input/output.
- A **game server**, a program written by the task authors, which launches the bots, communicates with them according to the protocol and keeps track of the game state. It informs the bots about all the relevant changes and prints a log of each action and state change. The server also acts as a judge that can terminate bots that drop out of the game or take too long to answer. If the interaction protocol is easy to read, the game server can also allow for human players that type in their commands interactively. This way, the participants can play against their own bots by hand.
- Some **sample bots** in various programming languages. These bots follow all the rules of the protocol, but they just pick a random move in every step. The participants can base their solutions on these sample bots to quickly learn how to implement the protocol and input/output. Moreover, they can evaluate their bots by letting them play against the sample bots.
- A **visualization**, which reads the log produced by the game server, and displays a graphical animation of the game. For some of our tasks, the game server already provides a rudimentary text-based visualization of the game.

After the end of the first round of the SOI, the organizers let the submitted bots compete against each other in a large number of games covering many different configurations, to make sure that the obtained results are representative and random decisions average out well enough. The results of all these games are then aggregated into a final score for each participant.

Between the first and the second round of the SOI, all participants are invited to an event called "SOI-Day" consisting of task discussions, talks and presentations. At this event, a shortened version of the the creativity task tournament is presented as if it was a live tournament, and the participants can thrill while watching their bots compete against the others.

2. Ten Years of Creativity Tasks

In this section, we give a brief presentation of each creativity task of the past ten years. We omit details such as the communication protocols, and refer the reader to the online task archive (`http://creativity.soi.ch`) for the full task descriptions.

2.1. *Connect Five (2007)*

Task description On a 20×20 grid, two players (black and white) alternate in placing a stone of their color on an empty square. As soon as a player succeeds in placing 5 stones in a row (horizontally, vertically or diagonally), he wins.

Note that unlike the game sold under the name "Connect Four", the stones do not "fall down" in their column, but stay exactly where they were placed.

Discussion This variant of the game is also known under the name "Gomoku", and appeared at the International Computer Games Association's *Computer Olympiad* in 1989, 1990, 1991 and 1992.[1]

2.2. *Fight of the Ant Populations (2008)*

Task description On a grid with obstacles, multiple ant colonies are fighting against each other. Each player controls all ants of a colony. In every turn, the player sees the 7×7 neighborhood of each of his ants and can move them individually to an adjacent tile. Some tiles contain food, others contain ants or the hill of the enemy. Collecting food and carrying it back to the own hill allows the colony to grow. If an ant attacks another ant both die. If an ant reaches an adversarial hill, three random ants of that colony die. Which colony survives the longest?

Discussion This game allows for a huge variety in strategies. Some submissions assigned different jobs to the ants, for instance: explorers that go to unknown territory, workers that collect the closest food, guards that stay near the hill and warriors that try to reach the hill of the enemy.

2.3. *Grand Theft Cake (Portal Maze) (2009)*

Task description The players are searching for a big cake in an unknown, polygonal maze with walls all around them. They have a handheld portal device that allows them to teleport from A to B instantly. They can walk around with a speed of 1 meter per second. In 0.1 seconds, they can look in a direction to learn how far away the wall is. It takes 5 seconds to shoot a portal in a direction (either of type A or B). The cake is a circle of one meter diameter. The goal is to find the cake and return to the starting position as quickly as possible.

Discussion The submitted solutions made creative use of the portal. Some just used it to return to the start quickly, others repeatedly used it as a shortcut throughout the entire search for the cake.

[1] `http://www.game-ai-forum.org/icga-tournaments/game.php?id=30`

2.4. *Tanks (2010)*

Task description Each player controls a tank, which can move on a line and fire one missile per round. The tanks can use different weapons, which are specified by their range, their impact radius, and their damage points. The tanks have an unlimited number of missiles of each weapon. When a missile hits a position, the damage points of all tanks within the impact radius increases by the number of damage points of the weapon, and tanks reaching a predefined number of damage points die. Moving around costs fuel, and the fuel is limited. The last surviving tank wins.

Discussion Most submissions implemented some simple ad hoc heuristics. The winning solution was 285 lines long and in each round, it chose the weapon and target position which would cause the highest total damage to all opponents, supposing that they would not move. If there was a tie between several possibilities, the one with the biggest impact radius was chosen.

2.5. *Multisnake (2011)*

Task description *Multisnake* is a multi-player version of the popular computer game called *snake* (Fig. 1). It is played on a rectangular grid which contains some obstacles. Each player controls a snake, and in each turn, all snakes move simultaneously by one step. When a snake moves onto a field occupied by an obstacle or by a snake, it dies. The winner is the last surviving snake. Some tiles contain a black or white ball. When a snake eats a white ball, it grows by one, and when it eats a black ball, it shrinks by one. To enforce termination of the game, all snakes grow by one each T turns, where T is a small positive integer.

Discussion The winning solution was 1005 lines long, implemented a complex scoring function for possible states, and explored them recursively.

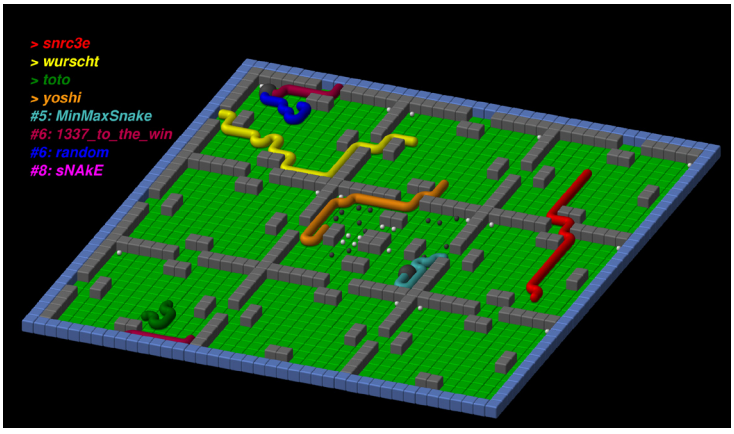


Fig. 1. Multisnake.

2.6. *Find the Anthill (2012)*

Task description In this second ant-themed task, the story was more peaceful: each ant just has to find the way back to its own ant hill (Fig. 2). The big difference is that this time each ant is controlled by a separate instance of the participant's submission. So the program controlling a single ant does not know where the other ants of its colony are, unless they are in its 7×7 field of view.
Like in nature, there is a distributed way of communication though: scents. The ants can place one out of 256 different scents on their current tile and they can sense the scents that other ants (of their own or of other colonies) put there. So if one ant sees the hill, it can leave hints for its colleagues.

Discussion Successful submissions made creative use of the scent hints to share knowledge between the ants. Some even tried to learn and imitate the marking patterns of the opponents to mess with it and cause confusion.

2.7. *Who wants to be a billionaire (2013)*

Task description The task is to write a bot which can interactively answer multiple choice questions with four possible answers, using Wikipedia articles as a knowledge base. For each question, the bot is allowed to consult up to 25 articles on the English Wikipedia. To do so, the bot has to provide a query string to the game server, and the game server will look up the article on Wikipedia (or in its cache), and feed a plain text version to the bot.

Discussion Most solutions were based on keyword search combined with some strategy to avoid frequent words which do not carry any meaning. The submitted bots would not have become billionaires, but the best was still twice as good as a random bot (which would have scored $\frac{1}{4}$ of the points).

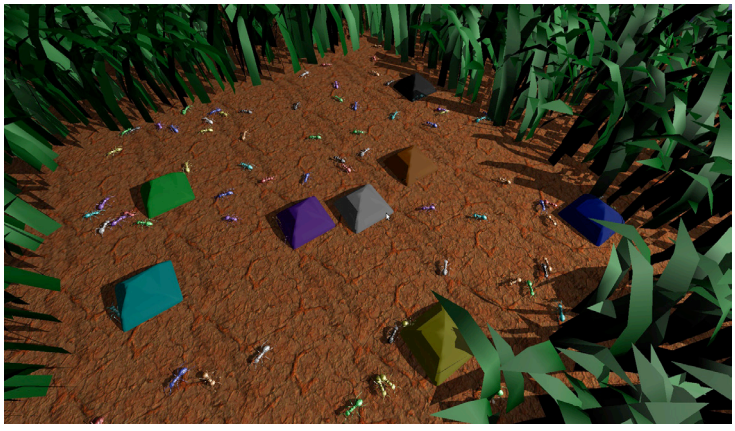


Fig. 2. Find the Anthill.

There were 11 submissions, and for the tournament, they were fed a total of 547 questions. Table 1 shows the ranking and their percentage of correct answers. Note that the primary ranking criterion was whether they were compilable and never crashed, and the number of correct answers was only the secondary criterion.

2.8. *Cops and Robbers (Scotland Yard) (2014)*

Task description Some cops are hunting a robber in a city whose streets and junctions are given as an undirected connected graph (Fig. 3). Initially, the cops and robbers are positioned on distinct junctions (nodes in the graph), and in each round, the robber and all cops move along an edge of the graph. As soon as a cop moves onto the same node as the robber, the cops win, and if this never happens during a predefined number of rounds, the robber wins.

One bot controls the robber, and another bot controls all cops. Every bot must be able to play both roles. At any time, the bots know the entire graph and the positions of all agents.

Discussion This is a classic problem in graph theory and we refer to Aigner and Fromme for a nice introduction (Aigner and Fromme, 1984).

The participants mostly implemented sophisticated scoring functions that would for instance weigh the distance between the cops and the robbers against the number of possible escape routes.

Table 1

The ranking and their percentage of correct answers

rank	1	2	3	4	5	6	7	8	9	10	10
% correct	50%	37%	35%	45%	36%	28%	27%	2%	0%	N/A	N/A
behavior	never crashed			sometimes crashed						did not compile	

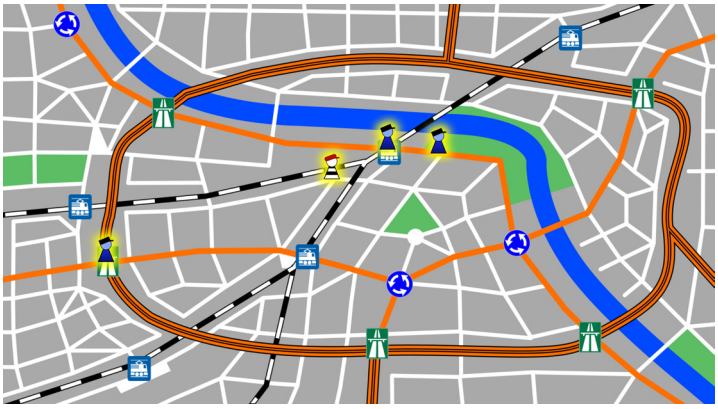


Fig. 3. Cops and Robbers.

2.9. *Diamond Auction (2015)*

Task description The players are at an auction bidding for diamonds. Each player starts with the same amount of gold and for each diamond everyone can bid an amount of his choosing. The highest bid gets the diamond, but all offered gold has to be paid. The goal is to maximize the number of bought diamonds using only the given amount of gold. To allow for more interesting strategies, there are several games played with the same players without restarting the participant's programs. This way, they can learn and exploit each other's strategies.

Discussion Being a very simple game with a simple protocol, we had many submissions for this task. Many of them implemented a random strategy sometimes even worse than our sample bots. Most of the more successful solutions tried to imitate the opponent or to predict the next move. The following graph (Fig. 4) shows the last game of the final, the lines showing the bids of the two players for the different rounds. Although the bids seem random the players are able to predict each others bid and bid just a little bit more. This results in a very clever use of the available gold.

2.10. *Vector Car Racing (2016)*

Task description Each player controls a car and in each round he can change his velocity vector by at most one in both directions. The game is played on a grid (i.e. a graph paper) with a start, a goal, some checkpoints and some walls enclosing a racetrack. The goal is to visit each checkpoint at least once and be the first to reach the goal. Should a player drive into the wall (e.g. if he didn't slow down early enough in a turn) he will be reset to the last visited checkpoint. To allow for some interaction, we added two items that can manipulate the velocity vector of other players.

Discussion This game is a well known pen and paper game and goes back to Jürg Nievergelt. It became widely known by an article in Martin Gardner's column in the Scientific American (Gardner, 1973).

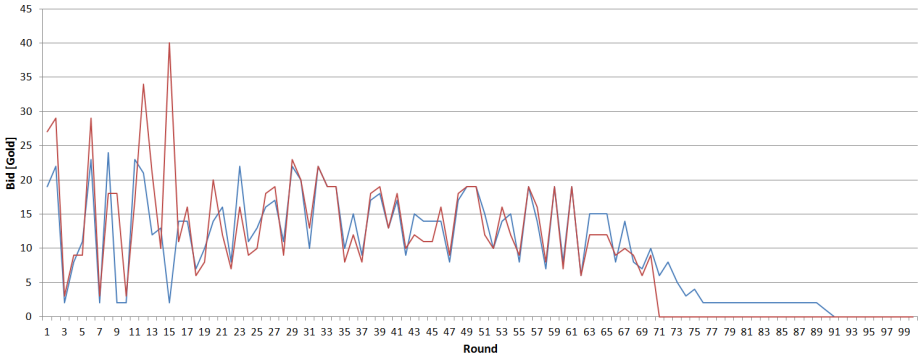


Fig. 4. Diamond Auction (the graph shows the last game of the final).

The submissions fell into one of two categories: sophisticated algorithms beating human players easily (3 submissions) or being barely able to play the game (4 submissions). One example from the second category used DFS to determine the "shortest" path resulting in some of the longest possible paths.

The winning submission consists of 1741 lines of code using a wide range of algorithms. On small maps it uses a four dimensional BFS (location and velocity) to find the optimal solution. On bigger maps, a heuristic is used to improve the runtime of this algorithm and only paths between two checkpoints are calculated. To find the best order of checkpoints it uses some more heuristics to find a good approximations for the traveling salesman problem.

2.11. *Statistics*

Table 2 shows our observed participation rates in the last ten years. Given the *bonus*-like character of the creativity tasks, participation rates in the past years fluctuated heavily. Usually, only students with enough time (and interest) left after solving the five *classical* problems tackled the creativity task. Tasks with a very simple interface (like the diamond auction in 2015) seem to have drawn additional interest, probably because writing a first solution was easily possible within an hour, also for beginners.

3. Implementation Details

3.1. *Our Task Selection and Creation Process*

The SOI is organized by a team of approximately twenty people, most of whom are former participants and now university students. Each year when the preparations of the first round of the SOI start, they discuss a few proposals for the creativity task on their internal mailing list. Once the task is chosen, two to three organizers are selected to write the detailed task description and implement the game server and the visualization. A few more organizers proofread the task description and write sample bots in as many languages as possible.

Table 2
Participation rates in the last ten years

year	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
task	c5	ants	maze	tanks	snake	ants	quiz	c&r	◇	race
creativity	17	N/A	4	6	2	10	11	9	25	7
total	**21**	**31**	**35**	**40**	**48**	**29**	**29**	**28**	**52**	**72**

The organizers of the creativity task are usually people who have successfully taken part in the creativity task a few years before when they were participants, and they are supported by older organizers who prepared the creativity task a few times before. This ensures a good knowledge transfer, which leads to good creativity tasks, even though this task format is more challenging for task creators than ordinary tasks and even though every year, many organizers join and leave the team.

3.2. *Evaluating the Submissions*

After the end of the first round of the SOI, the organizers review and compile all submissions for the creativity task, and then run a large number of games, covering a representative selection of different game parameter and player combinations. The results of all these games are then aggregated into a final score for each participant, and points for the creativity task are awarded as follows:

- A bot that follows the protocol and is capable of playing the game without violating any rules is awarded a predefined percentage p of the points, usually between 30% and 50%, depending on how difficult the task is. We also look at the source code to see if it witnesses more effort than just copying one of the provided sample bots.
- The rest of the points are awarded according to the aggregated score described above, in such a way that the participant ranked last gets p points, and the participant ranked first gets full score.
- If the bot contains minor bugs, such that it can still play in most games, but sometimes crashes, runs into an infinite loop, or violates the rules, some points are deducted.

This grading scheme ensures that every effort to write a program for this task is rewarded, even if it ranks last, which hopefully encourages beginners to tackle this task.

3.3. *Technical Considerations*

Ease of running the game server and the visualization It is crucial that all participants are able to run the game server on their own computers.[2] However, the game server needs to be able to invoke other programs (the bots) and read from their standard output and write to their standard input, a feature which is hard to get operating system independent. And even if the game server works on all major operating systems, the instructions on how to install the dependencies, compile and run it might look so long and complicated that beginners are quickly scared off. For the visualization, it can even get worse, if specific graphics libraries have to be installed and linked.

[2] We cannot provide an online game server instead, because in the first round of the SOI, no participants should be disadvantaged because they cannot use their favorite programming language. This would mean that we would have to support a very large number of programming languages, and moreover, it would require a big effort for the sandboxing.

In our experience, implementing the game server in Java, which is available on all platforms, seems to be the best solution. But even getting a Java game server to run was a bit challenging for some participants, because some could only install Java 6 (instead of newer versions) on their system, while others did not know how to specify the paths to the bots as command line arguments, etc. But we could always help them by email support or through our forum. What we do not know, however, is how many people did not succeed in getting the server to run, but did not ask for help.

For the visualization, the two best solutions seem to be either to integrate the visualization in the Java game server, or to implement it as an HTML5/JavaScript page, where one can paste the log of the game server, and then watch the visualization. An advantage of the latter is that it only requires a web browser, which is available on every computer, so we can provide some sample game logs, so that the participants can watch some games before even starting to implement their own solution and to bother getting the game server to run.

Fraud Detection The grading of the creativity task is not fully automated on purpose. The submissions are compiled manually to check for compatibility issues and the source code is read to get an idea of the participant's solution and to check that all the rules are followed. Given the small number of participants, this is feasible and also eliminates the need for a separate fraud detection or sandboxing of the submitted programs.

4. Discussion

4.1. *Task Selection Criteria*

We now present our criteria for the selection of the creativity task. This is more a collection of useful arguments that came up in past discussions rather than an ultimate list.

- **Interactivity:** It should be interactive, and interaction should not only happen between the game server and the bots, but also between the bots, i.e. the possible moves of a player should depend on the actions that the other players took before. This is mostly to make the task more fun and different from the standard tasks, but it is not strictly necessary. For instance, the *Grand Theft Cake* task (section 2.3), or the *Who wants to be a billionare?* task (section 2.7) do not have any interaction between the players, and still were successful creativity tasks. In the case of the *Vector Car Racing* task (section 2.10), however, some special effects were added to the traditional game to make it interactive, and to increase the size of the search space that optimal solutions would have to explore.
- **Hardness:** There should be no optimal solution, or optimal solutions should be too expensive to calculate.
- **Flat learning curve:** The task should be simple, with few, easy to understand rules, so that it is suitable for beginners, but it should also offer many interesting options for advanced participants.

- **Novelty:** The task should not be "too standard" to make sure one cannot just copy and adapt a standard solution found on the internet. For instance, chess or Nine Men's Morris were proposed in the past years, but not chosen for this reason.
- **Continuous Complexity Curve:** There should be a continuous spectrum of imagineable solutions between straightforward random bots and very sophisticated solutions.
- **Plethora of solutions:** There should be many different solutions that could work reasonably well and are feasible to implement. For instance, for some proposed tasks, it was feared that applying a standard minimax algorithm would be almost the only reasonable solution, and these tasks were thus not chosen.
- **Manual playing:** It is a plus if the game can be played by hand, such as *Multisnake* (section 2.5) or *Vector Car Racing* (section 2.10), so that the participants can compare their bots to their own intuitive playing style.
- **Visualizability:** A good task should allow for an appealing way of watching the interaction between the bots and giving a dramatic view on the progress of the game.

4.2. *Similar Task and Contest Types*

The IOI and other programming contest are always experimenting with new, less algorithmic task types. We refer to Verhoeff (Verhoeff, 2009) for an overview of *classical*, algorithmic IOI tasks. Forišek (Forišek, 2013) gives an overview of the programming contest landscape. He also presents some tasks from the IPSC and Slovak national competitions whose goal is different from just finding the fastest algorithm. Some contests entirely focus on hard optimization problems like Topcoder's Marathon matches and Google's Hash Code and other contests feature tournaments of competing bots for interactive problems like the AI Challenge or the Computer Olympiad.

Creativity-like tasks are also mentioned as *game-playing events* by Burton in (Burton, 2008), where many interesting suggestions for out-of-the-ordinary activities in Olympiad training camps are proposed.

4.3. *Advantages of this Task Format*

In our opinion, creativity tasks offer a number of unique possibilities. These nonstandard tasks can keep the participants busy for many weeks. They can try out all kinds of crazy algorithms and even the best ones are never really done. But also students with limited time can participate with some quick extension of the provided random bot.

The visualization of these tasks are ideal to be shown on a large screen during an award ceremony as parents, friends and teachers can really see what the contestants were doing. As related board and computer games are well-known, an audience that is not familiar with specifics of the algorithmic challenge can also appreciate the results.

The tournaments are fun to watch as the interaction between the bots suggests that the contestants are immediately fighting against each other.

Finally, we believe that creativity tasks are also a way to train for the newer, less-standard IOI tasks where heuristical solutions are required. Examples of such creative IOI tasks include *Languages* and *Maze* from IOI 2010, *Odometer* from IOI 2012 and *Art Class* from IOI 2013.

4.4. *Disadvantages of this Task Format*

It often proved challenging to find tasks that beginners can easily get started with but still leave a lot of options for the creativity of more advanced students. This higher initial hurdle lead to fairly small numbers of serious submissions for many of the presented tasks. From an organizer's point of view, creativity tasks take significantly more time to prepare and grade than regular tasks.

5. Conclusion

In summary, creativity tasks proved to be a beneficial addition to the types of algorithmic challenges given in the first national round of the Swiss Olympiad in Informatics.

We provide the full description and supplementary material for the presented tasks online at `http://creativity.soi.ch` and we want to encourage other delegations to experiment with such task types and look forward to learning about their experiences.

Acknowledgments

We would like to thank all the volunteers of the Swiss Olympiad in Informatics who created, prepared and graded these tasks over the years.

References

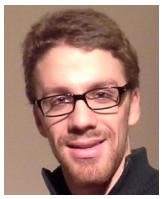
Aigner, M., Fromme, M. (1984). A game of cops and robbers. *Discrete Applied Mathematics*, 8(1), 1–12.

Burton, B. (2008). Breaking the routine: events to complement Informatics Olympiad training. *Olympiads in Informatics*, 2, 5–15.

Forišek, M. (2013). Pushing the boundary of programming contests. *Olympiads in Informatics*, 7, 23–35.

Gardner, M. (1973) Sim, Chomp and Race Track: new game for the intelligent (and not just for Lady Luck). *Scientific American*, 228(1), 108–115.

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



S. Grütter was a participant at CEOI 2009 and IOI 2010, and has helped with the organization of the Swiss Olympiad in Informatics since 2011. He was deputy leader of the Swiss team at IOI 2013.

He holds a BSc in Computer Science from Ecole Polytechnique Fédérale de Lausanne (EPFL), where he is currently pursuing his MSc and working at the Scala Lab under the supervision of Prof Martin Odersky.

His research interests are logic, formal languages, type systems and compilers.

D. Graf participated at CEOI 2009 and IOI 2009 (bronze) and returned to IOI as the delegation leader of the Swiss team in 2012, 2014 and 2015. He volunteers for the Swiss Olympiad in Informatics since 2010 and is its president since 2011.

He holds a MSc in Computer Science from the Swiss Federal Institute of Technology in Zurich (ETHZ), where he is currently pursuing his PhD in the group for Algorithms, Data Structures, and Applications of Prof. Peter Widmayer.

B. Schmid participated at IOI 2013, IOI 2014 (bronze) and CEOI 2013 (bronze). Since 2015 he helps organizing the Swiss Olympiad in Informatics.

He is currently pursuing a BSc in Computer Science from ETH Zurich.

Olympiads in Informatics, 2016, Vol. 10, 87–98

DOI: 10.15388/ioi.2016.06

Understanding Unsolvable Problem

Jonathan Irvin GUNAWAN
Undergraduate Student
School of Computing, National University of Singapore
Computing 1, 13 Computing Drive, Singapore 117417
e-mail: jonathan.irvin@yahoo.com

Abstract. In recent IOIs, there are several problems that seem unsolvable, until we realise that there is a special case to the problem that makes it tractable. In IOI 2014, the problem 'Friend' appears to be a standard NP-hard Maximum Independent Set problem. However, the graph is generated in a very special way, hence there is a way to solve the problem in polynomial time. There were several contestants who didn't identify the special case in this problem, and hence were stuck at the problem. In this paper, we will study a well-known technique called reduction to show that a problem we are currently tackling is intractable. In addition, we introduce techniques to identify special cases such that contestants will be prepared to tackle these problems.

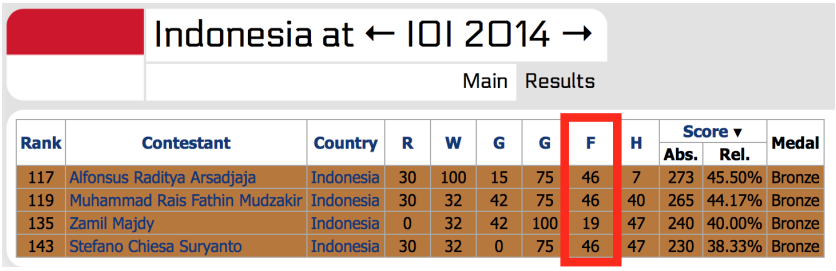
Keywords: special case, unsolvable, NP-hard.

1. Introduction

The problem 'Friend' in IOI 2014 required contestants to find a set of vertices with maximum total weight, such that no two vertices in the set are sharing a common edge. This is a classical Weighted Maximum Independent Set problem. We can show that Weighted Maximum Independent Set problem is NP-hard by reduction from 3-SAT (Cormen *et al.*, 2009). Since the formulation of NP-completeness 4 decades ago, no one has been able to propose a solution to any NP-hard problem in polynomial time. Clearly, it is not expected that a high school student can solve the problem in 5 hours. None of the Indonesian IOI 2014 team solved this problem during the contest. After returning from the competition, I asked the Indonesian team about this problem. None of the team members were aware of the fact that Maximum Independent Set is an NP-hard problem, and thus were stuck trying to solve a general Maximum Independent Set problem.

A similar problem also occurred in IOI 2008. The problem 'Island' required contestants to find a longest path in a graph with 1,000,000 vertices. The longest path problem is a classic NP-hard problem which can be reduced from the Hamiltonian path problem. If a contestant is not aware that the longest path problem is difficult to solve, the contestant may spend a lot of his/her time just to tackle the general longest path problem, without realising that there is a special case to the given graph.

Generally, some contestants spend too much thinking time trying to solve something that is believed to be unsolvable. If only they realise that their attempt is intractable, they may try a different approach and find a special case of this problem. In section 2 of this paper, we will introduce a classic reduction technique often used in theoretical computer science research. In the context of competitive programming, we may find out that a problem which we are attempting is unlikely to be solvable. After realizing that a problem is intractable, we are going to discuss how to proceed to solve the problem in section 3. Finally, in section 4 we will take a look at some common special cases in competitive programming that can be used to solve these kind of problems.

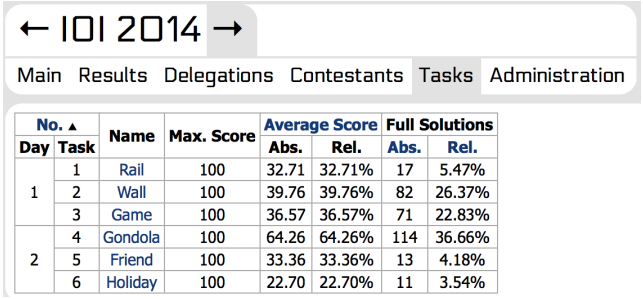


Indonesia at ← IOI 2014 →

Main Results

Rank	Contestant	Country	R	W	G	G	F	H	Score ▾		Medal
									Abs.	Rel.	
117	Alfonsus Raditya Arsadjaja	Indonesia	30	100	15	75	46	7	273	45.50%	Bronze
119	Muhammad Rais Fathin Mudzakir	Indonesia	30	32	42	75	46	40	265	44.17%	Bronze
135	Zamil Majdy	Indonesia	0	32	42	100	19	47	240	40.00%	Bronze
143	Stefano Chiesa Suryanto	Indonesia	30	32	0	75	46	47	230	38.33%	Bronze

Fig. 1. The result of Indonesian team in IOI 2014, taken from `http://stats.ioinformatics.org`. The red squared column highlights the 'Friend' problem.

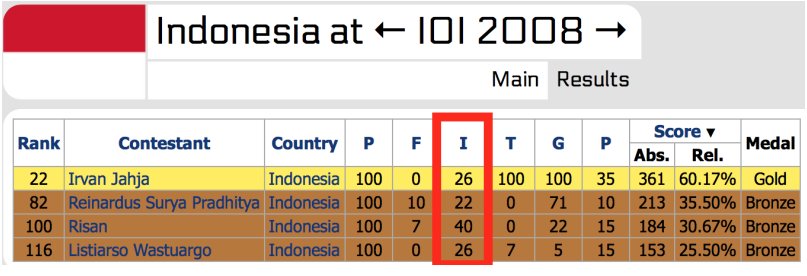


← IOI 2014 →

Main Results Delegations Contestants Tasks Administration

No. ▲		Name	Max. Score	Average Score		Full Solutions	
Day	Task			Abs.	Rel.	Abs.	Rel.
1	1	Rail	100	32.71	32.71%	17	5.47%
	2	Wall	100	39.76	39.76%	82	26.37%
	3	Game	100	36.57	36.57%	71	22.83%
2	4	Gondola	100	64.26	64.26%	114	36.66%
	5	Friend	100	33.36	33.36%	13	4.18%
	6	Holiday	100	22.70	22.70%	11	3.54%

Fig. 2. IOI 2014 tasks statistics, taken from `http://stats.ioinformatics.org`. 'Friend' problem is the second least accepted problem in IOI 2014. It may be because some contestants (at least all the Indonesians) were stuck at trying to solve a general case of Maximum Independent Set.



Indonesia at ← IOI 2008 →

Main Results

Rank	Contestant	Country	P	F	I	T	G	P	Score ▾		Medal
									Abs.	Rel.	
22	Irvan Jahja	Indonesia	100	0	26	100	100	35	361	60.17%	Gold
82	Reinardus Surya Pradhitya	Indonesia	100	10	22	0	71	10	213	35.50%	Bronze
100	Risan	Indonesia	100	7	40	0	22	15	184	30.67%	Bronze
116	Listiarso Wastuargo	Indonesia	100	0	26	7	5	15	153	25.50%	Bronze

Fig. 3. The result of Indonesian team in IOI 2008, taken from `http://stats.ioinformatics.org`. The red squared column highlights the 'Island' problem.

2. Identifying Intractability of a Problem through Reduction

We would like to know that the problem that we are attempting is unlikely to have an immediate solution. The most common way is to apply a well-known technique called reduction. Suppose we know that problem X is impossible to solve, and we also know that we can solve problem X by using problem Y as a black-box[1]. If we can solve problem Y, then we can solve problem X as well. Therefore, problem Y is also impossible to solve.

We are using NP-hard problems for illustration. Recall that NP-hard problems have yet to be solved in polynomial time for more than 4 decades. MIN-VERTEX-COVER is a graph problem that involves finding a minimum subset of nodes such that for every edge, at least one of its endpoint is in the subset. MAX-INDEPENDENT-SET is a graph problem of finding a maximum subset of nodes such that for every edge, at most one of its endpoint is in the subset. Suppose we already know that MIN-VERTEXCOVER is a NP-hard problem. Therefore, we can show that MAX-INDEPENDENT-SET is also a NP-hard problem by reducing a MIN-VERTEX-COVER problem into a MAX-INDEPENDENT-SET problem.

We will first prove the following two lemmas.

Lemma 1. *If $S \subseteq V$ is an INDEPENDENT-SET of graph $G(V, E)$, then $V - S$ is a VERTEX-COVER of the graph $G(V, E)$.*

Proof. Let us assume that $V - S$ is not a VERTEX-COVER. Therefore, there are two vertices $A, B \notin V - S$ and there is an edge connecting A and B. Since $A, B \notin V - S$, we have $A, B \in S$. As A and B are connected by an edge, we note that S is not an INDEPENDENT-SET. This is a contradiction. Therefore $V - S$ is a VERTEX-COVER.

Lemma 2. *If $S \subseteq V$ is a VERTEX-COVER of graph $G(V, E)$, then $V - S$ is an INDEPENDENT-SET of the graph $G(V, E)$.*

Proof. The proof is actually similar to the previous lemma. Let us assume that $V - S$ is not a INDEPENDENT-SET. Therefore, there are two vertices $A, B \in V - S$ and there is an edge connecting A and B. Since $A, B \in V - S$, we have $A, B \in S$. As A and B are connected by an edge, we note that S is not a VERTEX-COVER. This is a contradiction. Therefore $V - S$ is an INDEPENDENT-SET.

Theorem 1. *If $S \subseteq V$ is a MAX-INDEPENDENT-SET of graph $G(V, E)$, then $V - S$ is a MIN-VERTEXCOVER of the graph $G(V, E)$.*

Proof. We know that $V - S$ is a vertex cover by lemma 1. The only thing that remains for us to prove is its minimality. Suppose $V - S$ is not minimum vertex cover. Then, there is another vertex cover $V - S'$ where $|V - S'| < |V - S|$, which implies that $|S'| > |S|$. By lemma 2, S' is an independent set. Therefore, S is not a maximum independent set. This is a contradiction. Therefore $V - S$ is the minimum vertex cover.

[1] We say that problem X is reducible to problem Y

From the above theorem, we conclude that a MIN-VERTEX-COVER can be easily constructed if we have a MAX-INDEPENDENT-SET.

In the beginning of this section, we assume we know that MIN-VERTEX-COVER is a NP-hard problem. It is good to know as many NP-hard problem as possible. This is necessary so that if we encounter a new problem X, we can use any of the NP-hard problems that we know, reduce it to problem X, and thus prove that X is also NP-hard.

3. How to Proceed

Suppose we already know that a problem is unsolvable (i.e. any known algorithm will not solve this problem in time). In competition, it is impossible to complain that "This is unsolvable, can you eliminate this problem?" to the judges, since the judges believe they have a solution. Such a request is absurd when there are already several contestants who have solved that problem. Also, in a major competition (e.g. ACM International Collegiate Programming Contest World Finals, IOI), it is unlikely that the judges have incorrect solution.

3.1. *Approximation*

In real life, when we cannot find the optimal solution, we can try to find the solution that is close to the optimal solution. More specifically, we try to find a solution that is not larger than α (where $\alpha > 1$) times the optimal solution for a minimisation problem. The most common approximation algorithm I found in textbooks is the 2-approximation MIN-VERTEX-COVER problem, which means that the algorithm will not choose more than twice the number of vertices than the optimal solution. However, approximation problems rarely occur in competitive programming (especially IOI). One of the reason is because to create this kind of problem, the judges have to know the optimal solution in order to verify that the contestant's solution is indeed α-approximation. However, generating the optimal solution is impossible (or takes a long time). Since this approach is not really suitable for competitive programming, I will not discuss this approach in detail.

3.2. *Pruning*

This approach is useful in some competitive programming problems. In ACM International Collegiate Programming Contest (ICPC) World Finals 2010, problem I (Robots on Ice) required the contestant to count the number of Hamiltonian Paths with constraints (ACM ICPC World Finals 2010 problem statement, n.d.), which is known to be a NP-hard problem. While finding all possible paths is impossible, the solution

for this problem is to prune the exponential algorithm we use to find all possible paths (ACM ICPC World Finals 2010 Solutions. n.d.). If at some point we know that it is impossible to visit the rest of the unvisited points, then we can prune the path and backtrack immediately. However, it is not very suitable for IOI. Usually, IOI problems require deep analysis from the contestant. It is rare that we can get Accepted by only "hacking" a complete search algorithm. Therefore, we will not discuss this technique in detail.

3.3. *Finding Small Constraints*

Suppose there is an NP-hard problem with large N that is impossible to solve exponentially (e.g. $N > 50$). Sometimes we should also look for other small constraints that may help. For example, a SUBSET-SUM problem is considered an NP-hard problem, and will not be solvable with $N = 100$. When we are given the constraint that all the elements inside the array are small (e.g. $[0, 100]$), this problem can be solved using $O(NX^2)$ Dynamic Programming, where X is the upper bound of the elements inside the array. Even though the running time of the algorithm is exponential to the size of the input, the algorithm is still fast enough for the given constraint. Another way to apply this technique is when the value of N is not too large (e.g. $N < 40$). For example, while $O(2^N)$ algorithm for $N = 36$ is unlikely to run in one second, we can, for instance, use a $O(2^{N/2})$ Meet In The Middle algorithm for solving a problem like SUBSET-SUM. While $O(2^{N/2})$ is still exponential to N, it is much faster than $O(2^N)$, and the range of N that it can solve is twice the range of N using a $O(2^N)$ solution.

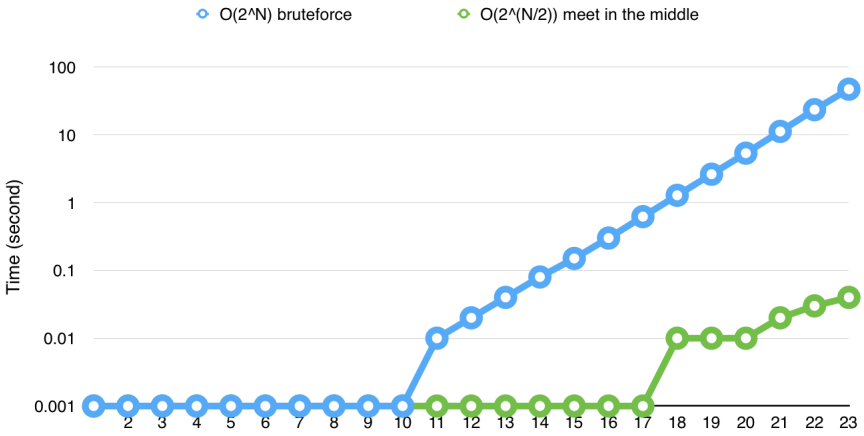


Fig. 4. Comparison of $O(2^N)$ and $O(2^{N/2})$ SUBSET-SUM algorithm running time for various input sizes. This experiment is run 100 times for each value of N on a MacBook Pro (Retina, 13-inch, Early 2015).

3.4. *Finding Special Cases*

This is the most suitable approach in IOI, and thus is the main focus of this paper. To solve IOI 2008 Island and IOI 2014 Friend, we need to use this approach. We must find a special constraint in the problem such that this constraint allows the problem to be solvable in polynomial time. We can check whether an additional constraint causes a problem to be solvable in polynomial time using the reduction proof of the original problem (without the additional constraint), and check whether the proof still holds given the additional constraint to the problem.

We will give we one example of a special case in a NP-hard problem. Suppose we have a function with the following formula

$$f(n) = \begin{cases} 1, & n = 1 \vee n = 2 \\ f(n-1) + f(n-2), & n > 2 \end{cases}$$

We consider the sequence $F = \{f(n)\}_{n=1}^{\infty}$, and we define $F(N)$ to be the first N terms of F. We want to know whether we can create a partition of $F(N)$ into two disjoint multisets A and B such that the sum of all elements in A is equal to the sum of all elements in B.

This looks like a classic PARTITION problem. PARTITION problem is NP-hard by reduction from SUBSET-SUM. Therefore, for a large value of N, it is unlikely to be able to find an algorithm that finds A and B in an efficient way. However, this sequence is defined in a very special way, in the sense that F is defined using the aforementioned recurrence. Therefore, we should inspect the recurrence formula more closely.

Lemma 3. *Any consecutive subsequence of F with length multiples of three can be partitioned into two multisets of equal sum.*

Proof. Pick any consecutive subsequence of F with length multiples of three, which we will denote by $F' = \{f(a), f(a+1), f(a+2), ...f(b)\}$ for some $a < b$. We can partition F' into

$$A = \{f(a+3k), 0 \leq k \leq \frac{b-a-2}{3}\} \cup \{f(a+1+3k), 0 \leq k \leq \frac{b-a-2}{3}\}$$

$$B = \{f(a+2+3k), 0 \leq k \leq \frac{b-a-2}{3}\}$$

A and B will have the same sum, as for every $0 \leq k \leq \frac{b-a-2}{3}$

$$f(a+3k) + f(a+1+3k) = f(a+2+3k)$$

by the construction of the function.

Theorem 2. *If* N *is divisible by three, then* $F(N)$ *can be partitioned into two multisets of equal sum.*

Proof. If N is divisible by three, then $F(N)$ is a prefix of F with length multiples of three. By lemma 3, $F(N)$ can be partitioned into two multisets of equal sum.

Theorem 3. *If* $N \equiv 1 \pmod 3$, $F(N)$ *cannot be partitioned into two multisets of equal sum.*

Proof. Suppose $F'(N) = F(N) - f(1)$. Note that $F'(N)$ contains $N - 1$ elements. Since $N \equiv 1 \pmod 3$, we have $N - 1 \equiv 0 \pmod 3$. By lemma 3, $F'(N)$ can be partitioned into two multisets of equal sum. Therefore, the sum of all elements in $F'(N)$ is even. However, the sum of all elements in $F(N) = F'(N) + f(1)$, which is odd because $F'(N)$ is even while $f(1)$ is odd. Therefore, there is no way to partition $F(N)$.

Theorem 4. *If* $N \equiv 2 \pmod 3$. $F(N)$ *can be partitioned into two multisets of equal sum.*

Proof. Assign $f(1)$ to A and $f(2)$ to B. Since $f(1) = f(2)$, we are now trying to partition $F'(N) = F(N) - f(1) - f(2)$. $F'(N)$ will have $N - 2$ elements. Since $N \equiv 2 \pmod 3$, we obtain $N - 2 \equiv 0 \pmod 3$. By lemma 3, $F'(N)$ can be partitioned into two multisets of equal sum, which implies our theorem.

Therefore, solving this problem is reduced to checking whether $N \equiv 1 \pmod 3$. We can solve this in $O(1)$.

We will provide more examples of special cases in the following section.

4. Some Example of Special Cases

We will look at some common examples of special cases that may occur in competitive programming problems.

4.1. *Planar Graphs*

Planar graph is a graph that can be drawn on a flat surface without having two edges crossing each other (West *et al.*, 2001). There are many graph problems which are easy to solve if the graph is planar. We will provide several examples.

4.1.1. *Number of Edges*

In simple general graph, the number of edges can be up to a quadratic order with respect to the number of vertices (i.e. $O(V^2)$). This is not the case for planar graph. In a planar graph, we may show that $E \leq 3V - 6$ holds for $V \geq 3$ by using the Euler's formula. Therefore, the number of edges is $O(V)$. Naturally, any algorithm that has $O(E)$ in its running time can be changed into $O(V)$. Computing shortest path in a general graph us-

ing Bellman-Ford algorithm takes $O(VE)$ (Halim and Halim, 2013), but it only takes $O(V^2)$ in a planar graph. Counting the number of connected components using DFS in a general graph takes $O(V + E)$, but it only takes $O(V)$ in a planar graph. Therefore, if the problem requires us to compute the number of connected components in a planar graph, even though the constraint states that $V \leq 100,000, E \leq 100,000^2$, the standard DFS solution still runs under one second (in competitive programming, we assume that 1 million operations can be done in 1 second (Halim and Halim 2013)).

4.1.2. *Maximum Clique Problem*

The Maximum Clique problem requires us to find the maximum set of vertices in a graph, such that every pair of vertices in the set is directly connected by an edge. By reduction from vertex cover, the maximum clique problem on general graph is NP-hard. However, it is easy to solve this problem in planar graph. Consider the problem of colouring a graph such that no two adjacent vertices have the same colour. Note that if there is a clique of size k in a graph, the set of the vertices inside the clique must be coloured with k colours. Therefore, the entire graph requires at least k colours. By the four colour theorem, any planar graph can be coloured with at most four colours. (Gonthier, 2005). Since at most four colours are required to colour a planar graph, there does not exist a clique with more than four vertices. Hence, we can solve the problem by checking whether there is a clique with four vertices. If there is no clique with four vertices, we check whether there is a clique with three vertices. Finding whether there is a clique with k vertices can be solved naively in $O(V^k)$. Therefore, the whole solution takes $O(V^4)$ time, which is polynomial. The solution can be improved to $O(V^2)$. Instead of having four nested loops to find four vertices independently, we can have two nested loops over the edges instead and test whether the four vertices (which are the endpoint of the two edges) form a clique. This solution takes $O(E^2)$, which is the same as $O(V^2)$ in planar graph.

4.1.3. *Problem Example*

We will use a past competitive programming problem to illustrate the importance of the properties in a planar graph. The 'Traffic' problem from Central European Olympiad in Informatics (CEOI) 2011 illustrates this concept well. In this problem, there is a directed graph with up to $V = 300,000$ vertices given in a 2D plane. There are two vertical lines denoted as 'left' and 'right'. All vertices are contained within the 'left' and 'right' lines, with some vertices possibly lying on these two lines. The problem requires the contestant to print the number of visitable[2] vertices lying on the 'right' line from every vertex lying on the 'left' line (which we shall call 'left' vertices and 'right' vertices for simplicity). An instance for this problem can be seen on Fig. 5.

The brute force solution for this problem is to run DFS from every 'left' vertex which gives us a running time of $O(V^2)$. It is very difficult to find a solution (if any) faster than $O(V^2)$ for this problem. However, there is an important constraint on the problem. The graph given in this problem is always a planar graph. We first relabel the 'right' vertices in ascending order according to the *y*-coordinate. The planar properties ensures that for

[2] vertex *v* is visitable from vertex *u* if there is a path from *u* to *v*

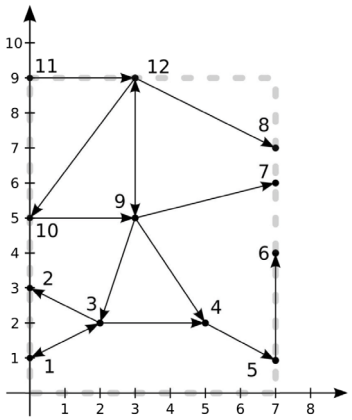


Fig. 5. An instance of problem 'Traffic'. In this example, the expected output is {4, 4, 0, 2}, since the top 'left' vertex can visit 4 'right' vertices, the second top 'left' vertex can visit 4 'right' vertices, the third top 'left' vertex cannot visit any 'right' vertex, and the bottom 'left' vertex can visit 2 'right' vertices.

every 'left' vertex, the sequence of 'right' visitable vertices from that 'left' vertex is a contiguous sequence, assuming that we have removed all 'right' vertices which are not visitable from any 'left' vertex. With this property, there is a $O(V \log V)$ solution (CEOI 2011 Solutions, n.d.).

4.2. *Bipartite Graph*

Bipartite graph is a graph in which the vertices can be partitioned into two disjoint sets U and V such that all edges connect a vertex from U and a vertex from V. Some problems have a bipartite graph as an input although the problem statement does not explicitly state that the given input graph must be bipartite. The problem that we used as an introduction for this paper, IOI 2014 Friend is a very good example. The construction of the graph in subtask 5 of this problem implicitly ensures that the final graph will always be bipartite. There are several graph problems that are NP-hard for general graph but solvable in polynomial time if the graph is bipartite. Since bipartite graph and bipartite matching was recently included in IOI 2015 syllabus (Forišek, 2015), we can expect that this type of problem may be conceived in the near future of IOI. We will take a look at several examples.

4.2.1. *Vertex Cover and Independent Set (and Maximum Matching)*

As we discussed in an earlier section, both of the MIN-VERTEX-COVER and MAX-INDEPENDENTSET problems are NP-hard. However, both of these problems are solvable in polynomial time on bipartite graph. By Konig's theorem, the size of the minimum vertex cover in bipartite graph is equal to the size of the maximum bipartite matching (Bondy and Murty, 1976), and the size of the maximum independent set in

bipartite graph is equal to the number of the vertices minus the size of the maximum bipartite matching. Therefore, both problems are equivalent to finding the size of the maximum bipartite matching, which can be solved in $O(V^3)$ time. Finding the size of the maximum matching in bipartite graph using Hopcroft-Karp algorithm or Maximum-Flow algorithm is much simpler to solve, as compared to using Edmonds Blossom algorithm on general graph. This is actually the solution of the 5th subtask of IOI 2014 Friend.

4.3. *Directed Acyclic Graph*

A directed acyclic graph is a directed graph that does not contain any cycle. Similar to planar and bipartite graphs, there are several graph problems that are much easier to solve if the graph is a directed acyclic graph.

4.3.1. *Minimum Path Cover*

MIN-PATH-COVER is a problem that requires us to find the minimum number of vertex-disjoint paths needed to cover all of the vertices in a graph. By a simple reduction from Hamiltonian Path, this problem is NP-hard. A graph has a Hamiltonian Path if and only if we only need one path to cover all of the vertices. However, this problem can be solved in polynomial time for a directed acyclic graph. For a graph $G = (V, E)$, we create a new graph $G' = (V_{out} \cup V_{in}, E')$, where $V_{out} = V_{in} = V$ and $E' = \{(u, v) \in V_{out} \times V_{in} : (u, v) \in E\}$. Then it can be shown by Konig's Theorem that G' has a matching of size m if and only if there exist $|V| - m$ vertex-disjoint paths that cover all of the vertices in G.

4.4. *Miscellaneous*

4.4.1. *Special Case of CNF-SAT Problem*

We will use Google Code Jam 2008 Round 1A 'Milkshake' problem for this example. This problem requires the contestant to find a solution with the minimum number of true variables that satisfy a CNF-SAT problem with up to 2,000 variables (Google Code Jam 2008 Round 1A, 'Milkshake' problem, n.d.). The CNF-SAT is a satisfiability problem given in a conjunctive normal form (i.e. conjunction of disjunction of literals) which was proven to be NP-hard (Cook, 1971). Therefore, it is unlikely that there is an algorithm to solve a CNF-SAT problem with 2,000 variables in less than 8 minutes[3]. However, there is a special property in this problem, in which at most one unnegated literal exists in each clause. Therefore, all clauses can be converted into Horn clauses. With this property, a linear time algorithm exists. (Google Code Jam 2008 Round 1A, 'Milkshake' solution, n.d.).

[3] In Google Code Jam, contestants are given 8 minutes to produce the output upon downloading the input.

5. Conclusion

In conclusion, we can use a well-known reduction technique to prove that a problem that we are currently attempting to solve is impossible (or at least it is very hard such that no people has been able to solve it for more than 40 years). In competitive programming (including IOI), understanding this technique is essential so that we will not be stuck at trying to solve an impossible problem, thus prompting us to find another way to solve the problem. To prove that a problem is NP-hard, it is good to know as many NP-hard problems as possible, so that we can reduce from any one of the problems that we know to the new problem. Some of the classic NP-hard problems include 3-SAT, Vertex Cover, Independent Set, and Subset Sum. After realizing that the problem is NP-hard, we must be able to find the special case that makes the problem solvable. We must be able to find a special property that breaks the reduction proof. Having a lot of practice on these kind of problems will help us to familiarize with the possibilities of a special case. Some of the common special cases include planar, bipartite, and directed acyclic graph.

References

ACM ICPC World Finals 2010 Problem Statement (n.d.). Available via internet:
`http://icpc.baylor.edu/download/worldfinals/problems/2010WorldFinalProblemSet.pdf`
ACM ICPC World Finals 2010 Solutions (n.d.). Available via internet:
`http://www.csc.kth.se/~austrin/icpc/finals2010solutions.pdf`
Bondy, J.A., Murty, U. S. R. (1976). *Graph Theory with Applications*, Vol. 290, London: Macmillan.
CEOI 2011 Solutions (n.d.). Available via internet:
`http://ceoi.inf.elte.hu/probarch/11/ceoi2011booklet.pdf`
CEOI 2011. 'Traffic' Problem (n.d.), available via internet:
`http://ceoi.inf.elte.hu/probarch/11/trazad.pdf`
Cook, S.A. (1971). The complexity of theorem-proving procedures. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. ACM, 151–158.
Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2009). *Introduction to Algorithms (3rd ed.)*. MIT Press.
Forišek, M. (2015). International olympiad in informatics 2015 syllabus. Available via internet:
`https://people.ksp.sk/~misof/ioi-syllabus/ioi-syllabus.pdf`
Gonthier, G. (2005). A computer-checked proof of the four colour theorem. Available via internet:
`http://research.microsoft.com/en-us/people/gonthier/4colproof.pdf`
Google Code Jam 2008 Round 1A, 'Milkshake' Problem (n.d.). Available via internet:
`https://code.google.com/codejam/contest/32016/dashboard#s=p1`
Google Code Jam 2008 Round 1A, 'Milkshake' solution (n.d.). Available via internet:
`https://code.google.com/codejam/contest/32016/dashboard#s=a&a=1`
Halim, S., Halim, F. (2013). *Competitve Programming 3*. `lulu.com`
IOI 2008, 'Island' Problem (n.d.). Available via internet:
`http://www.ioinformatics.org/locations/ioi08/contest/day1/islands.pdf`
IOI 2014, 'Friend' Problem (n.d.). Available via internet:
`http://www.ioinformatics.org/locations/ioi14/contest/day2/friend/friend.pdf`
West, D. B. *et al.* (2001). *Introduction to Graph Theory*, Vol. 2. Prentice hall Upper Saddle River.

J.I. Gunawan is an undergraduate student studying Computer Science in National University of Singapore. He participated a lot of programming contests, including IOI 2012 and 2013 and ACM ICPC World Finals 2014 and 2015. He is also the lead of the Scientific Committee of Indonesian Computing Olympiad team (TOKI) training Indonesian teams for IOI in 2015–2016.

Olympiads in Informatics, 2016, Vol. 10, 99–109

DOI: 10.15388/ioi.2016.07

Homo Informaticus – Why Computer Science Fundamentals are an Unavoidable Part of Human Culture and How to Teach Them

Juraj HROMKOVIČ
Department of Computer Science, ETH Zürich
e-mail: juraj.hromkovic@inf.ethz.ch

Abstract. The goal of this article is on one hand to introduce informatics as a scientific discipline in the general context of science and to outline its relationships especially to mathematics and engineering, and on the other hand to propose a way how to integrate computer science into school education.

Keywords: teaching, computer science, mathematics.

1. Introduction

The development of human society is mainly determined by the ability to derive knowledge and to find efficient ways of applying it. Let us explain this claim more carefully. Human beings sample experiences by observations and experiments and use them to generate knowledge by combining their experience with logical thinking. The derived knowledge is used to develop procedures in order to reach concrete goals. A crucial point in our considerations is that to use such procedures one usually does not need to fully understand the knowledge used to obtain such procedures. To an even higher extent, one does not need to understand the way in which this knowledge was derived and verified. Let us illustrate this on a simple example.

The famous theorem of Pythagoras claims $c^2 = a^2 + b^2$ in any right triangle, where c is the length of the longest side (hypotenuse). This theorem was used to fix the right angles when building houses and temples in the classical antiquity. The workers only needed to build a triangle of sizes 5 units, 4 units, and 3 units in order to get a right angle, and for sure they did not need to understand how the theorem of Pythago-

ras was discovered and how it was proven. Hence, to successfully apply the derived knowledge, one did not need to master the high qualification of an investigator. In this way, scientists changed and still change society and became a crucial factor in human development.

Computer science was created as a natural step forward in the above described development, when the following two conditions were satisfied:

1. Using the exact language of mathematics, one was able to discover and describe procedures in such a way that no intellect was needed to apply them. Executing them step by step, everything was unambiguous, and no educated or trained person was needed to find the right interpretation of the particular instructions of the procedure executed.
2. The technology enabling to execute the discovered procedures by machines was developed, and languages providing the opportunity to "explain" universal machines what they have to do were designed.

In this context, we speak about automation. We let the machine execute not only physical work, but also such human work considered as intellectual work in the past. This is the reason why computer science is a mixture of mathematics and engineering.

On the one hand, one uses the concepts, methods, and the language of mathematics in order to understand the considered entities and their relations so exactly that algorithms as unambiguously interpretable procedures can be developed in order to solve a variety of problems everywhere in science, technology, and everyday life. Here, mathematics is the instrument that has to be mastered together with the specific area in which one tries to solve problems.

On the other hand, developing and improving the enabling technology is mostly engineering. This is not only about hardware, but especially about software enabling us to conveniently communicate with computers in high-level programming languages.

We see that computer science is a part of the natural development of science, and it became a discipline that is crucial for the performance of the human society. This contextual view is important when thinking about what computer science actual is, and how to teach computer science in schools. We are asked not to teach specific isolated facts, methods, and other final products of scientific work, but the way how they were discovered, i. e., the paths from the motivation coming from the general context of science to the final products of the research work and engineering. As in its fundamentals computer science is very strongly related to mathematics as its basic research instrument, we start with the question "What is mathematics and how to teach it?" in the next section, and use the derived point of view in the final section to propose the way of teaching computer science in schools. Before the final section, however, we discuss engineering as a missing subject in schools in Section 3. Again, we use this discussion in order to show in the final section how teaching computer science can contribute to understanding some basic concepts of engineering and how it integrates them in the school curriculum.

2. Mathematics as the Language of Science and Consequences of this View for Teaching It

What is mathematics? If you pose this question, you can get a lot of different answers; frequently, even the response that this question is too hard to be answered satisfactorily. A mathematician or a university student can tell you with high probability that she or he proves theorems and thus investigates the structure and the properties of artificial mathematical objects, which can be useful to model reality, or the relationships between different such objects. A high-school student can tell you that mathematics consists of calculations that can be used to solve some classes of mathematical tasks (also called problems) such as solving quadratic equations or systems of linear equations or analyzing the properties of a curve. Obviously, you can also get a response that mathematics is something that is hard to understand, and that every "normal" human being can exist and be successful without it. This is a frequent answer, and unfortunately it is more of a rule than an exception.

Mathematics is the most powerful instrument humans ever have developed in order to investigate the world around us. But it is taught in such a way that the students do not realize this fact. Especially in high schools, they learn methods (algorithms) to solve some given problems (for instance, to find a maximum of a function). This may be also viewed positively as an intellectual challenge, because several methods are not easy to manage. But the bad news is that they can learn to successfully apply methods for solving different problems without really understanding why these methods work properly. High-school students often do not have a good intuition of what infinity or limits are about, but they use these concepts in a fuzzy way in order to analyze artificial functions without seeing any relation to reality. What are we doing wrong? A lot. We have to start to think first about what mathematics is, and then to try to find a way how to teach mathematics in a proper way.

From my point of view, the best way to view mathematics is as a special language developed for science, i. e., for knowledge generation. Going a few thousand years back, people wanted to discover "objective" knowledge. The important word here is the word objective. If you want to reach that, first of all you need a language in which each statement has an unambiguous interpretation for everybody who mastered this language. How to reach this? First of all, you need to give an absolutely exact meaning to the words (notions) you use, because the words are semantically the corner stones of each language. In this context, mathematicians speak about axioms. Many people have the wrong impression that axioms are claims in whose truthfulness we believe, but of which one is not able to prove that they are true. This is mostly not the case. Axioms are the precise definitions of basic notions that describe our intuition about the meaning of these notions. Probably the first concepts people tried to fix were notions such as number, equality, infinity, point, line, distance, etc. What is very important to observe is that people needed hundreds and in some cases thousands of years to come up with such definitions that the community of philosophers and later mathematicians has accepted. Why was all of this done? First of all, the language of mathematics is able to describe

objects, structures, properties, and relationships in an unambiguous way. In this context we speak about the "descriptional" power of mathematics. Thus, people were able to formulate exact claims this way, and so to express our knowledge unambiguously. But this was only one side of the language of mathematics. The language of mathematics was also used to derive new knowledge from the given knowledge, i. e., as a knowledge generator. Leibniz formulated this role of mathematics in a very nice way. He wanted to omit all political discussions and fighting in different committees by simply expressing the real problems in the language of mathematics, and then using calculations and logical derivations to obtain the right solution. Interestingly, he called this "automation" of the human work. By now we know that this dream of Leibniz cannot be achieved. There are two reasons for that. First of all, because of its exactness, the language of mathematics is restricted in its descriptional power, and so we cannot translate all real-world problems into this language. Secondly, one of the most important discoveries of the last century made by Gödel tells us that the argumentational power of the language of mathematics is smaller than its descriptional power. This means that one can formulate claims in current mathematics for which there do not exist proofs of whether they are true or not.

