Resurgence of Informatics Education in Schools
A Way to a Deeper Understanding of Informatics Concepts

Valentina Dagienė(✉)

Vilnius University, Vilnius, Lithuania
valentina.dagiene@mii.vu.lt

Abstract. Five decades ago, computing at several high schools was consistent with the academic and professional world. Nowadays, many countries have started to think about how to establish or re-establish informatics education in schools and how to attract children to learn informatics. Although informatics is not taught as a discipline in many countries, children are invited to participate in different contests and competitions in informatics, programming, or robotics organized all over the world. Educators are interested in motivating students to learn more about informatics, to understand informatics concepts, and to develop computational thinking. This paper discusses informatics education through the recent decades with its main focus on Logo activities and other strategies which support deeper understanding and promote attractive approaches of learning informatics in school. The worldwide Bebras informatics challenge is presented and discussed as an example of connecting formal and non-formal informatics education by using thousands of tasks based on informatics concepts and applying problem-solving strategies. As a case study, the statistical data from the 2017 Lithuanian Bebras challenge is presented and discussed.

1 Introduction

Significant changes in society do not begin at one particular day or even in a particular year. Changes come slowly, especially in school education. Teachers, policy makers, and researchers should work continuously for decades in order to gain significant results on children's achievements in informatics.

Teaching informatics (computer science or computing) in high schools started about four decades ago with the introduction of programming. Machines at that time were complicated to handle and programming was the exceptional possibility to manage them. The goal of teaching programming at schools was to show how a computer can help in routine work, mainly doing huge amounts of calculations. Computer programming is the best way to build a language for instructing (communicating with) a machine. Later, Juraj Hromkovič and others went deeper and declared: "We have to teach programming as a skill to

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 522–537, 2018.
https://doi.org/10.1007/978-3-319-98355-4_30

describe possibly complex behaviors by a sequence of clear, simple instructions" [11]. Then, Avi Cohen and Bruria Haberman promoted informatics (computer science) as a language of technology [4]. The history of informatics at school started with programming. However, programming is only a part of informatics. Why do we need to teach informatics at schools? What is informatics? What should we teach and how? These and similar questions are the core problems to everybody who has been thinking on bringing informatics to any school level.

The educational achievements that were obtained in the 1970s and 1980s might be explained by the implementation of computers in schools and by their impact to general education. In the beginning of the 1980s, informatics education, although to a different extent, was established in many schools across the USA and Europe, e.g., Austria, Bulgaria, Lithuania, and Slovakia.

One of the early Russian pioneers in the field of theoretical and systems programming, Andrei Ershov, said: "Programming is the second literacy", which became a slogan for several years [10]. Politicians and educators in the industrialized countries proclaimed "computer literacy" as an essential part of education, and they demanded the integration of new technologies into the school curriculum.

A significant role in designing a methodology for teaching programming has been played by researchers in various countries. More and more countries have been reconsidering the role of informatics in general education, discussing the curricula for teaching informatics in secondary schools, and developing new courses for teacher training in informatics. Informatics was significantly supported in countries which had a strong academic way of teaching. Lack of access to computers was compensated by providing theoretical knowledge on programming and the working principles of computer hardware. A lot of attention was paid to learn the syntax of a programming language, to understand commands, etc. These countries had implemented informatics education in secondary school education as a discipline and the main focus was on teaching programming.

Bringing informatics to schools through the curriculum in a formal track is quite important, but it is also necessary to support the informal ways of introducing students to informatics. Contests and olympiads on programming are informal ways of introducing informatics to students. There are some discussions and disagreements on the role of contests in education. An important aspect is how the contests are introduced: are they tools for attracting more students and to motivate them to participate and solve problems by collaborating and discussing results? In many cases, contests make learning and teaching programming more attractive for students. Furthermore, computer programming is one of the most appropriate and effective ways to develop problem-solving skills for students of computer science.

2 Logo

In 2006, when Lithuania celebrated twenty years of informatics in schools, it was written: "At the beginning there was Logo. Then everything happened" [5]. In

many countries, teaching informatics started with Logo, which was developed by Seymour Papert, the "father of Logo".

Logo is an educational language for children. Today the language is remembered mainly for its use of "turtle graphics", in which commands for movement and drawing produce line graphics. The language was originally conceived to teach concepts of programming.

For most people, learning Logo is not an end in itself, and programming tends to have a higher goal for the learner. Logo is accessible to novices, including young children, and also supports complex explorations and sophisticated projects by experienced users.

Seymour Papert's