What do scientists learn from that? The development of the language of mathematics is similar to the development of natural languages. You need to create new words and describe their meaning in order to increase its descriptional power, and to be able to speak about things you were not able to speak about before. Moreover, you need new concepts and words to be able to argue about matters you were not able to argue before, i. e., in order to strengthen your capability of deriving new knowledge by thinking. A very important point is that the process of developing the language of mathematics is infinite. As long the number of basic, axiomatic words of the language of mathematics is finite, one can always formulate claims that are not provable within this mathematical language, and one has to introduce new words in order to be able to prove them and so to make new discoveries.

A very nice example is the notion of infinity. Nobody saw anything infinite in the real world, and even most physicists believe that the universe is finite. This means that infinity is something artificial, simply an artificial product of mathematics. But without this concept, most of the current science would not exist. Without the notion of infinity, there would be no concept of limits and so we would not be able to exactly express notions like actual speed or actual acceleration. Our science simply would be somewhere before the discoveries of Newton. Hence, without the "artificial" concept of infinity, one is strongly restricted in the ability to discover our finite, real world.

Another example of a crucial notion is the concept of probability. Most of the sciences, even the non-exact ones such as didactics, psychology, medicine, and economy, heavily use this concept to model and investigate reality, and if they would make predictions without this concept, they would have serious trouble to convince society that their results are trustable to some extent and not only expert opinions.

Why did we focus on the view of mathematics presented above? Because it is the nature of mathematics that shows us which changes are necessary in order to improve

the education in mathematics for everybody. Based on this, we recommend to adopt the following concepts:

1. Focus more on the genesis of the fundamental notions (concepts) of mathematics. To define them took centuries, to prove most of the theorems took a few years. Each new concept enabled to investigate so many things that no single discovery can compete with introducing a fundamental concept. The strengthening of mathematics as a research instrument is the main job of mathematics, and deriving new concepts (not necessarily on the axiomatic level) in mathematics provides the best, true picture of its nature. Without this, nobody can really understand its role and usefulness. And only if one understands the genesis of mathematics as the development of a language of science and as a research instrument, one can be able to apply it regularly to all areas of our life. Teaching mathematics this way can completely change the behaviour of the members of our society. Instead of memorizing and sampling facts provided, one would start to verify the degree of trustability of claims sold as knowledge and to understand to which extent and under which conditions one is allowed to take them seriously.
2. Concrete examples first, abstraction as a final discovery. One first has to touch concrete problem instances and objects in order to get some intuition about their properties. Then, one can formalize her or his intuition into a formal concept. One has to follow the natural way of discovering that usually goes from concrete to abstract. To sell methods and theorems as final products is as poor as teaching manuals for washing machines or Microsoft office instead of teaching discoveries of physics, mechanics, and computer science that enabled to develop these products.
3. Teach algorithmics instead of training calculation methods. Pupils learn in schools to multiply arbitrarily large integers in decimal representations, to solve quadratic equations and systems of linear equations, or to analyze functions, etc. In all cases, most pupils learn to apply given methods, but most of them do not understand why they work. It is more of a challenging memorizing than a deep understanding of the nature of the algorithms used. One has to start to introduce problems instead of presenting methods solving them, and ask the pupils and students to solve concrete problem instances first and finally to discover an algorithm as a robust procedure that is able to solve any of the infinitely many concrete problem instances of the given problem. Discovering algorithms as well-functioning calculation procedures offers another quality of education in mathematics than executing a given calculation method, which any pocket calculator can do faster and more reliable. Teach programming as the art of exactly describing the methods discovered in an unambiguously interpretable way in the language of the machines, and strengthen the ability of exact communication this way.
4. Teach the principles of correct argumentation. Teach the notions of implication and quantifiers, and train direct and indirect proofs. Do not believe that the pupils in high schools cannot learn to verify and to derive simple proofs. They did not manage this in the past, because there was no effort made to teach proving claims, or most effort in this direction was done in a wrong way.

5. Guarantee the opportunity to the pupils to deal with the subjects as many time as needed at an individual speed. Mathematics is one of the sciences that needs a large number of iterative touchings of particular topics until one is allowed to say "Eureka," and gets a reasonable understanding of what it is about. The trouble is that no teacher can assure this by herself or himself for everybody in the class. Another problem is that most textbooks of mathematics are good collections of exercises, but explanations are written more for teachers than for pupils. One way out is to change the style of the textbooks. The textbooks should be written in such a way that pupils and students would be able to learn from them by their own with a minimal support from outside. Partitioning the discoveries into a number of small, natural steps written in the language mastered by the pupils at the corresponding age, and regularly giving the opportunity to verify whether one understands the topic up to now properly are some of the basic principles used to create good textbooks for mathematics.

Finally, one can ask how to reach the new teaching style for mathematics described above. For sure, one cannot ask the high-school teachers to make this change without showing them how to do this in detail. One also cannot ask educationalists, who do not have a sufficiently deep contextual knowledge of mathematics to master these changes. The movement has to start at the universities, where the teaching style has to change first. In order to speed up this process in Switzerland, in our department at ETH Zürich we develop new textbooks for teaching different topics of mathematics and computer science for all school levels. Our experiments prove that mathematics can become one of the most favorite subjects of pupils and students if taught in the way described above. Students can successfully master topics that were considered to be too hard for them before, and the marks in mathematics can be significantly higher than the average marks over all topics.

For me, it is not a question of whether the proposed evolution of the education in mathematics will come. It is only the question of the time at which particular countries will need to adopt it. Since this is a service for the future generation, the earlier the better.

3. Why Engineering is Not Allowed Not to Be a Part of Basic Education

As mentioned in the introduction, human society uses the knowledge discovered in order to reach different goals more efficiently by developing various procedures or different products. This is highly creative work that is beyond the pure learning of facts and making calculations that can be completely automatized. The whole process of engineering work starts with a description of the goals to be achieved. After that, one starts to combine experience and fundamental knowledge of science in order to design a solution that has to be implemented as a prototype. Next, one has to test this prototype, modify, and improve it until an acceptable product is produced.

In today's schools, we find almost nothing about the concept of iterative specifying, testing, modifying, and improving the product of our work, let alone about fundamental constructive ways of creating original products. But this is fundamental to human activities since the beginning of time. The current school systems ignore this fact to a high extent and are more about teaching to memorize than teaching to work in a creative way. One can explain this educational misconception by the fact that, in contrast to basic scientific models of reality, the engineering work is heavily dependent on experience that is often hard to formalize and thus to teach. The creative work of engineers as human experts cannot be described by an algorithm (a method). However, this must not be a reason to remove engineering from education, because students also have to learn to build their own experience over a longer period of time in order to become an expert for a special area.

The crucial fact we want to point out is that computer science mastered to formalize several basic concepts of engineering and made them available to our schools as a result. Teaching computer science is a chance to introduce engineering as a highly creative, constructive activity to our educational system.

Let us consider some concrete illustrations. Probably the most fundamental concept of technical sciences is modularity. One builds some simple units with clear and well verifiable functionalities and calls them modules. Then one uses these modules as fundamental building blocks to construct more complex systems that are again considered basic modules for constructing even more complex systems. There is no upper bound on the growth of the complexity of systems built in such a way. Unbelievably many human activities can be described by this modular concept. This includes learning. Ones learns some simple facts and methods and their applications in a restricted framework. After that, one masters them perfectly in such a way that one can use them as elementary operations in attacking more complex tasks. All spiral curricula run in a well-designed modular way. Thinking in a modular way when trying to reach given goals is the most fundamental instrument of creative human work. And it does not matter whether you apply it top-down or bottom-up. If one applies a top-down approach, then everything is clear and one designs only a transparent and well-verifiable plan for constructive work. More typical is the bottom-up approach where one builds more and more complex modules without knowing what the final product will be about.

Programming is an excellent instrument to teach modularity. Writing programs to automate solving different tasks provides modules for attacking much more complex tasks. Children of age between 10 and 12 are able to put five loops into each other without being disturbed by or even observing the complexity of their final product due to modularity. One simple program containing a loop gets a name and so becomes a new instruction. This instruction is used in the body of another loop that also becomes a new instruction used in another loop, etc. Here, one can train both, the top-down approach as well as bottom-up approaches devoted to open-end tasks. To practice the ability to bring a clear structure into complex processes is at least as important as any of the fundamental concepts of sciences or humanities contained in the school curricula.

Another important concept is teaching to test or verify products of our work. Typically, children verify their own products by asking teachers or adults to tell them whether their work resulted in what they were asked to reach. But this way, they cannot sufficiently build their self-confidence because they need a strong dependency on their supervisor. They have to learn to trust their solutions by being taught to verify the products of their work by themselves. Programming is again a wonderful instrument for this purpose using the platform or editor that fits the educational requirements. One can learn to test the functionality of programs by running them on several data sets as well as by the verification approach based on a transparent structuring of the program into small modules whose correct behavior can by easily verified.

In general, constructive engineering work within the educational system changes the behavior of young people in an essential way. They move from the role of customers searching for products with some desired property to the role of creative inventors for designing and developing products with new, original functionalities. The never-ending story of improving any human products with no limits on what can be approached in the future would be the most important, great consequence of embedding the creative way of thinking of engineers into school systems.

4. Teaching Computer Science as a Fundamental Step in the Evolution of Our Educational Systems

In the two previous sections, we already outlined, with respect to improving teaching of mathematics and introducing engineering, the principal contributions that could be offered by teaching computer science in schools in a proper way. This word "proper" is crucial for us, and thus we start by listing what we are not allowed to do if we want to avoid a disaster when introducing computer science to schools. In what follows, we present the most frequent mistakes that already caused frustrations in different countries.

1. To teach how to work with concrete software products and call it computer science. This activity destroyed the image of computer science as a scientific discipline in the past.
2. To let computer science be taught as a part of "social media" by teachers educated in human sciences only, and focusing on social, emotional and psychological aspects of communication by new technologies.
3. To choose the topic to be taught by committees of experts offering their favorite topic without looking at the whole context of science as presented above in Section 2.
4. To sell computer science as the ability to work with computers.
5. To sell computer science as a special branch of mathematics.
6. To sell computer science as a pure engineering discipline.
7. To try to teach the newest achievements of computer science. Think about what would happen if physics would try to do that instead of following the history, and thus developing step by step our view of the physical world.

8. To try to sell computer science as a joy much easier than mathematics and physics by avoiding any depth and thus a spiral curriculum, and instead presenting one simple application after the other.
9. Going too much into technical details about concrete programming languages, software systems, or hardware.

While 1, 2, and 4 have been the main reasons for destroying the image of computer science in the society in the past, currently the points 7 and 8 are the major danger for establishing computer science as a school subject.

After listing what we are not allowed to do, it is now time to switch to positive recommendations and conceptual work. We do not want to make a proposal for the content of a computer science curriculum, because our goal is not to go too much into detail, and because, for sure, there are various possibilities for good implementations of the computer science subject in schools. What we try here is to recommend a strategy and principles that are useful for designing a computer science curriculum that can be accepted as a fundamental part of education in its generality for everybody, and that essentially contributes to:

1. The understanding of our world (in this case with the focus on the artificial world created by humans).
2. Developing our way of thinking in a dimension that cannot be compensated by teaching other school subjects.
3. Providing knowledge that is useful and sometimes even expected as prior knowledge for the study of a variety of specialized scientific disciplines later (university studies, etc.).

As a byproduct, we have to aim to improve teaching overall, especially by strengthening the subjects mathematics and natural sciences. We already presented the basic strategy how to design a computer science curriculum in Section 2 about mathematics. We have to follow the genesis of computer science and think about motivations and fundamental concepts introduced and discovered by computer scientists from the point of view of science as a whole. For sure, we have to think about or even discover which concepts are available to which extent in which age, and to follow all the ideas for creating good teaching materials as presented in Section 2. Let us be more concrete and present a few examples.

One can decide to introduce programming at the age between 8 and 14. The first step is to deal with abstractions that enable us to unambiguously describe the problem instances. Then we teach to sample experience by trying to find solutions to concrete, special problem instances, whose size and complexity may grow with growing experience. After some time, one can develop a strategy that works successfully for a small collection of problem instances that we subsequently call a problem. Having a solution strategy, one has to learn to communicate it, i. e., to unambiguously describe it for anybody else. After that, we are allowed to start teaching proper programming by describing our strategy as a program in a suitable programming language. We are not allowed to teach the list of all instructions (fundamental words) of a programming language. We have to start with very few (10 to 15) fundamental instructions, and use modularity to

create new words (instructions) in order to make our communication with the computer more convenient. After writing programs, we let them run in order to verify their correct functionality and learn to correct, improve, and modify our programs in order to get a final product with which we are satisfied. Let us list some of the added values when teaching the introduction to programming in the way described above:

1. Training and strengthening the abstraction in representing real situations by drawing graphs, writing lists or tables with different kinds of elements.
2. Contributing to teaching mathematics by searching for a solving strategy, instead of simply learning a method as a given product of the work of others.
3. Strengthening the ability to express matters and procedures in an exact way, and so to improve the way used to communicate.
4. Recognizing that a language is not a given final product of human work, but that any language is continuously developed, and that in case of a programming language one can develop the language on her or his own with respect to her or his personal demand.
5. Defining new instructions by describing the meaning of the new words by subprograms, one learns the principle of a modular design that is common and fundamental in engineering.
6. Introducing the concepts of testing, verifying, modifying, and improving is the first contact with the creative, constructive work of engineers.

A really good teaching sequence for introductory programming can be created if one focuses on the above listed added values and not on technical details of programming languages and other software used or on a specific class of tasks.

Another nice example is teaching cryptography. Cryptography can be viewed as the history of developing the notion "secure cryptosystem." One can start with the historical examples in order to introduce the basic terms decryption, encryption, key, and cryptosystem with a lot of creative work by designing and breaking new, own cryptosystems. After defining the concept of "security" by Kerckhoff, one can build the bridge to probability theory. The concept of probability was used to design new cryptosystems and later to break them. One can wonderfully understand the importance and the usefulness of the concept of probability studying the history of secret communication in this way. Then one can introduce the formal mathematical definition of absolutely secure cryptosystems with respect to the concept of probability, and recognize that such system cannot be built for practical purposes. Finally, the concept of computational complexity offering public-key cryptosystems is the way out, leading to the recent e-commerce.

What we try to repeatedly present as the key strategy is to follow the history of the discoveries of particular concepts, methods, and ideas, and not to try to sell finalized products of science. The creative work is the most (and may be even the only really) exciting part of the study. Let us teach creativity by repeatedly discovering things that were already discovered, up to the point where one is able to discover something completely new. Forget about teaching facts, teach how to verify the trustability of claims made by others. We are lucky, because we are allowed to create a curriculum for a

completely new subject. We can implement principles, which the other subjects still did not recognize, and so contribute to the evolution of the system of education. For those who would like to see detailed implementations of the design principles presented above, we recommend to following textbooks from our production (Böckenhauer and Hromkovič, 2013; Freiermuth *et al.*, 2014; Hromkovič, 2011; Hromkovič, 2014) or the book "Algorithmic Adventures – from Knowledge to Magic" (Hromkovič, 2008; Hromkovič, 2009).

References

Böckenhauer, H.-J., Hromkovič, J. (2013). *Formale Sprachen.* Springer Vieweg.
Freiermuth, K., Hromkovič, J., Keller, L., Steffen, B. (2014). *Einführung in die Kryptologie.* 2nd Edition. Springer Vieweg.
Hromkovič, J. (2011). *Berechenbarkeit.* Vieweg+Teubner.
Hromkovič, J. (2008). *Sieben Wunder der Informatik – Eine Reise an die Grenze des Machbaren.* Vieweg+Teubner.
Hromkovič, J. (2009). *Algorithmic Adventures – From Knowledge to Magic.* Springer.
Hromkovič, J. (2014). *Einführung in die Programmierung mit LOGO.* 3rd Edition. Springer Vieweg.

J. Hromkovič is professor of informatics with a special added focus on computer science education at ETH Zurich. He is author of about 15 books published in 6 languages (English, German, Russian, Spanish, Japanese, and Slovak) and about 200 research articles. He is member of Academia Europaea and the Slovak Academic Society.

Olympiads in Informatics, 2016, Vol. 10, 111–124

DOI: 10.15388/ioi.2016.08

Examples of Algorithmic Thinking in Programming Education

Juraj Hromkovič, Tobias Kohn, Dennis Komm, Giovanni Serafini
Department of Computer Science, ETH Zürich
Universitätstrasse 6, 8092 Zürich, Switzerland
e-mail: {juraj.hromkovic, tobias.kohn, dennis.komm, giovanni.serafini}@inf.ethz.ch

Abstract. Algorithmic thinking and problem solving strategies are essential principles of computer science. Programming education should reflect this and emphasize different aspects of these principles rather than syntactical details of a concrete programming language. In this paper, we identify three major aspects of algorithmic thinking as objectives of our curricula: the notion of a formal language to express algorithms, abstraction and automation to transfer proven strategies to new instances, and the limits of practical computability.

The primary contribution of this paper are three examples that illustrate how general aspects of algorithmic thinking can be incorporated into programming classes. The examples are taken from our teaching materials for K-12 and university non-majors and have been extensively tested in the field.

Keywords: algorithmic thinking, K-12, spiral curriculum, programming education, Logo, Python.

1. Introduction

Algorithmic thinking constitutes one of the core concepts of computer science. It has proven a versatile and indispensable tool for problem solving and found applications far beyond science. Hence, sustainable computer science education should be built upon algorithmic thinking as its primary objective, thus unfolding benefits for a broad and general education. However, how do we bring algorithmic thinking to computer science education? In this paper, we identify a number of principles that we want to deliver to students at different levels. As the main contribution, we describe concrete examples of how to teach these paradigms, which have been proven successful in the past.

Our work is part of ubiquitous efforts towards establishing sustainable computer science in K-12 education. Particularly noteworthy and inspiring are "CS unplugged" approaches as proposed by Bell *et al.* or Gallenbacher, which do completely away with computers and solely focus on the underlying algorithmic principles (Bell *et al.*, 2012; Gallenbacher, 2008). By incorporating such ideas into programming education, we effectively combine the strengths of the two approaches, resulting in a truly sustainable education.

1.1. *The Setting*

The examples presented in this paper stem from teaching materials we have developed for primary school, high school, and university, respectively (Gebauer *et al.*, 2016; Böckenhauer *et al.*, 2015a; Böckenhauer *et al.*, 2015b; Kohn, 2016). The goal of our endeavours is to create a spiral curriculum that starts as early as fifth grade in primary school with iterations throughout mandatory school, and including computer science classes for non-majors at university level.

We use both Logo and Python in our classes and found that the simplicity of Logo is especially well-suited for primary school and complete beginners. At high school and university level, Python then allows us to discuss topics in more depth and to better link our programming classes to mathematics and the sciences. We have also extended our Python interpreter and included Logo's `repeat`-loop into Python. This allows us to introduce iteration at an early stage without the need for variables, getting the best of both worlds.

Our curricula and examples make heavy use of turtle graphics, both in Logo as well as in Python. Apart from the obvious benefits of direct visualization, the turtle is also a source of powerful didactical metaphors. In particular, the examples as presented in this paper all rely on turtle graphics to convey or visualize an algorithmic principle.

1.2. *Objectives*

Computer science is a vast field with algorithmic thinking at its core. Our curricula hence focus on the study of algorithms and its various aspects. Our approach comprises three major aspects of algorithmic thinking, as described in the following paragraphs: the notion of the programming language as a formal language to express algorithms, abstraction and automation as central problem solving strategies, and the limits of practical computability as a motivation for improving existing algorithms. More on the authors' goals, motivation, and approaches can be found in a complementing paper (Hromkovič *et al.*, 2016).

Concept of a Formal Language. Students are introduced to programming as a means to convey instructions to a machine – in our case the turtle. The initial set of instructions is strongly limited and restricted to basic movements such as moving forward and turning. Each instruction has a clearly defined syntax and semantics, avoiding any ambiguity. At first, then, programming is the activity of writing sequences of such instructions, encoding graphical shapes. From our perspective, this is to say that students use a formal language to combine words to sentences. Even though each valid sentence conveys the information of a graphical shape, not every sentence makes sense in the context of the interpretation of the resulting shape.

The initial vocabulary given to the students is not adequate to encode more complex shapes in a human-accessible form. Students are early required to extend the vocabulary by defining new words, i. e., by defining subroutines. In the context of the turtle, this can

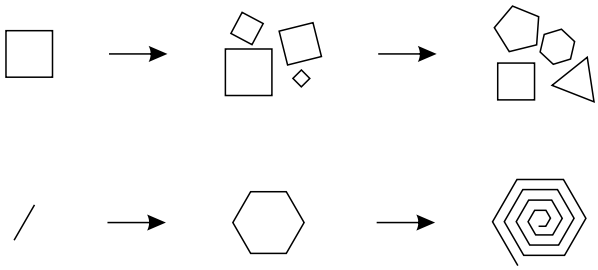


Fig. 1. By parametrizing programs, we gradually gain more versatile algorithms and procedures. Drawing a square of fixed size is the first step towards drawing arbitrary polygons of various sizes (above). Likewise, loops allow us to build ever more complex and larger programs out of simple and small parts (below).

be beautifully explained as "teaching the turtle new words" (Papert, 1993). The semantics of the new words is expressed algorithmically as a sentence over an already existing vocabulary. Think, for instance, of a house consisting of a triangle and a square. Both the triangle and the square themselves might be expressed as sequences of forward- and turning-instructions.

Hence, our objective is to provide students with a simple yet expandable base of instructions, the means to combine these instructions to sentences, and to define new words with associated unambiguous semantics. This way, the students are exposed to the concepts of modularization, formal languages, and expressing semantics in algorithmic form.

Abstraction and Automation. Programming is, of course, much more than combining instructions to form programs. Some of the most essential key concepts are abstraction and automation. Modularization, for instance, only unfolds its full potential in combination with parameters. Having a dedicated instruction to draw a square, say, helps to clarify the intent of a program. Allowing that very same instruction to draw squares of various sizes makes it versatile and open to applications beyond its initial conception. Further abstraction could even introduce a second parameter to pertain to the number of vertices to draw, resulting in one instruction capable of drawing all regular polygons (see Fig. 1).

Abstraction itself also requires the concept of automation. Even drawing a regular polygon with a given number of vertices is a tedious task without the notion of a loop. For the step to an abstract instruction encompassing all polygons, the loop becomes a necessity. Once this level of automation is mastered, students are introduced to loops with variations, allowing for figures such as spirals where even the "parameter" automatically varies (see Fig. 1).

Automation and abstraction are not just core concepts of programming but of computer science and algorithmic thinking in a much wider sense. Expressed in the context of problem solving, abstraction corresponds to the question "Can we adapt an already known or universally available strategy to solve the problem at hand?" Once we know how to solve a single instance, we then employ the concept of automation to apply our solution to a large set of instances.

Limits of Practical Computability. Finding solutions automatically is not always feasible. Indeed, the insight that there are problems that cannot be solved algorithmically (i. e., *undecidable problems*), shown in 1936 by Turing in his seminal paper "On Computable Numbers, with an Application to the Entscheidungsproblem" (Turing, 1936), is one of the deepest results of mathematics and laid the foundation for computer science itself. However, even computable problems can often only be solved under certain restrictions, e. g., using an unacceptable amount of resources (time and space), or without full precision due to numeric errors. This gives rise to numerous interesting research questions and solutions, which can both be explained to non-professionals.

For education, however, we need to make computability and its limitations visible and tangible. A prime example to serve this objective, as taken from turtle graphics, are circles. A computer cannot draw an exact circle, it must be drawn using an approximation such as a polygon (or Bézier curve). The cost of drawing an approximating polygon increases with the number of vertices, mainly due to the fact that the turtle needs time to turn at the vertices. Students therefore must find a compromise between more accurate representations and faster renderings, and eventually realize that the limitations of screen resolution quickly nullify additional precision beyond a certain point.

When seen in the light of modern applications, *intractability* is of particular importance to cryptography. In this regard, the inability to design efficient algorithms has far-reaching implications beyond computer science and its inclusion into the curriculum is well-warranted. At the same time, we found cryptography to be very motivating and well-suited as a subject of its own (Freiermuth *et al.*, 2010).

2. Modular Development: Building a Town Step by Step

Our group is actively involved in introducing young students to programming as soon as at fifth grade. To this end, we developed teaching material, hold classes, and, most importantly, introduce teachers to our didactic approach as well as to fundamental concepts of computer science. These school projects are based on the German textbook *An introduction to programming in Logo* (Hromkovič, 2014, German: *Einführung in die Programmierung mit Logo*), and on a Logo booklet (Gebauer *et al.*, 2016) covering the contents of its first seven chapters. The following example is taken from this booklet.

As already mentioned, one of the main objectives of our programming classes consists in making the students confident with the modular development of programs and teaching them how to systematically apply this problem solving strategy to complex problems. To illustrate our approach and the achievements of the students, we present a sequence of learning activities from the third (out of seven) unit of the courses.

At this point, the students already know how to move the turtle forward and backward on a straight line, how to rotate it as well as how to iterate over a sequence of

instructions for a predefined number of times. More precisely, the current vocabulary of the turtle comprises the following instructions as well as their abbreviations: `forward` (`fd`), `back` (`bk`), `left` (`lt`), `right` (`rt`), `clearscreen` (`cs`), `penup` (`pu`), `pendown` (`pd`), and `repeat`. Furthermore, the students are already used to giving their programs names and to reusing available programs as subprograms within main programs in an elementary way. In subsequent activities, they learn how to develop a table that consists of rows of identical squares in a proper modular way.

To reinforce the concept of modular development, the students are now challenged to write a program for drawing a small town, which consists of streets with identical houses. While the program for drawing a house is already available in the booklet, the students are expected to consequently apply the approach they intensively practiced. They are therefore expected to:

- Identify the next shape or pattern they can systematically reuse.
- Write a sequence of instructions for drawing it.
- Give this subprogram a name.
- Test and iteratively improve the code until the solution meets the assignment.

Afterwards, the students should reflect on how to adjust the position of the turtle in order to draw the pattern by simply reusing the program they developed above, to test and to iteratively improve their approach, and to finally integrate the two modules of their solution. In a following step, this new main program can be reused as a subprogram in other main programs of increased complexity. More specifically, the students are given the program shown in Listing 1.1 accompanied by the following exercise, which asks them to study the effects of each command in detail.

> ***Exercise.*** Where does the turtle start drawing the house? Think about the path the turtle follows when drawing the house using the program `HOUSE`. Where is the turtle located at the end of the execution? Draw the image and describe the effect of each command.

Next, they are told how to design a program `HOUSEROW` (Listing 1.2) that uses `HOUSE` as a subprogram. Here, the most difficult task is to position the turtle in such a way that, after each iteration, the new house is drawn at the correct coordinates.

Listing 1.1: Drawing a house using a simple `repeat`-loop.

```
to HOUSE
  rt 90
  repeat 4 [fd 50 rt 90]
  lt 60 fd 50 rt 120 fd 50 lt 150
end
```

Listing 1.2: Drawing a row of houses.

```
to HOUSEROW
  repeat 5 [HOUSE rt 90 pu fd 50 lt 90 pd]
end
```

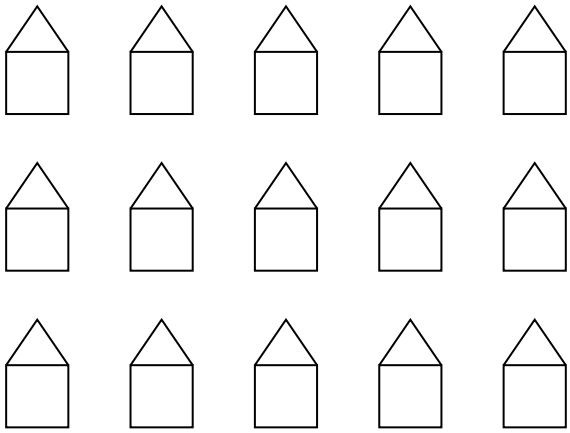


Fig. 2. A small town that consists of 15 houses.

Finally, the students are asked to use the modular approach in order to draw the town that consists of multiple streets. This way, the students learn how to extend the language of the computer step by step with more complex programs. The crucial observation is that the overall complexity is hidden in the smaller subprograms.

> ***Exercise.*** At this point, we would like to extend the complex of buildings by additional streets. Use the program `HOUSEROW` as a building block to draw the image shown in Fig. 2.
>
> *Hint:* After the completion of a row, the turtle has to be moved to the correct position to build the next street.

Modular development offers a didactically appealing platform for creative tasks. In the two following exercises, the students observe that even small changes such as adding a window, a door, or a chimney to their houses may have a considerable impact on the overall outcome of the streets and the town they are designing.

> ***Exercise.*** We decide to order the roof for the houses from another vendor. That is, we get two types of building blocks: One called `ROOF` and another one called `BASE`. Write two programs to draw the two building blocks. Combine those programs to form a new program `HOUSE1` that draws a house.

> ***Exercise.*** The houses in Listing 1.1 are very simple. Try to be creative and come up with a new design for a house. Use your house to build an entire complex of buildings.

The students learn that modular development is a systematic and efficient problem solving strategy. Moreover, they experience that subsequent changes in a basic module of a properly developed complex program require no or very limited additional programming effort.

3. Making Approximation Errors Visible with the Pac-Man

The turtle draws a circle by approximation, actually drawing a polygon with, say, 36 vertices (in practice, students often choose 360 vertices at first, building upon their knowledge that 360° stands for a complete circle). While the resulting figure is not discernible from a true circle on the screen, the approximation requires a couple of corrections when the circle is combined with other shapes. Most prominent is the question of finding the circle's center and the correct value of the radius. Both are slightly, but discernibly, off compared to a true mathematical circle.

A particularly illuminating example is drawing a Pac-Man shape. Students are asked to write a Python program that draws a Pac-Man and typically end up with a solution as shown in Listing 1.3 (note that the `repeat`-loop shown here has been added to Python in order to support our curriculum. A further discussion can be found in the aforementioned complementing paper (Hromkovič *et al.*, 2016)). However, their resulting pictures show that the shape is not closed as seen in Fig. 3: there is a small gap at the center of the shape. This discrepancy is subsequently discussed in a dedicated section and leads to a precise drawing of a pie chart.

Why does this gap in the center occur and how can we correct it? In a circle, any radius meets the circumference perpendicularly. This fact has been used twice in the program (Listing 1.3). For the approximation with a polygon, this does not hold anymore: the angle between the radius leading to a vertex and the circumference requires a small correction φ (Fig. 4). The correction φ is exactly half of the turtle's turning angle at each vertex. For our example with 36 vertices, this results in a 5°-correction (Listing 1.4). The correction of the angle can also be taken into the loop, resulting in a more symmetrical solution (Listing 1.5).

Listing 1.3: Drawing a Pac-Man.

```
from turtle import *
RADIUS = 100
right(45)
forward(RADIUS)
left(90)
repeat 27:
  forward(RADIUS * 3.1416 * 2 / 36)
  left(10)
left(90)
forward(RADIUS)
```

Listing 1.4: Correcting the angle (1).

```
left(90 + 5)
repeat 27:
  forward(RADIUS * 3.1416 * 2 / 36)
  left(10)
left(90 - 5)
```

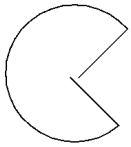


Fig. 3. An incomplete Pac-Man.

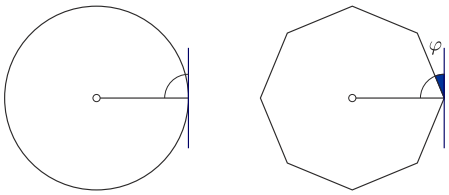


Fig. 4. As circles are approximated by polygons the radius does not meet the circumference in a right angle but is off by an angle φ.

Listing 1.5: Correcting the angle (2).

```
repeat 27:
  left(5)
  forward(RADIUS * 3.1416 * 2 / 36)
  left(5)
```

Listing 1.6: Starting the circumference not at a vertex but at the center of an edge instead.

```
repeat 27:
  forward(RADIUS * 3.15 / 36)
  left(10)
  forward(RADIUS * 3.15 / 36)
```

Finally, an alternative solution is to have the radius meet the circumference not at a vertex but at the center point of an edge. In this case, the radius does meet the circumference perpendicularly. The resulting code, again, has a very symmetrical form (Listing 1.6). Yet the ratio between radius and circumference now differs and requires to change the used approximation of the value of π.

4. Runtime Analysis Backed up by a Little Math

One of the authors is currently part of the team responsible at ETH Zurich for teaching computer science basics to non-computer science students (more specifically, students of biology, pharmacology, environmental sciences, health sciences and technology, agriculture, geology, and nutritional sciences).

We have been introducing students to Logo using the previously mentioned booklet (Gebauer *et al.*, 2016, see Section 2), whose main part is covered in roughly the first unit

Listing 1.7: A simple square.

```
to QUAD :WIDTH
  repeat 4 [fd :WIDTH rt 90]
end
```

Listing 1.8: Testing whether a given number is prime.

```
to PRIME :NUM
  ht
  make "IT 2
  make "ISPRIME 1
  while [:IT<:NUM] [
    make "RES mod :NUM :IT
    if :RES=0 [make "ISPRIME 0] []
    make "IT :IT+1
  ]
  if :ISPRIME=1 [setpc 1 QUAD 8] [setpc 0 QUAD 8]
end
```

of the lecture. After that, an advanced booklet is supplied that is specifically designed for this class (Böckenhauer *et al.*, 2015a). As part of this booklet, more involved concepts such as variables, conditional execution, and `while`-loops are introduced. The students are then given three *projects*, which consolidate what they have learned so far by designing small programs to solve specific tasks (Böckenhauer *et al.*, 2015b). The lecture is accompanied by exercise classes in which the students are asked to present and explain their solutions to a tutor.

As a first step towards the mathematical analysis of algorithms, we give the students the following project. The examples are taken from the project booklet (Böckenhauer *et al.*, 2015b) and the corresponding lecture notes. The goal is to show the students the idea of how to mathematically analyze how long a program will run depending on the input size. A typical example that does not need anything beyond high school mathematics is to test whether a given number is prime. Logo is especially suited to visualize the distribution of (small) prime numbers without much overhead.

The first component is a program called `QUAD` (Listing 1.7) that draws a square, and which essentially consists of a simple loop, which the students already know from previous lessons. The size of the square is determined by the value of the parameter `:WIDTH`.

Next, we can write a program that tests whether a given input is a prime number. This is done in the most straightforward fashion, i. e., by testing whether there is smaller number (except 1) that divides it. Depending on the result, either a red or black square is drawn on the screen using the instruction `setpencolor` (`setpc`). Before that, the turtle is hidden with the instruction `hideturtle` (`ht`). The corresponding algorithm `PRIME` is shown in Listing 1.8.

The students can easily follow the steps and try out different inputs. As a next step, we ask them to carefully check corner cases, and give them the following exercise.

Exercise. PRIME does not yet work correctly on all inputs as the value of :IT is initially set to 2. However, we know that 1 is by definition not a prime number. Thus, PRIME 1 creates an incorrect output. Extend the program such that a black square is drawn when the input is 1. Moreover, an error message should be output if 0 or a negative number is given.

Once the students familiarized themselves with the algorithm, we discuss its *running time*. It is obvious that the time grows with larger inputs, and this seems to be unavoidable on an intuitive level. Furthermore, it is easy to see that the running time directly depends on how often the body of the while-loop has been executed. We therefore agree on counting the number of these executions and neglect how many instructions are executed with each iteration. The above trivial attempt needs roughly 2^n iterations for inputs of length n (hence, n is the number of bits used to represent the input number). Now we show how to improve this running time using a little bit of math. The following exercise is well-suited to be presented to the students as part of the lecture.

Exercise. We can now make our algorithm PRIME "faster" (more efficient) by having it execute the while-loop less often. To this end, we make use of the following idea.

Suppose the input x is not a prime number. Then, by the definition of a prime number, there is a number a, which is neither 1 nor x, that divides x without remainder. But from this it also follows that there is a second number b, which is also neither 1 nor x, that also divides x without any remainder. An important point is that one of these two numbers is not larger than the square root $\sqrt{x}$ of x. If, e. g., a is larger than $\sqrt{x}$, then b has to be smaller, since otherwise $a \cdot b$ were larger than x.

We want to use this observation to improve PRIME. Write an algorithm PRIME2 which works exactly as PRIME, but in which the while-loop is modified such that the variable :IT does not take the values of all numbers smaller than :NUM, but only those that are smaller than or equal to $\sqrt{:\text{NUM}}$.

The students are asked to verify the speedup by trying different inputs. Good candidate inputs to observe the increase in speed of course depend on the computer used. Moreover, the students can try to make a simple running time analysis of PRIME2 themselves, which leads to the result that the loop is now executed at most (roughly) 1.41^n times for inputs of length n.

The above considerations give rise to another question, namely in which kind of analysis we are interested. To this end, the algorithm is modified such that the while-loop is left as soon as a divisor of the input is found. The resulting algorithm obviously still works correctly, but is it faster? Indeed, if the input is, say, an even number, the running time of the new algorithm is a lot better. However, if the input is prime, both algorithms take the same time. This is exactly the difference between a best case and a worst case analysis of the algorithm's running time.

Now we can follow the modular building of algorithms (see Section 2) and use PRIME2 as a building block to visualize the distribution of small prime numbers. More precisely, the algorithm PRIMES shown in Listing 1.9 uses PRIME2 as a subprogram to

Fig. 5. The distribution of prime numbers between 1 and 26. Instead of red and black squares, we use filled and unfilled ones.

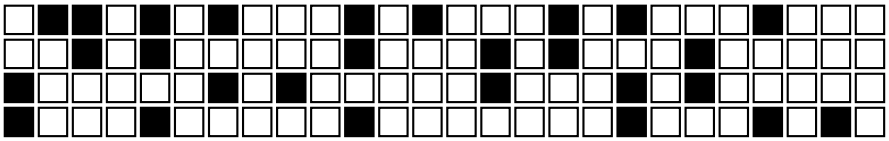


Fig. 6. The distribution of prime numbers between 1 and 104.

Listing 1.9: Visualizing primes.

```
to PRIMES :MAX
  pu lt 90 fd 300 rt 90 pd
  make "TEST 1
  repeat :MAX [
    pu rt 90 fd 10 lt 90 pd
    PRIME2 :TEST
    make "TEST :TEST+1
  ]
end
```

Listing 1.10: Visualizing primes more nicely.

```
to PRIMES2 :MAX :ROW
  pu lt 90 fd :ROW*10/2 rt 90 pd
  make "TEST 1
  repeat :MAX [
    pu rt 90 fd 10 lt 90 pd
    PRIME2 :TEST
    make "REST mod :TEST :ROW
    if :REST = 0 [
      pu lt 90 fd :ROW*10 lt 90 fd 10 rt 180 pd
    ] [ ]
    make "TEST :TEST+1
  ]
end
```

visualize the appearances of prime numbers among the first `:MAX` natural numbers.

The result of executing `PRIMES 26` is shown in Fig. 5. Next, we can improve the appearance by having the squares drawn in multiple rows (Listing 1.10). To this end, we write a new algorithm `PRIMES2` with an additional parameter `:ROW`. The turtle moves to the next row whenever it drew a number of squares that is divisible by the value of `:ROW`. With `PRIMES2 104 26` we obtain an output as shown in Fig. 6.

An advanced exercise then deals with prime powers. The students should solve this task at home, either alone or in small groups. The difficulty of this exercise is due to the usage of a return statement, which is implemented by the `output` instruction in Logo.

Exercise. A prime power is a natural number that has exactly one prime factor. For instance, 27 is a prime power since it has the prime factorization

$$27 = 3 \cdot 3 \cdot 3 = 3^3.$$

Clearly, every prime number is thus also a prime power.
In this project, you design a program PRIMEPOW, which checks whether a given number is a prime power. To this end, do the following steps:

1. Rewrite the program PRIME to obtain a program PRIMEOUT that uses the command output instead of drawing squares. If the value of :NUM is prime, the value 1 should be returned, otherwise 0.
2. PRIMEPOW has one parameter :TEST. We want to check whether the value assigned to :TEST is a prime power (possibly with the exponent being 1).
3. First, the program checks using PRIMEOUT whether :TEST is a prime number. If so, "Prime." is printed on the screen and the execution is ended using stopall.

```
make "ISPRIME PRIMEOUT :TEST
if :ISPRIME=1 [pr [Prime.] stopall] []
```

4. Otherwise, all numbers smaller than the value of :TEST are iterated, and again using PRIMEOUT it is checked whether the current number is a prime. If so, it is checked whether it divides the value of :TEST without remainder.
5. If such a prime number is found, PRIMEPOW takes note of this by setting the value of a variable :FOUND to 1. :FOUND is initialized with 0. If a second such prime number is found, this will be noted since the value of :FOUND is already 1. In this case, "More than one divisor." is printed on the screen and the execution is again ended with stopall.
6. Finally, if :FOUND is still 1 after all numbers were tested, "Prime Power. Base: " and the prime number that divides the value of :TEST without remainder are printed on the screen.
7. Check PRIMEPOW using small input values.

This introduction using Logo proved to be very valuable for the students in the succeeding lessons, where we implement more complex projects using Python. More precisely, they were able to learn important paradigms without having to worry too much about syntactical details.

5. Conclusion

Programming education is a great opportunity to teach important core concepts of computer science on various levels and to establish algorithmic thinking as part of a broad and general education. A necessary prerequisite is, of course, that we find ways to go beyond teaching the specifics of a programming language and rather put emphasis on those aspects of programming that lead to a deeper understanding of computer science.

In this article, we have provided three examples of how programming education can incorporate more general principles of algorithmic thinking. All three examples have been taken from our well-tested teaching materials for primary school, high school, and university level, respectively. Further details about our curricula are given in the complementing paper (Hromkovič *et al.*, 2016).

References

Bell, T., Rosamond, F., Casey, N. (2012). Computer science unplugged and related projects in math and computer science popularization. *The Multivariate Algorithmic Revolution and Beyond*, Springer-Verlag, 398–456.

Böckenhauer, H.-J., Hromkovič, J., Komm, D. (2015). *Programmieren mit LOGO – Projekte*. `http://abz.inf.ethz.ch/wp-content/uploads/unterrichtsmaterialien/primarschulen/logo_projekte.pdf`

Böckenhauer, H.-J., Hromkovič, J., Komm, D. (2015). *Programmieren mit LOGO für Fortgeschrittene*. `http://abz.inf.ethz.ch/wp-content/uploads/unterrichtsmaterialien/primarschulen/logo_heft_2_de.pdf`

Freiermuth, K., Hromkovič, J., Keller, L., Steffen, B. (2010). *Einführung in die Kryptologie*. Springer.

Gallenbacher, J. (2008). *Abenteuer Informatik. IT zum Anfassen – von Routenplaner bis Online-Banking*. Springer, 2 edition.

Gebauer, H., Hromkovič, J., Keller, L., Kosírová, I., Serafini, G., Steffen, B. (2016). *Programmieren mit LOGO*. `http://abz.inf.ethz.ch/wp-content/uploads/unterrichtsmaterialien/primarschulen/logo_heft_de.pdf`

Hromkovič, J. (2014). *Einführung in die Programmierung mit LOGO – Lehrbuch für Unterricht und Selbststudium*. Springer, 3 edition.

Hromkovič, J., Kohn, T., Komm, D., Serafini, G. (2016*). Combining the Power of Python with the Simplicity of Logo for a Sustainable Computer Science Education*. Unpublished Manuscript.

Kohn, T. (2016). *Python. Eine Einführung in die Computer-Programmierung*. `http://jython.tobiaskohn.ch/PythonScript.pdf`

Papert, S. (1993). *Mindstorms*. Basic Books, 2 edition.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2), 230–265.

J. Hromkovič is professor of informatics with a special added focus on computer science education at ETH Zurich. He is author of about 15 books published in 6 languages (English, German, Russian, Spanish, Japanese, and Slovak) and about 200 research articles. He is member of Academia Europaea and the Slovak Academic Society.

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

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.

G. Serafini is lecturer in the Computer Science Teaching Diploma Program at ETH Zurich. He holds a MSc in computer science and a teaching diploma in computer science from ETH Zurich. His research interests focus on the contribution of computational thinking to school education. He is a member of the board of the Swiss Computer Science Teacher Association and a member of the Swiss Olympiad in Informatics.

Olympiads in Informatics, 2016, Vol. 10, 125–159

DOI: 10.15388/ioi.2016.09

Programming in Slovak Primary Schools

Martina KABÁTOVÁ[1], Ivan KALAŠ[1,2], Monika TOMCSÁNYIOVÁ[1]
[1]*Department of Informatics Education, Comenius University*
Mlynska dolina, 842 48 Bratislava, Slovak Republic
[2]*UCL Knowledge Lab, Institute of Education*
23-29 Emerald Street, WC1N 3QS London
e-mail: martina.kabatova@gmail.com, {kalas; tomcsanyiova}@fmph.uniba.sk

Abstract. In our paper, we want to present the conception of elementary programming in primary Informatics education in Slovakia and the process of its integration into ordinary classrooms. First, we will familiarize the reader with the tradition of so called 'Informatics education' in Slovakia and with the various stages of the process of its integration. We will formulate the learning objectives of the elementary informatics as a school subject in Slovakia (referring to Blaho and Salanci, 2011) and give reasons why we believe that it offers an important opportunity for developing informatics knowledge, computational thinking and problem solving skills. We will primarily focus on the presentation of our arguments why we consider programming (in the form rigorously respecting the age of the primary pupils) to be appropriate and productive constituent of learning already for this age group. Several recent research findings, presented by Ackermann (2012) and others support our position here. In the next chapter, we will present in detail the conception of elementary programming and how it is implemented in the continuing professional development (CPD) of primary teachers in Slovakia. We will examine which programming environments are being used, what kind of pedagogies and which specific learning objectives our teachers apply. We will list programming concepts and identify corresponding cognitive operations, which we find appropriate for primary pupils. Then we will present and analyse the CPD of our in-service teachers (and the position of programming in this process) which we have recently implemented in Slovakia. Another important element of our CPD strategy is the well-known Bebras contest (in Slovakia it is called 'iBobor' or 'Informatics Beaver'). In the next chapter of our paper, we will apply qualitative educational inquiry methods to examine how our conception of elementary programming has really penetrated into primary classes in Slovakia. We are also interested in how it is being received by the teachers and pupils. Through interviews with the teachers we will identify different aspects of the whole process and main risk factors, which may complicate or hinder the implementation. In the final chapter, we will study the tendency to develop informatics and programming at the primary level in the context of various research projects presented in the academic research literature. We will compare various key findings of other research projects with our own experience.

Keywords: educational programming, primary education, computing, computational thinking.

1. Introduction

In recent years many educators, education policy makers and scientists call for integration of what they call computational thinking into primary education of all pupils. In influential documents like (The Royal Society, 2012) a need to distinguish computational thinking (or computer science, informatics or computing education) from the ICT education is declared. However, reasons for such reflections differ – computer scientists and industry leaders feel that not enough young people (and only very few women) choose to pursue a career in computer science and they believe that if pupils become familiar with some informatics concepts (within their primary education already) they will favour it later in their lives and careers. Others believe that computational thinking is equally important and key skill as literacy and mathematical thinking and they call for a redefinition of literacy and for integrating digital literacy development into primary education for the sake of educating a fully developed citizens to live in the digital world.

While ICT oriented education is included in most national curricula, many countries do not pay any special attention to including other (and in our opinion more interesting and more important) informatics concepts. However, we can observe important new step in the UK by establishing computing as a compulsory subject in every school year since September 2014.

The situation in Slovakia is different from most of the countries. Informatics as a separate mandatory subject was established many years ago and for many years we have been systematically preparing teachers for teaching it. The authors of our national curriculum took a great care to include topics from core informatics along with learning of basic ICT skills.

Since Comenius University plays a key role in building National Informatics Curriculum (2011, 2015) we have a lot of experience with integrating educational informatics both into schools (primary, lower secondary and upper secondary) and into teachers' professional development (PD). Some activities with digital technology were recently nation-wide integrated into early childhood education (or children 3 to 6) as well: see e.g. (Pekarova, 2008) and (Kalaš, 2010).

The whole conception of informatics is a broad and interesting topic to study, but for the purpose of this paper we fully focus only on one of its topics, namely, on **programming at primary level in Slovakia**. We will discuss its conception and the process of its integration into ordinary classrooms. We will present and explain:

(a) The reasons and short history of implementing primary informatics as a modern core subject taught in Slovak primary schools.
(b) Why we consider appropriate to have separate school subject focused on ICT and informatics, while we also support integration of ICT across curriculum.
(c) What role primary programming plays in our conception of informatics education and what forms, methods and pedagogies we consider appropriate in this context.
(d) How we proceed with the implementation of these objectives through in-service teacher development.

(e) What problems we have encountered, how this process actually evolves in our schools, how the teachers read our objectives, which objectives and corresponding skills they have mastered, how they interpret them and finally – which factors of the implementation we and the teachers perceive as risky or unfulfilled.

Brief History of Informatics at Slovak Schools

First we will familiarize the reader with the tradition of informatics education in Slovakia and various stages of the process of its implementation. We will briefly describe how the informatics education was established – from the period of experimental education at upper secondary schools in late 60s and early 70s, to the current stage of informatics as mandatory school subject for students from grade 2 (i.e. 7 to 8 year olds) up to the mid-upper secondary stage (i.e. 16 to 17 year olds).

In early 70s some of the vocational technical schools began to prepare students for operating industrial machinery via computers. In these schools some students learned basics of programming in Fortran and Cobol. Students first designed their programs using flowcharts, then they prepared corresponding punch cards which were then taken to the computer lab. Students never saw the computer themselves since it was usually located in a different institution and it took up several rooms. In the late 70s some schools built their own computer labs. In some industrial towns (where most of the vocational technical schools were located) special central computer labs were established. At that time new study programmes were launched at universities called informatics (at the beginning called cybernetics).

In 80s most of the upper secondary schools opened special classes focused on informatics. However, appropriately qualified teachers were absent. In school year 1982/83 Faculty of Mathematics and Physics of Comenius University opened a new study programme focused on upper secondary informatics teachers' pre-service education. Those students had access to the university computer EC1010 where they could write and run programs in Pascal. Soon after that several universities began to build computer labs equipped with 8-bit computers (e.g., PMD-85, Didaktik Alfa, PP-01), often using a version of Basic as a programming language

Many activities designed to attract young people to informatics emerged – in 1985 a P category of International Mathematical Olympiad started (later transformed into a stand-alone International Olympiad in Informatics). A series of articles on programming in environments like Karel and Logo were issued in the Zenit magazine targeted at secondary school students. Since 1986 a school subject "Informatics and computers" became part of the National Curriculum. Special classes focused on algorithms and programming were established at several grammar schools and vocational technical schools.

In early 90s most of the upper secondary schools taught informatics. A special computer lab with several PCs was usually dedicated to this subject, mostly taught by specialized teachers. The educators inspired by success at upper secondary school developed an experimental informatics curriculum also for lower secondary schools. For example,

Kosicka Str. Primary School in Bratislava opened a class focused on programming. Pupils aged 11 to 15 wrote game-like programs in a visual programming environment called Comenius Logo (Blaho *et al.*, 1995) and (Tomcsanyiova and Tomcsanyi, 1997). In the second half of 90s computers become more affordable and many businesses and households acquired them. In order to train students to use computers effectively, informatics in schools changed its orientation and became more user oriented – students learned how to create electronic documents (in T602, a text editor of that time), use spreadsheet editor, send and receive e-mails, navigate files and operating systems etc. In late 90s the National Curriculum was revised to incorporate five main topics – Information around us; Communication through ICT; Problem solving and algorithmic thinking; Principles of ICT; and Information society.

In 2008 a new National Curriculum for primary and secondary schools prescribed to teach informatics as a mandatory core subject from year 2 (7 to 8 year olds) to mid-upper secondary stage (16 to 17 year olds). At all school years five main topics of informatics remain the same and they cover basic digital literacy, ICT user skills, programming and core concepts of informatics, hardware and other digital technology related concepts, digital safety and other information society related concepts. At different school years the topics are taught differently – first and foremost respecting the pupils, their age and developmental stage.

There is an intense initiative in Slovakia to integrate digital technology into early years (pre-primary) education as well. Through an EU funded project teachers in early years education centres (kindergartens) are being educated to use digital technology appropriately with their children. Programmable toy Bee-Bot have been introduced, see (Pekarova, 2008) and (Kalaš, 2010).

According to our anecdotal information, programming at upper secondary level is mostly done in Delphi or Lazarus environments, with more and more schools gradually switching to Python. At lower secondary schools, Imagine Logo and Scratch are popular programming languages. At primary schools most widespread environments are Thomas the Clown, EasyLogo and several other microworlds that have been created in our department (we will present them in chapter 3).

2. Elementary Informatics, Computational Thinking and Programming

In accordance with the recent report of Informatics Europe and ACM Europe (2013) we will use the term **informatics** when we are speaking about the broad scientific field behind the digital technology. For us "informatics" is also an umbrella term that includes computing, ICT, and digital literacy – basically all concepts that have anything to do with digital technology, information or theory behind them.

An effort to distinguish various fields within school informatics is apparent in the Royal Society report (2012). However, we use these terms in a slightly different way from definitions provided there. By the term **ICT** we understand a set of user oriented skills (e.g., using a text editor, spreadsheet editor, creating graphics, animation, working with sounds …). **Digital literacy** in our context is understood as a set of basic skills that

everyone should acquire during their education in order to use digital technology (not only computers but all digital devices) effectively, safely and meaningfully to solve their everyday problems and tasks.

To understand what exactly is covered by our informatics (as a school subject) let's have a closer look at the five main topics in next chapter of our paper.

2.1. *Informatics as a School Subject in Slovakia*

Slovak National Curriculum (2011) codifies the following core school subjects rooted in informatics science (however, since 2015 both subjects are unified as Informatika):

- "**Informaticka vychova**", or Elementary Informatics in English, for school years 2 to 4 (pupils aged 7 to 10), while whole primary education consists of year 1 to 4 (i.e. pupils aged 6 to 10).
- "**Informatika**", which translates as Informatics, for school years 5 to 11 (pupils aged 10 to 17) at so called lower secondary and upper secondary schools.

For each of these subjects there is about 1 lesson per week, usually in a computer lab. Besides these dedicated subjects many ICT (and some informatics) elements are integrated across subjects as well, but that aspect will not be discussed in this paper.

Informatics as a core school subject is designed for every pupil regardless of their gender, future career or highest level of education they will reach. Great emphasis is on the age appropriateness – the content and form should always respect developmental stage of the pupils.

In all school years the five main topics of informatics remain the same, their content is always designed to fit the specific age group. National curriculum of primary informatics is presented in detail in Blaho and Salanci (2011). At primary schools the five topics cover:

- I**nformation around us** is the most comprehensive topic that includes working with text, graphics and multimedia. At primary school, pupils explore data structures – simple tables, graphs, dictionaries and mind maps.
- **Communication via ICT** – pupils work with websites relevant to their interests; they learn to use a web browser, e-mail client and chat.
- **Methods, problem solving and algorithmic thinking** – pupils learn to solve various problems and write down solutions (using words, icons or specific commands), they learn to control an agent directly and later by planning commands in advance. They learn to understand the causal connection between the program and behaviour of the agent. In this paper we will focus solely on this part of school informatics – and specifically on the elementary programming.
- **Principles of ICT** topic deals with hardware parts of the computer (keyboard, mouse, display) and external devices. Pupils also learn to work with folders and files.
- **Information society** – pupils learn about risks involved in using digital technology, about privacy and about the impact of information technology on the society.

We believe that these five topics cover the same concepts as three topics suggested in the Royal Society report (2012): digital literacy, information technology and computer science – which resulted into introducing a new compulsory subject computing in the UK since 2014.

In some other countries there is a strong initiative to include computational thinking and informatics concepts into all school subjects, see (Barr and Stephenson, 2011), instead of creating a separate dedicated subject. However, Slovak tradition of "informatics" as a school subject is a long one (including corresponding pre-service and in-service teacher development) and we believe that informatics is a distinct and important scientific field that should have a similar position in the education as mathematics or physics. Another contributing fact is that it seems to be unreasonable to demand from all the teachers to learn informatics concepts or how to incorporate computational thinking into their respective subjects – according to our experience they already struggle with integrating basic ICT elements into their teaching.

2.2. *Programming as a Component of School Informatics*

The core topic in the National Curriculum (2011, 2015) of school informatics that is most interconnected with informatics as a science is named Methods, problem solving, and algorithmic thinking. In it we expect pupils to learn how to solve various types of problems, externally represent a solution, and use such representation as an object to think with about the problem. Carefully chosen problems and well thought out pedagogy can lead directly to computational thinking development and even rather deep into elementary programming.

The term **computational thinking** was introduced and later developed by Wing, who understands it as

> *"a thought process involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent"* (Wing, 2011).

Recently, an interesting study by Selby (2013) refines the definition of computational thinking as...

> *"a focused approach to problem solving, incorporating thought process that utilizes abstraction, decomposition, algorithms, evaluation, and generalization."*

Wing and several other authors call for incorporating computational thinking into formative education of children (Wing 2008), (Lu and Fletcher, 2009), (Lee *et al.*, 2011) and (Hu, 2011). In some countries the focus is still on implementing informatics education only into secondary school, see e.g. (Hubwieser, 2012) and (Settle *et al.*, 2012). Our main interest lies in developing computation thinking "from the bottom" – i.e. form preschool and primary education. However, it is difficult to choose appropriate form

and content when it comes to this target group. According to Piaget's theory of cognitive development (1993), in accordance with Hu (2011) and also according to our own experience, most children reach the ability to work with abstractions only around their tenth year and some of them even later. It is agreed in the relevant literature that abstraction is the very essence of computational thinking. And yet we believe that supporting the development of computational thinking can productively start at the age of 5 or 6 by conducting well thought introductory steps leading to what we call elementary informatics.

Our first steps begin with direct manipulation with objects without the need to abstract or represent the process that is involved in their manipulation. While these activities may not resemble computational thinking at a first glance, we believe they are good preparation for development of higher order cognitive skills.

By **elementary programming** we understand activities in which pupils perform certain problem solving tasks of controlling an agent or planning its future behaviour – in a digital environment (programmable toy, microworld, programming environment...). We strongly believe that elementary programming is an excellent means for developing, implementing and verifying problem solving skills within the domain of computational thinking. If initiated at the primary stage of education, we also call it **primary programming**. An interesting study on connection of computational thinking and programming can be found in (Selby, 2012). There are many age-appropriate tools and environments that allow us to design meaningful and engaging elementary programming activities while respecting children's developmental stage. We believe we comply with the Blackwell's definition of programming (2002):

> *"Programming involves loss of direct manipulation as a result of abstraction over time, entities or situations. Interaction with abstractions is mediated by some representational notation."*

However, several problems arise if we want to define elementary programming activities. As we have already mentioned above – children in our target group have not yet developed their abstract thinking, and so abstracting over time itself is a problem. On the other hand, we believe that many valuable activities can be conducted before any kind of abstraction is involved. Moreover, these activities often have other features that are compatible with programming (e.g., some sort of representation is being used; planning future behaviour is expected etc.). We believe that learning to think computationally and to program one's solutions can be done gradually by doing specific activities that only slowly lead to a true abstraction, decomposition of problems and generalization of solutions.

We are aware that some authors consider programming at such an early age to be at least disputable, see (Lu and Fletcher, 2009), some regard programming as a significant form of computing but mathematical in its foundation, see (Hu, 2011). In this context we perceive elementary (or primary) programming as a tool for developing early computational thinking skills. We believe that carefully chosen tools, activities and pedagogies are an excellent way of integrating both – elementary programming and computation thinking – into primary education of all pupils. We – in accordance with Resnick (2012) – *"believe in Papert's dream of computational fluency for everyone",* that

> *"children should learn to program their own animations, games and simulations – and in the process learn important problem-solving skills and project-design strategies" and that it is necessary "to expand the conception of digital fluency to include designing and creating, not just browsing and interacting".*

However, we need to carefully build these skills gradually, always thinking about age-appropriateness.

There are several educators that promote programming as a productive and engaging activity even for very young children. They are looking for age-appropriate forms and study how do children approach various programming situations.

Mogardo *et al.*, (2006) deal with a pioneer experiment of Perlman who in 1970s designed a programming tool for preschool (and preliterate) children. The TORTIS system consisted of physical floor turtle that was controlled by logo-like commands depicted on plastic cards. Cards were placed into slots and after pushing a button they were executed by the floor turtle. Both the agent and the commands were tangible. Authors themselves admit that in the time of the described experiment there was only a little understanding of developmental psychology of a child and Perlman probably had not designed the tool in accordance with what we now would consider appropriate for such young children (the paper describes working with 3–5 year olds). However, some of her observations (analysed in 2006 by Mogardo *et al.*) are valuable even now – e.g., that children didn't manage to associate the screen commands with the movements of the turtle and even after adding the plastic cards (which were basically physical representations of screen commands for the turtle) children failed to understand that each card represents a movement of the turtle. We believe this problem is closely associated with a cognitive developmental stage of the children – at the age of 5 they definitely do not possess the ability to understand the connection between the plastic cards picturing commands and movements of the turtle on the floor. Our suggestion is to conduct different pre-programming activities that do not involve external representations (e.g. playing with Bee-Bots or even more trivial tools that involve "one command, one move" direct manipulations of the agent at a time) – and leave the programming activities involving abstractions and representations to later stages when pupils begin to develop the understanding of abstraction and external representations.

Ackermann deals with young children and their programming adventures in (2012) where she describes three aspects of programming as observed by the work with preschool children:

> *"1) making things do things (instruct them to follow and execute orders); 2) animating things (endow them with a mind of their own, teach them to look after themselves); 3) poking things (modulate how things act and interact by tweaking some parameters in their environment)."*

Ackermann admits that this is hardly a definition of programming per se and that the concept of programming is difficult, ever-changing and bearing many meanings to different people of different professions. However, she agrees that "*programming, at*

its core, is about giving instructions – or commands – to be executed by a machine". She presents several "*settings where youngsters are asked to give and execute orders, take over control*". For example – ambient programming is a new promising style that has a potential to attract many children, even those who in general do not incline to more traditional programming activities. Another take on ambient programming is described by Eisenberg (2009). On the other hand Ackermann refers to these activities as "*programming (in a weak sense)*" and she puts the word programming into quotation marks. In agreement with this approach we also distinguish our activities from *hard programming* and we will refer to them as *elementary programming* or *primary programming*. Another approach to programming, currently getting growing attention and becoming more widespread in all stages of education is physical computing and educational robotics programming, see e.g. (Przybylla, Romeike, 2014) or (Mayerova, Veselovska, 2016).

An interesting attempt at programming with primary school children is reported also by Gibson (2003). Probably the most successful initiative for programming for children is the Scratch community. Programming environment is being developed by researchers at the M.I.T. and it is continuously being improved and thoroughly studied, see Maloney *et al.* (2009) and Brennan *et al.* (2012).

3. Slovak Conception of Primary Programming

In this chapter we will present the conception of programming in Slovak primary education, based on current National Curriculum (2011, 2015) and materialised in the structure and content of the recent nation-wide professional development (PD) project for 700 primary teachers (see chapter 4). We will briefly characterise programming environments that have been used in the PD sessions and are currently being used in primary schools, what kind of pedagogies teachers apply and what are their learning objectives. We will analyse programming concepts and identify corresponding cognitive operations, which we consider appropriate for primary pupils.

In the Slovak approach to primary programming we can identify three domains with several sub-domains (with several overlaps and without any strictly predefined order of implementation, although with numerous dependencies in developing programming concepts and operations):

1) *Solving problems and handling solutions.*
2) *Controlling an agent:*
   - *Direct control of an agent.*
   - *Indirect control (building and handling future behaviours).*
   - *Some advanced concepts of primary programming (e.g. parameters, loops and procedures).*
3) *Tinkering with interactive environments:*
   - *Multiple agents and their properties.*
   - *Static scenarios.*
   - *Dynamic scenarios.*

3.1. *Solving Problems and Handling Solutions*

One of the main learning goals of primary informatics is to learn how to solve problems, and represent, evaluate, verify and reflect on their solutions. We consider concepts and practices in this domain to be exceptionally productive and developmentally appropriate. They can naturally contribute to all topics of primary informatics – including elementary programming.

In informatics education we focus especially on the procedure of problem solving which leads from the initial to the final state (the solution) while keeping given rules. Pupils probably do not perceive the procedure as the most important part of solving problems, but from the perspective of informatics education it is the core of the problem solving – the product (drawing the house, cannibals and missionaries transported to the opposite bank of the river, getting to the target square of the 'snake-and-ladder' game) is only a means of motivation. Therefore we choose problems that have interesting solving procedure (method or steps). We should always focus on the procedure of finding the solution and on its externalized representation (see Fig. 1). We should not neglect to verify if pupils are able to execute, communicate, analyse, evaluate and modify the discovered method of solution.

When designing lesson plans dealing with problem solving, it is important to choose both appropriate problems and learning activities to be conducted during the lesson. For some problems there exist supporting digital environments that enable pupils to solve them through direct interaction and visualization. This hands-on approach to solving problems supports experiments, iterative solutions, repeating solutions and trying out different solutions.

For example: the well-known puzzle about transporting the wolf, the goat and a cabbage across the river using one boat is an ideal problem for implementing via a software environment (Fig. 2 left). By clicking objects pupils experiment with transporting them.

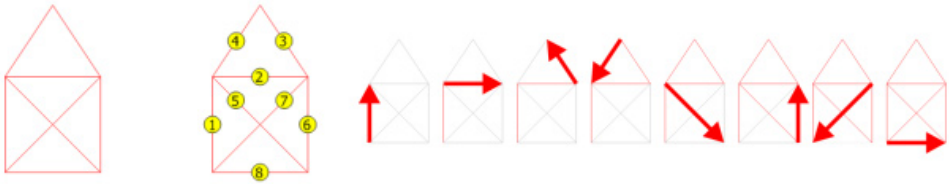


Fig 1. The first image is the required outcome – a one stroked drawing. The second and the third images (the third one being in fact a sequence of images) are possible notations of the procedure of how to solve it. Both solutions demonstrate how a solving procedure can be noted.

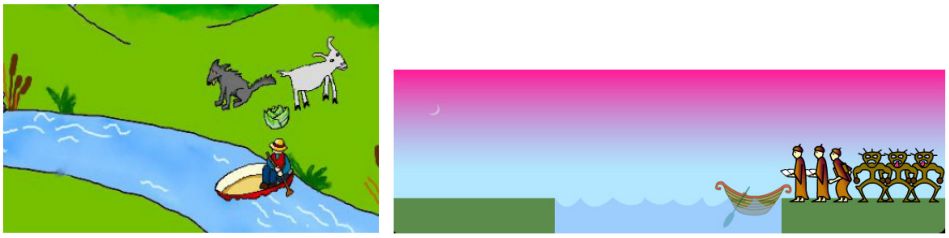


Fig. 2. On the left the Wolf, Goat and Cabbage puzzle environment. Right: screenshot from a similar puzzle with missionaries and cannibals, see `http://game-game.sk/18394/`.

Usually they solve the puzzle by trial-and-error method. Though, when they are asked how they have done it, they are motivated to reproduce and describe the solution and some of them are easily prompted also to put down the sequence of steps for transporting all items successfully.

If the environment is well designed it makes it easy to proceed from (1) **solving** the problem, through (2) **experimenting** with the solution procedure to the (3) **representing/recording** the procedure for future repeated solving of the same problem (maybe even without its immediate execution). These three steps in fact describe the advancement from solving problems to programming.

These are computational **cognitive operations** that are involved while solving problems and handling their solutions:

- Discuss and think about the core of a problem, about the relevant information provided by the problem assignment, about the conditions of solvability, about an appropriate procedure that will find a solution, about the difficulty level of the given problem, to look for similar problems that will help us to solve the problem.
- Use different strategies for finding the solution – like drawing a diagram, listing all combinations, guess-and-confirm, divide problem into smaller parts, find a similar problem, find a repeating pattern, look for the solution form the end etc., see Polya (1957).
- Explain the solution to someone else, to teach a friend how to solve it (verbally, by non-verbal means, using a specific language).
- Learn from someone else how to solve the problem (using verbal or non-verbal communication).
- Write down the solution (by a picture, or series of pictures, using icons, text, video or audio).
- Reason about the language and the form of notation of the solution in order to make it eligible for others.
- Execute the solution and verify its validity, correct wrong steps of the solution.
- Review certain properties of the solution (its eligibility, length, 'price'…), assess and compare it with several different solutions.
- Look for different solutions of the same problem.
- Reason about the non-existence of the solution.

3.1.1. *Activities and Examples*

An interactive microworld inspired by a task from the Bebras contest (see e.g. Dagienė and Stupurienė, 2016) enables pupils to experiment with sorting a group of children according to their heights (Fig. 3). It is possible to switch two children by clicking on the first one then the second one. This microworld also records the steps of the solution into a text file. A teacher or a researcher can use it to find out what strategy pupils used – if they all solved the problem similarly or if they applied different strategies – systematic or more random.

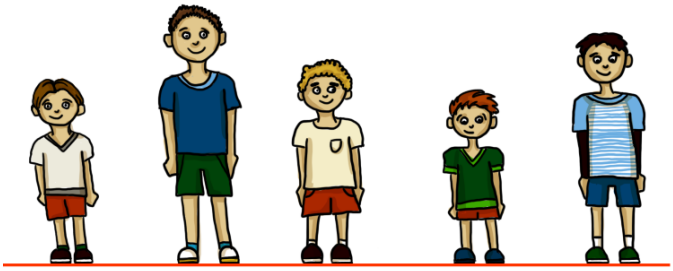


Fig. 3. Interactive application for a problem solving task.

3.1.2. *Pedagogy – Observations and Recommendations*

The specific organization of the lesson is up to the teacher – she is responsible for choosing the problems and selecting the activities according to the learning goals of the topic. Teacher can use many ready-to-use applications, microworlds and pre-made lesson plans. Many problems can be solved without the computer. Many tasks from the Bebras contest are suitable and they are available through the Bebras portal. The activities should be built around a direct manipulation with physical objects; or if they are implemented via some software application they should use appropriate pictures (dice, beads, building blocks, animals, persons …). It is crucial to motivate the pupils to actively think about the solution method and not to focus only on the end product – i.e. they should realize the difference between the procedure of drawing a house by one stroke and the resulting drawing where they can see no longer how it was done.

3.2. *Controlling an Agent*

Second domain of primary programming deals with two important and crucial concepts of pre-programming activities – direct and non-direct control of an agent. The first one is represented by a set of activities and suitable software applications that allow pupils to command an agent (a toy, another child, an on-screen character or animal etc.) to do something – mostly, to move to a given location. Each command is immediately executed and a result can be observed. Non-direct control of an agent gets pupils into real programming – they are asked to construct a sequence of commands in advance, which is then executed.

3.2.1. *Direct Control of an Agent*

Direct control of an agent can take place in a physical world where the teacher conducts an activity during which pupils give other pupils certain pre-defined commands (e.g. turn left, walk) to solve a given task (e.g. guide your friend from the desk to the door). Sometimes the commanded child can be replaced by a toy that is moved by hand according to the commands. There are also toys that can be controlled by a remote control, or digital toys with control buttons placed directly on them. Ambient programming

can also have similar characteristics, see (Eisenberg, 2009). In a software application a pupil controls a virtual agent. It is crucial for such microworlds to maintain age-appropriateness – using child-friendly graphics, presenting an engaging agent that a child can identify with or that can be perceived as a hero protagonist. In both cases (physical and virtual) the agent can execute only small set of well-defined basic commands – *make a step*, *turn left*, *turn right*, *play a sound*, *take an object*, *pick a colour*, *set a pen size*, *turn pen down* etc. In many software applications the virtual agent can be controlled directly but also by programming, i.e. without immediate execution of the command (see the next chapter 3.2.2.).

The most basic task for an agent is to move from one place to another. This task is often motivating enough and pupils are willing to carry it out and think about the sequence of commands to accomplish this goal. They gradually realise that:

- Only a limited set of commands is available to use in the solution (in a physical environment the teacher determines them, in a microworld they are usually set by the application itself).
- The current state of the agent is always represented by its visible attributes: rotation, position, pen colour etc.
- The execution of each command has a very concrete, specific and unambiguous effect on the agent and/or on the whole scene where it acts.

We choose the agents so that pupils are familiar with them and the activities they perform are more or less grounded in their reality (e.g. a bee flies to the flower, an ant moves objects) or at least actions of the agent should be believable (e.g. a turtle moves around and draws a line with its tail). Most agents therefore are animals, vehicles or human characters.

Many cognitive tasks listed in part 3.1. can be practised using activities mentioned in this chapter – by directly controlling the agent pupils can reason about the procedure of finding the solution, they can explain their solution to a friend, they can review specific properties of a given solution and it's correctness, or think about possible notation of the solution.

Activities and examples

Each of the software applications that will be presented in this chapter has its own specific features. They use different agents and different control interface, some of them record a sequence of commands. If the sequence of the steps can be recorded, we should consider its level of abstraction – the commands could be e.g. coloured pieces of paths (Thomas the Clown) or arrows that guide the agent (World of the Ant, Bee Tasks). Another significant difference in various microworlds and applications is whether the agent moves in a rectangular grid (Ice Cubes, Bee Tasks, World of the Ant, EasyLogo, Baltie), in a graph (Thomas the Clown) or with no visible constraints (Scratch). Rotation mode is closely related to the movement and the grid type – the agent can rotate either relatively or absolutely. Relative rotation means that the agent turns depends only on its previous heading; this is most common in open complex environments (Baltie, EasyLogo, Scratch). Absolute rotation is common in simpler applications where the agent moves in rectangular grid, usually only in four possible directions.

The agent in **Thomas the Clown** application is the clown on the bicycle. He moves in the graph-like network of roads. The child controls him by clicking the blue, yellow or red road piece in the right centre of the screen (see Fig. 4 left). The commands are executed immediately but the sequence is also recorded at the bottom of the screen. The task is to get Thomas from one place on the map to another.

In the **Ice Cubes** microworld the robot pushes the ice cubes (Fig. 4 right). It is controlled by the keyboard keys and the task is to move all ice cubes to their designated places. The sequence of moves is not recorded. Many similar microworlds are available on the web, though they are often perceived as games without educational dimension. Both applications check if the solution is correct.

The **World of the Ant** application features an Ant as the agent (see Fig. 5 left). The goal is to guide it through the maze to the door. Pink flower and blue star enable the Ant to walk through colourful walls. The Ant is controlled by the keyboard keys. The sequence is not recorded.

In the **Bee Tasks** the agent is a blue insect controlled by clicking the buttons with arrows (see Fig. 5 right). The goal is to guide it to the flower. The sequence of the commands is recorded on the bottom of the screen. Both microworlds verify whether the solution is correct.

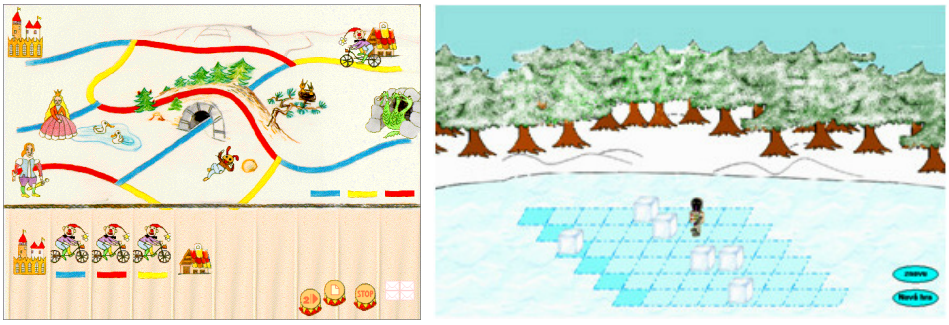


Fig. 4. On the left Thomas the Clown application, on the right an Ice Cubes microworld.

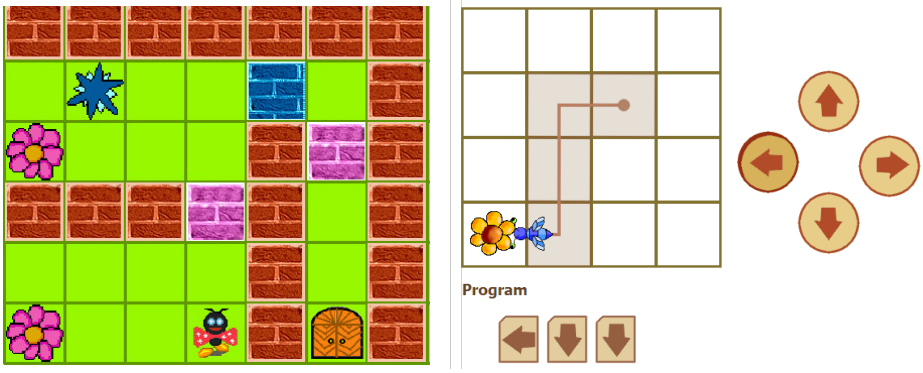


Fig. 5. On the left the World of the Ant application, on the right the Bee Tasks.

EasyLogo is open environment in which it is possible to change the appearance of the agent (in this case a frog). The agent is controlled by three buttons in the top right corner (Fig. 6 left). Forward arrow moves it one step forward on the grid – the agent moves along the grid lines, not from centre of the square to the next square. The left and right arrows turn the agent relatively to its current position. The goal is to guide the frog to the pond. This application does not check whether the solution is correct, nor does it record the sequence of commands in the direct mode.

Baltie is another open environment (Fig. 6 right). The grid where the agent (a sorcerer named Baltie) moves is not visible. Similarly to EasyLogo there are three buttons for the movement – first turns the agent relatively to the left, second moves Baltie one step forward and the third turns him relatively to the right. Baltie can conjure pictures – a child can choose pictures form the huge pre-prepared set. The picture will appear in front of Baltie and it is also possible to construct more complex images consisting of many smaller pictures. There is no in-built control of the correctness and the application does not record the sequence of commands in this mode.

Some of these applications allow creating and adding custom tasks for pupils – World of the Ant and EasyLogo. In World of the Ant we cannot choose a different agent or change the final goal of tasks, but we can design the maze and place object on different positions. EasyLogo is more opened – it allows to change the agent to any picture, set different backgrounds and completely rephrase the goal of the task (e.g. instead of guiding a frog to the pond we can ask pupils to move the frog along a square shape). We consider this an important feature – the teacher can design her own tasks which are better suited for the pupils and their skills, or match the motivation for the specific lesson. However, designing new tasks, creating custom pictures and related technical obstacles put a lot of demands on the teacher.

Open programming environments, such as **Baltie**, **Scratch** or **EasyLogo** can be used for the direct agent-controlling activities as well. However, a meaningful task has to be designed (or programmed) by the teacher first. There is no in-built solution checking and if the teacher needs such feature she has to virtually create the microworld to achieve this. A pre-made and partially programmed activity e.g. in Scratch can simulate desired

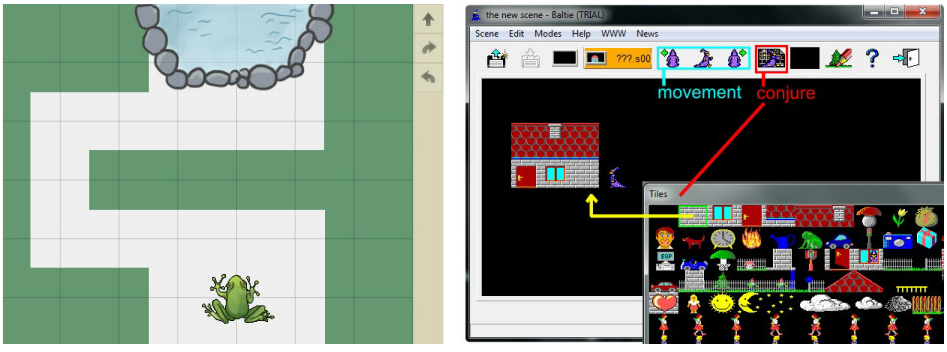


Fig. 6. On the left is EasyLogo, on the right is Baltie.

features of direct agent control. A path for the agent is in the following example a part of the backdrop (Fig. 7 right). The cat is controlled by keyboard arrows – this behaviour was programmed by the teacher in advance. Teacher also included a script that checks whether the cat is at the end of the path or whether it deviated from the path earlier. Similar assignments can be added to EasyLogo (Fig. 7 left), but it is not possible to add automatic solution checking.

Pedagogy – observations and recommendations

A teacher conducting these or similar activities must remember that the learning goal is to build basic understanding of controlling an agent by specific commands. The pupils should realize there is a causal connection between the commands and the behaviour of the agent. Since the control is direct and each command is immediately carried out, making this connection is possible for pupils before reaching formal operational stage of their cognitive development. In each application we presented the behaviour of the agent is visualized. This allows pupils to immediately see how they are progressing in the solution. We recommend using activities or environments that automatically check if the solution is correct. According to our experience, pupils are more motivated to solve problems if they have immediate feedback on their success. In our approach we always use direct control of an agent as an introductory activity to the very basics of elementary programming.

3.2.2. *Indirect Control of an Agent – Building and Handling Future Behaviours*

In the previous chapter we described activities in which the agent immediately carried out each command. Next step may naturally be focusing on planning the whole sequence of commands which will be executed only once it is complete. We call this approach an indirect control of an agent. Here again we can control either a physical agent (a classmate, a toy, or special programmable digital toy such as a Bee-Bot that is designed for that purpose) or a virtual one "living" in a software application on the screen. When working with physical agents pupils can write down the sequence of their commands on the paper, or draw it using pictures (an interesting activity by itself is to design the proper notation and discuss what 'proper' means in this context). E.g. programmable Bee-Bot

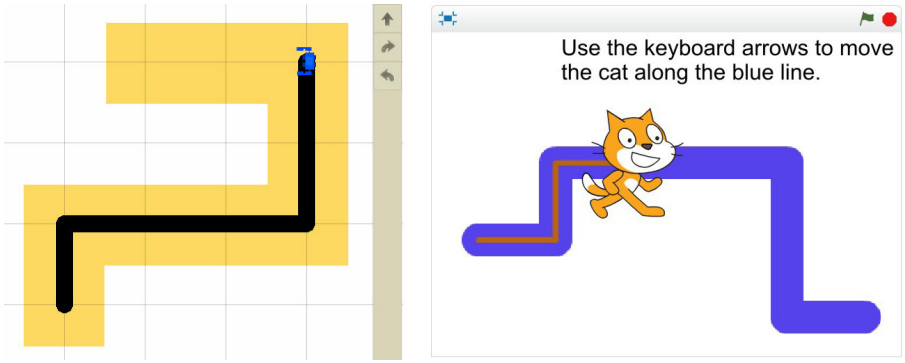


Fig. 7. Similar assignments in EasyLogo and Scratch.

does not display the sequence at all – it is entered by pressing the buttons atop the toy, but the child has to remember it or observe it when the toy moves according to the commands. When using software applications, notation is usually given by its designers – some applications use icons with arrows, icons combined with text or different kind of pictures (e.g. colour of the road in Thomas the Clown).

Indirect agent control assignments usually have the same goal as activities described in the previous part – to guide the agent from one place to another. Again it is possible to include also other actions e.g., picking and using items or avoiding obstacles. Since the sequence of commands is explicitly recorded – and thus visualized and editable – pupils can finish the sequence or add missing commands, or even correct the sequence, i.e. work with the representation. We can classify the cognitive operations according to what has to be done with the sequence of commands:

- Construct the sequence that guides the agent from its initial position to a final required position according to the assignment.
- Interpret a given sequence of commands (there are various ways of how to verify the interpretation, the most straightforward is when pupils move the agent according to the given sequence e.g., by clicking on the grid squares).
- Identify the final position of the agent after executing the commands.
- Identify the correct sequence among several sequences (more advanced version is to identify an incorrect sequence among several correct ones).
- Complete the sequence if the last step is missing, or two last steps are missing, or any step is missing.
- Identify and correct an incorrect command within the sequence.
- Find alternative solution, find a solution with specific properties (e.g. the path is the shortest possible, or on its path the agent will cross equal count of yellow and blue squares etc.).

Activities and examples

While most activities described above are suitable for implementation in a virtual microworld, it would probably be unreasonable to include all possible types of assignments into one environment, thus getting too complex or too much time consuming for primary pupils. Therefore several different applications are being used in our primary schools that focus on specific tasks or certain groups of tasks.

Indirect controlling of an agent in **Thomas the Clown** is implemented e.g. in the strawberry picking task (Fig. 8 left): the robot is waiting at the entrance to the garden, once a player completes a sequence of commands for moving and picking the strawberries, the robot will execute it. The goal is to pick all ripe red strawberries in the garden. Sequence is created by clicking the icons in the left part of the screen and it is recorded on the panel above the stage. When the sequence is being executed the active command is always highlighted. This microworld automatically generates different gardens of 2 by 3 grid squares.

In the **World of the Ant** (Fig. 8 right) we can also choose indirect control mode. At the bottom of the stage there is a set of commands – four arrows are for absolute rotation

and icon with legs represents moving one step in to the direction set previously. The Ant will carry out the sequence only after the child pushes the red button. The application is open to design many different mazes.

The **Bee Tasks** microworld was designed to offer several possible types of tasks we mentioned earlier. In its current version there are nine types – the first one was already described in part 3.2.1. as it is a direct control of the agent. The others are: interpreting a sequence of commands, constructing a sequence (the Bee has to get to the square with the flower, see Fig. 9 left), placing the flower on the square where the Bee will end up after completing the given sequence (again interpreting the sequence), adding a missing command (last one, last two, any in the middle), constructing a sequence (the Bee has to end in the square with the flower, but there are obstacles as well), and identifying a right sequence among several given (Fig. 9 right). A sequence is constructed by clicking on the icons with arrows – based on the same principle as in Thomas the Clown microworld.

Fig. 8. On the left Thomas the Clown, on the right World of the Ant.

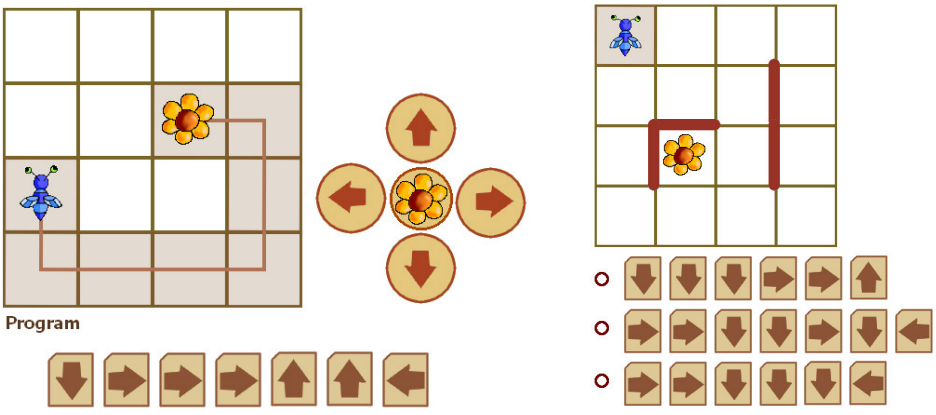


Fig. 9. Two different assignments in the Bee Tasks microworld.

EasyLogo offers a mode in which children plan and construct the sequence of commands (Fig. 10 left). However, this application behaves rather differently – it executes the commands immediately after they are dragged into the sequence – it doesn't wait for completing the sequence (commands are in the column at the right side of screen). In this sense the agent is directly controlled. However, there is a button "Run again" which re-executes the sequence after it is created.

If we want to use **Scratch** for this kind of activities, we first need to prepare a project – create a suitable backdrop, create sprite(s) and build the scripts for all functions, including script(s) to verify a solution (if we want to). An example below (Fig. 10 right) is an activity in which the goal is to build a sequence of commands for the Beatle to move from its green (start) square to another green (goal) square without even touching any other non-white square. The backdrop is a grid of white and coloured squares. In the Beatle's scripts area there are four blocks prepared for the pupils – already with their inputs properly set (move to the centre of a neighbouring square and turn left or right 90). Pupils will construct the whole solution (script) for that situation by duplicating and snapping the blocks into one script. This task has many variants of different levels of difficulty.

In both of these examples the task could be to fill in one or more missing commands into the incomplete sequence (solution). Some of the environments presented so far offer an option for the pupils or for the teacher to add their own tasks of the same kind. Scratch, World of the Ant and EasyLogo allow us to do so.

Similar tasks (where the goal is to guide the agent to a given goal) are used also in the **Bebras** contest. Since 2010 primary pupils can be involved in the contest in a special category specially designed for them. One task was inspired by Thomas the Clown (Fig. 11 left) – the farmer has to get to his cow, but on his way he needs to grab the bucket. All possible paths are depicted as a graph with edges of different colours. Pupils are prompted to select the track which meets the criteria.

Second example from the Bebras contest is quite difficult task that proved to be problematic as only 20 % of pupils solved it correctly, 17 % didn't answer at all. The story is: "A mouse is roaming in the maze, until it eventually reaches the cheese. Philip was observing the mouse and used small cards with arrows to record its movements. Unfortunately he dropped the cards and only two of them stayed at their places (see the

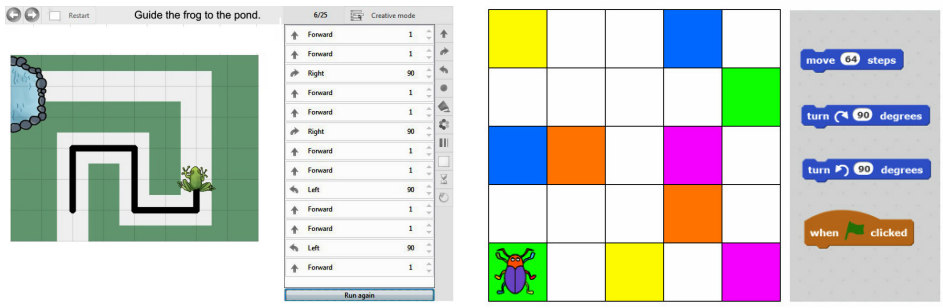


Fig. 10. EasyLogo and Scratch.

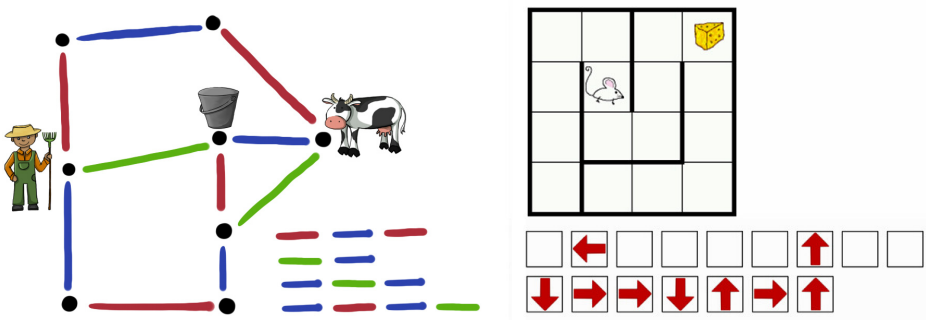


Fig. 11. Two Bebras tasks based on indirect control of an agent.

upper row of squares). Place the remaining cards and restore Philip's record." The task was interactive and pupils could drag the cards into empty slots using mouse.

The last described assignment illustrates that this kind of activity can be really difficult and can also be given to much older pupils. The variety of presented tasks show that guiding the agent is very rich context with many possible activities and a lot of potential. At the same time it is apparent that planning a sequence beforehand and executing it only after it's recorded is a programming-like activity that involves abstraction over time. Also the specific notation and execution of the sequence by some automatic machine-like agent is a feature of full-flagged programming activity. However, these tasks are still set in a concrete situations and their solutions do not require pupils to design universal solutions that involve this kind of abstraction.

Pedagogy – observations and recommendations

It proved to be crucial that the application itself verifies whether the solution is correct. If the microworld offers several tasks or several levels of difficulty, pupils should not be allowed to skip them freely. Most motivating environments have a game-like design presenting a bit more difficult task in each level. The designers of the microworld should always prepare a set of tasks to be solved by pupils. They should be ordered according to their cognitive demands, they should be motivating and engaging, prompting pupils to learn new concepts and challenging to engage more demanding (but still developmentally appropriate) cognitive operations. It is useful if the designer prepares several sets of tasks as they can be used for achieving different learning objectives, in different classes, for pupils at various stages of the learning process. Interesting option is to allow teachers to create their own tasks, however this approach has proven to be far too optimistic as only a small fraction of teachers are ready to do so.

3.2.3. *Classification of the Microworlds Used for Direct and Indirect Control of a Virtual Agent*

In part 3.2 we have presented several applications, microworlds and environments that are suitable for solving problems and learning computational thinking via programming-like activities. They are all suitable for primary school pupils as such or after certain preparatory steps. We summarize their features in Table 1.

Table 1
Features of microworlds

Movement commands	keyboard keys	World of the Ant, Ice Cubes
	icons with arrows	Bee Tasks, EasyLogo
	icons with agent image	Baltie
	icons with colours	Thomas the Clown
	cards with text	Scratch
Agent rotation style	without rotation	Thomas the Clown
	absolute rotation	World of the Ant, Ice Cubes, Bee Tasks
	relative rotation	EasyLogo, Baltie, Scratch
Grid type	graph	Thomas the Clown
	rectangles or squares	Thomas the Clown, World of the Ant, Ice Cubes, Bee Tasks, Baltie
	rectangular – lines	EasyLogo
	free movement (coordinates)	Scratch
Notation (in direct control mode)	without notation	World of the Ant, Ice Cubes, EasyLogo, Baltie, Scratch
	automatic notation	Thomas the Clown, Bee Tasks
Solution verification	no verification	EasyLogo, Baltie, Scratch
	automatic verification	Thomas the Clown, World of the Ant, Ice Cubes, Bee Tasks
Agent actions	only movement and/or rotation	Thomas the Clown, World of the Ant, Bee Tasks, EasyLogo, Baltie
	collecting objects	Thomas the Clown, World of the Ant
	moving objects	World of the Ant, Ice Cubes
	using objects	World of the Ant
	other	Baltie, Scratch
Goals	arrive at destination	Thomas the Clown, World of the Ant, Ice Cubes, Bee Tasks, EasyLogo
	other	EasyLogo, Baltie, Scratch
Pre-made activities	no in-built activities	Baltie, Scratch
	set of fixed inbuilt activities	Thomas the Clown, Ice Cubes, Bee Tasks
	set of activities provided, custom ones may be added	World of the Ant, EasyLogo

3.2.4. *Some Advanced Concepts of Elementary Programming*

In the previous two parts we focused on basic concepts that are in our opinion and according to our experience suitable and appropriate for all primary school pupils. Now we will present several others – more advance concepts – which still could fit into upper end of the primary programming, but probably not for the whole class and only with well experienced teacher.

Parameters

In some applications parameters in existing commands are rather intuitive and easy to use (e.g. in Scratch, Fig. 12 left). In this case there is no need to address this concept ex-

plicitly – children will understand immediately how to use them. Rather intuitive way to set the parameter is using a drop-down menu – in Scratch there are several – e.g. choosing a sound which will play by the "play sound" command. In this case pupils cannot make a mistake. Another child-friendly parameter is pen colour chosen from the palette (see EasyLogo displayed on Fig. 12 right). Parameters are present also in some modes of World of the Ant and in Baltie environment.

Loops

Several of described microworlds provide loops – Scratch, EasyLogo, Baltie. However, they are usually present in more complex open programming environments. Led by our department a small microworld focused on loop constructions was designed and developed (Fig. 13). In this application pupils control the Jumper who has to reach the door by jumping over platforms. The microworld is designed as a game – there are 24 levels in which the child solves more and more complex situations (it is also possible to design and add custom levels). Eventually the space for the commands becomes limited and pupils cannot solve the problem without using a repeat loop. This design proved to be highly motivating and pupils are deeply keen on completing the "game". On the other hand, one of the teachers using this microworld reported that only about a half of pupils aged 8 to 9 years were able to learn to use the loop themselves. Loops appear also in the LEGO WeDo programming language that is designed for primary schools. In this case,

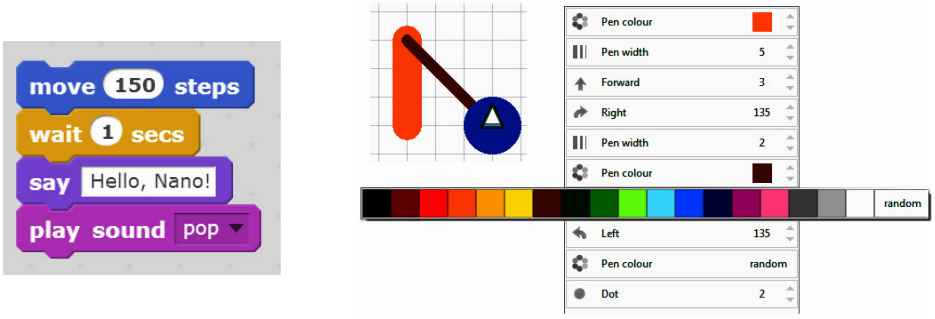


Fig. 12. Parameters used in Scratch and EasyLogo.

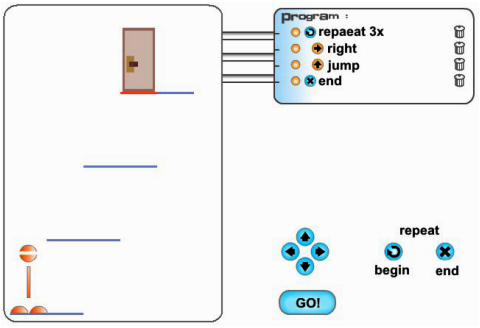


Fig. 13. A game-like microworld named Jumper is focused solely on loop constructions.

however, some commands implicitly contain repeated behaviour – turning on the motor means it will move until the program is stopped or some other action is assigned to it. As our research team reported pupils use the loops block rather easily and they intuitively understand their use in the programs they create for their robotic models.

Procedures

Some programming environments for primary pupils do not offer procedures (Scratch 1.4), others are designed to use them (EasyLogo, Scratch 2.0). According to our experience this concept is rather complex and is a good candidate to postpone to years 5 and 6. Here is an example of two procedures (Fig. 14 left) in EasyLogo. Pupils at first do not design them, as these loops are already prepared in the activity; she is prompted to use them in the program.

3.3. *Tinkering with Interactive Environments*

A programming environment named Living Pictures (influenced by Russian PervoLogo) has been specifically designed in our department to teach pupils some object-oriented concepts within elementary programming. In this environment pupils populate the virtual world (represented by a background) with moving objects – characters, animals, vehicles, plants or anything they choose from a pre-made set of pictures or draw them themselves. From the perspective of primary informatics pupils learn to control one or more objects, define their behaviours, clone them, set their properties and reactions to events.

Each object is at first depicted as a Logo turtle – the child should realise that this is in fact an abstract object that can take any form. Each object has different properties, its shape and position among them. Basic action of the object is its reaction to the onClick event (e.g. it can move few steps forward). Other events are triggered when the project is set to run and when objects collide, but we recommend to program these events later – with lower secondary school students, or with only some high achievers at the end of year 4. This programming environment is open – there are no pre-set activities. All

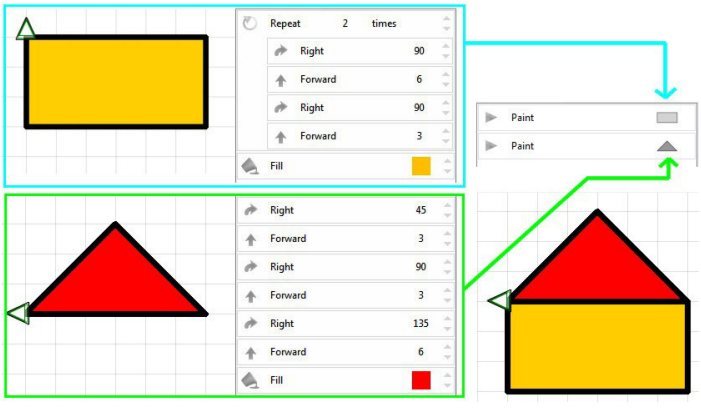


Fig. 14. EasyLogo procedures for drawing a yellow rectangle and red triangle are used to draw house.

assignments have to be designed and presented by the teacher. It is possible to add custom backgrounds and pictures. The teacher can adjust also the set of commands for the objects so that the pupils see only a limited sub set (Fig. 15). We consider this high level of customizability to be important especially if the application is designed for primary pupils. Setting up the user environment so that it is as simple as possible is crucial for the introductory lessons.

Another advantage of this application is that it is possible to export a project as an executable file (EXE). Thus pupils can be motivated to create moving pictures for their younger classmates (in accordance with Papert's principles of constructionist learning). Pupils can present the executable file to their friends or relatives.

In next parts we will illustrate several activities that can be done in this environment. We will focus on shape, position and rotation of the object, and we will use three events – onClick, onRunProject and onCollision. We believe that the outlined sequence of activities leads from designing a scene and setting the properties of objects to executing dynamic scenarios with multiple objects with different behaviour (which involves abstraction over time and over situation as well, see Blackwell, 2002).

3.3.1. *Multiple Agents and their Properties*

In parts 3.1 and 3.2 pupils controlled only a single agent. Using Living Pictures (or similar microworlds) it is possible to introduce multiple agents with different or identical properties. We prefer to tinker with properties that are visible – shapes, positions, and rotations. First, pupils should encounter objects with shapes that enable to see its rotation (character, animal …) later they will learn that for some shapes rotations are not observable (snowflake, sun). A good metaphor for describing such activity is a theatre – there are several actors on the stage, each of them has their own specific scenario and eventually they interact. This description helps with distinguishing the preparation phase (setting the properties, preparing sequences of commands) and the execution phase (the objects carry out their instructions).

Activities and examples

A good introductory activity is populating the world – pupils choose the background (green hills and sky) and place several objects on it, then change shapes of these objects

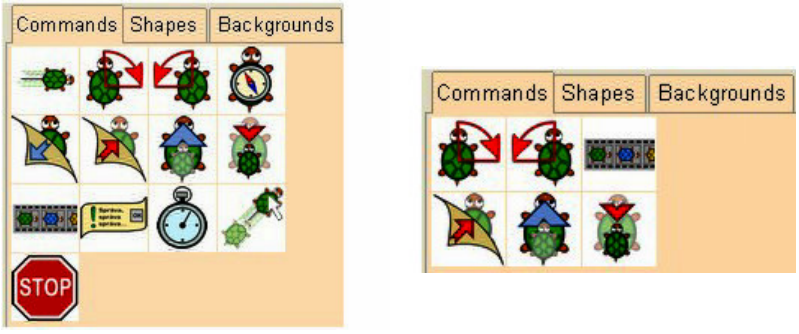


Fig. 15. Whole set of all commands (left) and limited set designed for a specific activity (right).

so they look like sun, clouds and trees. We can ask pupils to shrink or grow the objects according to their positions in the background so that an illusion of depth is created (Fig. 16 left). The features used here are: adding objects, changing their shapes, scaling down and scaling up and cloning objects. This activity can be done in many different settings – for example in the outer space (see Fig. 16 right). Rotation of the object can be also used in static scenarios (the astronaut is looking towards the aliens).

3.3.2. *Static Scenarios*

Clicking on or touching objects in the screen is nowadays the most intuitive way of interaction with the digital devices. This trend was set with the Windows interface and now is reinforced with touch screen technology. Objects in Living Pictures have a preset onClick even that is triggered if the object is clicked by a computer mouse. Pupils are already familiar with this event and assigning a reaction to the object when it is triggered is the next step.

Activities and examples

In Living Pictures each object has its own event window into which the commands for the object are dragged from the command palette. When designing static scenarios pupils will change the shape, size and rotation of the objects. The most straight forward activity is changing costumes. First the background is chosen, then objects are placed. For each object that is a piece of costume the pupils will set a behaviour – when it is clicked its shape will change to the next one form the chosen set of shapes.

As an example let us select a winter background with a snowman. Objects that will change their shapes with a mouse click are the hat, the broom, his face and buttons (Fig. 17 left). Similar projects are easily done in Scratch. The sprites have when clicked event and a single next costume block rotates a set of prepared shapes for the sprite. In our example (Fig. 17 right) three sprites can be clicked – clown's hat, eyes and mouth. Each object has the same and very simple script – switch the costume to the next one.

The greatest disadvantage of Living Pictures is that pupils can not immediately test the script and see what happens (they have to close the event window first and then run the project) – in Scratch it is possible. In Living Pictures it is also not possible to see the

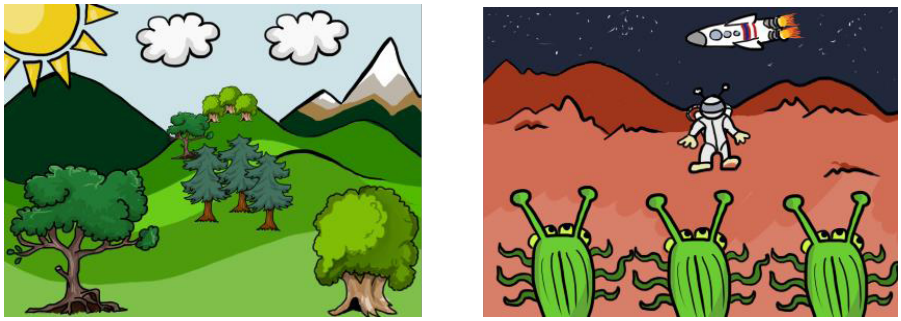


Fig. 16. Two static scenarios done in Living Pictures.

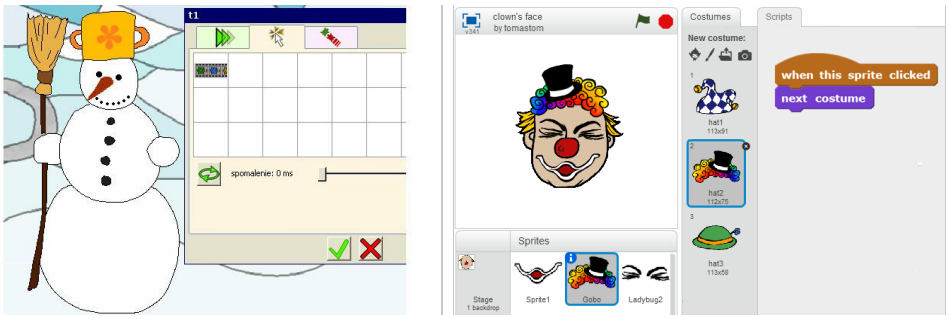


Fig. 17. Left: The "t1" window displays the onClick event with one command – change shape to the next one – which looks like a filmstrip. On the right similar project built in Scratch.

set of shapes for the object or which one comes next. In both environments we should encourage pupils to design the desired behaviour first for one object and only once it is tested and it works properly they clone it. This is done in accordance with object-oriented approach in programming where the programmer first designs the prototype object and its methods. Only after that is it reasonable to create inherited classes and objects with modified behaviour. Primary school pupils can learn to distinguish different objects with different properties and behaviour, or to tell what objects have in common. We believe this level of abstraction is appropriate for the primary pupils in the highest years (10–11 year olds).

3.3.3. *Dynamic Scenarios*

Dynamic scenarios in Living Pictures environment involve assigning a motion to the objects. Most common is setting an infinite loop for the motion, which is done by checking one of the options in the script (note that loop is not provided here as a programming structure). Throughout these activities pupils better understand the difference between preparation of the scene and running a project.

Activities and examples

In the Pond project pupils are prompted to set a background that will represent the pond. They place an object and change its shape so it looks like a fish. Then they set its direction – on our picture (Fig. 18 left) it will face right. In the event window they will command the fish to move forever forward. Default behaviour in Living Pictures is: if the fish is on the right edge of the screen it does not stop to move but it reappears on the left edge – objects do not bounce by default. This is a deliberate design choice that enables us to have an object which is forever moving to the right on a finite screen. After testing the fish's behaviour pupils clone it. Now they can change direction for some of them, or add some commands to onClick event (e.g. the fish disappears).

This is one possible set of activities in Living Pictures:

- Setting properties (shape, position, scale, direction) and cloning objects.
- One or more objects react to the onClick event.

- One object moves and reacts to the keyboard arrows; navigating this object is similar to direct agent control activities described in 3.2.
- Infinite movements of one or more objects; only one command is given to objects (usually move forward) and it is set to repeat forever.
- Infinite random movement of one or more objects on the scene, e.g. a butterfly is flying on a meadow, or Thomas the Clown is cycling on the plaza.
- Infinite (random) movement of one or more objects on the scene and their reaction to onClick event. This activity is on the edge of game design – pupils can prepare a scene where objects move randomly, when they are clicked they disappear. Aim of the game is to hit all the objects.
- Infinite (random) movement of one or more objects on the scene and their reaction to onCollision event. These kinds of activities are probably too complex for primary pupils, but if they are familiar with all previously listed concepts, they may be able to do them. An example: one object is a basket that reacts to the arrow keys, other objects in the scene are apples that are falling down (they move forever downwards), when the apple hits the basket it disappears. More complicated scenarios can be devised, but we believe there is too much abstraction involved and we do not consider this type of activity to be age-appropriate at Slovak primary level (consisting of only four years up to 10 years old pupils).

3.3.4. *Pedagogy – Observations and Recommendations*

We believe that tinkering objects, their properties and behaviours is an excellent opportunity for the primary pupils to learn the very basics of the object-oriented approach to programming. Activities in Scratch or Living Pictures are very intuitive. Pupils learn to change and set properties of objects, to distinguish the development phase form the running phase, to plan the future behaviour of objects, incorporate looping actions of objects and even begin to tinker with random values. It is crucial that these environments contain a large set of pre-made graphics and they should be opened to adding custom pictures. We believe that properly designed environment for tinkering with objects should:

- Allow to add object easily.
- Make changes in object properties (like shape, size or position) immediately visible.

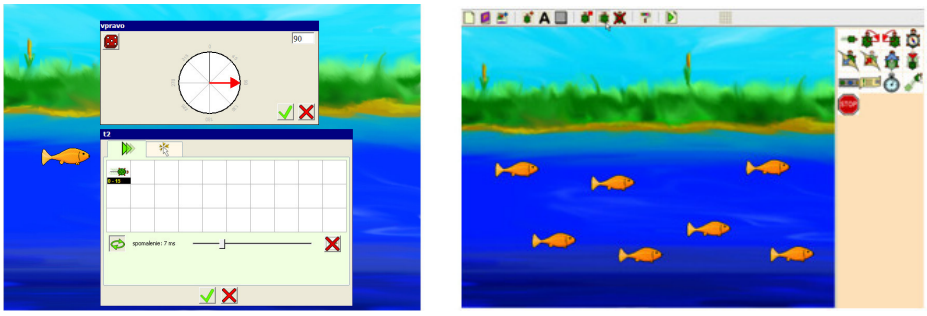


Fig. 18. The Pond project created in Living Pictures.

- Enable the object to react to at least three events: onClick, onRunProject, oOnCollision.
- Feature easy pupils-friendly manipulation with objects and their properties – by clicking and dragging.
- Enable cloning objects together with their behaviours.

3.4. *Implementation of Elementary Programming: Various Ways and Various Tools*

It is apparent from activities described in parts 3.1, 3.2 and 3.3 that core of primary programming can be learned using specialized software applications or microworlds that are (a) specifically designed for primary pupils and (b) designed to address the learning goals we have listed earlier. There are many similar applications being created around the world: Scratch (and its newer version Scratch 2.0), Scratch Jr., Microworlds JR, LEGO WeDo or Baltie. Several powerful microworlds have been developed in our department and made available for teachers and their pupils through various portals, websites, projects and PD sessions. Those include Thomas The Clown, World of the Ant, EasyLogo, Living Pictures, Jumper and Bee Tasks. Ice Cubes microworld and many similar ones originate from '*Infovekacik*' – an older Slovak on-line magazine for pupils created in cooperation with our department.

Another productive means to support implementation of primary programming into formal education for all children is the international contest Bebras. In Slovakia we initiated a special category for primary pupils and many contest tasks are deliberately designed to incorporate problem solving and elementary programming concepts. The national success and high number of contestants suggest that pupils and teachers are interested in this form of informatics.

4. Programming in Primary Teachers' Professional Development

All Slovak primary teachers have to get a master degree from a pedagogical faculty of one of our universities. They are not specialists – they teach pupils of years 1 to 4 (6 to 10 years old children) all subjects (sometimes excluding foreign languages and/or informatics). In 2008 a new compulsory school subject was introduced – primary informatics. However, pedagogical faculties have failed to update their study programs to include corresponding pre-service development for future teachers till today. Fortunately, a national project focused on professional development of in-service informatics teachers was launched in 2008 (till 2011) and authors of this paper were involved – together with the teams from five universities across the country – in developing its strategy and content and delivering it to 700 in-service teachers. The main goal of the project was to offer a modern, up-to-date, high quality education necessary for teaching this new subject at primary schools. Note that similar situation and PD strategy is being reported from the Czech Republic by Vanicek (2013).

Within the Slovak national project, 700 in-service primary teachers attended 18 study modules (each 6 hours long). Each module belonged to one of four tracks: *Digital literacy*; *Informatics*; *Didactics of primary informatics*; and *Modern school*. The Informatics track included six modules: *Computer and digital devices*; *Information around us 1–3*; and *Problem solving and basics of programming 1–2*. Teachers that graduated from the PD should be able to use digital technologies both in their classes and when preparing for them. They should perceive the elementary informatics as an important part of pupils' education and development, being able to meet the learning objectives of primary informatics as prescribed in the national curriculum. The study materials and whole lecturing process was designed to prepare the teachers for future development in digital technologies – for new microworlds and new operating systems, and also for new devices that would be used in the classrooms in near future. Authors of the study materials and lecturers took great care to introduce the teachers to a variety of software applications and appropriate teaching strategies. Teachers were learning how to evaluate appropriateness of software applications and microworlds and how to use them in the classroom.

From the perspective of this paper we find most relevant the modules dealing with **problem solving**, **elementary programming** and corresponding **didactical materials**. These areas had not been treated until then in any literature in our country (and hardly anywhere) and designing that content and delivering it to 700 in-service teachers was a real challenge and important innovative step towards new primary informatics. For the sake of the project, several new microworlds had been created, e.g. EasyLogo and Living Pictures, and participants used many other already existing microworlds and programming environments designed for primary pupils by experts in Slovakia.

One of the most successful new developments in the project was an idea and implementation of the Cards Tool (Tomcsanyi 2012). It is an authoring application that enables the teacher to design simple but vastly variable activities for any (primary) school subject. Participants of the project enthusiastically used the tool and created interesting activities that confirmed that primary teachers are creative and persistent and can use digital technology in their teaching. Since then, several thousand different activities created by the teachers themselves in the Cards Tool have been posted at Slovak portal `zborovna.sk`.

Although we lost touch with most of the participants when the project finished, we are interested in following how they manage to utilize new skills in their practice. Therefore we sporadically address a small sample of the participants and ask them to reflect about the project's longer term benefits. From that (mostly anecdotal) data we may formulate several interesting observations about the implementation of the *problem solving and programming activities* at primary level:

- Primary informatics lessons are usually run in a special computer lab, dedicated to primary key stage (older pupils usually use another computer lab).
- In each year group (2, 3 and 4) around 5 to 8 lessons are allocated to *problem solving and programming*. These are usually taught in a row, often towards the end of the school year.

- Teachers are using the study materials extensively and share them with other colleagues. They have altered their lesson plans to incorporate teaching methods applied in the project's PD sessions.
- Problem solving tasks (as presented earlier) are not highly popular among the teachers; many teachers simply skip them. They rarely realise that those tasks are not puzzles nor riddles, nor that their goal is not to find the solution by trial-and-error but systematically look for the solving method and reflect about the externally represented solution.
- Teachers enthusiastically use some microworlds that were created for the project – most of all The Jumper, World of the Ant and The Living Pictures while EasyLogo is less popular. Interestingly, each teacher has a strong preference for exactly one of the microworlds.
- Microworlds with the in-built sets of tasks of increasing difficulty and automatic verification of the solutions are used the most. Teachers often say they cannot provide immediate feedback for all pupils in the group and primary pupils are very keen on learning if they are progressing in the assignments. The sets should be designed to be solvable within one lesson (45 minutes). It should not be possible to skip the tasks in the set – only after the task is solved correctly the child can proceed to a harder one.
- Open programming environments are difficult to use and the teacher has to be better prepared for designing her own meaningful assignments and tasks within such environments (often it requires to attend extensive specialized training for the chosen environment). Our primary teachers probably have not reached that level of expertise yet.
- Most of the teachers are familiar with, visit and use the *Infovekacik website* – an on-line magazine for children with dozens of game-like microworlds. It would be probably useful to create a web portal with similar content and add lesson plans and recommended teaching methods. Teachers need good resources for their teaching that would inspire them to search for new suitable microworlds and software applications.
- Many teachers use The Cards Tool to design their own simple activities for other school subjects (mostly language and science, only rarely for primary informatics).
- All teachers are appreciative and see high value of the project's PD and of the new subject.

In conclusion, we believe that the national project and its PD programs were well designed and conducted. The participating teachers do incorporate learned skills and knowledge into their teaching. However, some of our plans proved to be too optimistic – most notably our inability to share with the primary teachers the importance and learning potential of the *problem solving tasks* (as described earlier in 3.1). Another failed expectation was to assume they would design their own sets of tasks for the pupils to support their informatics learning objectives. Teachers prefer to use the activities we prepared for them and their PD. Clearly it is vital to provide suitable series of activities with each microworld or digital toy/tool.

5. Discussion and Conclusion

Presented approach to primary programming has resulted from our previous experiences with teaching programming and developing programming interventions for all stages of schools, including university study programs for future teachers of informatics, for several decades. Our professional roots lie in Logo culture, into which our Comenius group has contributed by two internationally recognized versions of Logo: Comenius Logo and Imagine Logo. From that background we inherited our endeavour to respect the needs of students, together with other principles of Papert (1999) such as:

- *The Logo programming language is far from all there is to and in principle, we could imagine using a different language, but programming itself is a key element of this culture.*
- *So is the assumption that children can program at very young ages.*
- *And the assumption that children can program implies something much larger: in this culture we believe (correction: we know) that children of all ages and from all social backgrounds can do much more than they are believed capable of doing. Just give them the tools and the opportunity.*
- *Opportunity means more than just "access" to computers. It means an intellectual culture in which individual projects are encouraged and contact with powerful ideas is facilitated.*

We have also learned how important it is to integrate programming into pupils' learning experience only if they themselves see the meaning in doing so and perceive programming as a means to express themselves, to solve problems, *to make things happen*... In the case of primary pupils, such programming should most probably restrict to building simple future behaviours in certain notational system and solving tasks, which arise from handling such behaviours.

Although we consider elements of programming to be key constituent of informatics in primary education, we do not develop it as a means to attract more students to later Computer Science majors. We build it as a valued and legitimate core subject contributing to general education and complex development of every girl and every boy. Yet, we hope, that it may consequently play that role as well − the skills, knowledge, and attitudes, which pupils gain in elementary informatics may later help them build sound understanding of Computer Science principles.

Programming, which we consider appropriate for primary pupils, can be naturally divided into three domains (while first domain should proceed the other two, we believe that the second and the third ones can be implemented in any order or even in parallel). They are:

- Solving problems and handling solutions.
- Controlling an agent.
- Tinkering with interactive environments.

For each domain we have presented its main learning goals, corresponding computational concepts, computational practices, and essential cognitive operations to be performed; selection of activities and examples, which in detail illustrate various types of tasks and problems to be solved; several software applications that are being used;

and also several pedagogical observations and recommendations, which resulted from our collaboration with the primary teachers.

Most of the programming environments, which are being used at Slovak primary schools, are free applications, usually small microworlds focused on one of the domains listed above, and one or several cognitive operations belonging to that domain. As partly validated in chapter 4, teachers usually exploit environments which they find attractive (although often not being able to verbalize which criteria they apply for judging this). However, they clearly favour environments supplemented with teacher materials and activities for pupils, and environments, which they may give away to their pupils for their home work and play.

Our experience in implementing programming at the lower secondary stage ISCED 2 (although not based on systematic evidence yet) shows that three domains presented in the paper for primary stage can seamlessly be picked up and further elaborated in lower secondary years to cover further cognitive operations (like conditional steps in programs, abstractions, i.e. procedures without or with parameters etc.). However, extensive research to help us better understand cognitive demands of such programming and real values of educational programming for the complex development of primary and secondary students is inevitable. We have already undertaken some initial steps in this direction, see e.g. (Gujberova and Kalaš, 2013).

As we document in chapter 1, informatics in upper secondary education has considerably long tradition in Slovakia. In recent years, it has been extended as a mandatory subject to lower secondary level (2005) and primary level (2008). In chapter 2, we briefly characterized its curriculum and its learning goals and especially the key role of programming within the subject.

We fully focused on educational primary programming in the paper. In chapter 3, we presented in detail our approach to such programming together with corresponding computational concepts, cognitive operations, and programming environments employed in our classes. In chapter 4, we then described how the CPD for primary in-service teachers has been implemented – with partial successes and numerous obstacles and challenges that require permanent and intense support from the institutions responsible for education. In spite of many obstacles and slow progression, there are many positive and stimulating reactions from primary teachers who implement elementary informatics with exceptionally positive involvement. They also report positive attitudes of their pupils.

The development of the subject of informatics in primary school is a long-term process. In it, we must thoroughly respect the requirements of the developmental appropriateness, carefully observe and analyse the needs of the pupils, respect all stages of their learning processes, set correct priorities, and apply proper tools – so that we support the development of such programming, which our pupils will clearly benefit from. In this aspect, we deeply agree with Papert, Ackermann and other seminal authors when they advise not to learn programming for the sake of programming. Instead, we should...

> *use the knowledge of programming to create contexts where other playful learning can happen. Children will engage in programming if they can get something out of it right now – not later when they'll grow up,* (Ackermann, 2012).

References

Ackermann, K.E. (2012). Programming for the Natives: What is it? What's in it for the Kids? In: Kynigos, Ch., Clayson, J., Yiannoutsou, N. (Eds.), *Proceedings of Constructionism, Athens, Greece August 2012*. National & Kapodiststrian University of Athens, Athens, 1–10. Updated version obtained via CRN Japan: http://www.childresearch.net/papers/pdf/digital_2012_03_ACKERMANN.pdf

Barr, V., Stephenson, Ch. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48–54. DOI: 10.1145/1929887.1929905 http://doi.acm.org/10.1145/1929887.1929905

Blackwell, A.F.(2002). What is Programming? In: Kuljis, J., Baldwin, L., Scoble, R. (Eds.), *Proceedings of 14th Workshop of the Psychology of Programming Interest Group*. Brunel University, 204–218.

Blaho, A., Kalaš, I., Tomcsanyiova, M. (1995). Experimental curriculum of informatics for 11 year old children. In: *WCCE'95 Liberating the Learner: Proceedings of the sixth IFIP World Conference on Computers in Education*. Chapman & Hall, London, 829–841.

Blaho, A., Salanci, L. (2011). Informatics in primary schools: visions, experiences, and long-term research prospects. In: Kalaš, I., Mittermeir, R. (Eds.), *Informatics in Schools: Contributing to 21st Century Education*. LNCS 7013, Springer, 129–142. ISBN 978-3-642-24721-7.

Brennan, K., Resnick, M. (2012). *Using artefact-based interviews to study the development of computational thinking in interactive media design*. Paper presented at annual American Educational Research Association meeting. Vancouver, BC, Canada.

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.

Eisenberg, M., Elumeze, N., MacFerrin, M., Buechley, L. (2009). Children's programming, reconsidered: settings, stuff, and surfaces. In: *Proceedings of the 8th International Conference on Interaction Design and Children (IDC '09)*. ACM, New York, NY, USA, 1–8. DOI:10.1145/1551788.1551790 http://doi.acm.org/10.1145/1551788.1551790

Gibson, J.P. (2003). A noughts and crosses Java applet to teach programming to primary school children. In: *Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java (PPPJ '03)*. Computer Science Press, Inc., New York, NY, USA, 85–88.

Gujberova, M., Kalaš, I. (2013). Designing productive gradations of tasks in primary programming education. In: Proceedings of the 8th Workshop in Primary and Secondary Computing Education WiPSCE '13. ACM, New York, NY, USA, 108–117. DOI: 10.1145/2532748.2532750 http://dl.acm.org/citation.cfm?id=2532750

Hu, Ch., (2011). Computational thinking: what it might mean and what we might do about it. In: *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education (ITiCSE '11)*. ACM, New York, NY, USA, 223–227. DOI:10.1145/1999747.1999811 http://doi.acm.org/10.1145/1999747.1999811.

Hubwieser, P. (2012). Computer science education in secondary schools – the introduction of a new compulsory subject. *ACM Transactions on Computing Education*, 12(4), Article 16 (41 pages). DOI:10.1145/2382564.2382568 http://doi.acm.org/10.1145/2382564.2382568

Informatics Europe and ACM Europe (2013). *Informatics Education: Europe Cannot Afford to Miss the Boat*. Report of the joint Informatics Europe & ACM Europe Working Group on Informatics Education. http://www.informatics-europe.org/images/documents/informatics-education-europe-report.pdf.

Kalaš, I. (2010). *Recognizing the Potential of ICT in Early Childhood Education: Analytical Survey*. UNESCO Institute for Information Technologies in Education, Moscow. ISBN 987-5-905175-03-9.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, 2(1), 32–37. DOI:10.1145/1929887.1929902 http://doi.acm.org/10.1145/1929887.1929902.

Lu, J.J., Fletcher, G.H.L. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin*. DOI:10.1145/1539024.1508959.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education*, 10(4), Article 16 (15 pages). DOI:10.1145/1868358.1868363 http://doi.acm.org/10.1145/1868358.1868363

Mayerova, K., Veselovska, M. (2016). How to teach with LEGO WeDo at primary school. In: *Proceedings of 7th International Conference on Robotics in Education (RiE 2016)*, Vienna. To be published in the Springer Series: Advances in Intelligent Systems and Computing.

Mogardo, L., Cruz, M., Kahn, K. (2006). Radia Perlman – a pioneer of young children computer programming. In: *Current Developments in Technology-Assisted Education*, 1903–1908. CiteSeerX: 10.1.1.99.8166. http://citeseerx.ist.psu.edu/viewdoc□/summary?doi=10.1.1.99.8166

National Curriculum, Informatics for ISCED 1 (2011, 2015). (In Slovak). http://www.statpedu.sk/sites/default/files/dokumenty/inovovany-statny-vzdelavaci-program/informatika_pv_2014.pdf

Papert, S. (1999). What is Logo and who needs it. In: *Logo Philosophy and Implementation.* Highgate Springs, Vermont: Logo Computer Systems Inc. ISBN 2-89371-494-3.

Pekarova, J. (2008). Using a programmable toy at preschool age: Why and How? In: *Workshop Proceedings of SIMPAR 2008, International Conference*. 112–121. ISBN 978-88-95872-01-8.

Piaget, J., Inhelder, B. (1993). *La Psychologie de l'enfant*. Paris: Presses Universitares de France.

Polya, G. (2004). *How to Solve It.* Princeton University Press.

Przybylla, M., Romeike, R. (2014). Physical computing and its scope – towards a constructionist computer science curriculum with physical computing. *Informatics in Education*, 13(2), 241–254.

Resnick, M. (2012). Point of view: reviving papert's dream. *Educational Technology*, 52(4), 42–64.

Selby, C.C. (2012). Promoting computational thinking with programming. In: *Proceedings of the 7th Workshop in Primary and Secondary Computing Education (WiPSCE '12)*. ACM, New York, NY, USA, 74–77.

Selby, C.C. (2013). Computational thinking: the developing definition. In: *ITiCSE Conference 2013*, University of Kent, Canterbury, England (e-prints), 6 p.

Settle, M., Franke, B., Hansen, R., Spaltro, F., Jurisson, C., Rennert-May, C., Wildeman, B. (2012). Infusing computational thinking into the middle- and high-school curriculum. In: *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '12)*. ACM, New York, NY, USA, 22–27.

Tomcsanyiova, M., Tomcsanyi, P. (1997). Experimental IT education for lower secondary school using Windows and Comenius LOGO. In: *Learning and Exploring with LOGO: Proceedings 6th European LOGO Conference*. Budapest, 263–272. ISBN 963-8431-91-1. http://eurologo.web.elte.hu/lectures/tomcsa.htm

Tomcsanyi, P. (2012). Small interactive computer activities made by primary teachers. In: *Information and Communication Technology in Education 2012*. Ostrava: University of Ostrava, 263–272.

Vanicek, J. (2013). Introducing Topics from Informatics into Primary School Curricula: how do teachers take it? In: Diethelm, I., Arndt, J., Dunnebier, M., Syrbe, J. (Eds.), *Informatics in Schools: Local Proceedings of the 6th International Conference ISSEP 2013, Oldenburg, Germany – Selected Papers*. Universitätsverlag Potsdam, 41–51.

Wing, J.M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A*, 366, 3717–3725. DOI:10.1098/rsta.2008.0118

Wing, J.M. (2011). Research notebook: computational thinking – what and why? In: *The Link*. Pittsburgh, PA: Carnegie Mellon University.

Links to Microworlds and Programming Environments

Baltie: http://www.sgpsys.com/en/whatisbaltie.asp
Bee Tasks: see (Gujberova and Kalaš 2013), for the microworld itself contact the authors
Cards Tool: http://edi.fmph.uniba.sk/~tomcsanyi/Karticky/
EasyLogo: http://www.salanci.sk/EasyLogo/index.html
LEGO WeDo: http://www.legoeducation.us/eng/product/lego_education_wedo_software_v1_2_and_activity_pack/2239
Microworlds JR: http://www.microworlds.com/solutions/mwjunior.html
Scratch: http://scratch.mit.edu/
Thomas the Clown: http://www.r-e-m.co.uk/logo/?Titleno=7485

M. Kabátová, was an assistant professor of informatics education at Faculty of Mathematics, Physics and Informatics at Comenius University, Bratislava. Her research interests included elementary programming, educational robotics and qualitative research methodology applied to educational research in informatics education. She is a co-author of several research papers and learning materials dealing mostly with educational robotics. Since 2014 she runs an SME distributing cochlear implants in Slovakia and provides services for cochlear implants users. Currently she is planning to initiate a workshop providing a learning space for children with or without cochlear implants where they could explore programming and building autonomous LEGO robots.

I. Kalaš is a professor of informatics education at Comenius University, Bratislava. His professional interests include development of constructionist educational interfaces for learning for children and research in the field of the impact of digital technologies on learning. Ivan is a co-author of several programming environments for children, including SuperLogo, Imagine Logo, Thomas the Clown and RNA (Revelation Natural Arts) adopted by thousands of schools, home and abroad. He is also an author or co-author of several books and textbooks on children programming and informatics, which have been published in several languages and countries in Europe and beyond. He has also been active in several national and international policy efforts and initiatives. Ivan represents Slovakia in the IFIP Technical Committee for Education. From 2008 to 2013, he was a member of the International Advisory Board of the 'Microsoft Partners in Learning' initiative. From 2014 he is a visiting professor at UCL Knowledge Lab, London.

M. Tomcsányiová is an assistant professor of informatics education at Comenius University, Bratislava. She is a guarantor of a bachelor study programme for future teachers of informatics. She reads the courses on programming and educational aspects of programming and conducts corresponding research in the field of educational programming in all levels of education. Monika is a co-author of several textbooks and methodical teacher materials. She is also involved in designing small educational software environments supporting informatics education in Slovak primary and lower secondary schools. She develops tasks and organizes informatics challenges for primary and secondary students, including Imagine Logo Cup, Scratch Cup and national Bebras. Her area of research is didactics of programming for lower secondary schools. She is a co-author of research papers in this area.

Olympiads in Informatics, 2016, Vol. 10, 161–176

DOI: 10.15388/ioi.2016.10

Collecting, Processing and Maintaining IOI Statistics

Eduard KALINICENKO, Mārtiņš OPMANIS[1]
[1]*Institute of Mathematics and Computer Science, University of Latvia*
29 Raina Boulevard, Riga, LV-1459, Latvia
e-mail: eduardische@gmail.com, martins.opmanis@lumii.lv

Abstract. In this paper, we describe the whole process of creation, launching and maintaining the IOI Statistics website project. Special attention is paid to the data acquisition and correctness problems. Our experience may be useful for other International Science Olympiads still not having their statistics portals.

Keywords. IOI, statistics, data collection, data maintaining.

1. Introduction

International Olympiad in Informatics (*IOI*, IOI-WEB) is the annual International Science Olympiad for high school students. Since the first IOI held in 1989, a certain amount of data involving the competition is accumulated. However, due to a quite different quality of data for different years as well for different aspects of IOI, the data on its own is not useful.

Usually, we need the data to answer specific questions, like the following ones:

Question 1. What are the results of the competition for the given year?
Question 2. In what years did the given country participate?
Question 3. How many medals overall has a certain country achieved?
Question 4. What results did the particular person achieve?

As far as International Science Olympiads are concerned, Question 1 is answerable as publishing the results usually takes place. Sadly, however, publishing data about the event often ends here. About further questions, to the best of our knowledge, before the work on this project began, only International Olympiad in Mathematics (*IMO*, IMO-WEB) had a centralized system that could answer all these Questions. Most of the other efforts in collecting and maintaining the statistics were not made by the organizers. Most notably, Waldemar Gorzkowski and Ádám Tichy-Rács compiled the list of winners for International Physics Olympiad (IPhO), but it was done in a separate PDF file, which is not very customizable and exportable and included only medalists.

For International Olympiad in Informatics, the situation was even grimmer. Official IOI website (IOI-WEB) only collected individual IOI results, and often only for the medalists – the issue we will cover more in-depth in Chapter 4. There also existed two parallel projects – the one by Tom Verhoeff (TUE-NL) and another one by Stanislaw Waligorski (EDU-PL). Efforts to bring the results under similar format and process it to display results per country (EDU-PL) were made, but both projects were out-of-date and abandoned. Even the official website (IOI-WEB), despite several renovation attempts, often relied on official individual IOI websites, which have a tendency to go offline in some cases.

The best resource at the time was SnarkNews project managed by Oleg Hristenko (SnarkNews). However, despite being able to answer Questions 3 and 4 (on medalists only), for old IOIs it only had a list of names and countries of all medalists in that year, which was far from ideal. Furthermore, to the best of our knowledge, it did not appear to solve the problems about data aggregation we will touch upon in Chapter 2.

With this in mind, the goal of the project was to collect as many data as possible and aggregate them in a unified format. Further, the goal was extended to create and implement an IMO-like system to be able to explore the data collected. This paper tells the story of this project from its inception to becoming part of the IOI infrastructure. We hope that the lessons we learned could be useful to other International and Regional Science Olympiads looking how to overcome the same problems we tackled.

2. Data Collection

The work on this project began soon after IOI 2011. There was an interest in seeing the cutoffs for past IOIs to observe how they have changed percentile over the years. It was discovered that collecting the scores necessary for each kind of medal was not a trivial task, but after some time the necessary data was gathered (IOI-EDU).

At this time, the scale of the problem IOI had with statistics was observed, as some of the required data was only contained in one of the above-mentioned projects (EDU-PL), which in 2011 was already offline and was only available through the Wayback Machine (Archive). Despite the initial goal to collect only cutoff scores, it was quickly realized that in fact we have collected names, countries and at least total scores of every single IOI medalist. This meant that at least for some sense (in this case, IOI medalists) we can have some basic information, which led to the inception of this project. However, to illustrate the incompleteness of the data available, we can look at Fig. 1 and see that currently only 61% of IOIs have full per-task scores of all contestants available.

Again, limiting ourselves to IOI medalists only, it is trivial to process the raw data in such a way which could help us easily answer Questions 2 and 3. The biggest challenge with aggregating data, perhaps surprisingly, lies in Question 4. It has a couple of problems in its core and none of the resources available at the time appeared to have tackled it. First, the mapping function from a person to a name is not injective. So we cannot identify a person only by the name. Even more, as was discovered during the

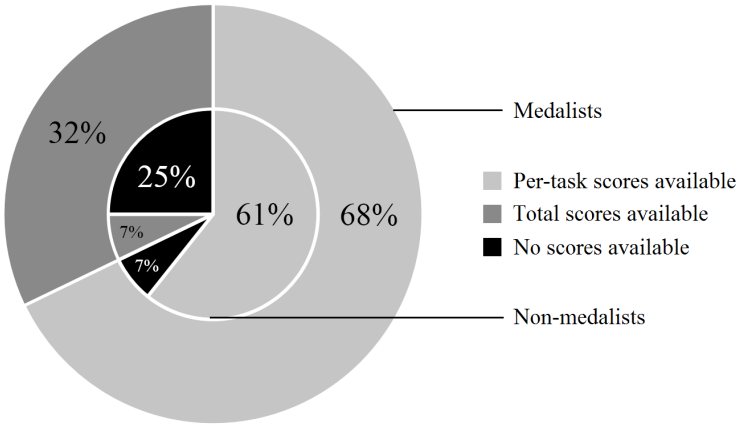


Fig. 1. Availability of contestants' scores for IOI 1989–2016.

collection of the data, even a tuple of name and a country is not enough to identify a person – there are two different people named Chen Zheng from China having participated in IOI.

Even bigger problem is the fact that the function mentioned above is not even a function. Putting aside the issue of a person changing a name (which in fact had occurred before – Jeppe Hallgren was known as Jeppe Petersen during IOI 2010), there is a serious issue of using a different spelling for the same person across different years. Most notably, Gennady Korotkevich was sometimes spelled as Henadzi Karatkevich. This whole issue is easily avoided with the help and understanding of the problem by the organizers – for example, the current IOI registration system asks leaders to pick specifically whether any delegation member is already in the registration database. However, until this support was added, this crucial part of information was lost, and so the biggest challenge in the whole data collection process was to recover this information.

In general, it was found that the same person is likely to represent the same country, although of course some exceptions are occurring like Fieke Dekkers (full name: Sophia Antonia Janna Dekkers), who was a member of a delegation from both Netherlands and the United Kingdom or Rob Kolstad representing the USA and the United Kingdom. Even limiting ourselves to contestants, which we are at this stage, Tomasz Czajka has won two gold medals – one for Poland and one for the United Kingdom. Furthermore, it is reasonable to expect that if it is the same person, then he has competed in close to consecutive IOIs – even if not strictly consecutive, a maximum gap of two IOIs is expected. Again, this does not hold if we are considering all delegation members, especially because it is common for former contestants to appear as a delegation member after some gap. However, at this stage, these two assumptions turned out to be a good basis for determining whether two differently spelled or written names are in fact the same person.

It was decided that it is not realistic to expect that decision can be made automatically without human input. However, it is certainly possible to provide a good set of

candidates for a human to look at in an attempt to minimize the human effort required. For the spelling differences mentioned above (which usually were a lot smaller than in the case of Gennady Korotkevich), Levenshtein distance (Levenshtein) between strings was quite a good indicator – if it were below a certain threshold, we would need to look at the pair to determine whether they are in fact the same person. However, it did not help to identify the second common case – if a person has several middle names, then often one set of middle names appeared in one year and a different one in another. In an attempt to tackle this issue, Levenshtein distance algorithm was run between all possible subsets (with the size at least 2) of individual words from the original strings. If any of those distances were small enough, the pair got triggered. In retrospective, while this did not catch all the cases, we feel that it generated a good enough accuracy in the original data set released.

Instead of unifying everything to a single format, as much data as we could find was released as a single Excel file (IOI-EDU). This included some tricky bits which were not documented well. For example, at IOI, the host can participate with the additional teams. At present, these teams are not competing officially, and so their representatives simply appear in the overall results at the proper position by the score but without any rank or medal. However, nothing of a sort was mentioned in the regulations earlier on. This led to a couple of precedents. Overall IOI 1989 winner, Teodor Tonchev, represented a second Bulgarian team. However, the official IOI 1989 printed booklet ranks him and declares a winner, so this interpretation was kept for that year. Secondly, in 1996 the second Hungarian team won four bronze medals. They were ranked in all the data we have seen and as per our inquiries, they appear to have received physical medals, so they were kept ranked as well.

To conclude, in this Chapter, we have discussed the challenges encountered regarding collecting and processing the data. The most important issue we have identified here is to avoid having a person's name and surname as the unique identifier. Even without a centralized system organizers should make efforts to record and keep information that can be used to tackle this problem in future.

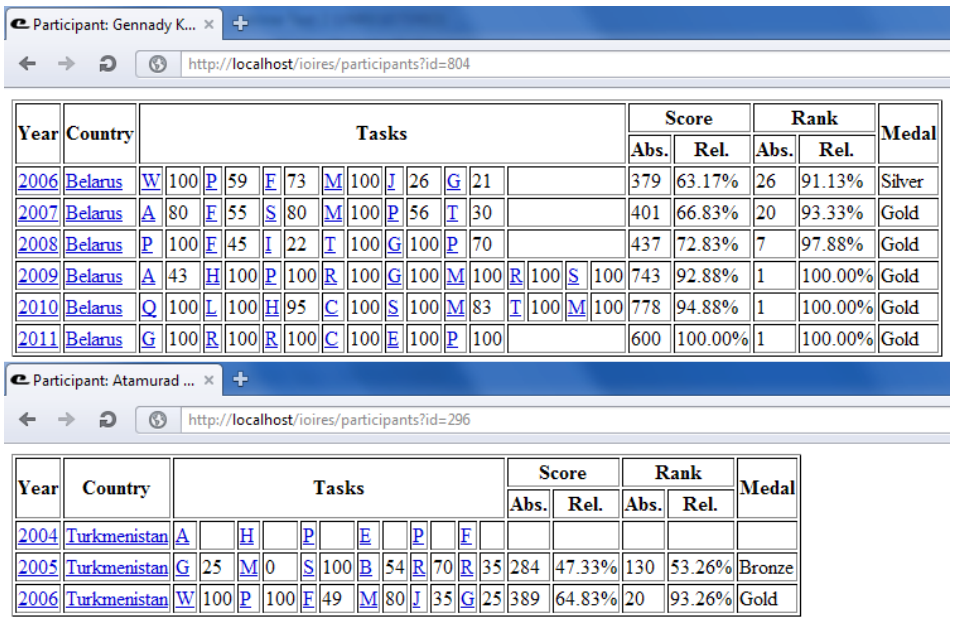
3. Website Planning

At this point, there were no immediate plans to carry on. The data has been collected, and it has even received sets of corrections, in particular by Ilham Kurnia and Mojca Miklavec. However, creating a convenient interface to host the data collected was the end goal – while the created resource (IOI-EDU) was still much more useful than anything available was at the time, it felt that there would be a huge potential wasted if it did not evolve further. To the best of our knowledge, while some efforts to create an IMO-like website based on the data collected were stated to us, they did not transform into anything concrete. That is the reason why in the spring of 2012 it was decided that the matter should be put in our hands and as such the plans for devising the website began to form.

The most crucial thing to get right was the database design. It is tempting to incorporate a lot of redundant information in the tables so that obtaining the data necessary is as easy as possible. However, that could lead to huge problems with maintaining the tables. As such, the effort was spent into carefully designing the database format to minimize the amount of redundant data kept.

However, there are some notable exceptions to that in the database design. For example, the rank and the medal contestant earned is kept in the competition record. Technically this information is redundant since it can be obtained by analyzing others results. Such decision was taken, firstly, to speed up the computation, as it eliminates the need to query all that olympiad's results if we are just interested in a single person; and secondly, to ease the handling of unranked teams, as now we can explicitly set a contestant to be unranked regardless of his position. The other example would be keeping the score of a contestant in his records directly. This is not always a redundant information since we do not always have full results (including the per-task scores). However, even if we had, the could have been some additional problems – at IOI 2000 to avoid zeroes at the end of the competition extra 50 points were awarded per day for simply turning up, so in that case even if we had full results, the contestant's total score would not simply be the sum.

The next big decision is about the cooperation between the database and the web server. It was thought that the database design was much less likely to change over time than the web server, and as such it was decided to write complicated queries which take care of everything, including composing and sorting the tables requested. This means that it is incredibly simple to write a website, as shown in Fig. 2, which queries the data-



Year	Country	Tasks	Score		Rank		Medal
			Abs.	Rel.	Abs.	Rel.	
2006	Belarus	W 100 P 59 F 73 M 100 J 26 G 21	379	63.17%	26	91.13%	Silver
2007	Belarus	A 80 F 55 S 80 M 100 P 56 T 30	401	66.83%	20	93.33%	Gold
2008	Belarus	P 100 F 45 I 22 T 100 G 100 P 70	437	72.83%	7	97.88%	Gold
2009	Belarus	A 43 H 100 P 100 R 100 G 100 M 100 R 100 S 100	743	92.88%	1	100.00%	Gold
2010	Belarus	Q 100 L 100 H 95 C 100 S 100 M 83 T 100 M 100	778	94.88%	1	100.00%	Gold
2011	Belarus	G 100 R 100 R 100 C 100 E 100 P 100	600	100.00%	1	100.00%	Gold

Year	Country	Tasks	Score		Rank		Medal
			Abs.	Rel.	Abs.	Rel.	
2004	Turkmenistan	A H P E P F					
2005	Turkmenistan	G 25 M 0 S 100 B 54 R 70 R 35	284	47.33%	130	53.26%	Bronze
2006	Turkmenistan	W 100 P 100 F 49 M 80 J 35 G 25	389	64.83%	20	93.26%	Gold

Fig. 2. Simple website to query the contestants' results from the database.

base, once the queries are known. The website itself looks quite different nowadays, but this table is remarkably similar to the one used today.

Initially, there was a third goal conceived, which was to unite all International Science Olympiads to be able to view a person's results across all competitions. Due to lack of workforce, we decided to focus specifically on IOI instead. However, the database was designed with more than one science olympiad in mind and as such the database was split into two schemes – the one with IOI-specific data, like competition records, and the one with the common information (like information about people and countries). As a result, it would be easy for some other olympiad to use the information from the common information scheme, allowing people and countries to share information across olympiads.

The website itself was coded in PHP, however as mentioned previously, most of it is just the code for displaying the information in a pretty way on the website. The biggest part of the website devoted to this part is request processing, where we need to combine all information obtained from forms to a single SQL query, resize photos, etc. However, it would not take a ridiculous amount of effort to replace the web server with another implementation of it. As for the actual design and structure of the website, it was deliberately designed similarly to IMO's statistics. However, certain aspects were improved. For example, the country delegation page with photos is only easily accessible while IMO has not finished yet, and we made sure that the accessibility of that page is improved in the IOI statistics.

Finally, what is important to consider but easy to miss is the URL lifetime. Ideally, you would like to make sure that the link created a long time ago would still work in the future and would not become dead. This means that it is desirable to create a good set of URLs at launch so that they are as human-friendly as possible. For example, compare `http://www.imo-official.org/country_individual_r.aspx?code=LVA` and `http://stats.ioinformatics.org/results/LVA`, where both of these links lead to the same type of page. The major changes are also mostly backward compatible. For example, the link which could have been made in 2012 (IOI-GDR) still works, despite the server move and a country code change to comply with ISO standards. The only dead links resulted after merging the entries of two persons (usually, after an e-mail that two entries which we treat as different people are in fact the same person), and then using the old link would not redirect to the new one, however this will get fixed it if we can reconstruct with confidence which entries were merged from backups.

In retrospective, there were also a couple of bad design decisions. One of such is the "Login" button. Its sole purpose is to allow the moderator to login into the system to view and act upon submitted requests. However, it sometimes creates confusion when users think that they are required to have an account to submit a change, while in fact "Edit" or "Add" interfaces are available to anyone. Secondly, in the "Edit" interface, there is a "Submit" button after every section (Basic Information, Contact Information, Participation Information and Photographs). The reasoning behind this was a separation of requests by category with the idea that if additional verification is needed for confirming an IOI role, we could accept the changes of contact information separately. It turned out to be a bad decision, as on several occasions people had missed multiple

"Submit" buttons and were pressing the last one, which resulted in the loss of data when users thought they submitted a change while only the photo was submitted. Finally, after submitting a request, one receives a password to revoke it if needed. Sometimes people seem to believe that this is the CAPTCHA-like process for the request and accidently cancel them instead. Perhaps, revoking requests in a suggested way is not usual, and additional warnings are required to prevent misunderstanding.

To conclude, in this Chapter, we have discussed the challenges encountered while designing a website. We feel that it is crucial to spend a considerable amount of time on planning the back-end, which can be easily overlooked in favor of the front-end. We feel that without a reasonable back-end, maintenance becomes unmanageable, which often leads to project abandonment.

4. Acquiring Data

The biggest challenge was obtaining the perfect (correct and complete) data. We currently have data about 5775 people in the database. However, a lot of data is still missing. Beginning from the first Olympiad at 1989 hosts usually produced a full list of the current IOI participants. Since 2000, this duty is stated in IOI Regulations (IOI-REG ver. 2014 S5.10(4)). Nowadays, publishing such a list is straightforward since all the data is in the IOI Registration system and it is given to the host for organizational purposes. However, now, we have no such official source of information for years 1990, 1993, 1995, 1998, 2002. Of course, now we can just point to the fact that these essential parts of IOI history are not kept in an appropriate manner.

During the initial project (IOI-EDU), only information about contestants was collected. Now, we were suddenly interested in obtaining information on team leaders and other delegation members. We limit ourselves just to "international part" of IOI participants. In the IOI statistics, you cannot find information about representatives of the host country (management, technicians, guides, volunteers, etc.) who help to organize the whole event. Also, from the historical perspective, IOI roles are changing. For example, during the first IOIs team leaders comprised an International Jury not existing anymore as a body. In 1993, there were such positions as "President of the International Olympic Committee" and "President of the Technical Committee". According to Regulations for that period, a president was nominated by the host country and today seems to correspond to the role of Chairman rather to the position of the President of IOI elected for a three-year term. Taking into account that it is almost impossible to solve all these semantic clashes of the past, we defined the following twelve roles of IOI participants: Contestant, Leader, Deputy Leader, Guest, Invited Guest, Observer, President, Executive Director, Chairman, International Committee (IC) Member, International Scientific Committee (ISC) Member, and International Technical Committee (ITC) Member.

However, even keeping names of roles as close to the current Regulations as possible, there appear unexpected problems. For example, according to the Regulations

current host country has one representative in the IC. But how to proceed if the host is represented by different people in the in-between meeting of IC and at the IOI itself? Are there two representatives of the host or one person (which one?) must be forgotten? Also, in an attempt to avoid expanding the list of roles too much, by "Invited Guests" there are marked different categories of people – task authors invited by the Scientific Committee, invited members of international organizations and associations and others.

We also provided functionality for the user to add some personal information to his profile like Codeforces/TopCoder handles, homepage, e-mail and social network profiles. This was implemented partially due to the anticipation that the website might be used for direct contacts and recruiting purposes by some parties and as such the ability to add contact information was desirable.

While the situation for contestants was not perfect, but still reasonable, the situation for other members of the delegations was close to appalling – even national websites listing all IOI teams for a specific country often omitted team leaders. However, it was anticipated that national delegations at large still have access to that information, so the natural choice would be to allow delegation leaders to provide and correct simply their data in the system. However, that did not quite solve the problem of contact information, and as well as that, it heavily relies on all delegation leaders being responsive to the cause. So it was decided that the editing access would not be limited to a single person for every country.

Instead, we decided to make the editing publicly available without any registration. Compared to the denied approach, we realized that there would also be bad and malicious edits. However, our estimations said that this way we would receive more data; and if we can filter the good requests from the bad ones, we would obtain more information this way, and since this was our primary goal, we decided to go with that. On the downside, every change submitted to the system has to be approved by moderators, which created an additional effort on our side of things. In retrospect, there were 12818 non-spam requests (including rejected malicious ones) at the time of writing out of 1295397 requests received overall, meaning that 99% of the requests we receive is spam. However the vast majority of them never reach our eyes, so despite that, we feel that it was the correct decision.

Furthermore, sometimes it benefited to go through a central point of moderation. For example, some confusion was created with Sweden and IOI 1989. The national website for Swedish Olympiads in Informatics lists IOI team for 1989 (SOI). Similarly, we have received requests to add this team to the system. However, we were in possession of the physical copy of the first IOI booklet having no records about a team from Sweden. After some conversations with Pär Söderhjelm, former Swedish team leader, it was learned that in 1989 there were two similar IOI-like initiatives. The Swedish team picked the "wrong" one (we would be interested get more information about this event!), so they did not participate in IOI 1989, and only joined IOI in 1990.

There were rare cases in the IOI history when teams mentioned in the official booklets did not participate in the competition due to visa issues or other, mostly political, reasons. Such cases of disagreement with official documents were resolved using direct communication with country representatives.

The biggest issue regarding missing data was generated by the rule that IOI had from 2000 – for non-medalists their names and represented countries should not be published (IOI-REG ver. 2014 S5.10(3)) in the results and as a consequence usually are lost from freely available sources of information. We believe that not mentioning names of non-medalists is contrary to Olympic spirit formulated by Pierre de Coubertin **"**The most important thing in the Olympic Games is not winning but taking part; the essential thing in life is not conquering but fighting well" (Coubertin). Contestants qualified to IOI as a rule passed many national and regional events and are good even if they do not obtain any medal at IOI. Furthermore, it creates a huge gap between the last bronze medal and the first person without the medal. Despite these two people being right next to each other in the standings, this rule made impossible for the second one to highlight his achievement easily. But for this project, it was decided that we would not allow individuals to de-anonymize their results if they desire as this was deemed to be impossible to verify. This concern turned out to be largely irrelevant, as we did not receive any such requests.

In an attempt to at least not to lose these names for the history, in the IOI Regulations since 2014 is stated that current IOI host is obliqued to "Produce a full result list containing the final scores of all contestants, which is made available to the OED and ISC, along with the data required to generate those scores;" (IOI-REG ver. 2014 S5.10(4)). On the public side of things, this rule began being pointless from IOI 2010, when the live scoreboard during the contest was introduced. As such, other projects like SnarkNews (SnarkNews) have full results available and so it became a pointless exercise to remove the non-medalists' names as the full results were already made available previously. In a hilarious display, IOI 2010 website still hosts both versions of results – anonymized and de-anonymized. As such, we have taken a strict stance on maintaining the full results of IOIs from 2010 onwards. Since then the risk of losing essential contest information is diminished.

However, we fully respect this rule while no live scoreboard was available. Most notably, during the data collection stage, we have managed to discover the cached full results of IOI 2004, which presumably were published by organizers by mistake and were taken down soon afterward. Including these data in the Excel file resulted in two e-mails from IOI 2004 contestants asking to remove their name from the results. Their wishes were respected, and this was also the main reason for the decision of not publishing the de-anonymized results of IOI 2004's non-medalists at all on the present website.

The biggest issue with that rule, however, is the fact that it made it easy for IOI organizers to neglect the data. Since 2000 in IOI Regulations it was specified that three lists should be published after the IOI:

1. The final scores of the medal winning contestants.
2. A list of all participants.
3. A list of all scores which contains no name/country information (IOI-REG, ver. 2014 S5.10(3)).

This rule does not oblique host to keep full scores for all contestants in any form. Moreover, often only the first list (as the most important list of the whole event) was published, meaning that non-medalists scores and all participants' names were lost even-

tually. We believe this is the main reason why the data we have today is so incomplete, which we believe was not anticipated at the time this rule was passed. Closing discussion about the availability of full results, it should be added that also rare cases of disqualifications are not clearly marked in results. Usually, such contestants appear as the last ones in the full results of the particular IOI together with contestants having no single submission into grading system.

Despite the fact that IOI is an individual competition there are also some cumulative statistics available regarding country participation in IOI: the total number of medals, the total number of participated contestants, years of participation (counting only the years when there were contestants on the team), etc. However, it is important to note that we still keep records of all years even when there was only a single observer in a delegation. Following the spirit of individual competition, countries are never explicitly ranked; there are different ways to sort data, but you will never find rank next to the name of a particular country.

To conclude, in this Chapter, we have discussed the challenges encountered while attempting to collect additional data to obtain the complete picture and why this was necessary in the first place. The takeaway message from this would be to make sure that hosts publish all required information, and then make sure that the information is duplicated in some centralized location – sometimes the last step would be forgotten, and the information would be lost after the host website would go offline. While Web Archive might save this information, it is not a good idea to rely on this.

5. Moderation

After reading the written above, a reader may have an impression that work is already completed and new data may appear only after the current IOI, and there will be no updates in-between IOIs. Concerning IOI statistics, this is far from the reality. There is still plenty of work to perform outside of IOI period. In this Chapter, we list several examples of such work in the decreasing order of importance.

First, the main effort goes into the attempts to fill in gaps in the history by trying to contact persons probably having missing documents. We are confident that it is impossible to get official data from hosts for five IOIs mentioned above. The most realistic way is to contact former team leaders and try to fill gaps on “delegation by delegation” basis. The main obstacle here is that during early IOIs generations are changed and without proper local archives it is almost impossible to restore information. As well, different countries are differently interested in completing information regarding their country. At the moment of writing paper statistics for 44 countries (from 101 or 43.56%) are completed.

Next, we have to deal with major requests concerning the website. These requests usually affect the whole website, so they have to be dealt with carefully. To give an example, in the last season, two of such requests stood out.

For the first one, a former contestant argued that his name must be removed due to “right to be forgotten”. We insist that IOI Statistics is the source of strictly statistical and

historical information, and we do our best to keep it correct and complete as possible. Removing any essential piece of correct IOI information is against our efforts to complete the whole picture of the IOI history. Also described "right to be forgotten" cases influenced just search engines (how easy is to find a particular piece of information) not the availability of information itself.

For the second example, S. Maggiolo suggested adding gender information to contestants' data on the official IOI statistics website (Maggiolo). Since we rely on authoritative sources of information and no gender information was kept until the creation of the IOI Registration system, we do not believe it's feasible to collect and verify the gender of individuals. Furthermore, addressing contestants according to their gender itself might be questionable – we are not attaching other personal information like race, religion, food preferences or disabilities; and we believe that some people might object to releasing information on their gender as well. We understand that these statistics might be considered useful to answer some research questions, like whether vegetarians tend to be better programmers, but we are not convinced at this point that the official IOI website is the right place for that. This matter was recently raised with International Committee.

Moving on, we have to deal with processing the usual requests. Commonly, these are about changing/adding photos or updating contact information. If there is a request to change contact information, requests are verified by checking a proposed link and its relevance to the addressee. In unclear cases if contact information is known, before approving, subject to the proposed changes or country representative is contacted. However, we periodically receive requests by e-mail (because these types of requests are not supported by the system). Most commonly, the request is to merge two of the profiles together because in reality it is the same person, which is usually quite easily verified.

Of course, not every request is legitimate and as such we have to struggle with hoaxes and improper requests. Relatively often, there are requests to change information in an appropriate manner. For example, a request to add "contestant of team X" for the year where we are confident that we have complete data (even more if suggested name is like "Anonymous"). Or request to remove all contact information for some participant, which appears to be fake after consultation with the addressed person. Or adding an improper photo like well-known Borat dressed in the green outfit (see Fig. 3). In general, almost all such requests seem to be malicious than just joyful. Finally, there is also an issue of dealing with spam. However, most of the work here is done automatically, and just most sophisticated cases pass through the spam filter.

For readers, it may be interesting to know how the moderation of requests submitted through the system looks from the inside. An example screenshot is shown in Fig. 3.

In the section "Administration" there is a possibility to clear spam, approve or reject a batch of requests according to submitter's IP address. Note that we can also change the availability of the data for certain IOIs. This comes useful during the IOIs when the results can be entered in the database long before they become official (and they will not "leak" to the public), and we can then publish them with one click as soon as they become official. Section "Country Data" offers a possibility to change color code for the specified country in the main table of countries describing information completeness:

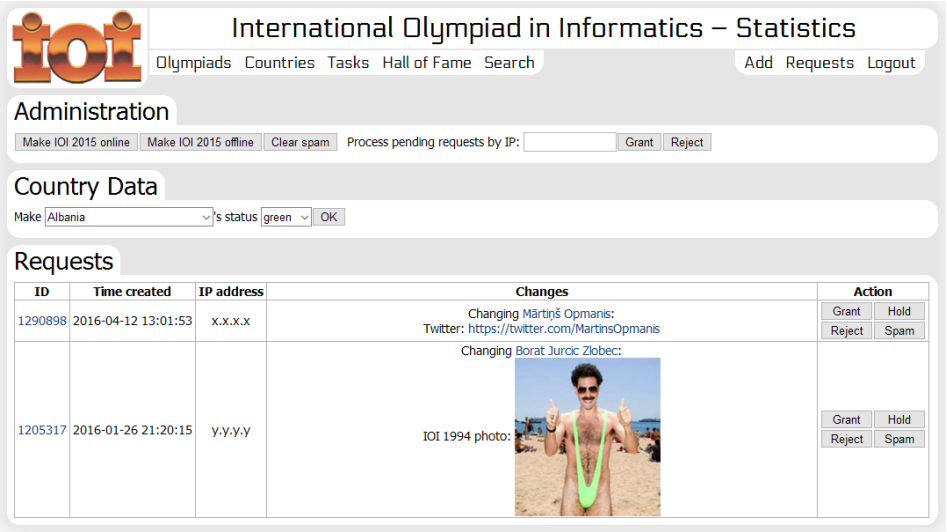


Fig. 3. Moderator interface screenshot (IP addresses are obfuscated).

green – information complete, yellow – information almost complete (usually completed regarding contestants), red – essential part of the information is missed.

All requests must be approved (button "Grant"), rejected ("Reject" or "Spam") or have the processing postponed ("Hold"). In the given example, both requests are real. The first one would like to add a Twitter profile to person's requisites and, therefore, will be approved. The second one is obviously non-legitimate so that it would be rejected. It is worth noting that even if the photograph were of the person in question, it would still get rejected as we aim to have more passport-like photographs on the website (although this desire is not expressed clearly), where the head occupies a significant percentage of photo's vertical space.

To conclude, in this Chapter, we have discussed the challenges encountered while maintaining the website. We feel that the most important issue is not to take accepting any edit requests lightly. While it is understandable that occasionally some false information provided by the community might slip through, if this information is false, one would lose the reputation which is crucial if you want to be a source of credible information.

6. Launch and Steps to Becoming Official

The statistics website was launched during IOI 2012 after a presentation during the IOI Conference. The results of IOI 2012 itself were added about a week after it was finished. The website accumulated an average of 140 unique daily IP visits during first two months after launch. The appeal to obtain more information was also fruitful. In the period until the next IOI we have granted 1233 requests, whether they were submitted

by the community entirely, or we have committed them ourselves based on the raw data submitted by the community.

Meanwhile, in the same period, we have received access to the official IOI website (IOI), which at the time was quite outdated. This meant that we could start to plan on how to move this project under the official domain. Initially, the idea was to incorporate essential information about IOI on the project and replace the original IOI website completely. However, this was later abandoned for cleaning up and maintaining the original website and hosting the statistics portal as a separate subdomain.

During IOI 2013, it was voted by the General Assembly (GA) to sponsor maintenance of the original IOI website and the development of this project from the IOI budget (IOI-2013-GA-MIN). This allowed us to fulfill the goal mentioned above and in April 2014 the website was moved to the place where it currently resides (IOI-STATS). While we were still supplementary to the official results published by the organizers, for IOI 2015 (IOI-2015) they did not do so and instead simply linked to our project, which we consider being a final step in becoming official.

One of the other aspects of becoming official outside of recognition is the ability to access parts of official data. This is best showcased by the process of releasing the list of participants before the IOI occurred. Usually, every year well in advance of IOI there appears a popular thread on Codeforces and TopCoder, where the community would share its knowledge on the upcoming IOI contestants, and that would be summarized, creating an unofficial IOI contestant list. The ability to display participant list was first added to the website for IOI 2013. At that stage, we did not have any access to the official data in the registration system (IOI-RS) and as such, we simply mirrored the thread on Codeforces for that year. After the organizers had released the official list, the website's data was modified to reflect that.

For IOI 2014, we decided to keep it official and did not take any data from the community, as some talks on accessing official data began already. Unfortunately, it did not reach the conclusion in time, and as such we have waited until the official data was released. Strangely enough, initially it got released exclusively on the mobile app for Android. Still, it was mirrored to the database and for some time, it was the only convenient way to view contestant list if you do not have an Android device. Subsequently, we received the necessary data (name, surname, role, and country – note that we always receive information on per-need basis and do not receive unrelated and personal information to the cause, like passport details, for obvious reasons, including the one of privacy) from the official IOI registration system, so we could add some additional delegation members (like guests, who were not displayed in an app). Also, we discovered a discrepancy between the registration data and the data published by the organizers, which was then addressed and fixed.

In the meantime, this issue was discussed within IC to ensure that we could receive the data from the IOI registration system sooner. Additionally, the check mark was added to the system to allow us to publish the participants' photographs as well. Because of these, for IOI 2015 we were able to publish the participant list soon after the registration deadline was over and we were the first resource to do so. We were able to publish 228 photos, where we were given permission to do so. Extra effort was made to avoid ac-

cidental leaks of personal information, where some people uploaded a full passport copy instead of a photo. Even despite the fact the we shrink the photographs to 180 pixels in height, and it would be very unlikely that there would have been any leaks because of that, we still manually cropped those photographs.

Finally, the process of becoming official means that you receive a lot more credibility over time and as a result people start relying on and linking to your information. At the moment of writing this paper, there are over 9000 links to "stats.ioinformatics.org" according to Google search. Some of them are not only simply linking to results or individual statistics, but also refer to actual statistics. For example, the post on Quora (Quora) was attempting to answer the question of the hardest IOI problem based on the average score per contestant available on the website. Of course, not all of the links are of the same level of importance. Moreover, there are incorrect ones. For example, link in the article (UG-RU, 2016) claims that there will be problems from the corresponding IOI where there is currently just statistical information about tasks.

Investigating pages with links, we found one, which may be the example why we cannot take any responsibility for the way how provided statistical information is used. For example, in the forum post (Apricity) photos from the IOI Statistics regarding Chilean contestants are extracted and an attempt to evaluate demographics based on ethnicity was made, which is not something that we approve or imagined that would happen based on the data we released.

To conclude, in this Chapter, we have discussed the process the website took to obtain an official status. Furthermore, some benefits of the official status were provided.

7. Future Work

One of the evergreen tasks is encouraging people to fill in missing parts of information. Just as a reminder for potential submitters – information can be submitted in a few simple steps. First, consult the "People" page of the particular country to find the particular person whose information you would like to update. If such a person already exists there, follow the link under that person's name and push "Edit" in the top-right corner of the page. After adding/changing appropriate information, click the "Submit" right after changed section. Be aware – there are four sections having separate "Submit" buttons, so if you want to edit information in multiple sections, you would need to submit multiple requests to the system! Only if you cannot find the person in the list of already known people, push "Add" and provide all known information about the particular person. In this case, "Submit" must be pushed just once at the very end. As a backup scenario, there is always a possibility to send all relevant information to us – we will add it by ourselves. The main principle – do not keep valuable information about former IOIs a secret!

As far as the further development of the website is concerned, the current plan is to refactor the web server code completely. When it was created, not many efforts were put into making it maintainable and as the result the code became quite unreadable over time, and it is now quite challenging to add new features without accidentally break-

ing something. After that, it would become feasible to considering open-sourcing this project so that other olympiads, like Regional ones, could use this platform to host their results. As well as that, perhaps some additional functionality could be contributed by the community then, as is currently happening with CMS (CMS), which is the competition system currently used at IOI.

8. Conclusions

In this paper, we have touched upon various aspects and challenges we have encountered in creating and maintaining a centralized place for collecting official IOI data. While it is not a small project and some International and Regional Science Olympiads might not have the resources to tackle this at the moment, we feel that it is still important to consider our experience. While the project itself is hard work, many challenges that made it hard are solvable easily if they would have been considered at the time. So we hope that after reading our experience some Olympiads might pay a bigger attention to the issues discussed here and make it a lot easier to execute the project of this scale.

Acknowledgements

We would like to thank Oleg Oshmyan for all the technical help received during the creation of this project and maintenance of the servers. We also extend our gratitude to Sergey Melnik and Vyacheslavs Kashcheyevs for supporting this project and giving us fresh ideas. We would also like to thank Ilham Kurnia and Mojca Miklavec for providing a massive amount of data in the early stages of this project. Finally, we thank IOI community for providing the financial support for this project.

References

Apricity. *The Apricity. Why do people still think that Chile doesn't have a mestizo majority?* `http://www.theapricity.com/forum/showthread.php?184231-Why-do-people-still-think-that-Chile-doesn-t-have-a-mestizo-majority`

Archive. *Internet Archive: Wayback Machine*. `https://archive.org/web/`

CMS. *Contest Management System.* `https://cms-dev.github.io/`

Coubertin, P. *Quotes.* `http://www.brainyquote.com/quotes/authors/p/pierre_de_coubertin.html`

EDU-PL. *Results of International Olympiads in Informatics, formerly at*: `http://www.oi.edu.pl/ioires/`

Levenshtein, V. I. (1965). Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4), 845–848.

IMO-WEB. *International Mathematics Olympiad.* `http://www.imo-official.org/`

IOI-2013-GA-MIN. *Minutes of the General Assembly of the 25th International Olympiad in Informatics.* `http://ioinformatics.org/a_d_m/ga/ioi13/GA-minutes-jul2013.pdf`

IOI-2015. *The 27th International Olympiad in Informatics.* `http://ioi2015.kz/`

IOI-EDU. *International Olympiad in Informatics: The Results*. `http://www.eduardische.com/ioi/main`

IOI-GDR. *Information regarding the former German Democratic Republic in the statistics of the International Olympiad in Informatics*. http://ioi.eduardische.com/countries/GDR
IOI-REG. *Official Regulations of the International Olympiad in Informatics.* http://ioinformatics.org/rules/index.shtml
IOI-RS. *Registration System of the International Olympiad in Informatics.* https://ioiregistration.org/
IOI-STATS. *Statistics of the International Olympiad in Informatics.* http://stats.ioinformatics.org
IOI-WEB. *International Olympiad in Informatics*. http://ioinformatics.org/
IPhO. *International Physics Olympiad statistics*. http://ipho.org/statictics.pdf
Maggiolo S. (2015). An Update on the Female Presence at the IOI. *Olympiads in Informatics*, 9, 127–137. http://ioinformatics.org/oi/pdf/v9_2015_127_137.pdf
Quora. *Thread "Specific Problems in International Olympiad in Informatics"* https://www.quora.com/What-are-the-hardest-problems-from-past-IOIs
SnarkNews. *SnarkNews on IOI*. http://ioi.snarknews.info/
SOI. *Swedish Olympiads in Informatics*. http://www.progolymp.se/1989
TUE-NL. *Former website of the International Olympiad in Informatics.* http://olympiads.win.tue.nl/ioi/
UG-RU (2016). Information about the 28th International Olympiad in Informatics*. Uchitelskaya Gazeta*, Number 4. (In Russian. Учительская газета). http://www.ug.ru/archive/63529

E. Kalinicenko is in his final year of studies at the University of Cambridge towards BA/MEng in Computer Science. He obtained four medals at IOIs, including a gold medal at the IOI 2011 and had obtained an honorable mention at IMO 2010. Participated in two ACM International Collegiate Programming Competition World Finals – one as a contestant from University of Latvia and one as an on-site coach of University of Cambridge team. He is a member of the Latvian jury for high school programming competitions and has been a deputy leader of Latvian delegation at IOI 2013.

M. Opmanis is a researcher at the Institute of Mathematics and Computer Science of the University of Latvia. He is one of the main organizers of Latvian Olympiad in Informatics, was deputy or team leader of Latvian IOI and Baltic OI teams. M.Opmanis was head of jury of Baltic Olympiad in Informatics at BOI 1996, 1999, 2004 and 2012. From 2012 till 2015 he was a member of IOI International Committee.

Olympiads in Informatics, 2016, Vol. 10, 177–194

DOI: 10.15388/ioi.2016.11

Distributed Tasks: Introducing Distributed Computing to Programming Competitions

Adam KARCZMARZ[1], Jakub ŁĄCKI[2], Adam POLAK[3]
Jakub RADOSZEWSKI[1,4], Jakub O. WOJTASZCZYK[5]

[1]*Faculty of Mathematics, Informatics and Mechanics, University of Warsaw*
ul. Banacha 2, 02-097 Warsaw, Poland
[2]*Department of Computer, Control, and Management Engineering Antonio Ruberti at Sapienza University of Rome*
via Ariosto 25, 00185 Roma, Italy
[3]*Department of Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University*
ul. Łojasiewicza 6, 30-348 Kraków, Poland
[4]*King's College London*
Strand, London WC2R 2LS
[5]*Google Warsaw*
Emilii Plater 53, 00-113 Warsaw, Poland
e-mail: a.karczmarz@mimuw.edu.pl, j.lacki@mimuw.edu.pl, polak@tcs.uj.edu.pl, jrad@mimuw.edu.pl, onufry@google.com

Abstract. In this paper we present *distributed tasks*, a new task type that can be used at programming competitions. In such tasks, a contestant is supposed to write a program which is then simultaneously executed on multiple computing nodes (machines). The instances of the program may communicate and use the joint computing power to solve the task presented to the contestant. We show a framework for running a contest with distributed tasks, that we believe to be accessible to contestants with no previous experience in distributed computing. Moreover, we give examples of distributed tasks that have been used in the last two editions of a Polish programming contest, Algorithmic Engagements, together with their intended solutions. Finally, we discuss the challenges of grading and preparing distributed tasks.

Keywords: programming contests, distributed tasks.

1. Introduction

When looking at major programming competitions, it is easy to notice that a large number of them are very similar in design. To name a few, the IOI, the ACM ICPC, Google Code Jam, TopCoder's algorithmic track, Facebook Hacker Cup, the CodeForces competitions and many more, are all focused on small, self-contained tasks, with automated

judging based on testing a program on a judge-provided set of testcases and mostly algorithmic in nature.

This model seems attractive both to the organizers and the participants. The organizers appreciate the very clear, non-subjective mechanism of judging and the automation of judging, which means the competition can scale out easily to a larger number of competitors. The participants also appreciate the automated judging (which means fast results) and the objective judgment criteria (which makes the competition more fair); additionally the entry barrier to such contests is pretty low, as the introductory-level tasks can be very simple.

If we consider programming competitions a way of educating future computer scientists and professional programmers, the model has its strengths, but also weaknesses. Some of these weaknesses come from the very nature of the small, self-contained tasks (which are a large part of the model's attractiveness): the participants do not learn about code maintainability and extensibility, they also do not learn anything about larger-scale system design. These weaknesses seem to be intrinsic to the model itself; and competitions that abandon the model moving to larger or less clear-cut tasks are frequently less successful (for example, TopCoder's Marathon Match track has over 10 times less registered participants than the Algorithm track).

However, there are also areas of programming expertise that the existing competitions do not teach, which are less intrinsically tied to the model of these competitions. In particular, the focus on algorithmic problems is not crucial to the model of the competition itself. Indeed, multiple authors already explored extending this classical model of competitions to other fields of computer science, for example visualization and precomputation (Kulczyński *et al.*, 2011), online algorithms (Komm, 2011), computer graphics and cryptography (Forišek, 2013).

1.1. *Distributed Programming*

Distributed (and cloud) computing has gained importance quickly in the recent years. The computing giants of today – Google, Amazon, Facebook and others – do not operate on the enormous mainframes that dominated computing in the past, but on networked farms of smaller servers. This is a model of computing that is not inherently in conflict with the algorithmic programming contest model, but is not taught by the dominant competitions of today; students that gained most of their programming skills through programming competitions will be totally unfamiliar with even the basic paradigms of distributed computing like the MapReduce framework (Dean and Ghemawat, 2008) or the CAP theorem (Fox and Brewer, 1999).

In this article we present a framework for running a contest focusing on distributed algorithms, developed by engineers in Google's Warsaw office in collaboration with the University of Warsaw, and show sample problems used in the Algorithmic Engagements (*Potyczki Algorytmiczne* in Polish) contest ran by the University of Warsaw. The same framework is used at the recently introduced Google's Distributed Code Jam competition.

1.2. *Designing a Distributed Programming Contest*

The primary focus of our design was simplicity from the contestant's point of view. The introduction of a new programming paradigm, likely unfamiliar to most participants, is clearly a challenge for the contestants, and we aimed at making the transition as smooth as possible.

Thus, the basic interface is similar to a programming contest like IOI. The participant submits a single program, which is compiled and executed by the framework. The same program is run on every *node* (computer) available to the participant.

Obviously, the nodes need to be able to communicate in order to collaborate in computing. We decided on a protocol based on simple message-passing ("send this array of bytes to this node"). The message passing methods are available to the program through a library that is common for all problems, and provided by the framework.

We also considered using an RPC-style interface. This, however, is more complex on an API level. A standard approach to RPC is that the programmer has to declare an interface (which will contain the remotely-callable methods). The server will just implement the methods of this interface. On the client side, the infrastructure needs to provide a way to generate a "stub" connected to some particular server; this stub will automagically have the method calls autogenerated. The language would have to provide some way to annotate that the interface is to be treated as a "remote" interface, increasing the complexity of the infrastructure implementation, and meaning more "magic" happening under the hood, which – in our perception – decreases the comprehensibility of the system. For instance, the "stub implementation" would need to make a choice of deep versus shallow copying of the arguments, each choice potentially leading to confusion for the participants.

Let us now present the functions of our library, together with their declarations in C++.

First, the library provides a function that returns the number of nodes M on which the solution is running, and the index (in the range $[0,\ M\ -\ 1]$) of the node on which the calling process is running.

```
int NumberOfNodes();
int MyNodeId();
```

The library maintains in each node a message buffer for each of the M nodes, which represents messages that are to be sent to this node. Messages are added to the buffer through the `Put`-methods.

```
// Append "value" to the message that is being prepared for
// the node with id "target". The "Int" in PutInt is
// interpreted as 32 bits, regardless of whether the actual
// int type will be 32 or 64 bits.
void PutChar(int target, char value);
void PutInt(int target, int value);
void PutLL(int target, long long value);
```

There is also a method that sends the message that was accumulated in the buffer for a given node and clears this buffer. This method is non-blocking, that is, it does not wait for the receiver to call `Receive`, but returns immediately after sending the message.

```
void Send(int target);
```

The following function is used for receiving messages.

```
int Receive(int source);
```

The library has a receiving buffer for each remote node. When you call `Receive` and retrieve a message from a remote node, the buffer tied to this remote node is overwritten. You can then retrieve individual parts of the message through the `Get`-methods. This method is blocking – if there is no message to receive, it will wait for the message to arrive.

You can call `Receive(-1)` to retrieve a message from any source, or set `source` to a number from $[0,\ M-1]$ to retrieve a message from a particular source. `Receive` returns the number of the node which sent the message, which is equal to `source`, unless `source` is `-1`.

Finally, for reading the buffer of incoming messages, the following three methods are provided.

```
// The "Int" in GetInt is interpreted as 32 bits, regardless
// of whether the actual int type is 32 or 64 bits.
char GetChar(int source);
int GetInt(int source);
long long GetLL(int source);
```

Each of these methods returns and consumes one item from the buffer of the appropriate node. You must call these methods in the order in which the elements were appended to the message (so, for instance, if the message was created with `PutChar`, `PutChar`, `PutLL`, you must call `GetChar`, `GetChar`, `GetLL` in this order). If you call them in a different order, or you call a `Get`-method after consuming all the contents of the buffer, the behaviour is undefined.

The serialization we decided to use is very basic, compared to models like Java's serialization mechanisms, Python's "pickle" or even Google's protocol buffer language. Again, we preferred to err on the side of simplicity, to minimize the entry barrier – this simple language turns out to be easy enough for the simple concurrency required to solve our problems, and is more straightforward to understand, both in terms of "how big will be the serialized message", and "what is actually serialized" (this, again, is the deep vs shallow copy question).

For correctness of execution, we chose what seemed the most natural model. The backend guarantees failure-less execution on all nodes, and requires all the instances of the program to execute correctly, within the specified time and memory limits. Note that the time used by the program is measured from the moment when the instances start until

all instances have finished execution. This assumption is justified with the following example involving two machines. The first one spends second of CPU time and then sends a message to the second one. The second machine first waits for a message from the first machine and then uses one second of CPU time. The total time used by the solution is then roughly two seconds, although each instance used only one second of CPU time.

In most of programming competitions, the programs access the input data by reading from standard input. (TopCoder's Algorithm contest breaks out of this by providing the input as an argument to the function the contestant is supposed to write.) However, a large fraction of the interesting distributed problems admit a solution that runs in $O(N/M)$ time, where N is the size of the input, and M is the number of nodes available to the contestant. This implies, in particular, that no node can afford to read the whole input (since that alone would take $O(N)$ time). Standard input is only accessible in a linear fashion, so it is not feasible for providing very large input data.

The approach we chose is what is typically done for interactive tasks on competitions like IOI (see, e.g., Chávez, 2015) – each problem with large inputs defines a set of input-access methods that are available to the program, similarly to the message-passing interface. These methods are guaranteed to return the same values on all nodes (so, each node has access to the same view of the input). Additionally, we provide upper bounds on the execution time of a single call to the input-providing methods.

Output is much simpler to handle (as it is often much smaller), so we went with the standard programming contest practice of expecting the output to be provided via standard output. We expect exactly one node to produce the output, while the others should not produce anything. This is a somewhat arbitrary decision (we could equally well have all the nodes output the exact same data to the standard output), however, as many solutions in practice have some sort of a "master" node that aggregates the work of the other nodes, it is convenient to the contestant to have one node output the result of the computation, and for the infrastructure not to prescribe which of the nodes it is.

As for the number of nodes we run the contestants' solutions on, we chose $M = 100$. This is large enough that a speedup by a factor of M is big enough to offset the extra time needed for inter-node communication, and yet small enough that providing that many nodes for judging is actually feasible.

The final issue that needs to be considered is the amount of data sent during the communication between the nodes, both in terms of the number of individual messages sent, and the size of those messages. Obviously, the limits here will be dependent on the infrastructure we run the contest on. Benchmarks on our framework show that a single message will take roughly 3–5ms (split between the processing time in the sender, the actual network latency and the processing time on the receiver). This number will be constant for messages from negligible to tens of kilobytes, and start growing linearly when the message size goes into hundreds of kilobytes.

While in the end, to fine-tune a solution the contestant will need to understand these patterns (for this purpose we provided the results of a few benchmarks to the contestants), we wanted the basic usecase not to require dealing with the calculations. To this end, we introduced an upper limit on the total number (around 1000) and size (a few megabytes) of messages a single node can send within the time limit, which roughly

corresponds to the total throughput it could achieve if it spent all its time on communication. This is a clear and concise way of informing the participants that if their solution stays well within those bounds, its performance should not be significantly affected by the communication overhead.

1.3. *Distributed Contest Judging Infrastructure*

From a research perspective, the most interesting challenge in preparing such a distributed programming contest is defining the exact programming model from the contestants' point of view. However, when doing it in practice, there is also a considerable amount of engineering effort involved in providing the infrastructure to run such a contest.

In the process of preparing problems the most interesting new challenge is writing a library that provides the input data. While in a standard programming contest the input is just a text file, and can be generated offline in an arbitrary fashion and almost arbitrary time, in the distributed contests the requirements are much more strict. The input-providing library needs to satisfy the following requirements:

- Access to an arbitrary element.
- Consistency across nodes, and across accesses.
- Access times on the order of 100ns.
- Ability to serve data with total size on the order of 10GB or more.

The requirements stem from the input model we chose. In particular, if we want to allow $O(N/M)$-time solutions, with runtimes on the order of 1–5 seconds, we need the input read by one node to be on the order of at least 10^7 items (which means 10^9 items in total – if each item is, say, two 64-bit integers, we get a total of 16GB), and we need the node to be able to read these 10^7 items within 1 second (so that the input reading does not dominate the computation).

The access times coupled with the data size mean it is infeasible to pregenerate all the input data – 10GB is too much to conveniently store in memory, while disk access (and even SSD access) is too slow. Thus, the input data is generated on the fly. This requires using pseudo-random generators that generate a sequence of numbers, and provide consistent and fast access to each element of the sequence (e.g., the CityHash family of functions).

In the judging system, the challenge is the scale. Judging a single testcase for a single solution requires, typically, 100 virtual machines. So, as an example, a deployment of 900 virtual machines on Google Compute Engine were used as the backend for the Online Round of Google's Distributed Code Jam. We think it is interesting that cloud computing, which is making distributed computing important as a topic of programming competitions, is also making distributed programming competitions much easier to organize: instead of buying physical hardware to support such a competition, it is easy to rent virtual machines and pay by the minute.

However, the real challenge in setting up a new competition type is in finding attractive problems: ones that challenge the contestants' creativity and problem-solving skill,

without requiring significant domain knowledge (in this case – knowledge of distributed programming paradigms), and that are fun, but not tedious, to implement, once you have the correct set of ideas. In the rest of this paper, we provide examples of problems that – in our opinion – satisfy these requirements.

We start with a simple task which lets us demonstrate the basics of the framework from the contestant's point of view. The remaining tasks were presented during the Algorithmic Engagements contest; their authors are: Jakub Łącki, Jakub Wojtaszczyk (task "Workshop"); Jakub Łącki (task "Assistant"); Adam Karczmarz (task "Sabotage").

2. Sample Task "Divisors"

In this task we are to count the number of divisors of a given positive integer $n \leq 10^{18}$.

Input

In this task the input data is provided via the standard input. The only line of the standard input contains n.

Output

Your program should print exactly one line to the standard output containing one integer: the number of divisors of n.

2.1. *Solution*

The easiest (sequential) solution that we can come up with is to check all candidates for a divisor d up to $\sqrt{n}$ and their counterparts of the form $\frac{n}{d}$. A sample C++ code follows.

```
int main() {
  long long n;
  int divisors_num = 0;
  cin >> n;
  for (long long d = 1; d * d <= n; ++d) {
    if (n % d == 0) {
      ++divisors_num;
      if (n / d != d)
        ++divisors_num;
    }
  }
  cout << divisors_num << endl;
}
```

Probably this is not the fastest sequential solution for this problem. We will, however, focus on how to speed it up by performing the computations using M nodes (machines).

A natural idea is to partition the set of possible divisors and let each of the machines look through each of the parts.

One way of performing this partition is as follows: machine 0 gets to check the candidates 1, $1 + M$, $1 + 2M$, ..., machine 1 gets to check the candidates 2, $2 + M$, $2 + 2M$, ..., etc. To make it work, it suffices to change the for-loop from the above code to:

```
for (long long d = 1 + MyNodeId(); d * d <= n; d +=
  NumberOfNodes()) {
```

In the end we need to aggregate the partial results. Let us select any of the machines (say, machine 0) as an aggregator. We just need to add code to be executed on each of the machines that sends the partial results from machines with positive numbers to the machine number 0.

```
  if (MyNodeId() > 0) {
    PutInt(0, divisors_num);
    Send(0);
  } else {  // MyNodeId == 0
    for (int node = 1; node < NumberOfNodes(); ++node) {
      Receive(node);
      divisors_num += GetInt(node);
    }
    cout << divisors_num << endl;
  }
}
```

The final solution works in $O(\sqrt{n} / M)$ time, i.e., this is the maximum of the time complexities of the instances running on each of the machines.

3. Task “Workshop” (2014)

During an algorithmic workshop students are sitting in a circle and solving problems. Whenever someone comes up with a solution to a problem, he or she shares the idea with his or her two neighbours (which takes exactly one minute), and then they pass it to their neighbours (which also takes exactly one minute), and so on.

If Johny comes up with a brilliant solution at quarter to twelve, what time will Chris hear about it? How many minutes does it take for a solution to reach from Kate to Tom? That is the kind of queries your program has to answer.

Input

The input data is provided by an interactive library. Your program can call six functions from the library:

- `int NumberOfStudents()`: returns the number n of students participating in the workshop ($3 \leq n \leq 10^9$). The students are numbered with consecutive integers from 1 to n.
- `int FirstNeighbour(int i)`: returns the number of the *first neighbour* of the i-th student ($1 \leq i \leq n$). The students have a hard time distinguishing left from right, therefore they prefer to call their neighbours in the order of their numbers. That is, the *first neighbour* of a given student always has a number smaller than his of her *second neighbour*.
- `int SecondNeighbour(int i)`: returns the number of the *second neighbour* of the i-th student ($1 \leq i \leq n$).
- `int NumberOfQueries()`: returns the number m of queries your program has to answer ($0 \leq m \leq 200$). The queries are numbered with consecutive integers from 1 to m.
- `int QueryFrom(int i)`: for the i-th query ($1 \leq i \leq m$), returns the number of the student who came up with a solution.
- `int QueryTo(int i)`: for the i-th query ($1 \leq i \leq m$), returns the number of the student willing to know when he or she is going to hear the solution.

Output

Your program should print exactly m lines to the standard output. The i-th line should contain the answer to the i-th query, i.e., the number of minutes it takes for a solution to reach from one student to the other.

3.1. *Solution*

In the problem a cycle is specified using an oracle which for a given vertex returns its two neighbours. The neighbours are returned in an order that is not necessarily consistent with the order of the vertices on the cycle. A number of queries are given, each consisting of two vertices, and the task is to compute for each query the length of the shortest path between the two vertices.

The first step of the model solution is to select a subset of vertices, which we call *checkpoints*. The checkpoints include all the vertices which are part of any query and additionally some number of randomly selected vertices. Later we discuss how to choose this number. After the checkpoints have been selected, each checkpoint is randomly assigned to some machine (node). All the random choices are made with a deterministic pseudorandom number generator so that all machines select the same checkpoints without needing to communicate.

In the second step each machine processes the checkpoints assigned to it. Starting from a checkpoint the process running on the machine traverses the cycle in both directions until it reaches (at both ends) any other checkpoints (they might be assigned to a different machine). While traversing the cycle, the process counts the number of visited vertices. Finally, it sends to the first machine a list of statements of the following form: the distance between the checkpoints a and b equals d and there is no other checkpoint between them.

The third step is run only on the first machine. First, it receives messages from all the other machines. Using information from the messages, the machine computes the order of checkpoints on the cycle and the distance between each two neighboring checkpoints. After linear-time processing of this information, it is easy to answer each query in constant time.

We are left with the problem of choosing the number of checkpoints. Recall that M denotes the number of machines. Each machine processes a random fraction of $\frac{1}{M}$ of the checkpoints which makes the expected running time of each machine $O(\frac{n}{M})$. However, from a theoretical point of view, this kind of a statement is worthless. Consider an imaginary situation in which we need to perform computations that take $\Theta(T)$ total time, and we pick a random machine to perform all of it. Then each machine spends $\Theta(T)$ time with probability $\frac{1}{M}$, which gives exactly $\Theta(T/M)$ expected time for each machine, just like in the case of our solution. At the same time, we are interested in the running time of the *slowest* machine, which is clearly still $\Theta(T)$.

For this reason, we study the performance of our solution experimentally. Luckily, the running time depends mostly on the size of the input data, not on its structure. Our simulations show that with M randomly selected checkpoints, the longest running machine uses $O(\frac{n \lg M}{M})$ time, compared to $O(\frac{n}{M})$ expected time, which is consistent with a theoretical analysis in (David and Nagaraja, 2003, p. 135). It is possible to reduce the variation between machines by increasing the number of checkpoints. In practice, our solution which always selects $10\ 000$ checkpoints is about 1.5–2 times faster than the one that selects M checkpoints.

3.2. *Tests*

It seems that virtually any nontrivial test is sufficient to distinguish solutions based on an incorrect algorithm. However, a bit more care is required to distinguish solutions that are correct but may be too slow – e.g. a variant of the model solution that selects as the checkpoints only the queries endpoints and the vertices $1, 2, \ldots, M$ instead of randomly selected vertices. To make such solutions exceed the time limit we need to pay attention to keep large contiguous fragments of the cycle without any query vertex and leave a large fragment of the initial $1, 2, \ldots, n$ cycle around the vertex 1 unaltered. The remaining part of the cycle is permuted either by performing a circular shift on the binary representations of vertex numbers or by xor-ing them with some fixed number. This method allows $O(1)$-time calculation of vertex neighbours and produces a cycle looking sufficiently random to make it difficult to come up with a clever incorrect solution exploiting this particular structure of the test.

4. Task “Assistant” (2014)

The life of an assistant is not easy. Not only did the professor order him to write a terribly long review, she also requested some corrections today.

Pushing keys of a keyboard is very tiring, so the primary goal of the assistant is to push keys as few times as possible, while correcting the review. The keyboard that he is using with a single click allows him to delete a character in the review, change a character to a different one or insert one character anywhere in the review.

To make things worse, the assistant has a very peculiar sense of esthetics. He likes letters from the beginning of the alphabet (like `a`, `b` or `c`), but is disgusted by the letters from the end of the alphabet (in particular, `y` and `z`). Each time he presses a key and changes a letter that comes earlier in the alphabet to a letter that comes later (for example, `m` to `p`), he suffers an esthetic shock, which is devastating for him. Because of that, the secondary goal of the assistant is to minimize the number of such changes.

Input

The first line of the standard input contains two integers k and l ($1 \leq k, l \leq 100\ 000$), that specify the lengths of the first and the second version of the review. The following two lines contain the two versions themselves. Each review consists only of lowercase letters.

Output

Your program should output a single line containing the minimal number of keyboard presses that the assistant has to perform in order to correct the review, followed by the minimal number of esthetic shocks that he will suffer.

4.1. *Solution*

The problem considered in this task is a variant of the well-known *edit distance* problem. Our solution will refer to the classical dynamic programming approach to this problem, which has been described in a number of textbooks (see, e.g., Cormen *et al.*, 2009). The solution that we obtain can be easily extended with minimizing the number of esthetic shocks. In short words, it suffices to, instead of storing only the edit distances, store integer pairs that describe the edit distance and the number of esthetic shocks, and compare them lexicographically.

Denote by $a_1, \ldots, a_k$ the characters in the first version of the review and by $b_1, \ldots, b_l$ the characters in the second version. Our goal is to compute a two dimensional $k \times l$ matrix D, where $D(i, j)$ contains the minimum number of changes that the assistant has to perform in order to change $a_1, \ldots, a_i$ into $b_1, \ldots, b_j$. Just like in the edit distance problem, the values $D(1, \cdot)$ and $D(\cdot, 1)$ can be computed in a straightforward way, whereas for $2 \leq i \leq k$ and $2 \leq j \leq l$, $D(i, j)$ can be computed in constant time, given $D(i-1, j)$, $D(i, j-1)$ and $D(i-1, j-1)$.

Assume that the topmost row of matrix D contains elements $D(1, \cdot)$ and the leftmost column contains $D(\cdot, 1)$. Partition the matrix into M *stripes* consisting of l/M consecutive columns (for simplicity, we assume that l is divisible by M), where M is the number of machines available. Each machine is responsible for filling in the entries of D in one stripe. Let the i-th machine (for $i \in \{1, \ldots, M\}$) be responsible for the i-th stripe from the left. See Fig. 1 for illustration.

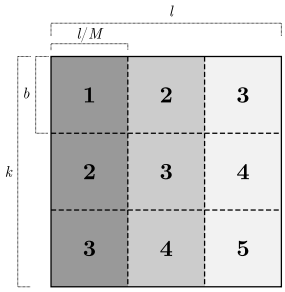


Fig. 1. The process of computing the matrix D. Stripes assigned to different machines have been marked with different shades of gray. The numbers inside the matrix specify the phase number, when the respective part of the matrix is computed (see below).

The first machine fills in the first (leftmost) stripe, starting from the topmost row. Consider the rightmost column in the first stripe. Observe that the contents of this column is everything the second machine needs to know, in order to fill in its stripe. The entries in the rightmost column of the first stripe are filled in by the first machine from top to bottom, and as they are being computed, the first machine sends them to the second machine. The second machine then fills in its stripe and sends the contents of the cells in the rightmost column of its stripe to the third machine, and so on.

The correctness of this approach should be clear. However, we need to improve it a little bit, in order to make it efficient. Clearly, each machine requires $O(kl/M)$ time to fill in its cells. However, all machines (except for the last one) send k messages, each containing a single number. This may be very inefficient, but can be fixed easily, as a machine may send the contents of cells in batches, each containing b numbers, thus reducing the number of messages to $\lceil k/b \rceil$. This obviously does not impact the running time of each machine.

However, there is one more efficiency aspect that we should take care of. Namely, we need to assure that the machines do not wait long for the numbers they need to have in order to perform computation. If each machine sends the contents of the rightmost column *after* filling in its entire stripe (i.e. sends batches of $b = k$ messages), then our solution becomes essentially sequential. On the other hand, we know that $b = 1$ is also not a good choice, for performance reasons.

Let us analyze how to pick a good value of b. For the analysis, assume that all the machines are perfectly synchronized, that is, in each time unit a machine can fill in exactly one cell of its stripe (or wait for the data it needs to continue working). Consider now a phase of exactly bl/M time units. Since each stripe has width l/M, the first machine sends the first message to the second machine exactly after the first phase. In the second phase, the second machine starts working (fills in the first b rows of its stripe) while the first one keeps on working on the following rows. From the second phase, the second machine no longer needs to wait, as it receives the necessary data exactly the moment when it needs it. In general, the i-th machine starts to work in the i-th phase after $(i-1)\,bl/M = O(ibl/M)$ time units. Thus, the M-th machine starts after $O(bl)$ time units and then works for $O(kl/M)$ time units. Hence, all the machines finish within $O(bl + kl/M)$ time units and send $O(k/b)$ messages each.

By setting $b = \lfloor k/M \rfloor$ we assure that the waiting time is dominated by the computing time, which means that our solution parallelizes the single-machine solution in a perfect way. At the same time, the total number of messages sent is moderate ($O(M^2)$).

5. Task "Sabotage" (2015)

The city of Megabyteopolis was built upon a large lake and consists of a number of isles connected with bridges. The bridges may run above other bridges.

A group of saboteurs wants the current president Byteasar not to be reelected. They plan to impact the public opinion by exposing Byteasar's administration's helplessness in the case of a major emergency. Specifically, they decided to blow up one of the bridges (they cannot afford blowing up more). The sabotage could be considered successful only if there was no other way between the isles previously connected by the destroyed bridge. Your task is to find the number of bridges that the saboteurs should consider when working out the details.

Input

- `int NumberOfIsles()`: returns n ($1 \leq n \leq 200\,000$) – the number of isles constituting the city of Megabyteopolis. The isles are numbered 0 through $n-1$.
- `int NumberOfBridges()`: returns m ($1 \leq m \leq 10^8$) – the number of bridges in the city. The bridges are numbered 0 through $m-1$.
- `int BridgeIntA(int i)`: returns the first isle connected by the bridge i.
- `int BridgeIntB(int i)`: returns the second isle connected by the bridge i.

Output

The output should contain a single integer – the number of bridges whose blowup could result in the sabotage being considered successful.

5.1. *Solution*

In this task we are asked to solve a basic graph problem: for a given undirected graph $G = (V, E)$ we need to compute the number of *bridges*. A bridge is defined here as an edge of G whose removal results in an increase of the number of connected components of G. Denote by $B(G)$ the set of bridges of G. A textbook algorithm (e.g., Sedgewick, 2002) for computing $B(G)$ is based on an extension of the *depth-first search* (DFS) algorithm and runs in $O(n + m)$ time. The number of vertices in our graph is quite small, i.e., the bound on the order of 10^5 is typical for graph tasks even in the traditional, non-distributed setting. The number of edges m in our case can be, however, much larger.

Unfortunately, DFS is not an algorithm that can be parallelized easily. Nevertheless, we do not need to entirely abandon the idea of using DFS: our strategy is to use the multiple machines to reduce our problem instance to an instance with only $O(n)$ edges. In such a reduced instance, we use DFS to find the bridges.

Definition 1. *Consider a graph $H = (V, E)$. We define a* bridge certificate *of H to be a set $X \subseteq E$ such that for any $Y \subseteq V \times V$, $B((V, E \cup Y)) = B((V, X \cup Y))$.*

As a result, replacing a subset of edges of G with its bridge certificate does not affect the set of bridges of G. We are going to use the bridge certificates to detect and remove edges of G. The following lemma describes a construction of a bridge certificate. For completeness we give its proof in the Appendix.

Lemma 1. *Let $H = (V, E)$ and $n = |V|$, $m = |E|$. Then there exists a bridge certificate X of H such that $|X| \leq 2n$ which can be computed in $O(n+m)$ time.*

By Lemma 1, we can take any subset E' of edges of G, find a bridge certificate X of (V, E') and replace E' in G with $X \subseteq E'$ in $O(n+|E'|)$ time. We call this step a *reduction* with respect to E'.

It turns out that we can easily perform the reductions in a parallel fashion. For simplicity, first assume that we have only two machines, i.e., $M = 2$. We partition the edge set E into two sets E_0, E_1 of roughly equal size. The machine i, for $i = 0, 1$, performs the reduction step on the set E_i, obtaining a certificate X_i of size at most $2n$, in $O(n+|E_i|)$ time. Next, machine 1 sends the set X_1 to machine 0. In the last step, machine 0 runs DFS to find the set $B((V, X_0 \cup X_1))$. This, however, takes only $O(n)$ time, as $|X_0 \cup X_1| \leq 4n$. By the definition of a certificate,

$$B(G) = B((V, E_0 \cup E_1)) = B((V, X_0 \cup E_1)) = B((V, X_0 \cup X_1)).$$

This concludes that indeed this approach finds all the bridges of G.

In order to develop a distributed algorithm using $M > 2$ machines, we perform multiple reduction phases. In the first phase, the set of edges is partitioned among the M machines, and each machine computes a certificate of the edges assigned to it. In the following phases the certificates are merged in pairs: two certificates produced in the previous phase are sent to a machine that takes their union and computes the certificate of the resulting graph.

Let us describe this process formally. Assuming that the machines are numbered 0 through $M-1$, we split the input edge set E arbitrarily into M parts $E_0, \ldots, E_{M-1}$, each of size $O(m/M)$. Our distributed algorithm runs in $K = \lceil \log_2 M \rceil + 1$ phases numbered 0 through $K-1$. In the k-th phase ($k = 0, \ldots, K-1$) only the machines with identifiers j divisible by 2^k are active and actually do perform some work. With each active machine j we associate two sets Y_j, X_j ($Y_j, X_j \subseteq E$) whose contents depend on the phase number k. Before the k-th phase:

- Y_j is a bridge certificate of the graph

$$H_j = (V, E_j \cup E_{j+1} \cup \ldots \cup E_{j+2^k-1}).$$

  In the above we set $E_{j+l} = \emptyset$ if $j + l \geq M$.
- If $k = 0$ then $Y_j = E_j$. Otherwise, $|Y_j| \leq 4n$.

After the k-th phase the set X_j is a bridge certificate of H_j and $|X_j| \leq 2n$. Note that it follows that after the phase $K-1$, X_0 is a bridge certificate of G and $|X_0| = O(n)$.

At that point the machine 0 runs DFS on (V, X_0) to compute the set of bridges of G in $O(n)$ time.

It remains to show how to implement the phases so that the invariants imposed on the sets Y_j, X_j are satisfied. Before the first phase we set $Y_j = E_j$. Assume that the phase $k-1$ has been completed. We perform a reduction of Lemma 1 on the set Y_j in order to obtain the set X_j. This takes $O(n + |Y_j|)$ time, which is $O(n)$ for $k > 0$ and $O(n + m/M)$ if $k = 0$. The last step is to initialize the sets Y_j before the next, $(k+1)$-th phase. To do that, for each j divisible by 2^{k+1}, we set $Y_j = X_j \cup X_{j+2^k}$ if $j + 2^k < M$ and $Y_j = X_j$ otherwise. As for all l, $|X_l| \leq 2n$, clearly we now have $|Y_j| \leq 4n$. To implement this step, the machine $j + 2^k$ sends the entire set X_{j+2^k} to the machine j. This requires a single message of $O(n)$ bytes. Fig. 2 depicts the phases of our distributed algorithm.

In each phase every machine remains idle or sends $O(n)$ bytes, or receives $O(n)$ bytes. Consequently, the first phase takes $O(n + m/M)$ time on each machine and each of the $K-1$ remaining phases runs in $O(n)$ time on each machine. Thus, the time complexity of this solution is $O(m/M + n \log M)$.

5.2. *Tests*

The library providing the test data had to be robust enough to serve graphs with large edge sets and nontrivial 2-edge-connected components (i.e., connected components of G formed after removing all the bridges), given limited time and space. It seems that a hard test case is a graph with a maximum number of edges and possibly large number of bridges. In such a case at least some of the 2-edge-connected components should be very dense.

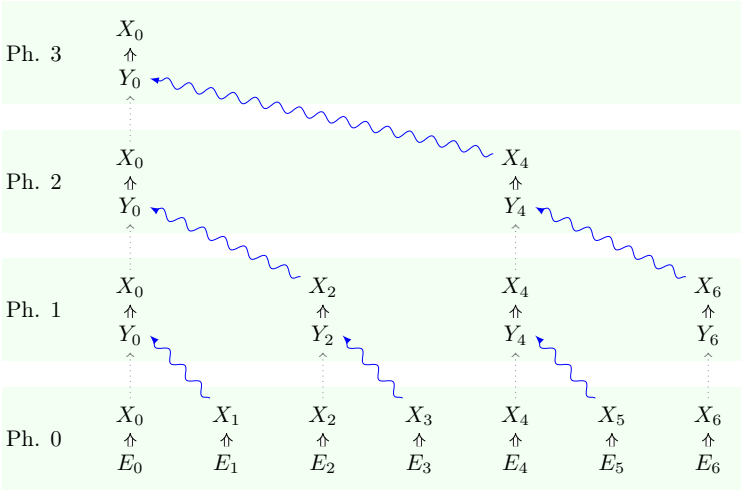


Fig. 2. The phases of the distributed algorithm when $M = 7$. In this case $K = 4$ phases are performed. The blue arrows illustrate the communication between the machines in the corresponding phases.

The infrastructure for generating test graphs provided a general graph interface along with a few specialized implementations (a vertex, a path, a cycle, a clique, a pseudo-random graph, a set of loops) that could serve the edges in $O(1)$ time with constant space consumption, regardless of the graph size. As an example, a cycle on n vertices numbered 0 through $n-1$ can be represented with a single integer n: when asked for the cycle's i-th edge, we just return $(i, (i+1) \bmod n)$.

Such graphs could be then combined into larger and more sophisticated graphs by unions and direct sums and also extended by adding specified edges, which were typically used to ensure the desired structural properties of the served graph. For example, this allowed to easily generate a tree of size 100 with each vertex replaced with a random 2-edge-connected graph with 1000 vertices and between $100\ 000$ and $500\ 000$ edges. Such a graph had on the order of 10^7 edges in total, 99 bridges and could be represented with the number of bytes on the order of 10^2. At the highest level, the vertices were assigned random identifiers, whereas the list of edges was randomly permuted.

The size of the in-memory representation of each test cases was $O(n)$ per machine and the sophistication level of the served graphs was limited only by the need to return the requested edge in time on the order of 100ns.

6. Conclusions

We described a novel format of programming competitions, aimed at familiarizing students with an increasingly important area of computer science – design of distributed algorithms. While there is a considerable engineering effort involved in preparing the backend for such a competition, we hope that an increasing number of competitions (maybe including IOI in the future) will feature tracks or problems of a distributed nature, to reflect the industry's shift toward cloud-based and distributed computing.

References

Chávez, L.H. (2015). *libinteractive*: a better way to write interactive tasks. *Olympiads in Informatics*, 9, 3–14.

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

David, H.A. Nagaraja, H.N. (2003). *Order Statistics* (3rd Edition). Wiley.

Dean, J. Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM – 50th anniversary issue: 1958–2008*, 51(1), 107–113.

Forišek M. (2013). Pushing the boundary of programming contests. *Olympiads in Informatics*, 7, 23–35.

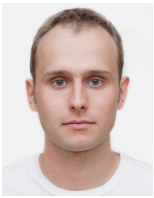
Fox, A. and Brewer E. (1999). Harvest, yield and scalable tolerant systems. In: *Proc. 7th Workshop Hot Topics in Operating Systems (HotOS 99)*. IEEE CS, 174–178.

Komm D. (2011). Teaching the concept of online algorithms. *Olympiads in Informatics*, 5, 58–70.

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

Sedgewick, R. (2002). *Algorithms in C++, Part 5: Graph Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston.

A. Karczmarz (1990), PhD student at Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland, jury member of multiple programming contests organized by University of Warsaw, coorganizer of Algorithmic Engagements 2015. In his research he focuses on algorithms and data structures.



J. Łącki (1986), postdoctoral researcher at Sapienza University of Rome, Italy, IOI Scientific Committee elected member, appointed chair for 2016, responsible for task selection at Algorithmic Engagements for many years, former head organizer of Polish Training Camp. In his research he focuses on graph algorithms.

A. Polak (1991), PhD student at Department of Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland, judge at the ACM Central Europe Regional Contest in 2012, 2013, and 2014. His research interests lie in algorithms, complexity theory, and computer vision.

J. Radoszewski (1984), assistant professor at Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland, and Newton International Fellow at King's College London, UK, chair of the jury of Polish Olympiad in Informatics, co–chair of the Scientific Committee of CEOI'2011 in Gdynia, former member of Host Scientific Committees of IOI'2005, CEOI'2004, BOI'2008, and BOI'2015. His research interests focus on text algorithms and combinatorics.

J.O. Wojtaszczyk (1980), Staff Software Engineer at Google, Warsaw, judge at the ACM ICPC World Finals in 2011, 2012, 2013, 2015 and 2016, coorganizer of the Google Code Jam since 2012, main organizer of the Distributed Code Jam. The primary focus of his engineering work is around cluster management.

Appendix: Proof of Lemma 1

Lemma 1. *Let $H = (V, E)$ and $n = |V|$, $m = |E|$. Then there exists a bridge certificate X of H such that $|X| \leq 2n$ which can be computed in $O(n+m)$ time.*

Proof. We compute the set X in the following way. First, compute some spanning forest F (we identify it with a set of its edges) of H using any graph search algorithm. This takes $O(n+m)$ time. Then, compute some spanning forest F' of $H' = (V, E \setminus F)$. Finally, set $X = F \cup F'$. Clearly, $|X| \leq 2n$ and $X \subseteq E$.

We now prove that X is indeed a bridge certificate of H. Let $Y \subseteq V \times V$. First, let us show that the graphs $H_1 = (V, E \cup Y)$ and $H_2 = (V, X \cup Y)$ have the same connected components. Clearly, if there exists a path $u \rightsquigarrow v$ in H_2 then a path $u \rightsquigarrow v$ exists in H_1, as H_2 is a subgraph of H_1. Conversely, if u and v are connected in H_1 by a path P, then any edge $(a, b) \in P \setminus Y$ such that $(a, b) \in E \setminus X$ can be replaced by a path $a \rightsquigarrow b$ contained entirely in F (recall that F is a spanning forest of H).

For a graph G, define $G - e$ as G with the edge e removed. Now assume that (u, v) is a bridge in H_1, i.e., $(u, v) \in B(H_1)$. Then, there is no path $u \rightsquigarrow v$ in $H_1 - (u, v)$. As $H_2 - (u, v)$ is a subgraph of $H_1 - (u, v)$, there is also no path $u \rightsquigarrow v$ in $H_2 - (u, v)$. Moreover, H_1 and H_2 have the same connected components and thus $(u, v) \in X \cup Y$. Thus, $(u, v) \in B(H_2)$ and consequently $B(H_1) \subseteq B(H_2)$.

Finally, suppose that $(u, v) \in B(H_2)$. Then, u and v are connected in H_2, but no path $u \rightsquigarrow v$ exists in $H_2 - (u, v)$. As H_2 is a subgraph of H_1, $(u, v) \in E \cup Y$. Let us show that no path $u \rightsquigarrow v$ exists in $H_1 - (u, v)$. Assume the contrary and let P be such a path. If $P \subseteq X \cup Y$, then P would be a $u \rightsquigarrow v$ path in $H_2 - (u, v)$, which is not possible. Hence, there exists some edge $(a, b) \in P$ such that $(a, b) \in E \setminus X \setminus \{(u, v)\}$. As $(a, b) \notin F$ and $(a, b) \notin F'$, there exist paths Q and Q' from a to b in both spanning forests F and F', correspondingly. At least one of these paths, say Q, does not contain (u, v). We may replace the edge (a, b) with the path $Q \subseteq X$. By replacing all such edges (a, b) with paths contained in X, we obtain a path $u \rightsquigarrow v$ in $H_2 - (u, v)$, a contradiction. Thus, $(u, v) \in B(H_1)$ and, consequently, $B(H_2) \subseteq B(H_1)$.

Olympiads in Informatics, 2016, Vol. 10, 195–205

DOI: 10.15388/ioi.2016.12

Reshaping Indonesian Students Training for IOI

M. M. Inggriani LIEM
Knowledge and Software Engineering Research Group
School of Electrical Engineering and Informatics, Institut Teknologi Bandung,
Jalan Ganesha 10, Bandung, Indonesia
e-mail: inge@informatika.org

Abstract. Indonesia has been participating at IOI (International Olympiad in Informatics) since 1995. This paper presents a result of qualitative study of Indonesian IOI participants. The supporting data are obtained from a questionnaire distributed to the IOI 2006 – IOI 2015 participants, interviews, training database, journals, and reports from the national training program. Main objective of the study is to investigate the suitability of our training program to IOI expectations. From 35 distributed questionnaires, we obtained 24 respondents. Without having prior knowledge in Computer Science, solid mathematical foundation, and algorithmic problem solving in formal education, the national training curriculum can only give the participants a foundation for general problem solving which is not enough for IOI due to increase of creativity, complexity and difficulty of IOI tasks. That is why Indonesia achievement is stabilized in bronze. Almost all of Indonesian IOI alumni are working or studying in the domain of informatics and still participating programming competitions after IOI. Most of all Indonesian IOI participants are studying in top universities and some of them are working in the worldwide prestigious IT companies.

Keywords: Indonesian IOI training program, qualitative research, IOI tasks.

1. Background

Amongst its objectives, IOI has two main objectives for the contestants: to give recognition to young people who are exceptionally talented in the fields of informatics, and to foster international relationships among them.

The first participation of Indonesia at IOI was in 1995, by the initiative of young lecturers of UI (Universitas Indonesia) who were studying in US. The training and selection of Indonesian participants was informal and voluntary, it then found its shape as an organization in 2004, two years after the first Indonesian National Science Olympiad (OSN). The support and sponsorship from the Indonesian Ministry of Education for IOI and other International Olympiad is important. But the policy of the government that aims to catch the potential candidates from all provinces of Indonesia becomes a problem. Indonesia has more than 4 million high school students spread into more than 17,000 islands, from remote area to big cities. The quality of education in remote area is

less in big cities like Java due to infrastructure gap. Students in big cities are more advantageous in education. Also, getting medals in OSN is considered prestigious and give good impact to the winners as well as to the school. Therefore, some schools in big cities (Jakarta, the capital, and some others province capitals) recognize the importance of the National Science Olympiad for their reputation, so that they are trying to get teachers or trainers. Most students who are interested in computer games became interested in programming, and by the support of their parents, they got private trainers for programming. This situation causes a broader gap.

The national selection process should keep a balance between "potential candidates" from remote areas and "ready to compete" candidates from the big cities. Up until now, Indonesia cannot be classified as a top performing country in the IOI. Nonetheless, its performance is improving and become more stable in bronze medals (Yugo *et al.*, 2014). The summary of Indonesian team achievement in IOI from 2006 to 2015 is figured in Table 1.

This study was conducted with aim to reshape Indonesian national training program so that improvement can be achieved by being stable in silver medals. We gathered information from the past Indonesian IOI participants by giving a questionnaire. We investigate what Indonesian IOI participants are doing after their experience in IOI. It is important for the IOI community to know about their alumni after 9–10 years.

2. Related Works

Our work was inspired by the usage of qualitative research in computer science education. Formerly, the qualitative research has been applied to social research. Applying it to science and technology is quite rare. Recently, this method became widely used for informatics in education. M. Knobelsdorf (2008) presented a research using qualitative method in teaching of programming. A. Theodoraki and S. Xinogalos (2014) presented a qualitative study on student's attitudes towards learning programming through

Table 1
Indonesian Participant Medal Achievement at IOI 2006 – IOI 2015

Year	Gold	Silver	Bronze
2006	0	1	0
2007	0	0	4
2008	1	0	3
2009	0	2	1
2010	0	2	1
2011	0	0	2
2012	0	1	2
2013	0	2	2
2014	0	0	4
2015	0	2	1

games. Hazzan *et al.* (2006) wrote about qualitative research in computer education. He also wrote about teaching and learning qualitative research by conducting qualitative research (Hazzan, 2014). In our case, this method is applied to the training and selection program of Indonesian Olympiad participants.

A survey done and reported by Nedkov *et al.* (2012) concerning the selection, preparation and participation of IOI teams of Bulgaria, Croatia, Latvia, Poland, and Slovakia. Those countries are the leading countries in the IOI. According to this survey, factors contributed to the successful participation are traditions, strong emphasis on mathematics in national education, targeted extra-curricular activities, early start and gaining experience by participating in competitions, systematics management and dedicated people, motivation and rewards.

3. Objectives

This study is conducted in order to obtain findings related to:

a. Factors that help the participants to get medal in IOI.
b. Fitness of Indonesian training topics to IOI tasks.
c. The influence of IOI to the career of IOI participants.

The first questions are the most important since one of the main targets of IOI participation is to win the competition that is getting medals, and the Indonesian education system has had bad results in mathematics in the PISA 2012 test. The Program for International Student Assessment (PISA) is a worldwide study by the Organization for Economic Cooperation and Development (OECD) in member and non-member nations of 15-year-old school pupil's scholastic performance on mathematics, science, and reading[1]. For winning a competition, a supporting curriculum is needed. The third question is trying to track the career of the Indonesian IOI participant, since life is going on after IOI.

4. Design Experiment

This paper analyse the data obtained from Indonesian IOI participants from 2006 to 2015. During these last ten years, Indonesia has sent 40 participants (IOI alumni), 5 of them have been participated twice. Therefore, we had 35 persons in the IOI during that period. From the 35 Indonesian IOI alumni, we obtained 24 responses in two weeks by contacting 3 alumni and asked them to contact the others. It makes over 70%. This is a proof that there is a close relation between the coaches and the alumni, as well as between alumni. Three years after IOI, they are still active in programming competition, as contestants in international competitions, Scientific Committee or Technical Committee on Indonesian National Training program and competition.

[1] http://www.oecd.org/pisa/keyfindings/pisa-2012-results.htm

The data are obtained from the following sources:

a. Indonesian National Training Program database, which are the participants, and also the training program archives (journal, schedule, report).
b. Questionnaires, via Google questionnaires as well as via email.
c. Literature reviews for getting information about IOI tasks and solutions.

5. Findings and Discussions

Based on the questionnaire, we identify some findings, which are described in the following sections.

5.1. *Effectiveness of the National Training Program*

In the beginning, Indonesia did not have a systematics training program. Talented high school students surround university faculty members were trained only a few weeks before the IOI. Since 2002, the Ministry of Education of Indonesia has initiated the National Science Olympiad (OSN), where programming is one of the competition subjects amongst 6 others (mathematics, physics, chemistry, economy, geology and astronomy). In OSN, 80 to 100 high school students that have passed the successive selection at the level of school, region, and then provinces are invited to come for a national competition held in one of the city in Indonesia. Each year, the scientific committee must run two concurrent programs: one for national selection (from school to OSN), while the other is a program for trainings and selections for the next IOI.

One cycle of trainings and selections of IOI participants takes one year consists of four phases (I, II, III, IV) training camps. Between two successive camps, candidates must join on-line mentoring and challenge, and also participate in other informatics contests. Thirty candidates of OSN winners enter the phase I. After three weeks of training, the scientific committee selects 16 top scorers to enter phase II. At the end of phase II, eight participants are selected for phase III. Finally, only four participants undergo phase IV becoming the official Indonesian IOI participants.

The organization of the training is getting better by having more IOI alumni and their involvement. Most of the alumni are graduated in informatics and then they work in worldwide prestigious information technology and software companies. Their experience both in competition, education as well as in their professional works helps to improve the training and selection programs. Some of the IOI alumni participate voluntarily in the scientific committee as well as the technical committee for the two concurrent programs. They become mentors and problem setters in both programs. They also set up an alumni organization (IA-TOKI) and have a close relationship with the scientific committee, though they are scattered all over the world. They are developing a contest management system which is used for on-line training program and some other regional/local programming competitions nowadays. The contribution of IA-TOKI to the national training program is significant.

Almost all of the respondents answer that the on-site training camps are very important and more effective than the on-line training:

1. They learn more from the mentors and from the peers during on-site training. Direct discussions are more effective than on-line learning.
2. They can focus more during the on-site training, compared to on-line training. The students in an on-site training are relieved from school so that they can fully concentrate on the IOI preparation. On the other hand, students must prioritize their school and daily tasks during on-line training.
3. They have a competitive environment during on-site training, especially in simulation sessions. Simulation trains the students how to manage their stresses, and builds a competitive sense by the presence of other students.
4. Being together with peers during on-site training for weeks made the students know each other better where they built a team spirit, and becomes friends.
5. They have a better internet connection.

There is one exception. One respondent stated that he prefer on-line training and self-studying, and learning from worldwide existing on-line competitions, books and on-line programming competition websites. It is then identified that his level is higher compared to his peers, so that he became bored during the on-site training, though he attended and completed the whole on-site training programs. He was the one that achieved a gold medal in IOI 2008.

5.2. *Other Contributing Factors*

Other factors that contribute the medal achievement are self-exercise, reading books, on-line resources, and other competitions that made them regularly connect to the world of competition.

When they are not in a training camp, 54% of the respondents were exercising themselves more than 10 hours per week, 17% only between 5 to 10 hours per week, 17% at least 3 hours per week, and 13 % exercise irregularly.

They also try to solve past IOI tasks, but not so much. 46% solve more than 10 tasks, 17 % between 5 to 10 tasks, 25% less than 5 tasks, and 8% does not try to solve in past IOI tasks, and 4% did not comment. Trying past IOI tasks seems not so interesting because they know that IOI tasks are very creative.

The following two figures (Fig. 1 and Fig. 2) illustrate those findings.

Other learning resources (Fig. 3) are websites and competition sites. 75% of the respondents state that learning from the websites such as Topcoder, Usaco and SPOJ are useful. 71% of them state that participation in competition is useful as preparation for IOI.

Learning from textbooks seems to be least favourite option for this young generation. Only 33% of the respondents were learning from textbooks. Only two textbooks are mentioned explicitly: Introduction to Algorithms (by Thomas H. Cormen, Charles E. And Leiserson), and Competitive Programming (by Steven Halim).

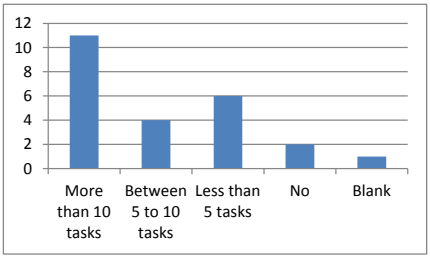


Fig. 1. Hours Spent by Indonesian IOI Participant for self-preparation.

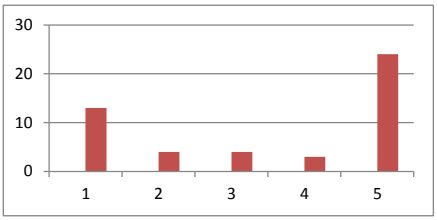


Fig. 2. Number of IOI Tasks exercised by Indonesian IOI Participant.

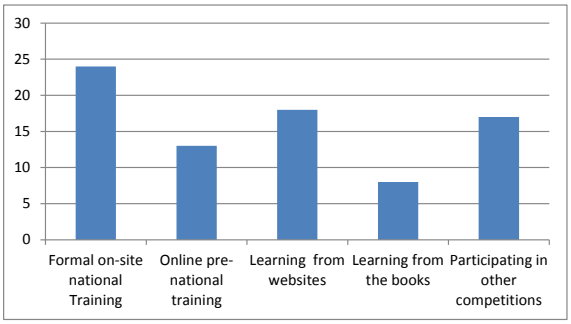


Fig. 3. Various Ways of Learning for IOI preparation.

Besides the national training program, all of the respondents participate regularly in the worldwide competitions, not only in the period of national training, but also few years after participating IOI. Competition becomes their hobbies, and they are getting more and more experience in that domain so that they become the national scientific members and trainers in a national training program or task contributors in prestigious international programming competitions. Their favourite competition websites are TopCoder, Codeforces, Usaco, APIO, COCI, CEOI, Joint, BEOI, Google Code Jam, ACM ICPC, and local competitions.

One of the alumni mentioned that the following ingredients are important for being successful in IOI: grit/ perseverance, time management and prioritization, attention to detail, creative thinking, the ability to recognize pattern on problems, experience with common tricks and avoiding common pitfalls, experience with a breadth of topics.

5.3. *Fitness of Training Topics to IOI Tasks*

A curriculum of the training program is progressing continuously since 2006, in three periods. During the first period (2006–2010), most of the trainers are the university faculty members. IOI alumni participation in the second period (2011–2013) and the third period (2014–2015) brings new colour to the program. The ambiance of programming competition becomes stronger. More than that, the creative problem solving exercises is introduced in addition to the classical training subject. The scientific committee is working together with IOI alumni closely for the training program subjects and tasks.

In the beginning, the phase I of training is dedicated to assure the basic foundation of CS (programming, advanced topics, while phase III is dedicated to simulation and the selection of four IOI participants. Phase IV is the final preparation before going to the IOI.

From one year to another, IOI tasks become more creative, unpredictable and difficult (Halim, 2013). All countries' training program must anticipate these changes and shifted accordingly. As a consequence, Indonesia decides to shift the basic skill gradually from on-site to on-line training by providing on-line course material in Indonesian contest management systems. The students should develop their self-learning capability. This solution works well for students living in big cities with good infrastructure as well as good internet reliability and availability, and good teachers are available. In addition to the shifting of basic skills in CS, exercise and simulations are designed to be harder and more challenging. Table 2 illustrates the different strategies of the three periods.

As illustrated above in Table 1, the achievement of Indonesian team is stabilized in Bronze medals. The year 2008 is a special year with one gold medal obtained by a prodigious student. The year 2011 is also exceptional with only one bronze obtained because of two reasons: a transition of the training method and too much given exercises that are not in the IOI style.

Table 2
Overview of Training Program Curriculum from 2006 to 2015

IOI year	Training Phase I	Training Phase II	Training Phase III	Training Phase IV
2006–2010	Basic programming Basic Exercises Simple simulation	CS topics, strategic problem solving Simulation	Advance Topics and Simulation	Final IOI preparation
2011–2013	Basic programming CS Topics, strategic problem solving, simulation	Advanced Topics and advanced problem Simulation	Advanced Topics and simulation	Final IOI preparation
2014–2015	CS Topics and Advanced Topics & strategic problem solving, simulation	Special advanced topics More Simulation	More Simulation, more ad-hocs problem	Final IOI preparation

This part is a deeper look to the subjects covered during the training programs, contains the analysis among the topics covered in the Indonesian training and IOI tasks.

As mentioned above, the Indonesian high school curriculum does not cover computer science as a compulsory subject, compared to other countries where the computer science concepts has been introduced since the earlier age, from 10–14 years (Dagienė and Futschek, 2010, Nedkov, 2012). Three phases of training, each lasts for 3 weeks can cover all the required topics suggested by IOI curriculum. However, topics such as strategic problem solving and thinking take time to be mastered. Many exercises are also needed in order to construct the pattern of problems as well as the pattern of solutions. These patterns are important for solving the creative IOI task.

The national training program topics covered the last version of the IOI syllabus, obtained from the website[2]. National Training topics exclude basic computer science and mathematics listed in the IOI syllabus and respondents made remarks regarding the importance of those topics (Fig. 4) for IOI.

In a questionnaire after IOI 2015, a list of topics is given. The country leaders were asked to rate the importance of them. In this study, we also try to explore the opinions of respondents regarding these topics, and we got the following results: Maximum flow, flow/cut duality theorem (67%); Strongly connected components, bridges and articulation points (54%); Heavy-light decomposition and separator structures for static trees (25%); Data structures for dynamically changing trees and their use in graph algorithms (54%); Topics in number theory (33 %); String algorithms (54%).

Additionally, IOI tasks and solutions from IOI 2006 to IOI 2015 were analysed. The respondents are asked to mention the most interesting, the most difficult, and the most memorable IOI tasks. From these, we obtained 74 tasks. These tasks are then cross checked to the topics listed.

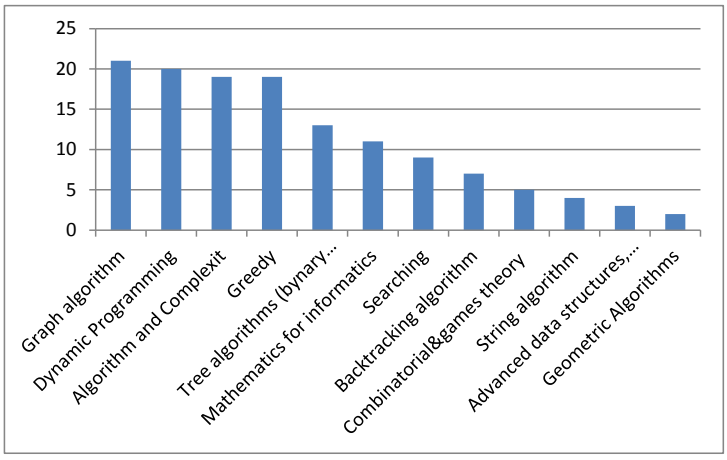


Fig. 4. Importance of Training Topics.

[2] http://ksp.sk/~misof/ioi-syllabus/

The summary of this cross check and analysis illustrate the applicability of our training topics for solving IOI tasks:

- The importance of the topics ranked in figure Fig. 4 has high correlations with the technics needs to solve the IOI tasks.
- Graph, DP, Greedy, sort and search are very important, but they alone are not enough, since the variation and specific condition of the graph can improve the algorithm.
- Algorithm complexity is important for measuring the performance of algorithm in order to solve more difficult subtasks.

5.4. *Influence of IOI to Participants Careers*

IOI influences strongly to the Indonesian participants careers. By winning medals in IOI, they can enter easily to top Indonesian universities in Informatics, even entering the top universities in the world. All of the participants are studying in the fields of informatics as shown by the following graphics in Fig. 5.

Most of the Indonesian alumni of IOI 2006–2008 have graduated from bachelor degree and now working or studying PhD level. Some of them are founders or CTO at top start-up Indonesian companies in the domain of software, offers internship and employees their juniors. Some medallists are working for prestigious companies in the US and UK as software developers. The younger alumni are now studying in bachelor or master program in informatics. Only 2 of 24 participants are studying in other domains: one is studying in medical faculty, and another is studying in the first common year (he is intended studying in informatics). This finding shows that IOI is a starting point for their study and then their career in informatics. The IOI gave them motivation for studying and working abroad. IOI medal is a free ticket to enter top universities and obtaining scholarships.

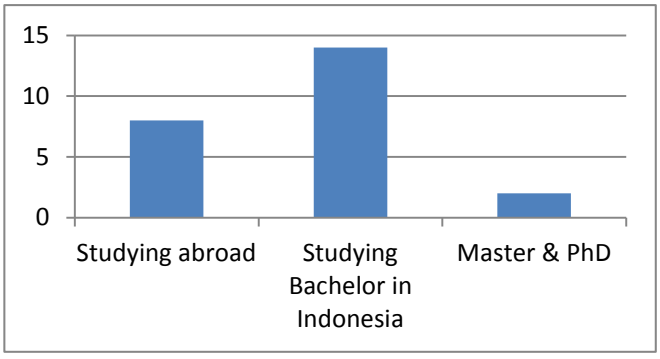


Fig. 5. Indonesian IOI Alumni Tracking.

6. Conclusions

This paper presents a study of Indonesian IOI participants from 2006 to 2015, where the data come from questionnaire distributed to them, interviews, national training database, journal of national training program, and from training reports. The study aimed to identify the appropriateness of our national training program to IOI participation, since we have only 1 gold in 2008, and in the last 4 years our achievement is stabilized in bronze. With the limitation of respondents, the findings from the pass IOI in this study will be used to reshape our strategy in the coming year, for moving at least to silver. We must also consider that IOI task is getting more unpredictable, difficult, and creative (Halim, 2013).

Without having computer science, solid mathematical foundation, and algorithmic problem solving in their formal education, three phases of training each last for three weeks on-site plus three weeks extra for final preparation are not enough for preparing IOI gold medallist, unless we can find a prodigious student. Bebras challenge[3] is a potential way to bridge the gap in computational thinking since an early age. Indonesia will join Bebras (Dagienė & Stupurienė, 2014) during the coming year. Indonesia will remain participate actively in IOI since the outcome of the participation and achievement in IOI also improves the spirit of competition amongst Indonesian senior high school students which in turn also means improvement in quality of high school education in Indonesia.

Last but not least, this paper highlights the difference of Computer Science, mathematics education in elementary, middle and high school of IOI countries. Formal education in Computer Science, mathematics and problem solving from an earlier age contributes to the success of the participants in IOI. One year, or more precisely four training camps three weeks each, is not enough to well prepare IOI participants unless we are lucky to find an extraordinary student.

Acknowledgment

I would like to thanks Adi Mulyanto who help me connecting to the Indonesian IOI alumni for getting the data, and also to the 24 respondents for their quick responses to the questionnaires. I would also like to show my special gratitude to Prof. Valentina Dagienė, who introduced to me about qualitative research, inspired and encouraged me for writing this paper.

[3] http://bebras.org/

References

Dagienė, V., Futcheck, G. (2010). Introducing informatics concepts through a contest. Presented in: *IFIP Workshop New Developments in ICT and Education held at Université de Picardie Jules Verne, Amiens, France 28–30 June*.

Dagienė, V., Stupurienė, G. (2014). Informatics education based on solving attractive tasks through a contest. *Commentarii informaticae didacticae*, 7, 97–115.

Halim, S. (2013). Expecting the Unexpected. *Olympiad in Informatics*, 7, 36–41.

Hazzan, O., Dubinsky, Y. *et al*. (2006). Qualitative research in computer science education. In: *SIGCSE '06 Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education.* 408–412.

Hazzan, O., Nutov, L. (2014). Teaching and Learning Qualitative Research ~ Conducting Qualitative Research, The Qualitative Report 2014, vol 19, Teaching & Learning Article 1, 1–29.

Knobelsdorf, M. (2008). A typology of CS preconditions for learning. In: *Koli '08 Proceedings of the 8th International Conference on Computing Education Research.* 62–71.

Nedkov, P. (2012). Young Talent in informatics: preliminary findings of an IOI survey launched by AICA in cooperation with IT STAR. *Olympiad in Informatics*, 6, 192–198.

Theodoraki, A., Xinogalos, S. (2014). Studying students' attitudes on using examples of game source code for learning programming. *Informatics in Education*, 13(2), 265–277.

M. M. I. Liem is a member of Knowledge and Software Engineering Research Group, School of Electrical and Engineering, Institut Teknologi Bandung (ITB). She has been teaching programming in ITB since 1977 and obtained her doctoral degree in University of Joseph Fourier Grenoble France in 1989, with teaching programming as major topics of her dissertation. From 2004, she is involved as a team member in national recruitment, training and IOI preparation for Indonesian team.

Olympiads in Informatics, 2016, Vol. 10, 207–222

DOI: 10.15388/ioi.2016.13

`oii-web`: an Interactive Online Programming Contest Training System*

William Di LUIGI[1], Gabriele FARINA[1], Luigi LAURA[1,2,3]
Umberto NANNI[2,3], Marco TEMPERINI[2,3], Luca VERSARI[1]
[1]*Italian Association for Informatics and Automatic Calculus (AICA)*
[2]*Department of Computer, Control, and Management Engineering "Sapienza" University of Roma, Italy*
[3]*Research Center for Distance Education and Technology Enhanced Learning (DETEL) – Unitelma University*
e-mail: {gabr.farina,williamdiluigi,veluca93}@gmail.com
{laura,nanni,marte}@dis.uniroma1.it

Abstract. In this paper we report our experience, related to the online training for the Italian and International Olympiads in Informatics. We developed an interactive online system, based on the Contest management System (CMS), the grading system used in several major programming contests including the International Olympiads in Informatics (IOI), and used it in three distinct context: training students for the Italian Olympiads in Informatics (OII), training teachers in order to be able to assist students for the OII, and training the Italian team for the IOI. The system, that is freely available, proved to be a game changer for the whole Italian Olympiads in informatics ecosystem: in one year, we almost doubled the participation to OII, from 13k to 21k secondary school students.

Being developed in CMS (`http://cms-dev.github.io/`), the system is highly available to extensions supporting, for instance, the production of feedback on problems solutions submitted by trainees. It is also freely available, with the idea of allowing for support to alternative necessities and developments.

Keywords: programming contest, Olympiads in informatics, programming training, problem recommendation model.

1. Introduction

The International Olympiads in Informatics (IOI) are an annual programming competition for secondary school students patronised by UNESCO. First IOI has been in Bul-

* A preliminary version of this paper appeared as W. Di Luigi, G. Farina, L. Laura, U. Nanni, M. Temperini, and L. Versari. Three Uses of the Online Social Programming Training System: On Nature and Purpose of Spreading Algorithmic Problem Solving. *Proceedings of the 8th International Workshop on Social and Personal Computing for Web-Supported Learning Communities, (SPEL 2015)*, State of the art and Future Directions in Smart Learning, 369–379, 2016.

garia in 1989. The 2015 IOI, held in Almaty, Kazakhstan, saw participation by 83 countries and 322 contestants (each country can have up to four contestants). Participants are usually the winners of national competitions.

Here we first introduce `oii-web`, an interactive online training platform, based on the *Contest Management System* (CMS, `http://cms-dev.github.io/` (Maggiolo and Mascellani, 2012; Maggiolo *et al.*, 2014)), that is the grading system used in several programming competitions, including IOI. We built, around `oii-web`, three distinct, in both target audience and functionalities, web based platforms: one dedicated to students preparing for the Italian Olympiads in Informatics (OII), one for the teachers, with a complete course on programming and several resources available, and the third to support the selection and the training of the Italian team for the IOI. We believe that our online training system fills a gap, since there are several open source grading systems and several online training platform, but to the best of our knowledge there is no open source solution if one wants to host his own training platform.

We report on our experience with the three platforms, designed around the common core, `oii-web`, that allows to navigate through problems, propose solutions, and get feedback about it.

The overall system is already apt to be fruitfully used, with educational aims, as a tool for competitive programming. Yet we are pursuing its enrichment with aspects of personalization to trainees characteristics and needs, aiming to better help them enhance their abilities to deal with contest problems: this would be novel, to our knowledge. So, in the last part of the paper we discuss the requirements of such extension showing an initial modeling schema for problems, solutions, and ultimately trainees.

2. Related Work

Here we deal with various topics connected to programming competitions and, more generally, computer programming learning for secondary school students: a web training platform, the organization of national Olympiads in informatics, and our experience in broadening the participation to it.

On these topics a crucial information source is the *Olympiads in Informatics* journal, founded in 2007, providing "*an international forum for presenting research and developments in the specific scope of teaching and learning informatics through Olympiads and other competitions*". Books such as (Skiena and Revilla, 2003) and (Halim and Halim, 2013) provide also essential material about algorithms, data structures, and heuristics needed in programming contests.

The importance and the effectiveness of programming contests in learning programming and, more generally, computer science has been observed and emphasized greatly in the literature: we mention the works of Dagienė (Dagienė, 2010) and Garcia-Mateos and Fernandez-Aleman (Garcia-Mateos and Fernandez-Aleman, 2009).

Various kinds of automated support to programming education are met in research since decades. The widest area of investigation seems to be related to introductory programming courses, where students learn to write programs, according to a programming language

syntax and semantics, and to solve problems. In this way students are trained on both basic algorithms and their coding. Programming errors are spotted basically in two phases: syntactic and static semantics errors are pointed out by the compiler, while logic/dynamic semantics erros are spotted by testing. So, program assessment is usually based on:

- **Static Analysis**, that gathers information about the program and produce feedback without execution. In this family fall approaches based on compiler error detection and explanation (Hristova *et al.*, 2003; Watson *et al.*, 2012), structured similarity between marked and unmarked programs (Naudé *et al.*, 2010), and also nonstructural analysis, keyword search and plagiarism detection (Khirulnizam and Md, 2007).
- **Dynamic Analysis**, that tests the program on accurately chosen input datasets and compares actual and expected output. One important application of this program analyses is in competitive learning tools, used to manage programming contests, such as (Leal and Silva, 2003).

In Wang *et al.* (2011) combine the two approaches: first the program undergoes static analysis, for compilation errors and to check similarity with "model programs". Then a dynamic testing is performed, and possibly the program adds in a set of model programs.

Grading systems such as CMS are mainly based on dynamic testing, and are many: amongst them are those used in ACM International Collegiate Programming Contest (ICPC), i.e. the proprietary Kattis[1], and the open source PC[2], available at `http://pc2.ecs.csus.edu/`. Other open source grading systems are Open Judge System[2] and DOMjudge[3].

If we focus on online training platforms, amongst several high quality ones are UVa Online Judge[4] and the more recent Sphere Online Judge[5] (SPOJ). Besides these training platform, there are several well-known programming contests platforms, including Codeforces, USACO, COCI, TopCoder, Codechef, and Hackerearth, that run contests with different periodicity. There are also events based on programming contests, like the Google Code Jam and the Facebook Hacker Cup. A detailed survey of programming contests is in (Combéfis and Wautelet, 2014).

3. Italian Olympiads in Informatics

The International Olympiads in Informatics started in Bulgaria in 1989, patronised by Unesco. They are considered one of the most important programming competition in the world. Each country can have four contestants, and the competition is divided in two competition days. On each day contestants will be given three tasks to complete in five hours. Each task is worth 100 points and, since IOI 2010, it is divided into subtasks, each

[1] `https://kth.kattis.com/`
[2] `https://github.com/NikolayIT/OpenJudgeSystem`
[3] `http://www.domjudge.org/`
[4] `https://uva.onlinejudge.org/`
[5] `http://www.spoj.com/`

worth a portion of the total points. There are time and memory limits for each subtask, and points are awarded only when all the tests in subtask yield correct results within the limits. There are also interactive tasks, like games, in which the contestant code alternates moves against an adversary.

In Fig. 1 we can see a graphical representation of a task, taken from OII 2014 final. The task, `taglialegna` (lumberjack), can be summarized in the following way: *there is a line of trees, with one meter of space between each of them. Each tree has a known height, in meters, and you can cut it aiming it toward its right or left. When an* m *meter tree falls, like in a domino game it forces the falling of its* $m - 1$ *close trees, and this in turn can force other tree to fall. You can decide which tree to cut, and for each of them you can choose in which direction it will fall. What is the minimum number of trees to cut in order to remove all the trees in the line?* For this task, the subtasks were designed to distinguish algorithms of different computational costs: if we denote with n the number of trees in the line, all the points were awarded to a (definitely not trivial) $O(n)$ solution, achieved by only one contestant, and decreasing points were assigned, respectively, to $O(n \log n)$, $O(n^2)$, and $O(n^3)$ solutions.

Italy participated in IOI for the first time in 2000, and since 2001 it started a national competition, promoted by a joint effort of the Italian Ministry of Education, University and Research (MIUR) and the Italian Association for Informatics and Automatic Calculus (AICA, a non-profit organization). The Italian Olympiads in Informatics (OII) are divided into three phases:

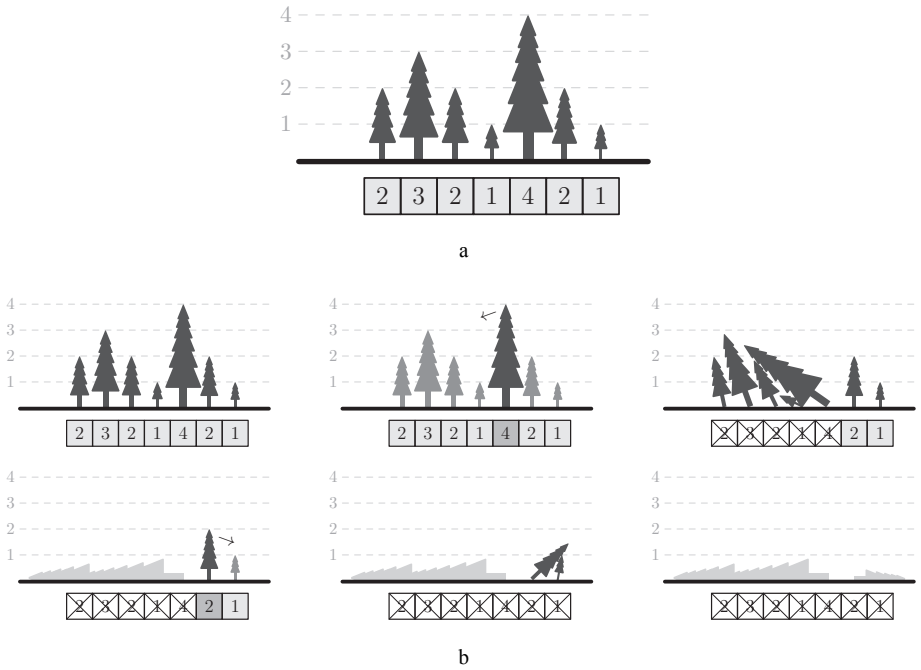


Fig. 1. The graphical representation of the task `taglialegna` (lumberjack), from OII 2014 final: an input instance (a) and a possible solution that uses two cuts (b).

1. **First Selection (Schools, November)**: in this phase, in their own schools, approximately 20k students compete to solve, on paper, a test that involves math, logic, and programming abilities; in particular, there are some fragments of code (C/C++ or Pascal), and the students are asked to understand the behavior of the fragments.
2. **Second Selection (Regions, April)**: in this phase there are approximately 40 venues, where approximately 1200 students, selected from the previous phase, compete by solving three programming tasks on the computer. In this phase points are awarded for solving the tasks, independently from the complexity of the solution.
3. **Third Selection (National Final, September)**: approximately 100 students are asked to solve *efficiently* three programming tasks on the computer. They compete for 5 gold, 10 silver and 20 bronze medals.

From the above description it should be clear how the required programming abilities are varying through the different steps: we first ask students to be able to *read* code, then to *write* code, and finally *efficiently write* code. A more detailed picture of the OII organization is described in (Casadei *et al.*, 2007).

The selection process does not end with the national final: the gold and silver medal winners, together with at most five bronze medal winners, selected by (young) age, form the group of IOI-candidates, and four of them will represent Italy in the next IOI (usually held in July or August). Thus, there is almost one year to train and select them, and this process is mainly done in four stages held nearby Volterra[6]. In each of the stages there are theoretical lessons, ranging from traditional algorithms and data structures to competitive programming tips and tricks, as well as programming contests. Besides the stages, there is a continuous on-line support for selfimprovement: the IOI-candidates are assisted by tutors (former IOI contestants) for assistance and guidance, and several training contest are organized, some of which focused on specific topics.

4. The Online Training System: oii-web

Our online training platform, oii-web, is based on the *Contest Management System* (CMS) (Maggiolo and Mascellani, 2012; Maggiolo *et al.*, 2014), the grading system used in several programming competitions, including IOI. CMS was designed and coded almost exclusively by three developers involved in the Italian Olympiads in Informatics: Italy hosted IOI 2012 and therefore, since 2010, it started the development of CMS, that was used/tested in the OII finals 2011 and, few month later, was the grading system of IOI 2012. CMS version 1.0 was released in March 2013, and since then has been used in both IOI 2013 and 2014, together with several other programming competitions in the world (Maggiolo *et al.*, 2014).

[6] The small city of Volterra in Tuscany is nowadays world-wide popular due to the fact that in the novels and movies of the Twilight vampire saga it is the origin place of Volturi, “the largest and most powerful coven of vampires”.

We began the development of `oii-web` during the preparation of the IOI-candidates for IOI 2012: why did we need an online training platform? The short answer is: in a programming competition there are very few (usually from 3 to 7) problems, to be solved in a short frame of time; in order to train the IOI-candidates we needed a system that allowed us to give them more problems they can solve whenever they want, so the first version of `oii-web` was simply an instance of CMS with one competition running, with several problems and unlimited time. For training the IOI-candidates and, later, the two[7] Italian teams competing in IOI 2012.

Later, we started using it consistently, and our feature list was growing almost daily, both for the front-end and for the back-end of the system:

1. It would be nice to provide some information about each problem, so the student can choose it without reading the whole description.
2. It would be nice to have a way to exchange messages, so students and tutors can chat about the problems.
3. It would be nice to be have a way to show/hide problems, so we can use some of them in contests to rank the students.
4. It would be nice to have stats about each problem, and who was able to solve it (in a grading system there are these stats but not visible to contestants).

Thus, we decide to include all these above mentioned features, together with others, and build an online grading system, `oii-web`. We integrated the open source *Discourse*[8] to provide forum functionalities. The source code of the system is freely available[9] in github and it is released under the GNU Affero General Public License[10]. Furthermore, it is also available[11] as a "dockerized" app for docker[12], an open platform to build, ship, and run distributed applications.

5. The Three Platforms

In this section we briefly describes the three platforms, based on `oii-web`, we developed, and their differences.

OII-training is the platform devoted to the students that are interested in OII. We can see a screenshot of the home page in Fig. 2 (a). In this platform there are approximately 180 problems spanning several techniques and difficulties, ranging from regional contests to IOI level. Furthermore, there are also the tests, from the first selection of OII (schools selection), available as interactive online forms. So far we did not advertise this platform in the schools, since we consider it in a beta testing phase. We allowed

[7] The nation that hosts IOI can have two teams of four elements, but only one team is eligible for medals.

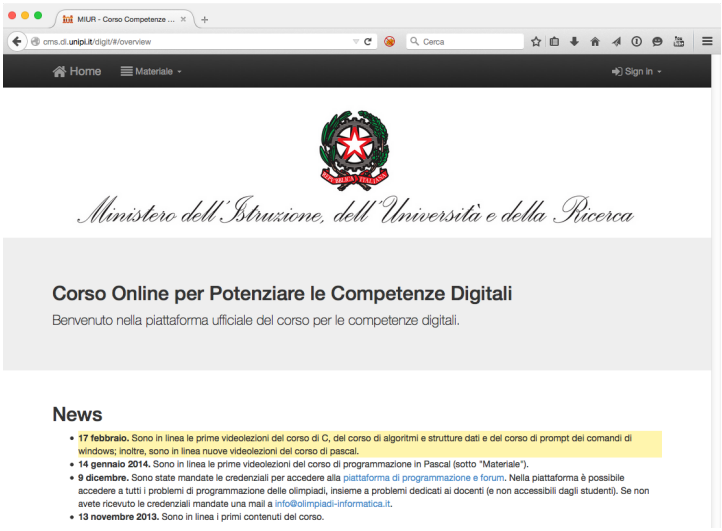
[8] `www.discourse.com`

[9] `https://github.com/veluca93/oii-web/`

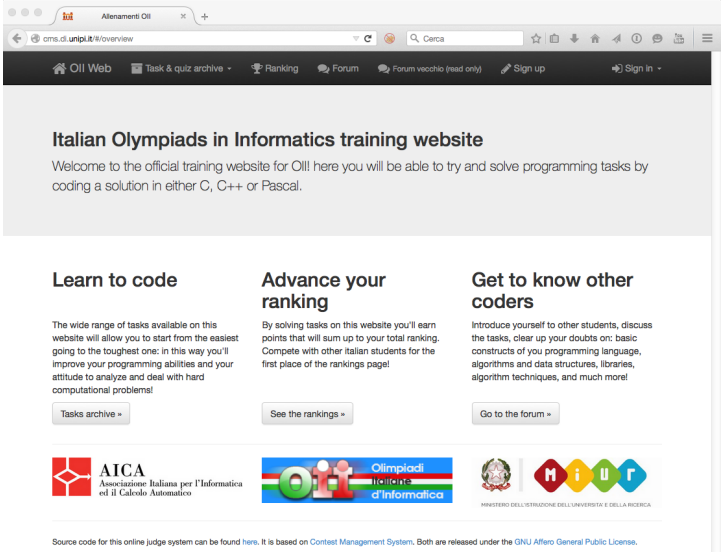
[10] `http://www.gnu.org/licenses/agpl`

[11] `https://registry.hub.docker.com/u/veluca93/oii-web/`

[12] `www.docker.com`



a



b

Fig. 2. The home pages of two of the platforms based on `oii-web`: the one for the teachers (a) and the one for the students (b).

students to register freely, and so far we have approximately 1.500 users despite the lack of promotion.

DIGIT is the platform dedicated to teachers: we realized this platform in a project sponsored by the MIUR, where the aim was to build a self-paced online course of computer programming, focused on the Olympiads in informatics. The idea was to train the teach-

ers so they would be able to train their students. Thus, this platform is currently the richest of the three, in terms of contents and functionalities. We can see a screenshot of the home page in Fig. 2 (b). There are video lectures on C/C++ and Pascal programming, Algorithms and Data structures, and some basic video tutorial as well including how to use the platform to submit a solution. There are also some lecture notes, and all the material can be distributed to students as well; the video lectures are also available on the OII channel on YouTube. The MIUR used this platform, since October 2013, in five distinct courses, with a sixth one scheduled to start in September 2016. So far approximately 3.000 teachers followed this course, and the effects on the OII were impressive: the participation of students in OII preliminary stages raised from 13k to 21k.

IOI-candidates is the last platform, and the only one not publicly available, since it is devoted to the IOI-candidates. This platform, as we mentioned before, has been the original motivation to develop the whole `oii-web`system. This platform has all the problems available to the other two platform, together with a *reserved* set of problems that we use in the contests to rank the students. The students are asked not to discuss these problems in public forum or social network, since we usually reuse them after few years.

6. Our Experience

The advantages of a training system are clear: without it, we need to give students, besides the text of the problem: the input cases, the rules for counting the points, and, in some cases[13], a code to check the correctness of the produced output. And the student has to: run its code against every input, run the checker against each input and matching output, check the time and memory limits, that can be a cumbersome operation for a beginner. Furthermore, even if we automatize this task, for example by a script given to the student, there is still the problem of measuring the running times in different machines: students can have very different hardware and it is meaningless to state time limits without knowing their hardware.

Our path began, as we mentioned before, with the needs of a training system able to assist us with the preparation of Italian IOI-candidates. We soon realized the advantages of such a system, as opposed to the use of an online platform like UVa Online Judge: we simply had more control, and this leads to a more effective teaching experience. We almost immediately decided to develop an online platform for the OII students as well, and we enriched our basic system with more features, in order to be able to deal with a much larger number of (averagely) less motivated students. In the beginning of 2013, the OII-training platform went online, in the form of a publicly available beta, as we were planning to add more features to make it more appealing for a larger audience. Almost concurrently, the MIUR asked us to design an online course for teachers, and we im-

[13] To check the correctness of some problems is enough to check that the output produced by the student is the same as the output produced by the correct solution; in other cases, usually when there is more than a unique solution for a problem, like finding a path in a graph under some constraint, it is necessary to write a checker code that verifies the solution proposed for the given input.

mediately decided to build it around our platform. So, in the next months, we adapted the system for the DIGIT platform, and realized the video lectures; in October 2013 we launched the first course: the MIUR opened a call for 250 teachers to be freely allowed to follow the course. The call was supposed to stay open for ten days, but we reached 250 teachers in the first day, and we decided to admit more. In subsequent courses, since we observed that the server was working fine, we raised the number of teachers per course to 700. At the end of each course there is a programming contest, and the ones that perform above a threshold (solving three problems out of seven) are awarded with a certification. Note that once a teacher has access to the platform, (s)he is allowed to use it also after the end of the course. Many teachers reported us that they had fun using the platform, and that they plan to keep on using it.

Our experience shows that the engagement in having or not a training system is completely different: we witnessed this at all the learners' levels; beginners were more involved, and advanced learners often joined the developers community (mostly made of tutors and former IOI contestants) to either contribute the system development or to propose new problems. The teachers were incredibly active in the forum, exchanging tips and solution strategies as well as mutual support. The IOI-candidates are literally eager to contribute to the system or in the design of new problems, maybe because they see the tutors as a model, or simply because they enjoy it so much that they want to be part of it.

We also asked the IOI-candidates to "adopt a past IOI problem": our goal is to have, in the system, all the problems from past IOIs, and therefore there is a (shrinking) list of the problems that need to be produced: in all the cases the text of the problem is available on the web, usually together with some solution, but we need to write input generators and fine-tune the time and space limits. Currently we have almost all the problems of the last ten IOIs.

7. Validation of the System: A Descriptive Analysis

In this section we discuss the results of a validation of the system; we performed our study by means of a survey technique, with a questionnaire as a tool. We focused on the users of the **OII-training** platform, and we report some stats in Table 1. With active user we denote a user that submitted at least one solution of a problem; with problem solved we denote the number of submission that completely solved a problem.

We sent the users of the platform the link of the questionnaire in May 2016; we had 171 users that answered, and this means almost half of the current active users. In Table 2 we report the questions and the statistics of the answers provided.

The experimental setup is based on the collection of general information about the respondents, and on scales aiming at *Satisfaction*, *Usability*, *Effectiveness*, *Active learning*, *Fun*.

Where possible we used a Likert scale, with five grades: two highest, two lowest, and an intermediate one. This allowed to separate clearly *mainly positive* judgements from *mainly negative* ones. Exceptions (questions Q6 and Q12) are motivated by their, less progressive nature.

Table 1
Some statistics about the OII-training platform

Number of registered users	1413
Number of active users in the period Jan. 2015 – May 2016	812
Number of active users Jan. 2016 – May 2016	399
Problems in the system	253
Problems solved by users in the period Jan. 2015 – May 2016	9754
Problems solved by users in the period Jan. 2016 – May 2016	6192
Average number of problems solved per user in the period Jan. 2015 – May 2016	≈12
Average number of problems solved per user in the period Jan. 2016 – May 2016	≈15,5

About the general satisfaction of the learners, we considered important the learner's feeling about the actual "learning results". In this respect it is quite satisfactory for us that 65% of the respondents selected mainly positive (the highest two) grades, while the mainly negative (lowest two grades) were chosen by a 6.5%.

Usability of the system was marked mainly positively by a 70%, with mainly negative scores below 4%.

Of course we were mainly interested in the effectiveness shown by the system, as witnessed by the higher number of questions dedicated to that topic. One main issue, in that respect, is the number of problems (meaning exercises) that the learner undertook/solved. A second issue regards the perception of the learner about having fruitful and not tiresome sessions of use of the system.

The first above issue is met by questions Q3 and Q4. As it was expectable, there are more exercises "tried" than "solved": the system is not a *panacea*. On the other hand, while only 1% of the respondents tried some exercises (probably between 1 and 5) with no success, data show that 56% of the students was able to give a try between 11 and 20 problems (one third of this share) or more than 20 (two thirds of them), succeeding in quite a respectable 47% of the whole sample. In this respect we notice that 3 learners out of 4 that tried more than 20 exercises, succeeded in more than 20 exercises.

The second issue above (regarding fruitful and not tiresome sessions of work in the system), is cared by questions Q5 through Q8: to some extent Q7 helps focusing the result of Q5, while Q8 does the same for Q6. From Q5 and Q7 we clearly see that (at least the learner's perception of) fruitfulness is high, with mainly negative results below 6% and 10%, respectively for Q5 and Q7. Q7 was indeed useful in pointing out one crucial aspect of fruitfulness (comprehension): its results sport a quite rewarding 61% of mainly positive marks. Finally, Q6 and Q8 tell us that there is scarce perception of a session of work in the system being tiresome or slow.

The approach to learning sought by the system is coherent with the concept of *active learning*, so we thought it would be interesting to probe the perception of learners in that respect. Question Q9 is quite direct in that respect: the results show mainly positive response (73%, equidistributed between the two highest marks). Questions Q10 and Q11 took a less direct route to the learner's attitude toward the system: the former question wanted to reveal the induced engagement, and scores almost 64% of mainly positive answers, while the mainly negative feedback is limited to 8%.

Table 2
List of the questions we used in the evaluation of the system.

Satisfaction
Q1. Did you find the system useful to fulfill your learning goals?
A1. Very Much: 25.1%, Quite So: 39.8%, Enough: 28.7%, A few: 4.7%, Not at all: 1.8%
Usability
Q2. Is the system simple to use?
A2. Very Much: 24%, Quite So: 46.2%, Enough: 26.3%, A few: 1.8%, Not at all: 1.8%
Effectiveness
Q3. How many problems did you tackle in the system?
A3. 0: 1.8%, From 1 to 5: 18.1%, From 6 to 10: 24%, From 11 to 20: 18.7%, More than 20: 37.4%
Q4. How many problems were you able to solve satisfactorily in the system?
A4. 0: 2.9%, From 1 to 5: 29.2%, From 6 to 10: 21.1%, From 11 to 20: 18.1%, More than 20: 28.7%
Q5. According to your perception, your sessions using the system were fruitful?
A5. Very Much: 15.2%, Quite So: 42.1%, Enough: 36.8%, A few: 4.7%, Not at all: 1.2%
Q6. According to your perception, your sessions using the system were long enough (to be fruitful), but not too long (to be tiring)?
A6. Too long (tiring): 8.8%, Not so long (not tiring): 81.3%, Short (not tiring): 9.9%
Q7. By solving a problem, did you improve your comprehension of the algorithm or the technique involved?
A7. Very Much: 24%, Quite So: 36.8%, Enough: 29.8%, A few: 7.6%, Not at all: 1.8%
Q8. Is the system quick enough in providing response?
A8. Very Much: 25.7%, Quite So: 35.1%, Enough: 29.2%, A few: 6.4%, Not at all: 3.5%
Active Learning
Q9. I felt active in approaching the problems with the aid of the system
A9. Very Much: 35.7%, Quite So: 37.4%, Enough: 17%, A few: 8.2%, Not at all: 1.8%
Q10. The study of algorithms and related techniques is more interesting with this approach
A10. Very Much: 33.9%, Quite So: 29.8%, Enough: 28.1%, A few: 4.7%, Not at all: 3.5%
Q11. Due to the interaction with the system I have identified and trained upon central issues in the problem solving activity, and important concepts in the solution of problems
A11. Very Much: 17.5%, Quite So: 38%, Enough: 33.3%, A few: 9.4%, Not at all: 1.8%
Fun
Q12. Using the system I felt mainly: i) Motivated, ii) Happy iii) Curious, iv) Relaxed, v) Other
A12. Happy: 15.2%, Motivated: 52.6%, Relaxed: 4.1%, Curious: 24%, Other: 4.1%
General questions
Q13. Which institution are you currently enrolled?
A13. High School: 69%, University: 9.9%, Work: 17%, Other: 4.1%
Suggestions
Q14. What would you improve in the platform?
A14. *See the text*
Q15. Other ideas or suggestions
A15. *See the text*

Question Q11 tried to connect the work in the system with the perceived gain in terms of problem solving skills. In this case we have still more than half the sample showing mainly positive response (56%), with 11% mainly negative and a third of the sample set on the intermediate grade.

Since the use of systems like ours is not usual in the Italian School, we liked the idea to fetch some reactions in relation to the “fun” factor. Such an investigation would be

more proper in game based, or gamified, systems; however, since we actually plan to add gamified aspects (namely a *badge* feature), it seemed good to add a question whose response would be more useful in future comparisons.

With respect to the present state of the system, the results are surely quite good, with feelings of motivation (53%) and curiosity (24%) encompassing more than three quarters of the sample's feedback.

We conclude this section by discussing the suggestions we received in questions Q14 and Q15. Question Q14 was *What would you improve in the platform?*, and users were allowed to choose one or more of the proposed answer, that are the directions we are working on. We report below the results (that were not mutually exclusive, as the other questions), in order of the expressed preferences:

77.2% I would add a wiki with documentation about algorithms and related techniques.
57.3% I would add a system to help the user to choose the next problem to solve.
36.3% I would add a badge system, to show achievements using distinct badges.
12.3% I would add more problems involving Mojito, the JackRussell mascotte of OII.

From the first of the above results we gather the obvious: of course a repository of centralized information, about algorithms and techniques needed in the solution of the exercises, is highly attractive. Such a development is actually in our plans, needing basically quite a lot of wear and tear in order to structure and feed the wiki, and not much more in terms or research. Being it a wiki, however, we are planning to make it available to contributions coming from all the members, so to make of it another opportunity for social collaboration, and social-collaborative learning.

On the contrary, the second preferred choice (recommending system for the next problem to solve, that would be based on the student model) is a topic for further research, met preliminary in the next section.

8. Further Developments

A work of Wang *et al.* (Wang *et al.*, 2011) states the following requirements for a comprehensive program assessment system:

1) *Sufficiently extended testing*, so to cover the various cases of computation.
2) *Checking on the program structure*, to see that the problem specification is met, and no cunning shortcuts bring to the correct output.
3) *Accepting and reasonably assessing programs with static errors.*
4) *Providing immediate and correcting feedback*.

We think that the developments in the `oii-web` system should ultimately fulfill these requirements, while the present directions should deal closely with the present purposes of the system, that is to allow non novice students to train for the contests. So here we try and define a model to support:

- A static analysis stage where solution strategies (algorithm, data structure and their mutual feasibility) rather than syntactic/semantics errors, are considered.
- An interactive communication between system and student, to help:
  - Developing one's capability to select solution strategies, by giving feedback on the actual choice.
  - Planning a path of growth of one's skills, by suggestions about next suitable problems to undertake.
- (And dynamic testing in the usual form, as it is already done).

Problems (the exercises proposed in the various contests, yearly), *Solution*s (the programs proposed by the students), and ultimately the *Student*s are modeled basing on a tagging mechanism. Tags are the names of problems (P), the algorithms (A) and data structures (DS) usable and/or used in the solutions (S) of problems, the contests (C), and levels of confidence (L) in the use of combinations of As and DSs.

Teachers in charge of the organization of a contest are named *guru*s; students that came out to be "exemplary" in a contest, and so "whose choices can count" when a solution to a problem is to be assessed, are called *Exemplary Peers* (EPs). EPs can be promoted as guru.

A problem is modeled as a family of strategy choices (A and DS), suitable for its solution. Differences in that suitability can be pointed out by a weight. The weight is computed basing on the frequency by which A/DS were chosen, and on the reputation of who performed that choice in the related contest. Notice that the reputation of the gurus and EPs is contextualized to the contest.

$$P = \{<A,\ DS,\ weight,\ contest>\}$$

A solution submitted by a student is modeled by the strategy chosen for it:

$$S = \{<person,\ P,\ A,\ DS,\ conteXt>\}$$

where $conteXt$ is either a contest name or "training" (off contest).

This metadata is provided by the student, in order to allow for a timely feedback from the dynamic analysis. On the other hand that metadata might be inaccurate, so it is subject to scrutiny: when a check points out that the data was wrong, it is changed accordingly, or (in the extreme case) the solution is removed altogether.

This check is done by gurus. A more social kind of scrutiny has been devised, yet it can't be applied, as the students solutions submitted to the system are not to be shown in public, at least for the time being.

A submitted solution is statically checked by comparing its specification S with that of the problem P. A feedback can be then given, about the appropriateness of the choice, its present weight, and possible better weighted alternatives.

The lightweight student model we can define in this framework define the skills shown by the student while solving problems in the system; it is a collection of "acquire-

ments" each one expressing the fact that a given problem has been solved, how, how well and in what $conteXt$

$$SM(person) = \{<P, A, DS, level, conteXt>\}$$

where *level* is a discrete variable in [1 ... 5] associated to the outcome of the dynamic analysis of the solution submitted by the student on the problem.

The above modeling framework can be used during a contest, in order to collect problems models and data on students (for instance to compute students reputation and define the set of EPs for that contest).

However here we are also interested in the possible use of this framework in a social webbased settings, to foster training in view of a next contest. The CMS would be the place for such training, organized by the following protocol.

1. A *Target Skills* is given. This is a set of triples designating the aim of the the training (at this stage for all the system): $TS = \{<A, DS, level>\}$.
2. The trainee can access the set of problems available from previous contest, her/his student model, and the TS.
3. While the trainee is entitled to select any problem and submit the related solution, the system can provide a list of suggestions, for "best next problems to undertake" in order to enhance SM(*trainee*) towards coverage of TS. This list is done by:
   (a) defining the set of elements in TS that are close to be covered by tuples in SM (*Proximal Coverage* – PC);
   (b) defining a set of problems, whose undertaking can bring to add elements in PC to the student model.
4. Upon submission of a solution, the trainee provides its initial modeling ($<A, DS>$).
5. The dynamic analysis of the solution establishes the *level* value for the t-uple

   $$<P, A, DS, level, conteXt>$$

   going to join the student model.

   Notice that the trainee specification of the solution can be subject only to late evaluation (by guru), in order to allow for a timely feedback coming from the dynamic analysis. So the new element in the SM is *sub-judice* and it could be modified or, in the extreme cases, deleted.

9. Conclusions

In this paper we introduced `oii-web`, an online training system for programming contests. The system is based on CMS, the grading system used currently in IOI competitions and other programming contests as well. We developed three distinct platforms,

based on oii-web, aimed at three distinct user sets: students enrolled in OII, their teachers, and IOI-candidates, i.e. the small set of students amongst which will be selected the four to represent Italy at IOI. We discussed briefly our experience, together with some current developments. We believe that, as happened in our case, the use of such a system can contribute to spread the algorithmic problem solving skills needed in programming contests.

We also believe that this tool can scale up toward being an educational support to refining students skills in "algorithm mastery", and we have presented lines of development in that direction.

Acknowledgements

We dedicate this work to Marta Genovie De Vita.

References

Casadei, G., Fadini, B., Vita, M. (2007). Italian olympiads in informatics. *Olympiads in Informatics*, 1, 24–30.

Combéfis, S., Wautelet, J. (2014). Programming trainings and informatics teaching through online contests. *Olympiads in Informatics*, p. 21.

Dagienė, V. (2010). Sustaining informatics education by contests. In: *Teaching Fundamentals Concepts of Informatics*. Springer, 1–12.

Garcia-Mateos, G., Fernandez-Aleman, J.L. (2009). Make learning fun with programming contests. In: *Transactions on Edutainment II*. Springer, 246–257.

Halim, S., Halim, F. (2013). *Competitive Programming,* Third Edition. Lulu.com.

Hristova, M., Misra, A., Rutter, M., Mercuri, R. (2003). Identifying and correcting java programming errors for introductory computer science students. In: *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '03)*. ACM, New York, NY, USA, 153–156.

Khirulnizam, A., Md, J. (2007). A review on the static analysis approach in the automated programming assessment systems. In: *Proc. Nat. Conf. on Software Engineering and Computer Systems*. Pahang, Malaysia.

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

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

Maggiolo, S., Mascellani, G., Wehrstedt, L. (2014). Cms: a growing grading system. *Olympiads in Informatics*, 8, 123–131.

Naudé, K., Greyling, J., Vogts, D. (2010). Marking student programs using graph similarity. *Computers and Education*, 54, 545–561.

Skiena, S.S., Revilla, M.A. (2003). *Programming Challenges: The Programming Contest Training Manual*. Springer Science & Business Media.

Wang, T., Su, P., Ma, X., Wang, Y., Wang, K. (2011). Ability-training-oriented automated assessment in introductory programming course. *Computers and Education*, 56, 220–226.

Watson, C., Li, F., Godwin, J. (2012). Bluefix: Using crowd-sourced feedback to support programming students in error diagnosis and repair. In: *Proc. Int. Conf. on Web-based Learning, ICWL 2012*, LNCS (vol. 7558). Springer Verlag, 228–239.

W. Di Luigi holds a Bachelor of Science with Honours in Computer Science and Engineering from the University of Bologna, and he's now attending a Master's degree in Computer Engineering at Politecnico di Milano. He has been involved in the training of the Italian IOI team for the last four years, and helped organizing three Italian Olympiads in Informatics. He is one of the core developers of `oii-web`.

G. Farina participated in IOI 2012 and IOI 2013. He is attending a Bachelor degree in Control Engineering at Politecnico di Milano. He has been actively involved in the training of high school students and of the Italian IOI team for the past four years. He helped organizing several Italian competitions in Informatics, including the Italian Olympiads in Informatics.

L. Laura is involved in the training of the italian team for IOI since 2007, and since 2012 he is in the organizing committee of the Italian Olympiads in Informatics. He got a Ph.D. in Computer Science in "Sapienza", and currently teaches "Web-based Systems Design" in Tor Vergata university of Rome and "Information Systems" in LUISS university of Rome.

U. Nanni is professor of Enterprise Information System in "Sapienza University", President of the Information Engineering bachelor degree in Latina location, Director of the Research Center for Distance Education and Technology Enhanced Learning in "Unitelma Sapienza". His research interests include: graph algorithms and their applications, elearning models and technologies, data- text- and business- intelligence. He was PI of the LLP project eLF-eLearning Fitness.

M. Temperini is an associate professor of Engineering in Computer Science at "Sapienza" University. He teaches programming techniques and programming languages for the Web. He got a Ph.D. in Computer Science at "Sapienza". His research activity is on Web-based distance learning, adaptive e-learning, social and collaborative learning, and Web-based participatory planning. He has been workpackage leader and/or national research unit coordinator in several international projects.

L. Versari participated in IOI 2012, winning a silver medal. He is attending a Masters degree in Informatics at the University of Pisa and he is a student at the Scuola Normale Superiore. He has been involved in the training of the Italian IOI team for the last four years, helped organizing three Italian Olympiads in Informatics. He is one of the core developers of `oii-web`.

Olympiads in Informatics, 2016, Vol. 10, 223–230

DOI: 10.15388/ioi.2016.14

Bridging the Gap Between Bebras and Olympiad: Experiences from the Netherlands

Willem van der VEGT
Dutch Olympiad in Informatics
Windesheim University for Applied Sciences
PO Box 10090, 8000 GB Zwolle, The Netherlands
e-mail: w.van.der.vegt@windesheim.nl

Abstract. While the number of contestants in the Dutch Olympiad in Informatics was declining, the number of participants in the Bebras contest grew rapidly. In order to reach these Bebras participants for joining the Olympiad, several steps were taken. We analyzed the differences between the contests. We offered Bebras contestants an introductory course in programming. And we changed he contest format of the first round of the Olympiad, introducing two new types of tasks. As a result, the number of contestants increased and girls returned to the Olympiad.

Key words: Bebras contest, Olympiad in Informatics.

1. Introduction

The Dutch Olympiad in Informatics was initiated in 1991 as a contest for selecting participants for the International Olympiad in Informatics. In the early years of its existence, the number of participants grew to just below 200. Since 2000 the number of contestants in the first round of the contest has varied, but it is proven to be hard to attract more pupils in high school for the Olympiad.

In 2005 the Netherlands was among the first countries to join Lithuania in the Bebras Challenge. Over the last 10 years the number of participants has grown rapidly to over 21000. A lot of contestants of the Olympiad also participated in the Bebras contest. We started looking into how we could persuade other Bebras-contestants into joining the Olympiad as well.

Naturally, there are large differences between the Olympiad and the Bebras contest. It is a bit like comparing the contestants in a marathon run that are aiming for a national championship with the grand total of recreational runners; the participation of contestants serves different goals. However, there are also similarities; both of our contests are about algorithmic and computational thinking, and they aim to challenge the participants to show what they are capable of.

In section 2 we will give a short history of the Olympiad and Bebras in the Netherlands and elaborate on the similarities and differences between the two contests. In sec-

tion 3 we will show the measures we took to bridge the gap between these contests. In section 4 a sample problem in our new approach will be shown. In section 5 we will present the first results of the new contest format and section 6 presents some discussion.

2. Olympiad and Bebras in the Netherlands

From 1999 till 2014 the Dutch Olympiad in Informatics (2016) had a fixed format with three stages.

Three or four programming tasks for the first round are published on our website in September; the final task is to design a program that can enter a tournament. Contestants can register themselves and they are able to submit their solutions till January 15th. Pupils are allowed, even encouraged, to co-operate. Our submission system is able to handle over 10 different programming languages; evaluation is also done by this system. However, all submissions that failed the test are inspected manually, and if the jury is able to fix a small bug, like an IO-format, the program is re-evaluated and a small amount of points will be subtracted (like 3 out of 100 for every fixed error). Contestants that score at least 50% of the maximum score are allowed to join the second round. All contestants that proceed to the second round get a certificate, and there are special prizes for early submitters and the winner of the game tournament (Codecup, 2016).

This second round is in March at a university; two or three problems with subtasks (van der Vegt, 2009) have to be solved and sometimes there are one or two more theoretical questions (van der Vegt, 2012). All languages that our system can handle are allowed. Only in very rare cases the results of the automated evaluation are overruled. The best performing contestants are invited for a trainings course in April on algorithms and problem solving. They will need to switch to C++ or Pascal for the training course and to prepare themselves for the IOI. Finally, the third round in May or June with a limited number of contestants is used to determine the team for the next IOI.

Like in many countries, organizing the Bebras contest is done by the organizers of the Dutch Olympiad in Informatics. The Bebras Challenge was first held in Lithuania (Dagienė, 2006). The Netherlands started with a test contest in 2005. The contest grew rapidly, in 2015 over 1.3 million pupils from more than forty countries participated in their national Bebras. The questions used in these contests are chosen from an international task pool. The contest is about computer science, algorithms, structures, information processing and applications. No prior knowledge is required. Criteria for good Bebras tasks are formulated by Dagienė and Futchek (2008). Dagienė and Stupurienė (2016) give an overview of current research on Bebras.

Contestants compete in their own age division. In the Netherlands contestants have 40 minutes to complete 15 tasks. These can be multiple choice questions, questions where an answer has to be given in the form of an integer or a short string, or interactive questions. The contest runs for a week; the best performing contestants for every age division are invited at a university for a second round (Beverwedstrijd, 2016).

All contestants in the second round get a certificate. In the IOI-style, 1/12 of them get a gold certificate, 1/6 a silver one and 1/4 bronze. The overall winner in each age category wins a gadget with a text inscription showing he or she was the winner of this year's Bebras contest in a specific agegroup.

Performing in the Olympiad takes a reasonable amount of time; our research suggests that a typical contestant can use 20 to 80 hours to write the programs for the first round, tough experienced pupils can easily, within a few hours, create solutions that score well enough to proceed to the second round. A contestant in the Bebras contest does one round in 40 minutes.

The questions differ accordingly. Most of the tasks in the Olympiad require writing either a batch or a reactive program. So knowledge of a programming language is required. For Bebras, no prior knowledge is needed. The tasks however tend to test the perception of concepts of computer science and computational thinking. Barendsen et al. (2015) showed that it is possible to use Bebras tasks to assess the understanding of these concepts; in their research they focused especially on algorithms.

Another difference was the participation of girls. Until 2004 the Dutch delegation for the IOI used to be a mixed team (Maggiolo, 2015). In later years there was no possibility to create such a team, because the girls almost completely disappeared out of the contest. In Bebras we reached many girls, around 40% of the contestants every year. In the highest age group this is around 20%.

3. Bridging the Gap

We decided to work on different changes to try to get more Bebras contestants in the Olympiad. Now we are connected with the contestants in Bebras, we can invite them as talented pupils to try the tasks of the Olympiad. But since most of them will not be able to write a program, we now offer a programming course for interested pupils. We give contestants a link to the Dutch translation of the Canadian Computer Science Circles (2016). This is an interactive website designed for student to learn programming in Python.

To give these newcomers a challenge, we had to add a series of tasks on a lower difficulty level. We call these tasks the A-tasks.

We also introduced a new kind of tasks in the first round of the Olympiad, tasks that have the look and feel of a Bebras task, but with a problem that we think will not be solved within a few minutes. You can solve the task without using a computer, but it could also be possible to write a computer program to help solving the problem for you. These tasks are called the B-tasks.

The three tasks of the type we used to offer in the Olympiad are now the C- and D-tasks. An overview is given in Table 1. So we still offer a few hard programming tasks,

Table 1
Tasks in the first round of the Dutch Olympiad in Informatics

Task type	Description	Number of tasks	Points per task	Points per group
A	Introductory programming tasks	5	40	200
B	Theoretical, Bebras-like tasks	4	50	200
C	Advanced programming tasks	2	100	200
D	Game programming task	1	100	100

but we want to attract new contestants in two ways: as starting programmers, after taking the course, solving A-tasks and getting curious about the other tasks we are offering, or as experienced Bebras-contestant, with a few challenging, more time consuming puzzles. We hope that this entices the pupils to try a few programming tasks.

When we started this approach in 2014, we had no clue how many contestants we would be able to attract. We stated in our contest description that the top 100 contestants with a certificate would be invited to the second round, tough the last year we had over 100 contestants in the Olympiad was 2003.

We decided to combine the certification system of Olympiad and Bebras, but we set boundaries in advance, so a contestant will know where to aim. We give every contestant that solved all A-tasks or all B-tasks bronze certificate. Performing well for both types is enough for silver. The boundaries are stated in Table 2.

4. Output only Problems: an Example

The Bebras-like theoretical tasks are like the output only tasks at the IOI. On the Wiki of the International Olympiad in Informatics (2016) this definition is given. "A task is of type 'output only' if the contestant is provided with the input files, and must only submit the corresponding output files. The contestant can solve each test case by hand or by writing one or more programs in the language of his choice, and doesn't need to submit these programs." In our contest we will not use input and output files; the contestant has to download the problem description and to submit the answer in our contest system.

For the first tasks we used exercises that were based on the tasks Dungeon, Palindromes and Cities from Burton (2010) and Coins from Kubica and Radoszewski (2010). Since there is a large period in time that these tasks can be solved in the first round, we could not simple use one instance of these tasks. So we decided to make these tasks personalized. The contest ID of a contestant was used as a random seed to create a problem instance.

Two other restrictions were made:

1. For a full score (50 points) a submission had to be made within a week after the problem statement was generated by the contest system. Each day of delay gives a penalty of 1 point.
2. To discourage guessing, submitting a wrong answer gives a penalty of 10 points. A new submission is only allowed after 24 hours.

Table 2
Boundaries for certificates

Type of certificate	Total score
Bronze	200–399
Silver	400–599
Gold	600–700

The header of a downloaded problem instance is shown in figure Fig. 1. It shows the date and time of production, as well as the username of the contestant.

For task B4: "Woorden leggen", Crosswords, a contestant had to place six give words on a 10 by 10 grid, each letter in a different cell, in the way a crossword puzzle or a Scrabble board is filled. All words had to be connected and no unintended short words should appear. The number of cells in the smallest enclosing rectangle had to be submitted.

Four of the six given words were the same for all contestants. The other two were selected out of a dictionary; the total length of the two words was 13 and each of the words had at least a length of 5. If our program discovered that this problem instance had no valid solution, another one was created.

This is of course a tricky problem. There are many suboptimal solutions, so a contestant has to be really convinced before submitting. The average score for this task was much below that of other B-tasks, as shown in Table 3.

In our second round, we take one of the B-tasks of the first round and we extend it to a programming task with several subtasks. This way contestants are already familiar with the problem behind the task. In the second round in 2016 we had a programming task Crosswords with 6 subtasks. Input were a diagram with a filled in crossword and a dictionary file with allowed words.

A. Count the empty cells.
B. Count the number of different letters used.
C. Count the number of different words used.
D. Make a list of all words, in a specific order.

Fig. 1. Header of a problem instance.

Table 3
Scores for B-tasks

ID	Title	No solution	Incorrect	<=20%	<=50%	<=100%	100%	Average
2014–2015 B1	Maze	26	45	0	4	24	83	78,94%
2014–2015 B2	Palindromes	36	14	1	2	42	87	72,52%
2014–2015 B3	Cities	39	8	0	1	24	110	74,99%
2014–2015 B4	Coins	43	8	0	1	23	107	72,37%
2015–2016 B1	Radio mast	29	26	1	1	56	107	78,41%
2015–2016 B2	Connections	42	15	6	3	37	117	72,43%
2015–2016 B3	Subsequence	47	17	0	5	30	121	73,97%
2015–2016 B4	Crossword	63	53	8	3	56	37	53,11%

E. Make a list of all words in the dictionary that can be added in the diagram.

F. Try to fill the diagram, minimizing the number of empty cells.

A seventh subtask was output only: Produce a 10 by 10 diagram with only valid words, and minimize the number of empty cells.

5. Results

After applying these changes in the Olympiad, the number of participants grew rapidly. Fig. 2 shows the number of contestants that earned points with their submissions.

The boundaries we used for giving certificates proved their value. A part of the participants with a bronze certificate was invited to the second round. Table 4 provides an overview of certificates and participation.

In 2015 only 76 out of the 100 invitees competed in the second round, in 2016 we invited 124 pupils and 84 of them joined the contest.

The introductory course in programming was a success. Between 2011 and 2014 the average number of submissions using Python was 18%. In the last two contests it was 45%. So a lot of the new participants use the language that they were trained in. Other contestants kept using the languages they knew already; we had submissions in C++, C#, Java, Pascal, PHP, Visual Basic and Haskell.

It was nice to see how some teachers use the A-tasks as part of their assessment for computer science education. We had at least three classes that submitted some of these tasks, with unique solutions for the participants. This is a form of collaboration that we encourage; we think that especially the A- and B-tasks should find their way to the classroom.

Some of the contestants started with the B-tasks. About 10 % of the contestants restricted themselves to these tasks. Since the cut-off for the second round was 240 points

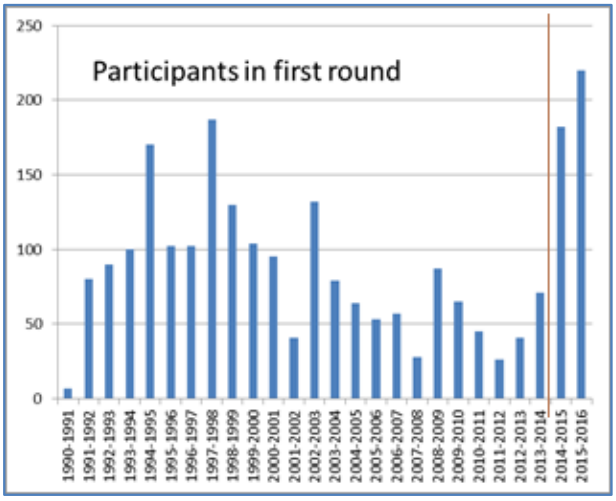


Fig. 2. Participation in the Olympiad before and after the new format.

Table 4
Results of the first round of the Olympiad

Results first round	2014–2015		2015–2016	
	Total	Girls	Total	Girls
Gold	13	0	23	0
Silver	35	0	65	1
Bronze, proceed	54	9	36	9
Bronze, not proceed	9	2	25	6
No certificate	71	12	71	9
Total with score	182	23	220	25
No score	47	12	94	16
Total users	229	35	314	41

in 2015 or 297 point in 2016, competing only with A- or with B-tasks could not get you an invitation for the second round.

And the girls returned to the Olympiad! Between 2005 and 2014 we had only three girls in our contests. None of them performed well enough that we could consider to let them advance to the final round. Table 4 shows that we reach a reasonable number of girls in this contest, about 12% of the participants. Alas, last year only 2 of the girls actually joined the second round. One of them scored very reasonable, the other one is still young and has a lot of years ahead to improve.

6. Discussion

We changed the contest format for the Olympiad in order to attract more Bebras-contestants to the Olympiad. This turned out well. The number of participants was at least tripled, the girls returned to the Olympiad and we welcomed many newcomers in programming, due to the introductory course. The certification method worked out well.

Which challenges remain?

1. We want to attract still more contestants. Given the discussions on the role of programming in education and the emphasis on computational thinking, both Bebras and the Olympiad offer possibilities to discuss tasks and backgrounds in a classroom. In Bebras we have a good working relation with many teachers. Getting the computer science teachers involved in the Olympiad, using for instance A-tasks as part of the assessment, can attract more participants.
2. New forms of tasks will be needed in the near future. Informatics as a subject is changing and developing all the time, for instance by introducing physical computing (Przybylla and Romeike, 2014) and the use of constructivists learning environments (Weigend, 2014). The contest format of the Olympiad gives a focus on algorithms. Other topics need to find a place. So we need to keep experimenting with new question types.

3. The Olympiad is still mostly a man's world. Finding partners, like focus groups on girls and technology, is a condition to improve the participation of girls in the Olympiad.

The tasks we introduced in our new contest were based on the work of colleagues in the international community. We found ready-to-use ideas, that we only had to fit into our new approach.

Exchanging experiences within this community is and will be an important base for further improvements.

References

Barendsen, E., Manilla, L., Demo, B., Grgurina, N., Izu, C., Mirolo, C., Sentence, S., Settle, A. Stupurienė, G. (2015). Concepts in K-9 computer science education. In: Dagienė, V. (Ed.), *ITICSE'15: Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, New York, NY, USA, 85–116. DOI: `http://dx.doi.org/10.1145/2858796.2858800`

Bebras website (2016). `http://bebras.org/`

Beverwedstrijd (2016). (In Dutch). `http://www.beverwedstrijd.nl/`

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

Codecup (2016). (In Dutch). `www.codecup.nl` or `nio3.codecup.nl`

Computer Science Circles (2016). `http://cscircles.cemc.uwaterloo.ca/`

Dagienė, V. (2006). Information technology contests – introduction to computer science in an attractive way. *Informatics in Education*, 5(1), 37–46.

Dagienė, V., Futschek, G. (2008). Bebras international contest on informatics and computer literacy: criteria for good tasks. In: R.T. Mittermeier, M.M. Syslo (Eds.), *ISSEP 2008*, LNCS 5090. Springer-Verlag Berlin Heidelberg, 19–30.

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.

Dutch Olympiad in Informatics (2016). (In Dutch). `http://www.informaticaolympiade.nl`

International Olympiad in Informatics (2016). *Output only*. `http://wiki.ioinformatics.org/wiki/Output_only`

Kubica, M., Radoszewski, J. (2010). More algorithms without programming. *Olympiads in Informatics*, 4, 157–168.

Maggiolo, S. (2015). An update on the female presence at the IOI. *Olympiads in Informatics*, 9, 127–137.

Przybylla, M., Romeike, R. (2014). Physical computing and its scope – towards a constructionist computer science curriculum with physical computing. *Informatics in Education*, 13(2), 241–254.

van der Vegt, W. (2009). Using subtasks. *Olympiads in Informatics*, 3, 44–48.

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

Weigend, M. (2014). The digital woodlouse – scaffolding in science-related Scratch projects. *Informatics in Education*, 13(2), 293–305.

W. van der Vegt is teacher's trainer in mathematics and computer science at Windesheim University for Applied Sciences in Zwolle, the Netherlands. He is one of the organizers of the Dutch Olympiad in Informatics and he joined the International Olympiad in Informatics since 1992. He was involved in the IOI-workshops on tasks in Dagstuhl (2006, 2010) and Enschede (2008). He also is one of the task designers for the Bebras contest.

Olympiads in Informatics, 2016, Vol. 10, 231–235

DOI: 10.15388/ioi.2016.15

Problem Solving, Presenting, and Programming: A Matter of Giving and Taking

Tom VERHOEFF
Dept. of Math. and CS, Eindhoven University of Technology
Eindhoven, The Netherlands
e-mail: t.verhoeff@tue.nl

Abstract. Nim is a well-known two-person game, where players alternate *taking* items from one of multiple piles. Finding a winning strategy for such games is a nice exercise in *problem solving*. Typically, the winning strategy for classical nim is explained in terms of nim sums, involving binary notation of numbers. I explain how to understand and play a winning strategy without prior knowledge of binary notation, which is useful when *presenting* this strategy in primary school. *Programming* that strategy is also an interesting challenge. This can be done elegantly in a functional language that supports patterns, such as the Wolfram Language. I conclude by *giving* you a variant of nim to work out yourself.

Keywords: impartial games, nim, functional programming.

1. Introduction of Problems to Solve

I teach enrichment classes in a primary school. The focus is on problem solving. One of the themes is puzzles and games that allow a mathematical analysis. Sometimes we work on 'exotic' (less known) puzzles and games, such as 'the princess on a graph', Perfect (2011), and others, Dore *et al.* (2010). But of course it is also important to address the classics.

One of these all-time classics is the following two-person game, known as *misère nim*[1], played as follows:

- Start with several piles of items (go stones, pennies, toothpicks, etc.).
- Players alternate turns.
- At each turn, a player takes one or more items from one pile.
- The player taking the last item loses.

Nim is an *impartial* game, in the sense that the set of available moves depends only on that state of the piles, and not on whose turn it is. The problem is to find a winning strategy: decide which positions are a win for the first player, and how to play in such positions.

[1] `https://en.wikipedia.org/wiki/Nim`

This can be formulated as a programming problem, where you need design and implement a function to select a move for a given game position. It can also be offered as a *reactive task*, Verhoeff (2009).
However, the problem that I first want to address is how to present a winning strategy that does not require prior knowledge of binary notation, so that it can be explained to pupils in primary school.

2. Presenting a Simple Solution

Impartial games are mathematically well understood via the *Sprague-Grundy Theorem*[2], especially for *normal* play, where the player who cannot move loses. But in this article we use the misère rule: the player who cannot move wins, making it a bit more interesting.

When only piles of size 1 remain, it is easy to see how the game proceeds. When the number of size-1 piles is even, the player to move wins, and loses when it is odd. Therefore, when all piles *except one* have size 1, the player to move can win, viz. by taking away from the larger pile such that an odd number of piles of size 1 remain, i.e., by taking away all or all-but-one items from the larger pile.

What remains to analyze are game positions with at least two piles whose size is larger than 1. It turns out (some explanation will follow) that in this case we need to break each pile into distinct groups whose size is a power of 2. Traditionally, this is accomplished by writing the pile size in binary notation.

However, I felt that too complicated to explain in primary school. And then it struck me that there is an easy way to avoid binary notation. For each pile, we are going to form groups as follows:

1. Break the pile down into groups of size 1.
2. Repeatedly combine two groups of equal size into one group.
3. This terminates when all group sizes are different.

Observe that it is an *invariant* of this process that the group sizes are powers of 2: Initially, in Step 1, the group sizes are $1 = 2^0$, and in Step 2 the group size doubles: $2 \times 2^i = 2^{i+1}$. The number of groups is a suitable *variant function*, which decreases in each iteration of Step 2, proving loop termination. Note that this is a non-deterministic algorithm.

Once all piles have been broken down into such groups, proceed as follows:

1. For each group size, determine how often such groups occur.
2. Consider the largest group size G that occurs an *odd* number of times.
3. If no such group exists, the position is lost; otherwise, proceed with Step 4.
4. Start by removing already 1 item from a pile P that has a size-G group.
5. Regroup the remaining items of that group as explained above.
6. For all group sizes $S < G$ in pile P, do the following:

[2] https://en.wikipedia.org/wiki/Sprague?Grundy_theorem

# size-S groups in P	in total	Remove from P
0	...	does not happen
1	even	0 size-S groups
1	odd	1 size-S group
2	even	2 size-S groups
2	odd	1 size-S group

7. After this move, all group sizes occur an *even* number of times. Hence, it leaves a lost position.

Breaking down the group of size $G - 1 = 2^k - 1$ in Step 5, you obtain k groups of all sizes 2^i with $i < k$. Therefore, in Step 6 every group size $S < G$ occurs at least once.

By the way, this also explains why these groups sizes, being powers of 2, are useful. But do note that there is no need to know about powers of 2 to carry out these algorithms.

3. Programming the Simple Solution

In a functional programming language that supports patterns, the first algorithm, which splits a pile into groups, can be elegantly expressed. Here it is using the Wolfram Language, Wolfram (2016):

```
(* Create groups of 1, given the pile size n *)
singletons[n_Integer] := Table[{1}, n];

(* Combine two equal groups, using a pattern *)
combine[{x___, a_, y___, a_, z___}] := {x, Join[a, a], y, z};
(* Do nothing if no equal groups are present *)
combine[list_List] := list;

(* Repeatedly combine equal groups until no more change *)
combineStar[list_List] := FixedPoint[combine, list];

(* Split a pile of size n *)
split[n_Integer] := combineStar @ singletons @ n;

(* Split all piles in a position *)
split[position_List] := Map[split, position];
```

Some *examples*:

```
singletons[5]
   {{1}, {1}, {1}, {1}, {1}}

combine[{{1}, {1}, {1}, {1}, {1}}]
   {{1, 1}, {1}, {1}, {1}}
```

```
split[5]
   {{1, 1, 1, 1}, {1}}

split[{3, 4, 5}]
   {{{1, 1}, {1}}, {{1, 1, 1, 1}}, {{1, 1, 1, 1}, {1}}}
```

The second algorithm, which determines whether a position is won and how to move is equally elegant.

```
(* Reduce by removing a pair of equal groups *)
reduce[{x___, a_, y___, a_, z___}] := {x, y, z};
(* Do nothing if no equal groups are present *)
reduce[list_List] := list;

(* Repatedly reduce a list until no more change *)
reduceStar[list_List] := FixedPoint[reduce, list];

(* Determine the nimsum of a list *)
nimsum[list_List] := reduceStar @ Catenate @ list;

(* Determine whether a position is won *)
won[position_List] := nimsum @ split @ position != {};

(* Move to make, if position is won and
   has at least two groups of size > 1 *)
move2[position_] :=
   Block[{s, ns, mx, i},
      s = split @ position;
      ns = nimsum @ s;
      (* Determine largest group in ns *)
      mx = First @ MaximalBy[ns, Length];
      (* Determine index of pile with group mx *)
      i = First @ FirstPosition[s, mx];
      (* Replace pile i with nimsum of that pile and ns *)
      ReplacePart[position, i -> Total[nimsum @ {ns, s[[i]]}, 2]]
   ];
```

An *example*

```
nimsum @ split @ {3, 4, 5}]
   {{1, 1}}

won[{3,4,5}]
   True

move2[{3,4,5}]
   {1, 4, 5}
```

Programming `move1` that moves optimally when at most one group has a size greater than 1 is left as an exercise.

I admit that it takes some time to get acquainted with functional programming. But once you do, it does pay off.

4. Conclusion

Finding a simple way for presenting a solution to a problem can in itself benefit from problem solving. I have applied this to presenting an optimal strategy for classical nim, a well-known taking game. Finally, I have shown how this strategy can be expressed in the Wolfram Language using functional programming.

Let me finish by giving you a new challenge. In *classical nim*, a player must take one or more items from *exactly one* pile. Here is a less well-known variant: at each turn, the player must take one or more items from *one or two* piles. Find the winning positions for the first player and how to determine a winning move.

While you are at it, the game *Chomp*[3], is still unsolved, in the sense, that we do not know how to determine a winning move for the first player (although it has been proven that the first player can win).

References

Dore, R. *et al.* (2010). "Math puzzles for dinner". *MathOverow*, 24 Jun. 2010. (Accessed 19 Jun. 2016) `http://mathoverflow.net/questions/29323/math-puzzles-for-dinner/`

Perfect, C. (2011). *Solving the* "*princess on a graph*" *puzzle*. Blog, 15 Dec. 2011. (Accessed 19 Jun. 2016) `http://checkmyworking.com/2011/12/solving-the-princess-on-a-graph-puz`

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

Wolfram (2016). *Wolfram Language*. (Accessed 20 Jun. 2016) `http://www.wolfram.com/language`

T. Verhoeff is Assistant Professor in Computer Science at Eindhoven University of Technology, where he works in the group Software Engineering & Technology. His research interests are support tools for verified software development and model driven engineering. He received the IOI Distinguished Service Award at IOI 2007 in Zagreb, Croatia, in particular for his role in setting up and maintaining a web archive of IOI-related material and facilities for communication in the IOI community, and in establishing, developing, chairing, and contributing to the IOI Scientific Committee from 1999 until 2007.

[3] `https://en.wikipedia.org/wiki/Chomp`

Olympiads in Informatics, 2016, Vol. 10, 237–247

DOI: 10.15388/ioi.2016.16

REPORTS

Gomel Training School for 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 technology used to teach programming and preparing for olympiads in informatics. Remarkable feature is a balanced education of four age ranges (preschoolers, grades 1–4, grades 5–8, grades 9–11) in four directions – thinking, mathematics, programming, algorithmization – and permanent Internet competitions to improve motivation. Distance learning system DL.GSU.BY is the effective technical base of the teaching.

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

1. Introduction

The author has been training pupils of different ages for the in programming and preparing them for olympiads in informatics in Gomel School 27 since September of 1996 and with help of the Distance Learning system DL.GSU.BY (further DL) since September of 1999.

The primary goals and objectives of this process are as follows:

- To develop in each child such properties as willingness to learn, analytical skills, self-dependence, and creativity.
- To give each child the base computer literacy.

- To help each child to understand what is "programming" and to decide whether his/her professional future will be with computer programming or without it.
- To prepare everybody who invests sufficient time to reach medals of national and international olympiads in informatics.

The results of this work during 1996–2015 are reflected in the following achievements of pupils of Gomel and Gomel region (Performance Statistics, 2015):

- More than one hundred pupils chosen programming as profession.
- Dozens of pupils entered universities without exams.
- 189 diplomas and 32 honorable mentions from national Olympiad in Informatics of Belarus.
- 37 diplomas of international collegiate programming contests (Sankt-Petersburg, Russia).
- 8 gold, 11 silver and 8 bronze medal of International Olympiad in Informatics (IOI).

Table 1 below lists the medal results from IOI 1997–2015 of the teams of different regions of Belarus as well as Minsk team and team of the Lyceum of Belarus State University separately represented on national Olympiad in Informatics of Belarus.

Important hallmark of the Gomel training school for olympiads in informatics is the start of learning as early as possible. It provides earlier (grade 10 or earlier) successes in the olympiads including IOI (all from city of Gomel unless noted otherwise):

- Aliaksei Danchanka (grade 9): 1998 – participation, 1999 – silver, 2000 – silver.
- Mikhail Svarycheuvski (10): 2000 – silver, 2001 – bronze.
- Raman Dzvinkouski (9): 2002 – bronze.
- Uladzimir Miniailau (9): 2005 – silver, 2006 – gold, 2007 – silver.
- Henadzi Karatkevich (5): 2006 – silver, 2007–2012 – gold.
- Uladzislau Padtsiolkin (9): 2011 – silver, 2012 – silver, 2013 – bronze.
- Siarhei Kulik (9, from Mozyr[1]): 2001 – silver, 2012 – bronze, 2013 – gold.

Table 1
Medals from the national olympiad

Region	Total	Gold	Silver	Bronze
Gomel region	28	9	11	8
Minsk region	11	1	5	5
Lyceum of BSU	9	1	3	5
Vitebsk region	7	0	3	4
Minsk	5	0	3	2
Grodno region	4	0	2	2
Brest region	1	0	0	1
Mogilev region	0	0	0	0
Total	**65**	**11**	**27**	**27**

[1] Mozyr – town in Gomel Region.

- Adam Bardashevich (10, Mozyr): 2001 – silver, 2012 – gold.
- Fedar Karabeinikau (8): 2014 – participation, the first under bronze cutoff.

Unconditionally, their success is based on their own hard work as well as their parents' efforts on education and the creation of conditions for rapid growth. In addition, the success of Mozyr pupils is strongly connected to Alexey Borunov, Mozyr coach for programming contests. Nevertheless, the author believes that the education environment described in this paper essentially helped everybody mentioned above and many other medalists of national and international olympiads in informatics to achieve such remarkable results.

The remainder of the paper is structured as follows: Chapter 2 represents the author's educational principles; Chapter 3 describes the elaborated education courses, exercise packages and permanent Internet competitions; Chapter 4 contains the author's educational strategy; Chapter 5 represents Gomel Regional Olympiad in Informatics; Chapter 6 is devoted to newly elaborated "Accelerated Learning" approach to teaching; finally Chapter 7 contains conclusions.

2. The Author's Educational Principles

This chapter describes the principles used by author to organize the teaching process.

Constancy. Lessons are held in informatics cabinet of Gomel School 27 every Wednesday and every Sunday, even on holidays (including summer). When the author is absent from Gomel, he gets somebody to fill in for him. The site DL.GSU.BY supports the education process around the clock, so everyone can study anytime and anyplace at their own convenience.

Inclusiveness. The author does not deny anyone who comes to learn. Now the lessons are organized in such a way that one can begin studying not only from the first grade of school but also from preschool ages – as young as 4 years (Dolinsky, 2013).

Efficiency. The main criteria of efficiency of lessons is the ratio of the study time to total usage time. The author tries to organize the study so that each pupil works hard each minute of each lesson, either learning something new or consolidating their skills.

Individuality. The author has for many years believed that efficiency in the sense described above can be achieved only through individual and differentiated instruction. That is, each pupil moves on at their own rate and accordingly at any given time all pupils can be at different points of the learning process.

Self-dependence. Obviously such individuality can be achieved only though independence of learning, since a teacher physically can't immediately teach (to explain, to listen, etc.) all pupils simultaneously, particularly if the pupils work with different materials. Furthermore, the self-dependence is an important objective of the education. By and large, the author's view is that teaching independence is even more important than teaching any specific knowledge. It is especially true in the area of programming where one needs to learn and relearn all professional life.

Optimality (of material selection). The author tries to select the most useful material and construct the process of studying it in the most effective system. The author can't claim to have achieved perfection, but it is true that he does try to teach today better than yesterday and tomorrow better than today.

Demands (to comply with the rules). Unfortunately, the list of rules is ever expanding, complicating the compliance: keep silence in the audience; know where the classes are held; bring the notebook and pen; come to class before lesson starts, leave when the lesson is over; do exercises himself, (don't cheat); etc.

Using the site DL.GSU.BY. Since 1999, the author has headed the development of this project and actively used it for teaching. This approach has the following advantages:

- Fast verification of solutions (from a few seconds to a few minutes) and as a result – multiplying the intensity of the teaching process.
- Variety of task types – differentiating the study and keeping interest of students with different levels of preparing and motivation.
- Automatic presentation of tasks and differential study (Dolinsky, 2012) – the technical basis for customized education.

Teacher activity. The author tries to organize the studying process in such a way that each student works independently as much as possible.

3. Educational Courses, Exercise Packages, and Permanent Competitions

The teaching process is based on using the distance learning system DL and includes the following learning and training courses: "Programming-professionals (individual)", "Programming-professionals (collegiate)", "Programming-beginners", "Programming-professionals (individual) work on errors (W/E)", "Programming-professionals (collegiate) W/E", "Programming-beginners W/E", "Preparing for IOI", "Methods of algorithmization", "Basic programming", "Start to program", "Informatics", and "Mathematics".

The course **"Programming-professionals (individual)"** is open from beginning of December to middle of July/August – until IOI. It contains weekly Sunday 5-hour (9:00–14:00 GMT+3) individual contests – tasks from past olympiads of regional level until the first decade of January (when Gomel regional olympiad in informatics is held), then from olympiads of national level until end of March (when national olympiad in informatics of Belarus is held), and from olympiads of international level until the close.

The course **"Programming-professionals (collegiate)"** is open from the middle of July/August (after IOI is finished) until the end of November, when Collegiate Programming Contest in Sankt-Petersburg is held. It has weekly Sunday 5-hour (9:00–14:00 GMT+3) contest with tasks from past collegiate contests.

So, the whole year (including holidays) every Sunday there is 5-hour individual or collegiate open on-line contest for all comers. Immediately after the olympiad is finished the tasks become available for practice in courses **"Programming-professionals (individual) (W/E)"** and **"Programming-professionals (collegiate) W/E"**, respectively.

For participants in informatics cabinet of Gomel School 27, there is additionally interactive discussion of solutions.

Tasks are classified by themes and copied into appropriate branches of the course **"Methods of algorithmization"**. Thus, the latter is adjusted to the trends in the development of competitions and provides targeted preparation for olympiads. In addition, the tasks that were not solved are copied to the course **"Preparing for IOI"**.

To activate self-study and practice in the courses "Methods of algorithmization" and "Preparing for IOI" we hold the permanent Internet competitions: Autumn Cup, Winter Cup, Spring Cup, Summer Cup, and Whole Year Prize. In the Cups we award the three the best pupils who solve most tasks for the season (autumn, winter, spring, summer). In the "Whole Year Prize" we awarded the one pupil who solves the most tasks for the whole year (autumn-winter-spring-summer). In the course "Preparing for IOI" it is insufficient to solve one task, but in addition one needs to describe their solution in the DL forum corresponding to the subject of the task.

The courses "Programming-professionals (individual) (W/E)", "Programming-professionals (collegiate) W/E", "Methods of algorithmization", "Preparing for IOI", feature "Tests grants". That is, a pupil can get the test data (input data, model answer, and program-checker) on which their solution failed. There's a special FAQ and a dedicated DL forum to support pupils in the situation "I got the right answer on my computer, but my solution was rejected".

For individual contests (after their end) as well as for all the courses, there are permanent result tables with links to solutions of all participants (as images to prevent copying and resubmitting).

The DL forums have links to authors' systematic descriptions of solution as well as to descriptions by pupils of Gomel and university students.

The author elaborated a set of tutorials, two from them have been published as books in Sankt-Petersburg (Dolinsky, 2005, 2006).

Another important aspect is the joint participation of pupils and university students in the weekly Sunday olympiads where students are training for ACM ICPC (only for quarter-finals and semi-finals so far).

The weekly Sunday olympiads as well as practice in other weekdays is targeted towards medalists of national olympiad in informatics preparing for International Collegiate Programming Contests and IOI, as a rule pupils of grades 9–11.

For pupils of grades 5–8 preparing for regional olympiads is the course **"Basic programming"** with possibilities of automatic task presentation and differential study. The exercise system has tree-like structure. Correct solution provides transition to the next exercise. Wrong solution or pressing the button "I don't know" transitions the student into subtree teaching to solve the problem. Own teaching subtrees may be settled for any such teaching tasks. This way we provide individual differential teaching that adapts not only to the level of preparation of a pupil but also to their current emotional and physical state. Pupils who know more and are in better shape branch less and thus advance faster. The course "Basic programming" has exercise packages with differential teaching on the following topics: introduction to programming, one-dimensional arrays, two-dimensional arrays, geometry, strings, sorting, queues. To increase the motivation

of beginners, there are weekly Sunday olympiads in the course **"Programming-beginners"** from 7:00 to 20:00 (GMT+3). One can solve the same tasks for practice in the course **"Programming-beginners W/E"**.

For pupils in grades 1–4 there is the learning course **"Start to program"** that largely contains the same tasks as "Basic Programming", but in essentially linear form. Most of teaching trees are reorganized into sequences of tasks for simpler learning. There are permanent Internet-competitions "Season Cups" and "Whole Year Prize" for learning courses "Basic Programming" and "Start to program". Note that in the competitions the tasks from the learning subtrees are not counted. In the course "Start to Program" only pupils of grades 1–4 and in the course "Basic Programming" only pupils of grades 1–8 are awarded.

The training course **"Informatics"** was created for pupils of grades 1–4 of Gomel School 27. Originally there was only one package of exercises: "Learning to think" (Dolinsky, 2013). But soon we started to copy exercises from "Start to Program" to for more advanced pupils in "Informatics". Moreover, we re-open "Informatics" every academic year, which gives more possibilities to change it. So, now the course "Informatics" has practically the same material as the course "Start to Program" has, but in a better methodological form. The course "Informatics" also has permanent Internet-competitions "Season Cups" and "Whole Year Prize" where pupils of grades 1–4 are awarded.

The "Basic Programming", "Start to program", and "Informatics" courses are based on the Pascal programming language.

Finally, the course **"Mathematics"** contains different tasks in mathematics, including such exercise packages as flash tasks on mathematics of grades 1–5, tasks from the international mathematics contest "Kangaroo" (2001–2015, all grades), Canadian Math Contests (1998–2015, grades 7–11), "Math from informatics". In the latter, the tasks are from the course "Programming-beginners", but reformatted so that the pupil needs to manually enter the answer for given input data. The answers can be computed manually or using an appropriate program. For the course "Mathematics" there also are also permanent Internet-competitions "Season Cups" and "Whole Year Prize" where pupils of grades 1–8 are awarded.

Tables 2 and 3 represent participation statistics from 2008 to 2014.

Table 2

	2008/09	2009/10	2010/11	2011/12	2012/13	2013/14	2014/15
Mathematics	–	–	–	–	146	175	214
Informatics	–	–	–	249	301	179	212
Start to program	22	182	142	118	174	189	300
Basic programming	10	263	205	159	290	309	468
Methods of algorithmization	205	269	340	332	491	425	441
Preparing for IOI	31	25	8	7	12	17	3
Total	**237**	**739**	**695**	**820**	**1424**	**1294**	**1638**

Table 3

	2008/09	2009/10	2010/11	2011/12	2012/13	2013/14	2014/15
Programming-prof. (ind.)	98	116	128	165	157	141	133
Programming-prof. (col.)	51	59	73	101	55	113	71
Programming-beginners	77	67	59	89	108	69	211

4. The Educational Strategy

Currently we are providing education for four age ranges: preschoolers (from 4 years), junior school (grades 1–4), middle school (grades 5–8), and high school (grades 9–11), in four directions: thinking, mathematics, programming, algorithmization:

- **Thinking.** The author believes that development of thinking needs to be first to provide higher learning efficiency. For preschoolers and juniors the training course "Informatics" has special packages of exercises "Learning to think", "Differences", "Analogy", "Learning to Count", and "Tangram". The courses "Start to Program" and "Basic Programming" have embedded exercises for development of thinking.
- **Mathematics.** It is clear that knowledge of mathematics is important in itself, but it is more important that development of abilities to solve mathematical problems automatically develops effective thinking skills. On one hand, there is training course "Mathematics" containing different mathematical tasks for pupils from grade 1 to grade 11. On the other hand, training courses "Informatics", "Start to Program", "Basic programming" have special packages of exercises where mathematics and programming are integrated. For example, the course "Informatics" has packages "Math (programs)" for grades 1–5. Each of them contains tasks from appropriate mathematics textbook, converted to programming tasks by parameterization. To solve the task, one need to solve it mathematically and then write the corresponding program. The course "Basic Programming" contains packages "Kangaroo grades 3–4" (2001–2008) and "Kangaroo grades 5–6" (2001–2009) where appropriate mathematics tasks are converted into programming tasks.
- **Programming.** We are using the Pascal programming language to teach pupils in grades 1–8. But the language is a means, not a goal. Special attention is given to debugging technology as well as structuring of program sources to improve their readability and understandability.
- **Algorithmization.** Best of all we are trying to develop skills for algorithm elaboration. The second direction is study of standard algorithms. Both directions are supported by weekly olympiad solving as well as the following practice on unsolved tasks and studying the needed theory. In addition, the course "Methods of algorithmization" is strongly structured around themes and subthemes and contains information about task sources (year, country, etc.).

5. Regional Olympiads in Informatics

Since 2010 the tasks for the Gomel regional olympiad that qualifies 15 best participants to the national olympiad are created by the national Scientific Committee. But in Gomel region we have at least five olympiads whose tasks we prepare ourselves: in autumn two olympiads (school and city level) for grades 1–11 and in spring three olympiads (school, city, and regional level) for grades 1–9. For those five olympiads we introduce three age divisions: in autumn grades 1–4, grades 5–8, grades 9–11; and in spring grades 1–4, grades 5–7, grades 8–9. We use the tasks of olympiads for:

a. Targeted training for higher level olympiads.
b. More accurate indication to students and teachers of what and how to learn.

Tasks for grades 9–11 (in spring grades 8–9) include three groups of increasing complexity (note that each pupil needs to try solve all tasks from all groups):

- *First group (5 tasks)* – tasks on topics from the course "Basic Programming" (one-dimensional arrays, two-dimensional arrays, geometry, strings, greedy (simple algorithm based on sorting)).
- *Second group (5 tasks)* – tasks based on the course "Methods of algorithmization": queues, recursion, dynamic programming, graphs, and brute force.
- *Third group (2 tasks)* – tasks from the course "Preparing for IOI", as a rule on the following topics (or their combinations): research, complex dynamic programming, complex data structures, and complex problems on graphs.

This approach allows participation "with interest" for pupils who only began to study programming (first group of tasks). At the same time we can determine the preparation level of those who spent more time (second group of tasks) and point everyone to the topics that need more detailed work. Finally the third group of tasks provides engagement for all 5-hours for the most prepared students, the strategic goal for whom is preparing for international olympiad in informatics.

Tasks for grades 5–8 (in spring grades 5–7) also include three groups of increasing complexity:

- *First group (5 tasks)* – tasks on topics from course "Basic Programming": introduction to programming, one-dimensional arrays, two-dimensional arrays, geometry, sorting.
- *Second group (3 tasks)* – strings, story problem, research task.
- *Third group (2 tasks)* – implementation task from a real olympiad, queues.

For the first group the simplest tasks from each topic are chosen. It is a way to check whether a pupil has studied the topic at all, because the task has a minimum of text and is standard for the topic.

The second group of tasks is to differentiate pupils (who have the preparation level to solve tasks from the first group) on the skills needed to solve the olympiad tasks. *The first important such skill* is development and debugging of own algorithms. The string processing task is one where the problem formulation is very simple and easy to understand. The main difficulty is to formulate the solution process (algorithm) in a

programming language. *The second important skill* for solving is the ability to read and understand the task text, to differentiate the important, minor, and insignificant things, to reformulate the problem in mathematical and programming terms. So our story problem is the task with detailed (or even cumbersome) formulation but with very simple solution. We use simplified versions of tasks from national olympiads. *The third important skill* is ability to research the solution space. The task statement may be very brief so that it is clear what needs to be done. The main problem is to invent a way to get the result. To get such tasks, we reformulate in programming terms tasks from international mathematics contest "Kangaroo" (grades 5–6).

Finally, the third group of tasks checks students' readiness to solve original problems from real olympiads. First task is on implementation details from one the following topics: one-dimensional arrays, two-dimensional arrays, geometry, strings. The second is an original task on queues.

Tasks for grades 1–4 also include three groups of increasing complexity:

- *First group (10 tasks)* includes tasks from "Introduction to programming" (three tasks on integers, one task on each of characters, strings, length of strings, number of characters in string) and three tasks on using Pascal standard subroutines DELETE, COPY, POS to delete and copy strings and to find position one string within another.
- *Second group (5 tasks)* includes tasks on one-dimensional arrays: sum of all elements, number of elements with some feature, minimal/maximal elements, search for the first element with some feature.
- *Third group (5 tasks)* is to differentiate more prepared students and includes the tasks on the following topics: two-dimensional arrays, geometry, strings, research (base on "Kangaroo" tasks for grades 2–3), and a story problem.

Systematic and purposeful development of tasks for regional olympiads is an important means of improving the student and teacher preparation in the region.

Note that all five regional olympiads in all three age divisions are on-line events, so usually students from all over Belarus participate in them.

6. Accelerated Learning

To improve the teaching results, recently a few new packages of tasks and exercises were introduced:

- **"Accelerated course – 2013"** includes the following topics: introduction to programming, one-dimensional arrays, two-dimensional arrays, geometry, sorting, strings, story problem, research task. Each topic includes three parts: theoretical minimum, tasks from Gomel olympiad for grades 1–4, tasks from Gomel olympiads for grades 5–8. Such approach allows the most capable pupils to move with maximal speed. At the same time pupils who encounter difficulties in this course can use the standard learning approach.

- **"Olympiads grades 5–8 by topic"** includes tasks from corresponding Gomel olympiads, grouped by topics. It allows to see potential rating of each student and it definitely shows to him and to his teacher the areas for further work.
- **"Olympiads grades 9–11 by topic"** is similar to the above in composition and purpose. It includes additional topics such as recursion, dynamic programming, graphs, brute force, and more complex tasks (with complexity level corresponding to national and international olympiads).
- **"Belarus olympiads"** includes tasks from national olympiads of Belarus (qualification and final stages) grouped by topic and in the order of increasing complexity. Note that we include in that course a special set of tasks with incomplete solutions. To get the full solution of such task one needs to know special theory as well as have good skills of developing and debugging complex algorithms. At the same time the olympiads up to IOI don't demand full solutions for all tasks because winners are defined by sum of points. So it is important to develop skills for solving tasks partially. We gather such tasks into special theme "Incomplete solutions" and remove the tests cases that can't be solved by partial solutions. Despite the simplicity, such solution can get from 20 to 80 points and can essentially improve the final result of a contestant in an olympiad. Moreover such simplified solutions may be useful to verify full solution.

We believe that development of these courses is essential not only for preparing to olympiads but also to identify "flaws" in the education system with a view to their eventual elimination.

Note that the author has written and documented the solutions for all tasks (for medal minimum points) from the course "Belarus olympiad" and also for more complex tasks for courses "Olympiads grades 9–11 by topics" and "Olympiads grades 5–8 by topic". So, if a pupil can't solve some task there are two options: ask for help from somebody who already solved it or read the description of the author's solution.

The shortest way for a pupil from grades 5–8 to reach the medal level of national olympiad in informatics of Belarus is: "Accelerated course", "Olympiads 9–11", "Belarus Olympiads".

7. Conclusion

We represented the current state of the system for preparing students of Gomel and Gomel region for olympiads in informatics as well as strategic development directions.

Note that the system is also used by pupils outside of Gomel region. A remarkable feature is permanent monitoring of preparation state for all pupils as well as a balanced education of four age ranges (preschoolers, grades 1–4, grades 5–8, grades 9–11) in four directions: thinking, mathematics, programming, algorithmization.

To improve motivation for ongoing education, we organize the permanent Internet competitions (Autumn-Winter-Spring-Summer, Whole Year).

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 children. *Olympiads in Informatics*, 8, 63–68.

Dolinsky M. (2005). *Algorithmization and Programming with TURBO PASCAL: From Simple to Olympiad Problems: Tutorial*. Sankt-Petersburg "Piter". (In Russian).

Dolinsky M. (2006). *Solving of Sophisticated Olympiad Programming Problems: Tutorial*. Sankt-Petersburg "Piter". (In Russian).

Performance Statistics of Gomel Pupils in International and National Olympiads in Informatics from 1997 to 2015. (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 pupils in Gomel to participate in programming contests and olympiads in informatics. He was the 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, 2016, Vol. 10, 249–254

DOI: 10.15388/ioi.2016.17

Armenia: IOI Participation and National Olympiads in Informatics

Vahram DUMANYAN, Armen ANDREASYAN
Department of Informatics and Applied Mathematics, Yerevan State University
1 Alex Manoogian, Yerevan, 0025, Armenia
e-mail: duman@ysu.am, andreas@ysu.am

Abstract. The article describes technology used to teach programming and preparing for Olympiads in Informatics in Armenia. It describes the current model of organizing the competition: steps, target groups, methodologies of development, evaluation, and the preparation of participants for the different stages of the competition.
Keywords: IOI, olympiad in informatics, teaching programming.

1. IOI Participation

Armenia started participating in the IOI in 1996 in Hungary. Next year Armenian delegation could not go to the South Africa. Our students won first medal (bronze) in 2002. Since then they returned from an IOI with at least one medal. For us especially significant are IOI'2005 when Vahe Musoyan won gold medal, and IOI'2014 when all four students won medals (2 silver and 2 bronze). Since now Armenia has 1 gold , 4 silver and 20 bronze medals. You can find all details in the Statistics page (International Olympiad in Informatics – Statistics).

Medalists of an international olympiads may enter appropriate departments of universities of the country without exams and study for free.

In the first years, team leaders found sponsors for the trip expenses. For the last 15 years, these costs are covered by the Ministry of Education.

Armenian students participate in International Zhautykov Olympiad (Iglikov *et. al.*, 2013) – 3 silver and 7 bronze medals, and APIO – 3 bronze medals.

2. National Olympiad Procedure

The National Olympiads among school student are held by the Ministry of Education and Science. Every year the schedule of Olympiads is made. The assignments and the

results of all the school Olympiads held in Armenia are placed in (School Olympiads) and in (Armenian Educational Portal) websites.

YSU Faculty of Informatics and Applied Mathematics is responsible for the National Olympiad of Informatics.

National Olympiad is held in three stages. There are no divisions according to the grade and age. The majority of the participants are students of grades 10–12, but there are also lower grade students. The first phase is the school stage and is held at schools in February. Until 2014–2015 school year, the schools conducted that stage independently and chose their best students. Schoolteachers made the assignments themselves and held the competitions. Sometimes the content and form of their assignments did not correspond to the assignments of next rounds. There are no statistics on the number of participants of those years.

The second stage is regional and is held in early March. Armenia is divided into 10 provinces. On this stage, the capital Yerevan is considered as a separate entity. The students of the provinces gather on the same day and the same time. They solve the problems compiles by the Commission. Before 2014–2015 school year, the juries of every province checked the results according to the provided tests and chose the best students themselves. The biggest turnout was in Yerevan. Yerevan municipality and regional administrations reward the winners of their regions with diplomas. Each province and the capital have their quotas for the participation on the next stage. Yerevan has 20–25 seats and each of the provinces has 4–6 seats. However, some regions sent fewer students or did not send at all.

The third stage is the National. It is held in late March or early April in the computer center of YSU. In the last 10 years, the competition is held for 2 days. As a result, best students get rewards from the Ministry (diplomas, certificates). The Committee decides the number of diplomas. The students that got diplomas, form the selection group.

3. New Format of Providing National Olympiads

In 2014, one of the participants of IOI'2013 made new contest management system. It is qualitatively higher than the previous system. The new system makes it possible to hold online contests, to put interactive problems and IOI – format (with subtasks) problems. It was decided to hold the competition of all three stages of the Olympiad with this system, and to hold the first two stages online. Any student mastering programming languages C, C++, Pascal can take part in the first stage, informing the school administration beforehand. The list of the participants is published in the `olymp.am` according to the school information.

The second stage of Olympiad is held in certain places given by the regional administrations (in Yerevan the Municipality decides the place). The students with best results from the first stage take part in the second stage. According to the decision of the Organizing Committee, the school of the student is not taken into consideration.

The organizing committee also determines the list of participants of the third stage, based on the results of the second one. The quotas of the provinces have been eliminated. As a result, the number of students from Yerevan has been significantly increased.

4. Selection Competitions and the Preparation for the IOI

The selection group is formed according to the results of the national stage of Olympiad. Theoretical and practical classes are held for the group. Then, according to three competitions, four students who will represent Armenia in IOI are determined. For the preparation for the IOI 1–2 week summer camp is held. Last years, not only the members of IOI group take part in Summer Camp, but also the students of selection group, that still study at school. Not only the group leaders held the lessons but also previous years' IOI medalists and participants. The Summer Camp is held at Camp "Ughetsir" of Quantum College, located in the mountain resort of Aghavnadzor, as well as in the YSU Rest House, Tsakhkadzor.

5. The Role of Online Competitions and the Online Judges

Under current conditions, it is impossible to succeed without systematic long-term studies. We think that online competitions held for the students are very useful.

Armenian National Olympiad problems can be solved in am.spoj.com. We are grateful to Spoj team for this opportunity. During our training, we use other archives too. Initially, we use Michael Dolinski's website (DOLINSKY, 2013). We would like to single out St. Petersburg cycle of online Olympiads (Olympiads in Informatics: St. Petersburg, Russia), USACO online contests (USA Computing Olympiad), Croatia online contests (Croatian Open Competition in Informatics). Thanks to the organizers for giving the Armenian translations of the competition assignments in their websites. Codeforces' competitions (Codeforces) are useful, too. It also contains numerous resources (descriptions of algorithms, virtual competitions), which are suitable for trainings.

6. Specifics of National Olympiads

Programming is a small part of School program of informatics. It is included in 11th grade program, and it has quite superficial presentation. Informatics is not among the examination subjects. Applicants for admission to university programming departments have to pass mathematics and physics or English. Only a few schools have Olympic circles of informatics.

In Armenia everything is concentrated in Yerevan: economy, science, culture, education. One can say that Yerevan dominates the provinces. The population of the provinces is almost twice more than in Yerevan.

For last two years, it has become possible to get information about participation in first two stages due to the online competition of regional stage. We have the following statistics: Fig. 1, Fig. 2, Fig. 3.

The number of the participants from provinces has decreased especially in 3rd stage.

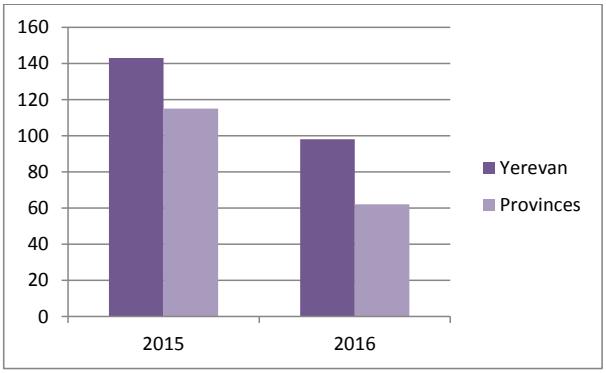


Fig. 1. Number of participants at the 1st stage.

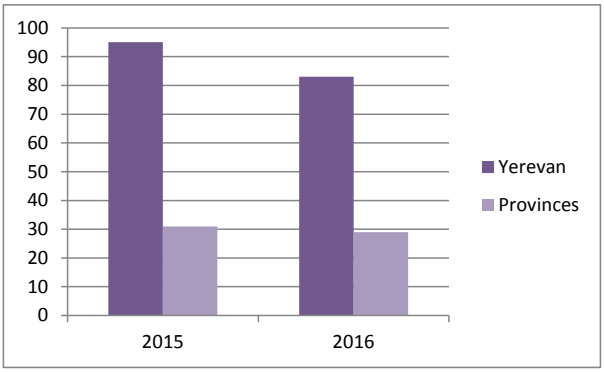


Fig. 2. Number of participants at the 2nd stage.

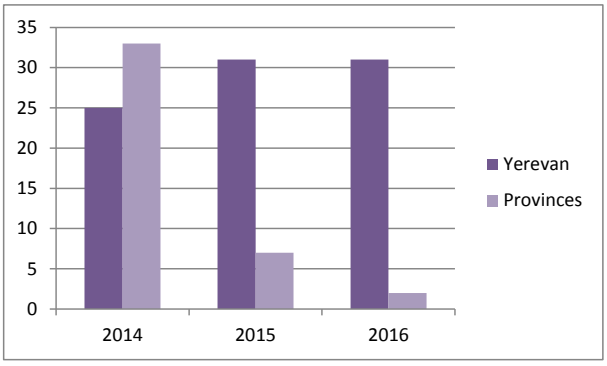


Fig. 3. Number of participants at the 3rd stage.

In the capital city there is majority of two schools. In the following chart you can see representations of schools in the republic stage of National Olympiads in 2016: Fig. 4.

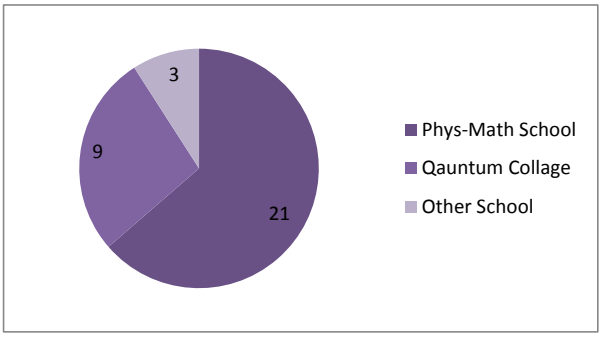


Fig. 4. Participants of 3rd stage of National Olympiads in 2016.

Physics and Mathematics Specialized School is founded in 1965 by academic Artashes Shahinyan. Every year students of the school represent Armenia in the international Olympiads on Informatics, Mathematics, Physics, Chemistry, Biology, Astronomy. In 2015 students of the Phys-Math school got 14 medals from these International Olympiads. Quantum Collage is the first private school and founded in 1991. This school also has good Olympic history and traditions. Until 2014, only 12 participants from each of these two schools were allowed to take part in the second Stage of Olympiads. Now this restriction is removed. The students representing Armenia in the IOI are mostly from those schools. 13 of Armenian 25 medals are of Quantum College and 8 of Physic-Mathematical School. 24 students from 52 participants from 2003 to 2015 were of Quantum College, and 20 students of Physic-Mathematical School. This is explained by the fact that both schools have Olympic groups that work throughout the school year.

However, we are sure that there are talented students in provinces, too, as there are exclusive precedents. School student from the second city of Armenia, Gyumri, won the first medal. In 2012, a student from the southern mountainous part of Armenia, Syunik, was involved in the group and won bronze medal in IOI. He gained his basic knowledge himself through the Internet.

Now we think to take measures to increase the interest and the participation of province students, organize in these two strong schools some kind of distance lessons, online materials accessible for all students in the country. Also we consider switch to the two level contests opening second division for students up to 9-th grade like in Lithuania (Dagienė, Skupienė, 2007), Serbia (Ilić, 2012).

7. Conclusion

Armenia has been participating in IOI since 1995 and has achievements. In recent years, great work for the improvement of the quality of National Olympiads has been done. There are some problems with the promotion of the Olympiads. Measures should be taken to spread the necessary knowledge of Informatics and to involve more students and schools in the National Olympiads.

References

Armenian Educational Portal. `http://armedu.am`
Codeforces. `http://codeforces.com`
Croatian Open Competition in Informatics. `http://hsin.hr/coci`
Dagienė, V., Skupienė, J. (2007). Contests in programming: quarter century of Lithuanian experience. *Olympiad in Informatics,* 1, 37–49.
Dolinsky, M. (2013). An approach to teaching introductory-level computer programming. *Olympiads in Informatics*, 7, 14–22.
Iglikov, A., Gamezardashvili, Z., Matkarimov, B. (2013). International Olympiads in Informatics in Kazakhstan. *Olympiads in Informatics*, 7, 153–162.
Ilić, A., Ilić, A. (2012). IOI trainings and Serbian competitions in informatics. *Olympiad in Informatics*, 6, 158–169.
International Olympiad in Informatics – Statistics. `http://stats.ioinformatics.org`
Olympiads in Informatics: St. Petersburg, Russia. `http://neerc.ifmo.ru/school/io/index.html`
School Olympiads (in Armenian). `http://olymp.am`
USA Computing Olympiad. `http://usaco.org`

V. Dumanyan, Doctor of Sciences (Mathematics), Yerevan State University; Dean of the Faculty of Informatics and Applied Mathematics.
Organizer of National Olympiads in Informatics (since 2013).
Leader of national team on International Olympiad in Informatics (2007, 2010, 2013–2015).
Director of Armenia Subregion in the ACM ICPC Olympiads (since 2004).

A. Andreasyan, MSc in computer science from Moscow State University. Assistant professor at the Department of Programming and Information Technologies of Faculty of Informatics and Applied Mathematics of the Yerevan State University.
Organizer of the summer camps for preparing the Olympic teams. Organizer of National Olympiads in Informatics. Leader of national team on International Olympiad in Informatics (since 2003). Coach of Yerevan State University teams in the ACM ICPC (since 2003).

Olympiads in Informatics, 2016, Vol. 10, 255–262

DOI: 10.15388/ioi.2016.18

Olympiads in Informatics in Republic of Moldova

Anatol GREMALSCHI[1], Angela PRISACARU[2], Sergiu CORLAT[3]
[1]*Institute for Public Policy, Chisinau, Republic of Moldova*
[2]*Department of Pre-University Education, Ministry of Education, Republic of Moldova*
[3]*Department of Informatics and Information Technologies, Tiraspol State University, Chisinau, Republic of Moldova*
e-mail: anatol_gremalschi@ipp.md, aprisacaru@gmail.com, sergiu.corlat@math.md

Abstract. This article presents the retrospective of organizing the Republican Olympiad in Informatics from the first editions. It describes the current model of organizing the competition: steps, target groups, methodologies of development, evaluation, and the preparation of participants for the different stages of the competition.

Keywords: informatics curricula, IOI, olympiad in informatics, teaching programming.

1. The History

Informatics was introduced into the undergraduate studies in 1985. From the beginning, the curriculum was oriented to systematic studying the basic computing concepts: the structure of information and the basic principles of programming. This allowed the placement of informatics in the science field. Consequently, the discipline has benefited from lots of opportunities of organizing school contests at various levels, as well as other sciences.

1.1. *Year 1987*

The first Olympiad at the national level was organized in 1987. At that time, the pre-university educational institutions had practically no computers; first editions of the Olympiad were held based on theoretical examinations, as well as mathematical competitions. Competition topics derived from various areas of applied mathematics, theoretical computer science, logic. From edition to edition, algorithmic component became more pronounced, the subjects were moving towards practical component – to identify computer models for problem solving in various fields.

The advent of computers in schools set the beginning of a qualitatively new stage in informatics competitions. It became possible to implement the theoretical solutions in a programming language and to test these solutions. The first automated evaluation systems appeared practically immediately.

1.2. *Year 1996*

Since 1996, students from Moldova have participated in the International Olympiad in Informatics. This also imposed adjustments to the organizational model for the Republican Olympiad in Informatics. The IOI is the reference point from which the development of all structural components of the national Olympiad flows: scientific component (competition tasks, evaluation), organizational (cultural, communication, scientific sessions), technical (computer network).

During the same period, schools were implementing individual model of using computers in labs (one student – one computer). This resulted in increase of the number of pupils fond of informatics and of programming in particular. The motivation of participants at the Olympiad increased at both national and local level. The winners of the Olympiad got national testing facilities, preferential admission to universities in the country, various awards; and a chance to compete for participation in international competitions.

1.3. *Year 2000+*

The organizational principles and scientific assessment were finalized, supported by rules and other normative acts of the Ministry of Education. The Olympiad "matured" and continued to develop in line with the national curriculum of the computer science and the Olympic principles promoted by the Organizing Committee of the IOI. In 2004 Moldova became a member of the community of the Balkan Olympiad in Informatics and in 2007 hosted it for the first time.

In recent years the Republican Olympiad in Informatics has generated a number of local competitions organized by various educational institutions: individual and team programming competitions; online contests; ICT usage competitions; robotics competitions, etc. So, the number of students fond of various fields of Computing is growing.

2. Moldavian Republican Olympiad in Informatics

2.1. *The Organizational Model*

The Olympiad is held annually during the academic year: from September to May. The National Olympic Committee, the composition of which is approved annually by the Minister of Education, organizes the Olympiad. The competition is conducted in three stages: local, regional, and national (Fig. 1).

On the local and regional stage, each grade is considered a separate age group. There are two age groups on the national stage: middle school (up to 9th grade) and high school (grades 10–12).

Local Stage. There is a preparation period from September to December and internal competitions in educational institutions take place in January (p. 1.a and 1.b on Fig. 1).

Regional Stage. It is organized separately in each administrative unit. Teams of the educational institutions selected at the local stage participate on the regional level. Teams for the national stage are formed based on the results. To ensure more rigorous selection of the teams, the contest can take place in two rounds (p. 2.a and 2.b on Fig. 1).

National Stage. The participants are the teams selected in the previous stage and National Olympiad winners from the previous year. The total number of contestants varies between 130 and 160. Traditionally a stage lasts four days, two of which are reserved for competition. Competition sessions last between 4 and 5 hours, depending on the age group (middle school, high school) and the difficulty of the problems. After finishing the contest session, the participants can attend a self-assessment session, where they check their own solutions. Any disputes that may appear are presented to the Scientific Committee of the Olympiad. After the completion of the competition tests and resolution of disputes, the official solutions are presented. In parallel with the contest, training sessions, book launches, meetings with representatives of software companies and other activities are organized for team leaders (Fig. 1).

Winners of the national stage form the Extended National Team from which the National Team that participates in international competitions is selected (Fig. 1).

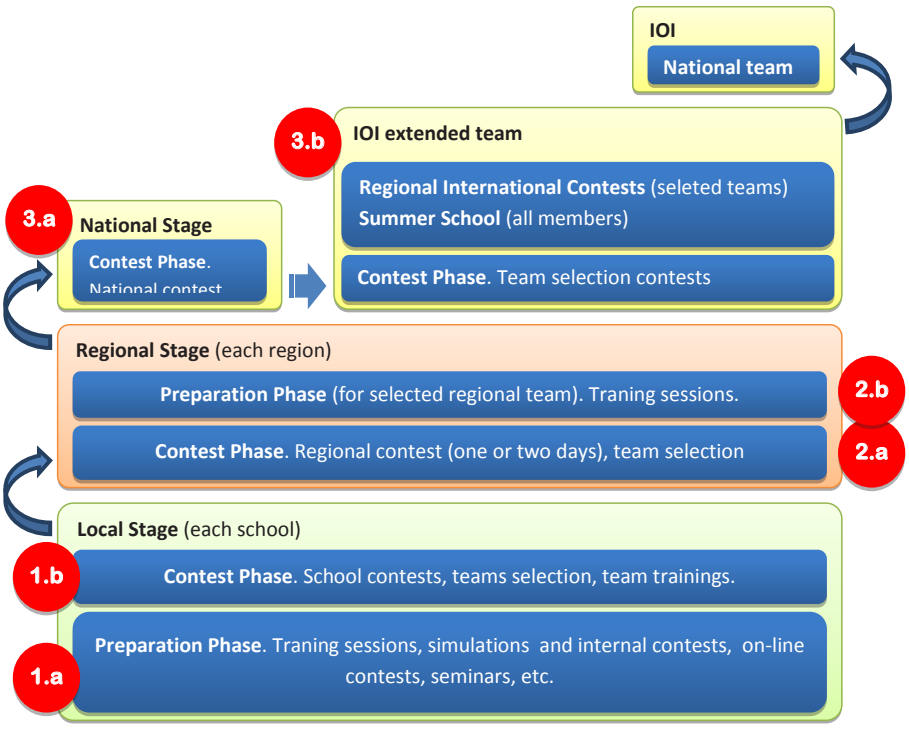


Fig.1. Stages of Republican Olympiad in Informatics.

2.2. *Target Group*

According to the curriculum, the study of software modules begins only in the 9th grade. However, taking into account the students' interest in programming and the capacity of self-learning of the gifted and highly gifted pupils, also younger children are accepted in the competition. The youngest participant in the history of the National Olympiad was a 6th grade pupil (13 years old)[1].

Informatics teachers of junior grades try to identify the pupils with skills in computer science through a variety of methods and techniques. In many cases the selection is performed by the teacher of Mathematics, in the 6th–7th grades. A study conducted by the authors in 2011[2] established a high degree of correlation between the performance in Mathematics and in Computer Science among the participants of the National Olympiad. Also, pupils who have good results in Physics (the study of this discipline begins in the 6th grade) show increased interest towards programming elements. Statistics compiled over the last 12 years (Fig. 2) indicate relative stabilization in the numbers of participants in the national stage in both categories: Senior (110 on average) and Juniors (40 on average).

2.3. *Competition Topics*

Competition topics are selected according to the National Curriculum in Computer Science. The problems for Local and Regional stages are prepared separately for each grade. The problems sets for the national stage are prepared separately for the two age groups. Usually 3–4 problems are proposed in each stage, to be solved by developing programs in Pascal, C, or C++. Six problems are proposed on the national stage, in two days of competition.

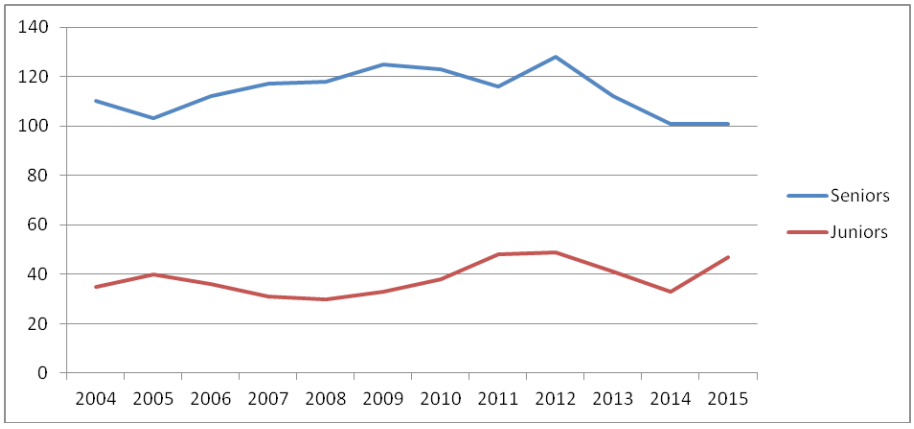


Fig. 2. Number of participants at National Stage, 2004–2015, by age categories.

[1] Municipal Olympiad in Informatics, 2013

[2] Didactica Pro, Nr. 4 (68), September, 2011, p.46–49

Task topics include data structures, combinatorics, programming techniques, elements of analysis and design of algorithms, complexity, algorithms on graphs, elementary algorithms of computational geometry, mathematical modeling, heuristics, etc. The degree of difficulty and statements match the contestants' age.

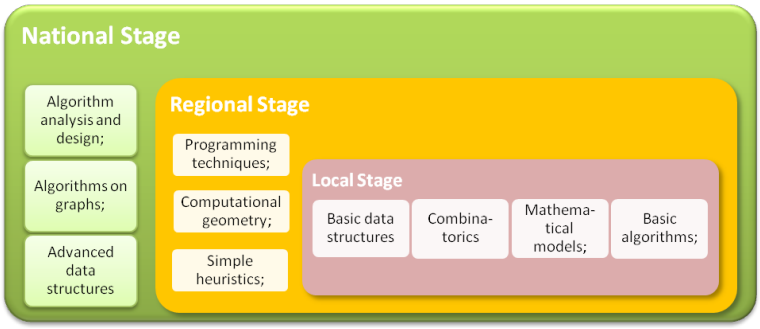


Fig. 3. The extension model of tasks domains, based on competition stages.

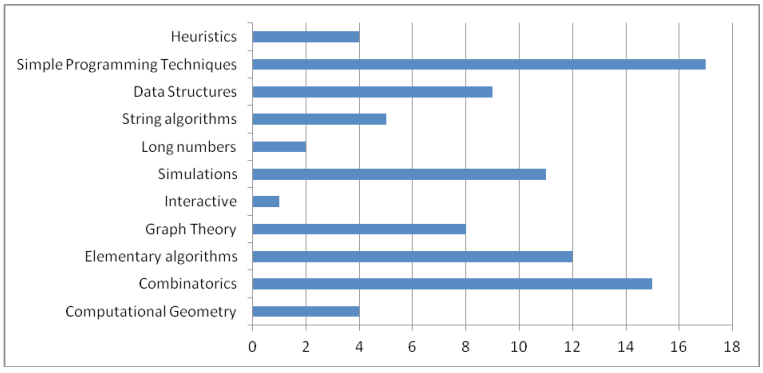


Fig. 4. Distribution of tasks by domains (last 15 years, Juniors).

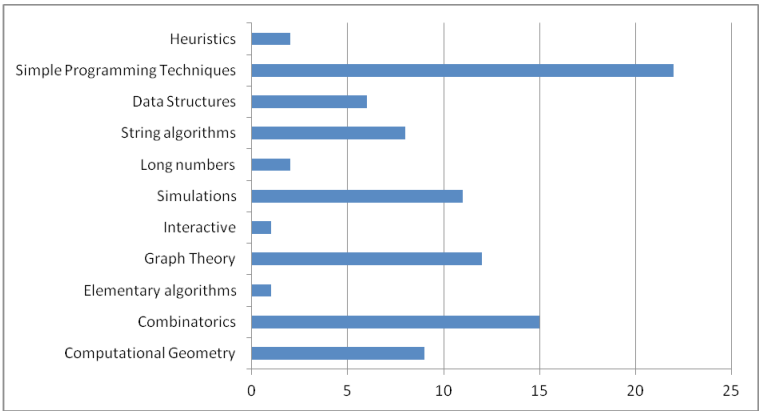


Fig. 5. Distribution of tasks by domains (last 15 years, Seniors).

Content of tasks is developed by complexity categories, to separate competitors according to the efficiency of their solutions. The number of test cases per task ranges from 5 (local and regional stages) to 10–20 (national stage). The tests check the accuracy and efficiency of the solutions by testing special cases and enforcing time limits. The proposed model of the problem selection allows reducing the share of participants who receive zero scores. Thus, after statistical analysis of the results of the past 10 years, a quantitative distribution of the problems depending on the complexity was established (Fig. 6, 7).

Evaluation: automatic, based on a local server. Participants are identified in the system based on user accounts, generated by the system. Human intervention in the evaluation process is excluded.

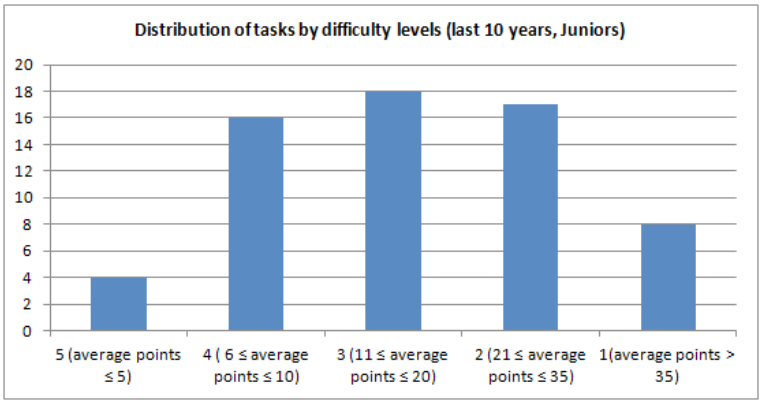


Fig. 6. Distribution of tasks by difficulty levels (last 10 years, Juniors).

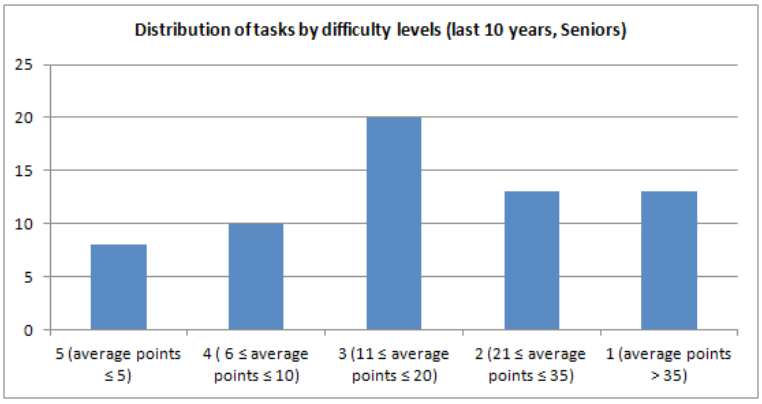


Fig. 7. Distribution of tasks by difficulty levels (last 10 years, Seniors).

3. Conclusions

- The organizational model of the National Olympiad provides the opportunity to participate in the initial stage to all pupils who are interested in computer science, regardless of gender, locality, type of school, nationality.
- Distributions of the competition stages allow teams time to prepare between competitions.
- The automatic evaluation is scalable in both directions. It was used to assess the results of international competitions and is used regionally for training and regional competitions[3];
- The organization of the post Olympiad activities provides qualitative preparation of National Team, as demonstrated by the results achieved by Moldova during the participation in IOI (22 bronze medals, 2 silver medals).

References:

Chistruga, Gh., Lupu, I., Gremalschi, A. (2015). Didactic digital supports for preparation in informatics. In: *Proceedings of The National Scientific Conference "85 Years of Higher Education in Moldova", Chisinau, 24–25 September, 2015*. 117–122.

Gremalschi, A. (2001, 2002, 2003, 2004, …, 2013). *Republican Olympiad in Informatics. Tasks and solutions. 2001 (2002, 2003, 2004, …, 2013)*. Chisinau, ASEM.

Prisacaru, A., Besliu, V., Bolun, I., *et. al.* (2014). *Republican Olympiad in Informatics. Tasks and solutions*. Chisinau.

Prisacaru, A., Besliu, V., Bolun, I. *et al.* (2015). *Republican Olympiad in Informatics. Tasks and solutions*. Chisinau.

Gremalschi, A., Corlat, S. (2011). The role of interdisciplinary in mathematics-informatics in preparing students performance. *Didactica Pro*, 4(68), 46–49.

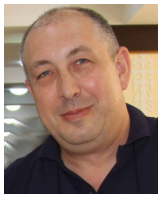
Informatics. Curriculum for Secondary Education, High School. (2010). Chisinau, Stiinta.

Informatics. Curriculum for Secondary Education, Gymnasium. (2010). Chisinau, Lyceum.

[3] Proceedings of the national scientific conference "85 years of higher education in Moldova", Chisinau, 24–25 September, 2015.

A. Gremalschi, PhD, professor, Technical University of Moldova; Director, Institute for Public Policy.
Author of textbooks in Computer Science.
Coordinator of educational projects.
Organizer of National Olympiads in Informatics (since 1997).
Leader of national team on Balkan Olympiad in Informatics, International Olympiad in Informatics (since 1996).

A. Prisacaru, consultant, Department of Pre-University Education, Ministry of Education of the Republic of Moldova.
Professor of Informatics, MSc in Informatics.
Organizer of the summer camps for preparing the Olympic teams.
Organizer of National Olympiads in Informatics (since 2013).
Leader of national team on Balkan Olympiad in Informatics (2014), International Olympiad in Informatics (2015).



S. Corlat, MSc in Exact Sciences. Lecturer at Tiraspol State University. Research areas: computational geometry, graph theory, computer graphics. Also involved in institutional, national and international projects for e-Learning. Trainer of Informatics team of ”Orizont” Lyceum. Leader of national team on Balkan Olympiad in Informatics (2003, 2004, 2006, 2009), International Olympiad in Informatics (2014, 2015).

Olympiads in Informatics, 2016, Vol. 10, 263–278

DOI: 10.15388/ioi.2016.19

IOI 2015 Report

Artem IGLIKOV, Mansur KUTYBAYEV, Bakhyt MATKARIMOV

Nazarbayev University
53 Kabanbay batyr ave., Astana 010000, Kazakhstan
e-mail: {artem.iglikov, mansur165, bakhyt.matkarimov}@gmail.com

Abstract. The International Olympiad in Informatics (IOI) is an annual international informatics competition for individual students at schools for secondary education from various invited countries, accompanied by social and cultural programmes. We present a report on the 27th International Olympiad in Informatics, July 26 – August 2, 2015, Almaty, Kazakhstan (IOI'15), organized by the Ministry of Education and Science of the Republic of Kazakhstan, Republican Scientific and Practical center "Daryn", al-Farabi Kazakh National University and supported by Mayor of Almaty and Mayor of Almaty region. IOI'15 established a new IOI record with 322 contestants from 83 countries, participated in IOI'15 and awarded by 161 medals (27 gold, 55 silver and 79 bronze), Jeehak Yoon from the Republic of Korea is absolute winner of IOI'15. At IOI'15 Java was first time introduced as IOI official programming language. In this report we pointed attention on issues happen as well as things that done well.

Keywords: IOI, programming contest, International event organization and management.

1. Introduction

IOI is one of the world's top level Olympiads for secondary schools students, among International Mathematical (since 1959) / Physics (since 1967) / Chemistry (since 1968) / Biology (since 1990) Olympiads. Initiated by UNESCO and starting from 1989 in Pravetz, Bulgaria, IOI constantly develops, especially in the level of scientific and technical solutions. The IOI's official site is `http://ioinformatics.org`, for general information on IOI we refer readers to the website and the following IOI documents: IOI Regulation[1], IOI syllabus[2] and the ITC/ITWG guidelines[3].

The President of the Republic of Kazakhstan, Dr. Nursultan Nazarbayev, made order in 1996 on governmental support and development of secondary schools for gifted students, and in 1998 the Government of the Republic of Kazakhstan established a new state enterprise, Republican Scientific and Practical center "Daryn" with primary goal to discover, encourage and give recognition to gifted students by developing and supporting

[1] `http://ioinformatics.org/rules/index.shtml`
[2] `http://www.ioinformatics.org/a_d_m/isc/iscdocuments/ioi-syllabus.pdf`
[3] `http://wiki.ioinformatics.org/wiki/HostingAnIOI`

special educational programs and activities. Kazakhstan hosted 36th International Mendeleev Chemistry Olympiad in 2002, Almaty city. Kazakhstan subregion of the Northeastern European Regional Contest of the ACM International Collegiate Programming Contest was created in 2003. From 2004 in Almaty city was organized annual International Zhautykov Olympiad on Mathematics, Physics and Computer Science (IZhO, n.d.), hosted by the Zautykov Republican Specialized Physics-Mathematics Secondary Boarding School for Gifted Students. Kazakhstan hosted 7th Asian Physics Olympiad in 2006, Almaty, and 51st International Mathematical Olympiad in 2010, Almaty-Kokshetau-Astana. Based on these achievements, Kazakhstan applied to IOI Executive Director as IOI potential host in 2010. At IOI'10 in Waterloo, Canada, Republic of Kazakhstan was selected by IOI International Committee as IOI'15 future host. Before IOI'15 Kazakhstan hosted 46th International Mendeleev Chemistry Olympiad in 2012, Astana city, `http://mendeleev.kz/` and 45th International Physics Olympiad in 2014, Astana city `http://ipho2014.kz/`.

2. IOI Host Committees

Steering Committee: Aslan Sarinzhipov, Minister of Education and Science, Chairman; Yessengazy Imangaliyev, Vice-Minister of Education and Science, Vice-Chairman; Galymkair Mutanov, Rector of the Al-Farabi Kazakh National University; Sholpan Kirabayeva, Director of the Republican Scientific and Practical center" Daryn", Secretary; Akhmetzhan Yessimov, Mayor of Almaty city; Yerbolat Dossayev, Minister of National Economy; Tamara Duissenova, Minister of Health Care; Kalmukhanbet Kassymov, Minister of Internal Affairs; Asset Issekeshev, Minister of Investment and Development; Bakhyt Sultanov, Minister of Finance; Erlan Idrissov, Minister of Foreign Affairs.

Host Scientific and Technical Committee: Bakhyt Matkarimov, Adilet Zhaksybai, Sergazy Kalmyrzayev, Ulugbek Adilbekov, Myrzakerei Miras – Nazarbayev University; Darkhan Akhmed-Zaki, Zhanl Mamykova, Natalya Surina, Erbolat Kalaman, Shyngyz Rabat, Pavel Chekanov, Askar Akshabayev – Al-Farabi Kazakh National University; Artem Iglikov, Azizkhan Almakhan, Madyar Aitbayev, Nurlan Zhussupov, Askar Aitzhan, Yesskendir Sultanov, Bektur Suleimenov – Kazakh-British Technical University; Mansur Kutybaev – International Information Technologies University; Fuad Hajiyev – ADA University, Azerbaijan; Georgiy Korneev, Nikolay Vedernikov, Gennady Korotkevich – ITMO University, Russia; Egor Kulikov, Elena Andreeva – Moscow State University, Russian Federation; Michael Mirzayanov – Saratov State University, Russia; Alexander Klenin – Far-Eastern State University, Russia; Ali-Amir Aldan – Massachusetts Institute of Technology, USA.

50+ staff specialists from involved organizations works full time during IOI'15 week. 150+ volunteers selected from active members of the host University volunteers club (team guides, organization staff) and from former participants of IOI and other Olympiads (HSTC volunteers).

IOI'15 was fully supported by the Government of the Republic of Kazakhstan. About 2000K was reserved/allocated for IOI'15 from the Ministry of Education and Science of the Republic of Kazakhstan, $1556K was spent for the main event, including purchase

of facilities $652K and $97K for IT Infrastructure works/HSTC/Technical staff. In total, guest fee was $93K.

3. Preparatory and IOI Present Host Actions

IOI'15 website created in 2011 and general information with photos about Kazakhstan was first time distributed at IOI'11, Pattaya, Thailand. Initially IOI'15 was planned in the capital Astana, in 2014 Almaty city and al-Farabi Kazakh National University were announced as the host of IOI'15. Programming contest for 300+ onsite contestants and 10000+ submits for evaluation with full feedback is a hard task, and IOI'15 was a first case in Kazakhstan. Previous to IOI Kazakhstan experience in programming contests organization was reported in (Iglikov *et al*., 2013). The following preparatory actions was crucial: 1) using IOI CMS (Mares and Blackham, 2012; Maggiolo *et al*., 2014) at national and international programming contests in Kazakhstan, Artem Iglikov recognized as IOI CMS developer; 2) organization of Asian-Pacific Informatics Olympiad APIO'14, hosted by Kazakh-British Technical University, Almaty (APIO'14, 2014); 3) hosting all-Russia team Olympiad in programming for secondary schools in Almaty at K.Satpayev Kazakh National Technical University[4]. Programming language Java accepted as official programming language in various programming contests, including ACM ICPC. Martin Mares, ISC/ITWG, evaluated Java solutions for IOI'13 tasks, created by Egor Kulikov, Pavel Mavrin and others, and reported to ISC at IOI'14, Taipei, Taiwan. ISC recommended Java as official IOI'15 programming language. In 2014 Egor Kulikov accepted IOI'15 HSTC member invitation. In 2014–2015 IOI CMS development team added full support for Java.

Communications with IOI countries representatives was by e-mail and private Google group "IOI 2015 Team Leaders". List of country contact emails extracted from the IOI Registration system, managed by Eljakim Schrijvers `https://ioiregistration.org/`. All foreign participants, including IOI committees and Host scientific and technical committee (HSTC), were registered in IOI Registration system. IOI'15 invitation letters was generated through IOI registration system, as well as various reports, e.g. list of participants with relevant information, including email, meals preferences, travel data, etc. For visa support procedures Host created forms and instructions, published on website and distributed by email two month before IOI, visa support procedures takes up to 30 days. Free of charge landing visa at Almaty International airport was organized for all teams who completed required procedures.

Call for tasks was made in December 2014 with submission deadline at January 31, 2015. Contestants machines specifications and operation system image was published in June 2015. Significant changes in IOI'15 competition rules from previous IOI was Java as official programming language, allowing multi-threaded programs at the contest time, and limits on competition print job size. 3 practice session tasks published 2 weeks before IOI'15. IOI'15 and 9th IOI conference programmes was published at the host website in July.

[4] `http://neerc.ifmo.ru/school/russia-team/index.html`

4. IOI Committees Meeting

IOI IC/ISC committees meeting organized in Almaty at February 26 – March 2, 2015. All IC/ISC members participated this event, for newly created ITC committee Host decided to organize meeting few days before IOI. IC inspected IOI'15 venue, making many objective suggestions with short checklist. IC minutes was published at IOI official web site. 43 unique tasks submitted for IOI'15, including 6 submissions from HSTC. HSTC rejected 11 submissions, due to various reasons, and 25 tasks shortlisted for selection by ISC. Finally, ISC selects 9 tasks for IOI'15, including 3 backup tasks: Scales, Eryk Kopczyński, Poland; Teams, Adam Karczmarz, Poland; Boxes with souvenirs, Monika Steinova, Slovakia; Towns, Bang Ye Wu, Taiwan (IOI'14 backup task); Sorting, Weidong Hu, China; Horses, Mansur Kutybayev, Kazakhstan; Liar, Ulugbek Adilbekov and Sergazy Kalmurzayev, Kazakhstan (IOI'15 backup task, opened at IZhO'16 (IZhO, n.d.)). Cultural programme of IOI committees meeting includes opera presentation Tosca and visit of Shymbulak mountain area.

5. IOI Organization

IOI'15 venue formed by nearly located al-Farabi Kazakh National University campus with newly built Student hotel (contestants), Atakent Park Hotel (team leaders) and Ritz-Carlton Hotel (IOI committees, tasks authors, invited guests). Walking time between team leaders and contestant's hotels is about 25 minutes. Student hotel was opened free of charge without meals for early arrived teams contestants from July 23. For IOI'15 needs host University allocated al-Farabi library building, Director Kalima Tuenbayeva. Team's registration was at al-Farabi library. Opening/Closing ceremonies performed in the U.Dzholdasbekov Palace of Students. Most of activities organized in Atakent Park Hotel for team leaders, and in host University campus for contestants. IOI Doctor (medical) room allocated in al-Farabi library and Student hotel, as well as host University medical center was ready to serve for IOI. At IOI'15 3 medical treatment cases registered. 10 air conditioning and climate control facilities was installed in competition hall (6) and Student hotel (4). We reserved at least two rooms/halls for any IOI activity, except single competition hall. IOI committees rooms, IOI office, translation session hall, GA meeting hall was allocated both in Atakent Park Hotel and al-Farabi library. Rooms in Student hotel was not identical, e.g. for 2 or 4 persons, mirrors was only on 1st floor, etc. Rooms for girls was allocated on the 1st floor, mixing contestants from different countries, also we allocated contestants rooms starting from 1st floor based on IOI total medals rank of countries. In hotels for adult participants most of rooms was single, and we sorted participants by age to allocate rooms. Student cafeteria at host University campus was allocated for contestants and team guides. For leaders breakfast was on a residence, lunch and dinner was at Atakent Park Hotel. Lunches at contest days was organized for all participants at host University campus. Farewell party was at Atakent Park Hotel. Transportation was organized within IOI venue and for all excursions, for guest excursions two big busses was allocated. There was a shortage of minibuses, and

planned hourly shuttle at IOI venue was not implemented. Life stream translation was not at IOI'15, daily video distributed with YouTube by Azhar Rakhletova, easily found by search "IOI 2015", also video/photo materials published at IOI'15 website. IOI'15 schedule was traditional 8 days for IOI.

6. Cultural Programme

IOI'15 cultural programme includes two whole day excursions for all participants, two whole day and 3 Almaty city excursions for guests, various sport/entertainment events and invited lectures. Whole day guest excursions were to the Big Alma-Arasan gorge with Nursery "Sunkar", and Kazakh aul "Huns". Almaty city excursions were to the Central State Museum of Kazakhstan, «Kok Tobe» mountain park with panoramic view of Almaty city, Park of 28 Guardians of Panfilov's division, Saint Ascension Cathedral, Ykhylas Museum of Folk Musical Instruments. Day 4 excursion was to the high-mountain sports complex "Medeu", including visit of Shymbulak glacier at 3000+ m above the sea level; and Kazakh State Circus 45 years anniversary presentation. Unfortunately, day 6 Turgen gorge mountains area excursion was canceled due to official emergency notification on flood flow, not happen actually, and excursion was limited to acquaintance with National customs and traditions.

IOI'15 Host organizing committee thanks to Chris Peterson from Massachusetts Institute of Technology for lecture "How to apply to MIT (and other USA colleges)" at July 30 with various presents, e.g. books signed by MIT professors, including "Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Kazakhstan governmental program Bolashak supports education abroad and our students are very interested in presentations of this kind. We did not consider this presentation as a promotion of MIT, in fact we also invited lecturers from few other Universities and ICPC community, and only Chris Peterson comes to IOI'15. It would be better to make open announcement before IOI for similar activities.

7. 9th IOI Conference

IOI conference traditionally organized in two days during contest time in parallel with Question/ Answers sessions at July 28–30, 2015. IOI conference program published at host website and includes 15 presentations with workshop on IOI CMS contest management system, moderated by Stefano Maggiolo, ITS member. Special session of IOI conference was organized for Kazakhstan teachers.

8. Quarantine

During Quarantine time contestants allowed to enter Student hotel garden and IOI volunteers organized Dance club / Entertainment programme.

9. Translation Sessions

Translation sessions organized at Atakent Park Hotel. For translation we asked team leaders to bring their own laptops, and less than 10 laptops were requested from organizers. At IOI'15 was used IOI Translation system created in Taiwan for IOI'14 with minor modifications. At the first translation session a large printing queue was caused by inefficiently organized printing procedures with operator intervention, this was improved for the second session. In the evening before contest day 2 a power outage happen in the hotel, causing one hour delay in GA meeting start time. Fortunately, both backup and central power recovered by power service engineers on duty, and we did not switch to the backup plan – translation hall at host University without coffee break. Translation nights continued up to 5 a.m. In total, 454 task statements with notices was prepared during translation sessions and published at IOI'15 host and IOI official websites.

10. Question/Answer Sessions

Question/Answer sessions organized at Atakent Park Hotel in parallel with IOI Conference. At IOI'15 clarification forms from contestants was scanned and distributed within "IOI 2015 Team Leaders" Google group, being visible to all team leaders, and text translation from any team leader who knows question language was accepted. Most of contestant's questions in fact were handled within IOI CMS, e.g. questions written in English or any other language, known by ISC members. Only six questions from contestants were handled at Question/Answer sessions.

11. Contest Tasks

For IOI tasks preparation we used automated platform for creating programming contest problems Polygon[5], developed at Saratov State University, Russia, by Michael Mirzayanov team from 2008. Polygon automates contest tasks preparation, organizing effective team work on tasks and preventing typical errors, and supporting user access management, version control, issue-tracking, integration with popular test systems, contest tasks archive compilation, long-time online backups, tasks search and classification capabilities. For security reasons separate instance of Polygon was installed by Michael Mirzayanov on our servers with two levels of authorization.

Boxes with souvenirs. Number of subtasks: 6. First two subtasks are easy and require basic knowledge of programming with simple logic. Third subtask could be solved by "brute force" algorithm. Starting from fourth subtask, contestants have to make simple observations about the structure of the problem. Fourth subtask has a lot of different solutions; the most popular one could be dynamic programming with 2 states. Subtask 5 and 6 could be solved only with most important observation of the task. The difference is in implementation details of the algorithm which could give better performance. Tests

[5] `https://polygon.codeforces.com`

development for this problem was relatively easy. Problem has special tests for checking correctness of solutions on important observations.

Scales. Number of subtasks: 1. This problems has no subtasks. Score for this problem depends on effectiveness of the algorithm. We used all the available tests for this problem, which could be easily generated and grouped by subtests. 56 different scores was on this task at IOI'15 competition.

Teams. Number of subtasks: 4. First subtask could be easily solved by "brute force" algorithm. The second subtask needs knowledge of well-known sorting algorithms. Third and fourth subtasks need advanced geometry and data structure algorithms. A lot of tests were prepared for this problem, including all corner cases. Most of tests were prepared to test programs on time limits, which was crucial for this problem.

Horses. Number of subtasks: 5. First subtask very easy one. Starting from second subtask contestants should make observation about algorithm structure. Second one is just easy implementation of this algorithm. Third subtask needs another crucial observation, which helps to solve this subtask. Difference between third and second subtask is only in constraints. The last subtasks could be solved only with advanced data structures and programming techniques. Tests for this problem were quite tricky, because it consist of a huge amount of corner cases. And there is a lot of space to make some mistake in program. There are a lot of different tests that covers huge amount of occasions.

Sorting. Number of subtasks: 6. This problem has a lot of different subtasks, because it could be solved in many different ways. Of course not all of them are efficient enough. Most of the subtasks have some unique constraints which makes the problem easier than the problem itself. First three subtasks don't require main observation to solve the whole problem. Difference between these subtasks is implementation difficulty. For the next three one needs observation about the problem structure to be successfully solved. Each time contestant needs to make more effort using additional technique or implement additional data structure. Problem needed only some restricted amount of tests, which was prepared using small test generators. They cover almost all cases, including corner cases, small cases and large testcases.

Towns. Number of subtasks: 6. This problem may be divided into more than 6 subtasks, it was decided to fix the number of subtasks on 6. Each subtask could be treated as a different problem, because each of them has some specific constraint on some parameters. Depending on the contestant observations there might be slightly different algorithm to solve with different score. The problem was divided by most interesting cases to solve. Tests preparation for this problem was hardest one in the contest, consisted from the algorithm which is hard to test with fixed amount of tests, because of the randomized solutions. Jury tried to prepare a lot of different tests that increase probability of failure of wrong solutions. Most of them were prepared depending on wrong solutions written by jury itself and by beta testers.

When all task statements was completed we did not write tests verification programs, based on final constraints and separated from tests verificator, already created by tasks developers, this caused formally invalid test cases. Initially prepared graders was not enough secured to prevent attacks. IOI tasks analysis with test data published at IOI'15 host and IOI official websites.

12. IOI System

IOI system needs low feedback time and high availability/fault tolerance and high performance/ throughput networking evaluation system, built on relatively low cost facilities. All parts of IOI'15 system was reserved, 10% of laptops, 100+% of servers, 100% network core switches and trunk lines, and 10% of network switches and network cables to end-point devices. IOI'15 facilities were purchased in 2015 by open bid according to the laws of the Republic of Kazakhstan. Many kinds of facilities were provided by supporting organizations. IOI'15 facilities includes 1) uninterruptible power supplies: built-in with laptops, 20 new 2kVA UPS for every network switch, 1 new Fujitsu APC Online 20 kVA in server room, external 800kVA mobile power generator for competition building; 2) various facilities for installation of power network in competition hall; 3) 415 HP ProBook 450 G2 laptops with external keyboard, mouse, mousepad as contestants machines/grading system workers; 4) 30+ external monitors provided for contestants by request; 5) servers: 4 new HP/ProLiant DL380Gen9, 6 new (6+) Fujitsu blade servers; 6) 1 new Fujitsu NetApp backup storage system; 7) 14 Alcatel-Lucent 48 port layer-2 network switches, supporting VLANs and 1 GigE Ethernet; 8) 4 Alcatel-Lucent layer-3 10 GigE switches at network concentration points; 9) Fujitsu rack; 10) network cables, testers, cable channels, telecommunication boxes, etc. for network installation; 11) 5 new fast color printers; 12) 5 new (+5) fast black-white printers; 13) 3 new (+7) big Samsung monitors as information desks, network monitor, contest results online presentation, etc.; 14) 5+ projectors for presentations; 15) 10 new climate control facilities installed, 6 in competition hall, 4 in student hotel; 16) 1 audio system for announcements in competition hall; 17) various video/photo translation/publication facilities for public media coverage; 18) 6 mobile communications jammers for quarantine; 19) 1 voting system for GA meetings.

ITC/HSTC report follows: Laptops was chosen to have more modern technical characteristics than at previous IOI, and to maximize work time on battery, e.g. 15.6" display and Core i5 CPU, and to fit in the budget. 336 laptops were allocated for contest hall, 60 for grading system, 19 for translation session and other needs. Host received similar (but not exact) laptops about 3 months before IOI, to prepare the system and test battery work time. The laptops arrived about 1 month before IOI. The network was not yet set up, so we couldn't fully test the laptops at that time, we just checked that every laptop can successfully be turned on and boots the pre-installed system. The primary servers used for the contest were HP DL380 Gen9. Everything was installed on one server. Another server was a full copy of the main and had database was being replicated to it, so in case of problems we could switch to it in minutes (and nothing would be lost). Third server was used for live backups of contestant machines; we couldn't do this on the first server due to high load on the file system. Fourth server was used for translation sessions. We also had blade servers and tower machines ready to use as a backup system. We used 10 Gbps network interfaces on the server with 10 Gbps switches in the server room. Switches in the contest hall all had 1 Gbps links to contestant's laptops and 2 10 Gbps uplinks to the server room. All switches outside of the contest hall were backed up, so if any one

fails network would be still up. We had several spare switches for the contest hall and we could configure them in few minutes to replace any failed switch. All switches were managed remotely by university network administrator. Cable network built safely using dual camera cable raceways for both power and digital networks. Competition hall has 6 video cameras to monitor from ISC/ITC rooms.

Contestant sample images preparation: once we had the IOI'15 laptops, we took 5 of them for setting up the contestant software. Initial software setup was performed using Ubuntu package-management software (APT). Basic software installation was scripted (INSERT REPO LINK). Some manual configuration had to be done for setting up help in some IDEs (INSERT MANUAL LINK). Also, several days before the IOI we updated the image. Two versions of Eclipse 3.8 and 4.4 were provided. The 3.x branch considered to have better performance, while the 4.x branch has more features. After the installation of software following issues was detected 1) KWrite could not open any file for reading (fixed by installing missing KDE packages); 2) Default destination of Java API for Eclipse, NetBeans and IntelliJ IDEA points to Oracle internet site (JDK links was fixed in configuration); 3) Default C++ help files are 15 years old (replaced by actual StdLibC++ help files); 4) StdLibC++ help files are not complete (additional help files was download from `http://en.cppreference.com/`); 5) Free Pascal IDE has no help files (help files was downloaded and installed); 6) Free Pascal IDE fails on debugging complex programs (not fixed); 6) Code::Blocks hangs when multiple instances started in short period of time (not fixed). For each Editor/IDE following requirements was checked: 1) It is possible to write, build, run and debug a programs solving "Search" practice session problem on all supported languages (a lot of minor misconfigurations found and fixed); 2) Help files are available for all supported languages (required help files was downloaded and configured); 3) It is possible to save/load files (failed by KWrite); 4) Printing are supported (printing is not implemented in Sublime Text editor and have bugs in Code::Blocks, to alleviate later issue contestants was instructed to use "Print to PDF" or use different IDE/Text editor for printing). Keylogging software (`logkeys`) was installed on contestants' workstations. It allows monitoring contestant activity before and during the contest. Some of the contestants performed activity on theirs workstations before start of the contest (most of them – unintentionally, like sleeping on keyboard) and was warned about that. The side-effect of the key logging was ability to determine times when a contestant computer was hang. It became possible due to timestamps that accompanies pressed keys and clear message of key logger restart. Therefore, it is possible to determine the time of last key press before the hang-up and first key press after it. So we have a good upper bound on hand-up time.

Imaging contestant machines: for imaging we follow Bernard Blackham report for IOI 2013 (Blackham, 2013). We have set up a dhcp boot server (dnsmasq), which forwarded laptops to boot TinyCoreLinux from tftp server (atftpd). The imager script was being downloaded after TCL is booted, so we didn't have to update TCL image each time we change the imager script. The imager script wipes the partition table, re-partitions the drive, and starts udp-receiver to receive the main partition image. We were able to re-image up to 90 machines with the speed about 900 Mbps, which for 10 GB image took about 1–2 minutes. We tried to re-image all the machines at once, but this didn't

work well (probably required some nicer setup). We decided not to lose time on this and reimaged the hall by 3–4 rows (each row was up to 25 machines). Even in this case it took about 10–15 minutes to reimage everything. We also didn't bother on TCL image size and contestant image size and had no issues with it. During imaging the addressing was the following: all laptops in the VLAN 18 with addresses 10.18.row.place / 16. Server had one interface in VLAN 18 with address 10.18.0.1 / 16. During the contest the addressing was the following: each laptop in different VLAN (1600 + (row – 1) * 100 + place – 1) and address 10.row.place.2 / 24. Server had over 300 virtual interfaces with addresses 10.row.place.1 / 24.

All managing of contestant's machines and workers was done remotely (after several tries). We could: turn on laptops remotely with wake-on-lan (which saved us from a lot of walking), selectively re-image laptops, start memtest or badblocks on all the laptops, do anything we want on the laptops during the imaging script is running (thanks to Bernard for embedding a backdoor), do anything we want on the laptops after imaging with SSH (laptop contained servers public SSH key) or NetAdmin. To control and monitor contestant workstations we used Java-based NetAdmin tool by Georgiy Korneev. It monitors current state of each computer in the network several times per second and allows issuing control command to all workstations or selected part of them. There is a command queue for each workstation, so the command executes only when workstation is accessible and all previous commands has been finished successfully. NetAdmin allows issuing of server-side (local) and workstation-side commands (remote). Server-side command executed on server by NetAdmin itself. For example, *Copy contestant data* command looks as follows: `scp -r {day} {ip}:/home/ contestant`, where `{day}` and `{ip}` are placeholders for current contest day (`day0`, `day1`, or `day2`) and IP-address of the computer to copy data to. By default, command executes on all computers simultaneously, but this is not the case for *Copy Contestant Data* command – if you try to copy 100M simultaneously on 360 computers this will result in terrible network performance. NetAdmin is able to throttle execution of such kind of commands to specified number of simultaneous executions. Workstations-side (remote) commands are executed by special remote execution service, installed on controlled workstations. Both remote execution service and NetAdmin server authorizes each other using TLS certificates. Another option is to use `ssh` command on server side. An example of client-side command is *Reboot*: `reboot -fn`. Notice, that there is no need to specify exact computer to reboot, as long as the command is run on that specific computer. NetAdmin support offline commands that executes event when workstations are not accessible. Offline commands are useful for wake-on-lan and similar scenarios. NetAdmin also has been used to monitor and control grading system invocation workstations.

During the competition, we performed backups of contestant home directory on each workstation. On days 1 and 2 backups scheduled to run each three minutes. During the practice session backups was performed each minute to test network throughput and to measure influence on contestants. Average backup of all contestant workstations took 40–50 seconds, with peaks up to 60 seconds. Backups was performed using `rsync` incremental backups (`--link-dest`), this allows to make snapshots of home directory, while preserving hard drive space and network throughput. Using this scheme the initial

backup of the single workstation has size of 300+MB. To reduce this size we implemented cross-workstation incremental backups. In this scheme we took the full backup of only one «original» workstations (still 300+MB), while backups of the other workstations was made as incremental relative to «original». This allows us to reduce the size of the initial backup to 5MB/workstation. Most of this 5MB are files regenerated by Gnome/Unity after detection of new hardware ids after reimaging. New backup scheme allows us to dramatically reduce time and space required for the initial backup. Unfortunately, the backup time of all workstations was 5–7 mins, even when there are no actual changes was made. Profiling shown than workstations CPU and network usage was almost negligible, while performance was capped by server hard disc. Further investigation shown, that most of the time is spend by `rsync` to create hard link to each unmodified file during backup. To alleviate this issue we switched from Ext4 to BTRFS that gave us 5 times boost. To speed up backups even more, we decided not to backup large amount of files that should (almost) never change or are regenerated on startup. This two tweaks combined reduced backup time of all workstations to 40–50 sec, while decreasing workstation and network load. Test backup restore was performed several times, and no issue was found. Unfortunately, backup restore during day 1 analysis resulted in hang of heavily used workstations. After reboot all workstations become ok. Further investigations shown that hangs was due simultaneous update of Gnome configuration files and cache by both user and `rsync`. There are two scenarios to work around this issue: 1) Perform backup restore only when workstation is not in use. This is the main scenario in the case of unrecoverable contestant workstations failure. In this case, restore took about 30 sec, which is small compared to time to try recovering workstation, and moving contestant to another one; 2) Restore backup to different directory. This scenario was used on day 2 result discussions, when backups were restored to `dayX/backup` directories.

We used master version of CMS (which has evolved greatly during the year before IOI'15) with some modifications. Main improvement of this year was testing a submission in parallel on all the workers on different test cases. So if there are some idle workers, contestant would receive the result much faster than if it would be tested on one worker as it was before. This lead to an issue with ES performance, but it was solved by Stefano and Bernard before day 2. Proper patches for CMS will be posted in the official repository. Other CMS modifications included: 1) specifying subtasks inclusion in dataset options (when subtask 2 can include all the test cases from subtask 1 according to the task statement, that could be specified in the dataset options, and results of evaluation on test cases of subtask 1 will be included into results of evaluation on test cases of subtask 2; 2) displaying results was done in an aggregated form (for each appeared evaluation outcome we showed number of test cases and maximal used memory and time); 3) Oracle JDK support; 4) many improvements for Polygon importer. There were several problems with Java support: multi-threading, memory limit, time limit. Oracle JDK is multithreaded by nature. During testing we observed about 17 threads required to just start the program for Oracle JDK 8 on Ubuntu 15.10 x64. We had several choices: 1) limit number of threads with the sandbox (this would be problematic, because JVM can start GC any moment on a separate thread); 2) limit the number of threads the contestant program is allowed to start (this should work with GC which is not being started

by contestant program, but there was a theoretical possibility that standard libraries used by the contestant could start some threads); 3) allow multi-threading. We have chosen to allow multi-threading and counted the execution time as the sum of execution times of all threads of the program. This lead to an interesting issue: if the program used amount of memory close to the JVM heap size, then JVM would start GC on a separate thread, and its working time will be added to the total execution time. To overcome this issue we raised memory limits and JVM heap size (both -Xmx and -Xms). Another problem is that standard data input classes (Scanner, BufferedReader) in Java are quite slow. With help of Egor Kulikov we re-wrote all graders to use the raw FileInputStream, buffering and parsing the input in the grader itself. After this, surprisingly, Java input become much faster than input on C++. So we had to rewrite C++ input procedures as well (reading number of bytes from the input and parsing them manually). After that the same technique was applied to FreePascal graders.

13. Competition

Day 1 and 2 contests were started at 9:00 as planned by schedule. IOI'15 CMS evaluation statistics: 5762 solutions at competition day 1 evaluated in total on 325118 tests, 8845 solutions at competition day 2 evaluated in total on 470563 tests. In total 371 print jobs from 114 contestants was executed, 9 pages was a maximum per single job. Submits statistics by tasks vs. programming languages with number of different contestants and submits presented in Table 1. C++ used by most of contestants, and Java usage approaches C/Pascal usage. Java perspectives may be only estimated locking usage dynamics on next IOI's.

ISC/ITC/HSTC reports on issues happen at contest time: Hardware setup, including servers, workstations and networking was very solid. There was no major issues was found, while several minor issues was fixed: 1) glitches on workstation screen when the lid is moving (2 contestant, workstations were replaced); 2) heavy glitches for few

Table 1

Programming languages usage at IOI'15

	C++		Java		Pascal		C	
day1	**312**	**5278**	**5**	**90**	**5**	**81**	**4**	**58**
boxes	308	2380	5	42	5	27	3	33
scales	281	1503	4	19	4	23	4	15
teams	263	1395	5	29	4	31	3	10
day2	**312**	**8107**	**11**	**148**	**5**	**179**	**4**	**116**
horses	305	3339	11	61	5	70	3	22
sorting	304	3006	6	66	4	72	4	93
towns	177	1762	4	21	2	37	1	1

seconds when video mode is changed (probably bug in video card drivers, does not affect contestants); 3) two contestants was provided with additional monitors (by request); 4) no mouse pads on practice session (mouse pads were provided by request on days 1 and 2; 5) workstation hang-ups (real origin is undetermined, 8 contestants was affected during day 1, and 4 during day 2).

Day 1:

At the beginning one of 12 CMS servers was not working properly due to misconfiguration. As a result electronic statements and grading was unavailable for roughly 1/12 of all contestants. They could still read the hard copies. This was reported around 9:15 and fixed within ~5 minutes.

Around 9:40, a cheating case was detected. In problem Scales a contestant managed to read the internal data of the grader. We fixed the grader, so that this type of hacking would be much more difficult. When doing that we faced a CMS bug, which resulted in grading not working for ~20 minutes. After that, the outstanding solutions were graded quickly.

At 10:30 we discovered a problem in test data for task Teams. A task statement condition was not satisfied in one test case, 2 students got affected. The first one was the one who reported the issue, so he added a workaround within few minutes. The second student submitted a solution that solved the incorrect test 13 minutes later after his incorrectly graded submission. Some other students already passed this test. A rejudge was on the way, but we realized that it would not finish before the end of the contest. Because of that, around 13:40 we have announced the details of the problem.

Around 12:30 the same CMS bug has shown up. Before it got fixed, around 12:33 we had a power problem with UPS on network switches, which caused the CMS server to be unavailable for everybody. It was fixed in about 10 minutes. During this time, the contestants could work on their computers. After few minutes the CMS bug was "fixed" (graders were restarted). Unfortunately, one contestant has to reboot his computer during power failure period. This reboot takes a lot of time, since DHCP server was inaccessible at this moment. It is recommended to pin dynamic IP addresses in future installations.

At 12:55 the allowed interval between submissions increased from 1 to 5 minutes (per task). This was dropped down to 1 for the last 8 minutes of the contest. During the second half of the contest the grading time was around 18 minutes. To the best of our knowledge, all grading results were reported within 25 minutes.

Several computers locked up during the contest and they had to be rebooted. We have analyzed the logs, but the reason remains mysterious (we are still investigating). In one case, it happened during the network outage, so the machine did not come up immediately. The student has lost about 5 minutes, so we decided to extend the contest by 5 minutes for him. However, additional analysis of log files revealed that some other machines could have been unusable for up to 13 minutes. They were rebooted by the volunteers, who did neither tell us nor recorded the details, and the contestants did not complain, so we cannot be sure how serious the problem was. The volunteers will receive better instructions for day 2.

A large number of clarification requests were received due to students not knowing how to compile their program in their preferred environment (mostly Code::Blocks).

They had included graderlib.c and grader.cpp into their project, which caused multiple symbols to be defined.

Public clarification actions: 1) In the beginning: Aman can give multiple souvenirs when he is in a section; 2) Graders read I/O from file; 3) Scales grader behavior for incorrect queries; 4) Late: sample grader prints not only the sequence, but also the number of queries; 5) Late: Teams may give bad results.

8 machines froze and got rebooted incidents of rebooting from 3 to 15 minutes.

Questions received: 1) lots of technical questions – we assisted the students; 2) questions regarding the rules – we gave the full answered; 3) task related questions – often answered ANSWERED IN TASK DESCRIPTION or INVALID QUESTIONS. Exceptions: questions regarding something that was clarified – then we tried to provide a very helpful answer. Question/Answer translations worked fine.

Analysis of day 1 issues: 1) some cases of freezing computers may have been caused by Code::Blocks grabbing the keyboard and mouse and then locking. However, there may have been other causes, too; 2) There were 4 students affected by the regrade on Teams. To gather more information on sporadic hang-ups of contestant's workstations, the remote syslog facility was set up. Syslog analysis allows attributing some of hang-ups to Code::Blocks and finding the way to alleviate them. To test performance of remote syslog workstations was simultaneously rebooted several times. No missing log messages or substantial network load was detected during reboots.

Day 2:

During the day 2 all announcements were dubbed on the screen as popups using `notify-send`.

There was a small issue in the sample grader for Towns. This was updated in CMS before the contest and pushed to the machines at 09:05. 09:11: E-1 report could not extract zip file, confirmed, using local versions for sorting at 09:15. Several issues rose about day 2 graders, first at 09:13. There were two versions of graders; both of them were the same. 09:32 clarifications notice that horse.out is 0/0. The .out files are results of running sample graders, announcements made at 09:46. 09:20 ranking issue: submissions with ids from day 2 overwrote submissions from day 1. Fixed at 09:40. 11:40 Found and fixed a typo in the announcement sorting.1.in –> sorting1.in. 13:07 another ranking issue: some scores became lower after refresh. Fixed in 2 minutes. This was likely caused by a faulty restart of one the ranking servers.

During the contest day 2, one more issue was found: the bug in Code::Blocks that caused UI to hand while debugging some programs. Two affected contestants were instructed to use `pkill` in such circumstances. Recorded issues at day 2: 1) L-5 hardware 12:36–12:41; 2) G-15 multiple Code::Blocks freezes (~15 min total lost) caused by the contestant repeatedly trying the same thing; 3) J-10: 12:30–12:35; 4) I-13: 13:40–13:44 Code::Blocks.

ITC investigated CMS issues and reported: After the ISC confirmed the problem on a test case for Teams, the ITC inserted new test dataset in CMS and started the background judging. The system handling the queue of submissions saw an increase in workload that exposed an application level deadlock causing frequent freezes. Therefore the background judging proceeded slowly, and eventually it was decided to switch to the

new dataset before its judging caught up. As the rate of submission increased, the deadlock was triggering again, requesting manual intervention each time. As judging became slower, contestants increased their submission rate as they could not wait for the (potentially positive) previous results. This forced us to increase the minimum time between submissions to 5 minutes. The cause of the deadlock was found and fixed by Bernard between day 1 and day 2. In parallel to this, one reason for the higher than expected workload was CMS change to distribute each test case to a single worker, which has the advantage of giving much lower latency when the system is relatively free. The unexpected drawback of this change, in a setting with high number of workers such as the IOI, was that the system, handling the queue, was overwhelmed by the communication with the workers. This problem was patched by Stefano between day 1 and day 2 (where each evaluation was split in about half a dozen packets rather than one per test case).

14. Award Ceremony

322 IOI'15 contestants awarded 161 medals, 27 gold (rounded up), 55 silver (rounded up) and 79 bronze. Jeehak Yoon from the Republic of Korea is single absolute winner of IOI'15 with perfect score 600 from 600, he was awarded IOI trophy. IOI Distinguished award was to Don Piele, USA, post mortem. Special presents were given to girls – IOI'15 contestants (Kazakhstan tradition).

15. Conclusion

In IOI'15 participated 83 Official Teams (77 teams with 4 contestants, 3 teams with 3 contestants, 2 teams with 2 contestants, 1 team with 1 contestant), 0 Observing Countries, 1 President, 1 ED, 10 IC, 8 ISC, 7 ITC, 24 HSTC; 322 Official Contestants, 161 Leaders/Deputy Leaders, 62 Guests + 7 Invited Guests + 2 children. Hosting IOI is an exceptional event in a lifetime experience. Thank you very much IOI community for giving us the opportunity to show a small part of our beautiful country! Welcome Kazakhstan again!

Acknowledgements

We greatly appreciate efforts and hard work of more than 200 specialists to organize IOI'15, including staff, volunteers, engineers from the Ministry of Education and Science of the Republic of Kazakhstan, Republican Scientific and Practical center "Daryn", al-Farabi Kazakh National University, administration of Almaty and Almaty region, ITMO, Saratov SU, Moscow SU, Far-Eastern State University, ADA University, MIT, Kazakh-British Technical University, International Information Technologies University, Kazakhstan Hewlett-Packard office, ALSI, Albeta Kazakhstan, Emergency service Almaty, Kazakh State Circus, Medeo & Shymbulak administration, Kazakhtelecom and many others. Thank you very much!

References

IZhO (n.d.). *International Zhautykov Olympiad*. `http://izho.kz/`

Iglikov, A., Gamezardashvili, Z., Matkarimov, B. (2013). International Olympiads in Informatics in Kazakhstan. *Olympiads in Informatics*, 7, 153–162.

Mares, M., Blackham, B. (2012). Introducing CMS: a contest management system. *Olympiads in Informatics*, 6, 86–99.

Maggiolo, S., Mascellani, G., Wehrstedt, L. (2014). CMS: a growing grading system. *Olympiads in Informatics*, 8, 123–131.

APIO'14 (2014). *Asian-Pacific Informatics Olympiad 2014*. `http://olympiads.kz/apio2014/`

Blackham, B. (2013). *Bernard Blackham report on IOI'13*. `http://www.ioinformatics.org/locations/ioi13/`

A.V. Iglikov – ISC member, Taiwan 2014, ITC member, Host Country 2015. Technical director of Kazakhstan subregion of the Northeastern European Regional Contest of the ACM International Collegiate Programming Contest. Jury chairman of the Kazakhstan National Olympiad in Informatics, International Zhautykov Olympiad (on Computer Science).

M.U. Kutybayev – IOI Team leader of Kazakhstan 2014. ISC member, Host Country 2015. Author of task "Horses" (IOI 2015). Jury chairman of the Kazakhstan National Olympiad in Informatics, International Zhautykov Olympiad (on Computer Science).

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. Jury chairman of the Kazakhstan National Olympiad in Informatics, International Zhautykov Olympiad (on Computer Science).

Olympiads in Informatics, 2016, Vol. 10, 279–283

DOI: 10.15388/ioi.2016.20

The Informatics Olympiad in Mongolia: Training Resources for non-English Speaking Students

Altangerel KHUDER[1], Danzan TSEDEVSUREN[2]
[1]*School of Information, Communication Technology, Mongolian University of Science and Technology*
[2]*School of Mathematics and Natural Sciences, Mongolian National University of Education*
e-mail: khuder@must.edu.mn, tsedevsuren@msue.edu.mn

Abstract. In this country report we present activities related to Mongolian team training for national and International Olympiads in Informatics. First we will outline the current conditions and problems Mongolian team is facing today. Then we describe possible solutions to those problems. At the end we will compare our achievements to our country report published in IOI conference in 2007. We hope this paper will be interesting for team leaders and teachers from non-English speaking countries.

Keywords: information and communication technology, informatics education, informatics competitions, programming contests, Informatics Olympiad, Mongolia.

1. Introduction

There are about 565 high schools and 497 000 students in Mongolia (Purevjal and Altantuya, 2013). Each year Mongolian National Informatics Olympiad is organized in four stages: School competition, District competition, City/Province competition and National competition. Fig. 1. shows the participant numbers for each stage. Students with top scores are admitted to the next level.

The first National Informatics Olympiad was organized in 1987. Mongolian Informatics Association is responsible for organizing whole annual national informatics Olympiads in cooperation with the Ministry of Education and Science, other universities and ICT companies. In 2015 we've started using CMS (Contest Management System) in our National Competition which reduced the time needed for judging and made it less human dependable (Maggiolo, Mascellani, and Wehrstedt, 2014).

The first four winners of the national Olympiad participate in the International Olympiad in Informatics. The national winners receive scholarships to study IT at local universities (Choijoovanchig, Uyanga, and Dashnyam, 2007).

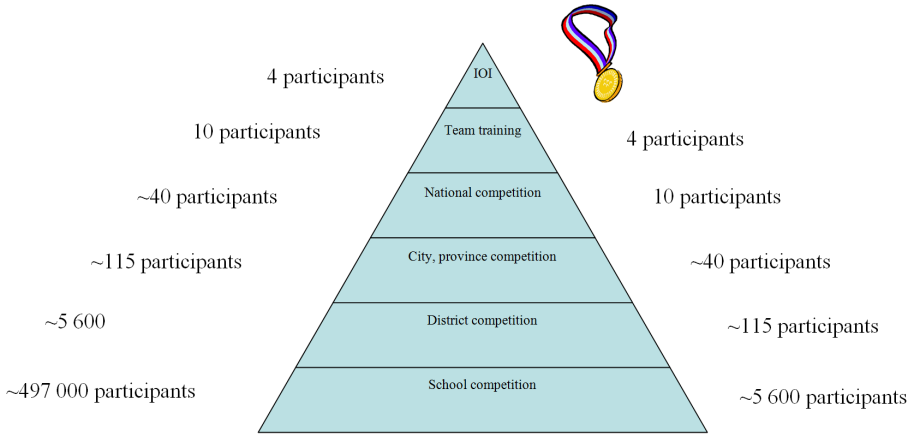


Fig. 1. Participants and structure of National Olympiad.

Professors from leading IT universities such as Mongolian National University of Education, Mongolian University of Science and Technology and National University of Mongolian, participate in trainings for IOI team.

Students solve 6 programming problems in two days for National competition. The maximum score at the competition is 300 or 600 points. Fig. 2 shows the total participant number and the percentage of the maximum score from absolute high score.

Fig. 3. shows results of the Mongolian team participation in IOIs from 2011 through 2015.

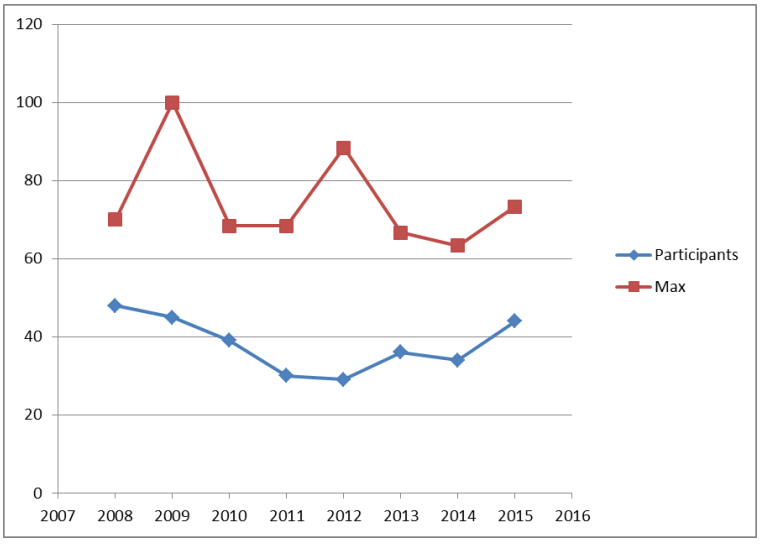


Fig. 2. Maximum score (in percentage from absolute high score) and participant number.

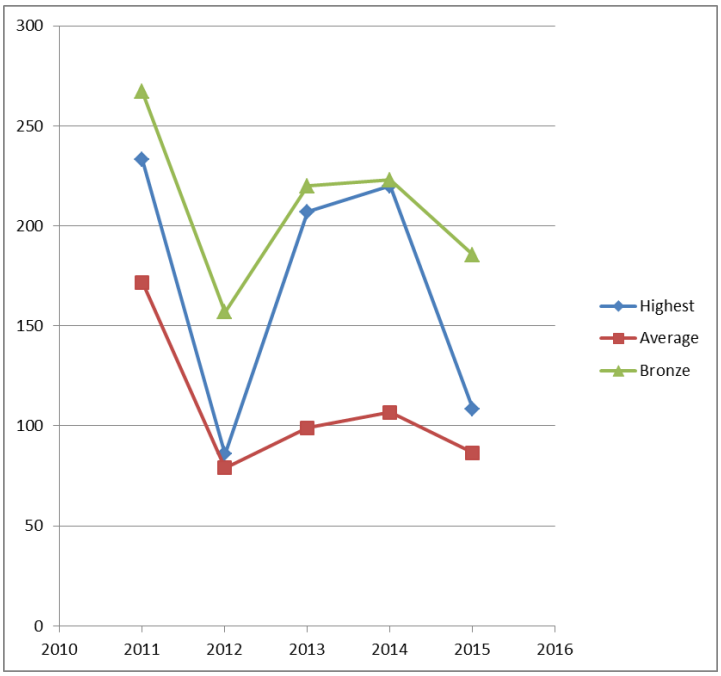


Fig. 3. Mongolian team results in IOI.

2. Key Issues and Problems

The following issues are the key challenges that hinder the success of Mongolian team at the international competition:

1. Weak English language. Due to language barriers students cannot fully understand tasks, use online internet sources and participate in online contests. Regular participation in online contests can be very useful for the development of students' programming and algorithmic skills.
   In addition, online contest rankings show us the preparedness of our students at the international level.
2. Lack of student's skills and knowledge related to IOI syllabus. IOI syllabus gives us list of knowledge that is required to solve most of the IOI problems. Lack of knowledge on IOI syllabus results badly planned training and subsequently lack of success at IOI.
3. Lack of highly qualified teachers to train students. The most successive former IOI participants are going abroad after graduation to study and work. Now they work for Google, Microsoft, Facebook, Amazon etc. This kind of "brain drain" takes most of our teachers away. If former IOI participants would share their experience with new participants it could help them much.

In the next section we will list how we have tried to improve students' skills and knowledge of IOI.

3. Improvements

Most of the problems stated in the previous section related to English language skills. Though teaching English is not our main purpose we've started looking for websites, online contests, which can support Mongolian language. To solve a problem student must translate the problem into Mongolian. However, it takes a lot of time and in addition student can misunderstand the problem.

There are many online contests organized almost every day on the internet. As most of our students are not strong in English we were interested in websites which allow us to add problems in Mongolian language and upload the tests or just translate the problems into Mongolian language.

So far we've found following websites and contests:

1. Bebras. Last year our students were able to do the Bebras problems in their native language and they were excited because there were enough problems to challenge themselves. In this contest we had wide choice for the easy and hard problems (Vegt, 2013). We think many high school students were stimulated to participate in informatics Olympiad after their Bebras participation.
2. SPHERE online judge (`www.spoj.com`). In 2008 one of our committee members contacted Andrew Kosowski from Gdansk University, Poland and asked permission to translate the problems on the website into Mongolian language. Now most of the basic level online problems in Mongolian language are on SPHERE. Now several teachers from Mongolian can add problems and organize contest on SPHERE servers.
3. Croatian Open Competition in Informatics (`www.hsin.hr/coci`). There are 7 online contests in a year and the organizers allow translators from other countries to get the problems in English and translate into their native language before each online contest.
4. Hackerrank. The website allows to its users to be problem setter and organize a contest.
5. USACO. One of our contestants contacted a personal from USACO and they agreed him to get problems before the contest and translate them into Mongolian.
6. Asia-Pacific Informatics Olympiad. The Asia-Pacific Informatics Olympiad (APIO) is an IOI-like competition for delegations within the South Asian / Western Pacific region.

All these websites and online resources became available for us thanks to international cooperation. Therefore we think one of the best ways to improve quality of our national team is international cooperation.

As a result of a fruitful cooperation with Russian team we got their full syllabus for IOI preparation. Now we have translated the syllabus in Mongolian language and we hope this will help us to organize well-planned training.

Recently some of the former contestants are coming back to Mongolia to share their experience with other contestants. That is a good way to share their knowledge with the next generation and this kind of feedback will help us to support continuous development of Mongolian team.

Also we have a dedicated website for our National Olympiad in Informatics so we can save our results for later analysis (Mongolian National Olympiad NGO, 2015).

4. Summary

Compared to Mongolian report in 2007 there were following main improvements:

- We introduced full IOI syllabus to students.
- We have an official website for National contest.
- We started using IOI judge system in National contest (CMS).

Skills and knowledge of Mongolian students are improving each year and there are more and more students interested in Informatics Olympiad. To support them further we will need some text book in the future.

Another future work for us is to develop international cooperation further so that our students can participate in many more online contests in their native language. We hope this way they can achieve good placements in IOI and become good specialists.

References

Mongolian National Olympiad NGO (2015). (In Mongolian). Retrieved 2016.04.21 from: `www.informatics.mn`

Choijoovanchig, L., Uyanga, S., Dashnyam, M. (2007). The Informatics Olympiad in Mongolia. *Olympiads in Informatics,* 1, 31–36.

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

Purevjal, A., Altantuya, Y. (2013). *Statistical Report 2013.* Ulaanbaatar: OchirPress LLC.

Vegt, W. v. (2013). Predicting the difficulty level of a Bebras task. *Olympiads in Informatics*, 132–139.

A. Khuder is an associate professor of the School of Information, Communication Technology, Mongolian University of Science and Technology. He is PhD in ICT and actively participates in the organization of Mongolian Informatics Association. His research interests include Natural Language Processing, Operating Systems and Artificial Intelligence.

D. Tsedevsuren is Professor at School of Mathematics and Natural Sciences, Mongolian National University of Education. He is PhD in ICT and Educational Studies, and he is currently working as a President of this Mongolian Informatics Association. His research interests include Informatics education, ICT in eduaction, theory and methodology of eLarning and electronic learning content development.

Olympiads in Informatics, 2016, Vol. 10, 285–290

DOI: 10.15388/ioi.2016.21

Belarusian Olympiad in Informatics

Iryna KIRYNOVICH[1], Aliaksei TOLSTSIKAU[2]
[1]*Faculty of Computer-Aided Design, Belarusian State University of Informatics and Radioelectronics*
Brovki 6, Minsk, Belarus, 220013
[2]*Faculty of Applied Mathematics and Computer Science, Belarusian State University*
Nezavisimosty Ave. 4, Minsk, Belarus, 220030
e-mail: kirinovich.irina@yandex.ru, tolstikov@bsu.by

Abstract. Olympiad in Informatics has been held in Belarus since 1988 and is a high priority direction in the work with gifted students. National Olympiad in Informatics consists of four stages. It is the most popular intellectual competition among youth in the Republic of Belarus.

Keywords: Olympiad in Informatics, trainings, selection for international competition, International Olympiad in Informatics, IOI.

1. Introduction

In 2016 the XXIX Belarusian Olympiad in Informatics was held in Mogilev, Belarus. Belarusian team has taken part in IOI since 1990, when the II International Olympiad in Informatics was held in Minsk. For 26 years the Republic of Belarus was represented by 63 members who were awarded by 79 medals (gold 14, silver 33, and bronze 32).[1]

Participation in the International Olympiads is the final step of multi-stage competition among gifted students. Participants of the first selection and training camp are determined during the final round of the national contest. In 30 days teams of the Republic of Belarus prepare for participation in IOI, IMO, IPhO, IChO, IBO. The national contest and the preparation for international competitions are provided by employees and students of the leading universities

National Olympiads are organized by the Ministry of education of the Republic of Belarus.

The winners (Fig. 1) of the final stage of the competition have the opportunity to enroll in the state universities without exams.

[1] `http://stats.ioinformatics.org/countries/BLR`

Fig. 1. The final stage of the Belarussian Olympiad in Informatics in 2016 – winners of the 1st degree diplomas. Head of Department of education of regional Executive Committee Vladimir Ryzhkov is standing in the middle of the photo.

2. National Belarusian Olympiad in Informatics

National Olympiad in Informatics is held with the aim of identifying and improving the capabilities of the gifted students in the field of informatics and programming. The main goals of the National Olympiad are to increase interest in learning programming, to deep their theoretical knowledge and practical skills, to prepare students for participating in the international competitions. Participation in all stages of the Olympiad is free for students.

National Olympiad is held in each academic year in four stages: the first stage is the educational establishment level, the second stage is the district (city) level, the third stage is the regional competition, the fourth stage is the final national round.

Problems of three types are used at the stages of the National Olympiad in Informatics; each of them requires developing an algorithm for the problem and implementing it in one of the available programming languages (C++ or Pascal). The problem of the first type is the classic one: contestant has to write a program that reads the input and produces the answer, i.e. the contestant doesn't know which test data will be used to check his program. The problem of the second type is a question with open tests. In problems of this type participants have all input data, and it is necessary to provide the results of the solution for known test data.

Solution grading for the second type problems usually uses partial scoring, the solution is compared with the optimal or best solution among all participants of the contest. The problem of the third type is the interactive one. In problems of this type the source code of a participant interacts with unknown to him library. Solutions obtain input and the list of actions that should be performed using known protocol. These problems allow

Fig. 2. Awarding 3rd degree diplomas and medals to the holders of the final stage of the Belarussian Olympiad on Informatics 2016.

using questions required processing of input data in real-time, games, tasks, "black box" interaction etc.

Between the stages of the Olympiad training camps are held to prepare students to participate in the future stages of the competition.

The winners of each round of the Olympiad are awarded with I, II, and III diplomas (Fig. 2). The number of winners at each stage of the contest does not exceed 45% of the number of participants. The number of I diplomas does not exceed 20% of the number of winners. Share of II diplomas is no more than 30%, III diplomas – no more than 50%. The actual number of winners at each stage is determined by the jury.

2.1. *First and Second Stages*

The **first stage** of the National Olympiad should be held not later than November of the current academic year and is an internal stage in each institution of the Republic of Belarus. At this stage all interested students take part in the competition. The best students qualify to participate in the second stage of the Olympiad in accordance with the established quota. The problems preparation for this stage is not kept centrally so each institution determines the format and prepares questions independently.

The **second stage** is the city or district level. At this stage the contest is held no later than December of the current academic year and consists of one practical exam. The contest problems for the second stage of the Olympiad are prepared centrally by each regional department of education and contain usually only the classic problems. After the second stage of the Olympiad each district forms a team to participate in the third stage of the Olympiad.

2.2. *Third Stage*

The **third stage** of the National Olympiad is held in each region of the Republic of Belarus (7 in total). All participants compete on the same problem set prepared by the Ministry of education of the Republic of Belarus. The problem of the second type (open tests) adds to the classic problems at this stage. The competition is held in 2 practical exams, each of them requires solving 3–4 questions. The problem complexity is significantly higher at this stage than at the previous one.

Each region forms a team of 15 students to perform at the final stage. In addition, the team may be expanded at the expense of the last year contest winners (final stage), who won diploma this year (third stage), but was not ranked in the top 15 in the region, as well as the winners of the International Zhautykov Olympiad (Kazakhstan).

In January/February each regional team receives additional training camp for 14 days to prepare for the final stage. Students listen to the theoretical lectures on constructing algorithms, how to use data structures in the efficient way, and various branches of mathematics and computer science. In addition to theoretical studies there are practical exams with analysis.

Team of the Lyceum of the Belarusian State University participates in addition to the seven regions of the Republic of Belarus in the final stage of the Olympiad.

2.3. *Final Stage*

The **final stage** of the National Olympiad is carried out by the Ministry of education within 5 days in one of the host cities in March/April (Table 1). The contest problems at this stage contain the interactive task in addition to the classical and open tests.

In the National Olympiad in Informatics all students shares a common rank list regardless of grade (Table 2). About 50% of the participants of the final stage of the Olympiad are graduates of secondary schools, gymnasiums and lyceums.

All students (grade 11) who won the final stage are eligible for admission to the state universities of the Republic of Belarus without exams.

3. Preparation for International Competitions

According to the results of the final stage of the National Olympiad the list of 10–12 most successful students is formed for selection and preparation of the participants of the International Olympiad in Informatics. Camps are held in 2–3 phases with a total duration of 30 days.

At the first phase of the trainings participants compete in from 10 to 15 rounds, the complexity of the problems is close to the international level. After each round analysis is provided by coaches. In addition, training sessions are conducted with rounds of solving different types of problems: object recognition, optimization problems, game problems, etc. The list of the International Olympiad in Informatics participants forms

Table 1

Number of participants of the final stage in 2012–2016

2012	2013	2014	2015	2016
123	123	120	122	120

Table 2

Distribution of participants of the final stage in 2012–2016

Grade	2012		2013		2014		2015		2016	
	Partici-pants	winners	Partici-pants	winners	Partici-pants	winners	Partici-pants	winners	Partici-pants	winners
6–8	9	2	5	3	9	4	7	2	7	1
9	14	5	18	6	14	4	22	8	19	7
10	43	21	35	12	32	15	29	12	43	22
11	57	28	65	34	65	31	64	32	51	24

after the first camp. The team is approved by the Ministry of education of the Republic of Belarus.

During the next camps coaches make lectures with different topics based on the complex data structures and algorithms, usage of a contest system image provided by IOI organizers and other. Participants continue preparation solving the competitions from other countries and regions. The participation in training camps for students is free.The results of the team of the Republic of Belarus in IOI over the last 5 years are presented in Table 3.

In Belarus operates the high-tech Park, which includes more than 150 resident companies. Every year, the resident companies of the Park of high technologies take part in rewarding winners of the final stage of the Olympiad in Informatics (Fig. 3).

The winners of the final stage of the Republican Olympiad will be awarded with cash prize of the special Fund of the President of the Republic of Belarus for social support of gifted students and shall be included in a national data Bank of talented youth.

The winners of the international Olympiad receive the title of Laureate of the special Fund of the President of the Republic of Belarus for social support of gifted pupils and students.

The presence of this title gives the right to receive a number of social benefits.

Table 3

Performance of the team of the Republic of Belarus in IOI in 2011–2015

2011		2012		2013		2014		2015	
10 grade	Gold	11 grade	Gold	11 grade	Gold	11 grade	Bronze	11 grade	Silver
10 grade	Silver	11 grade	Gold	11 grade	Silver	10 grade	Silver	11 grade	Silver
9 grade	Silver	10 grade	Silver	11 grade	Bronze	10 grade	Bronze	11 grade	Bronze
9 grade	Silver	10 grade	Bronze	11 grade	Bronze	8 grade	–	10 grade	Bronze

Fig. 3. Nine girls participated in the final stage of the Republican Olympiad on Informatics 2016. Some of them won medals. Director of the company "Yandex Bel" Alex Sikorsky has congratulated the participants.

4. Conclusion

This article briefly describes the procedure for conducting the National Olympiad in Informatics in Belarus. Description of all four stages of the competition and the schedule of training and selection camps are provided. In the Republic of Belarus National Olympiad in Informatics is an important and significant event. Belarusian students won lots medals at the international level.

I. Kirynovich, associate professor of the department of engineering psychology and ergonomics, faculty of computer design, Belarusian State University of Informatics and Radioelectronics, PhD in Physics and Mathematics, national coordinator of the Bebras contest in Belarus, the Chairman of the jury of the final stage of the Republican Olympiad in Informatics scientific and pedagogical leader for the preparation team of the Republic of Belarus for participation in the international Olympiad in Informatics.

A. Tolstsikau, senior lecturer at Department of computational mathematics, Belarusian State University. Member of the jury of the final stage of National Olympiad in Informatics and various student programming contests. Research interests include parallel and distributed computing, computational complexity theory and algorithms and systems processing large volumes of information.

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, 2016

Olympiads in Informatics

Volume 10, 2016

ISSN 1822-7732

9 771822 773007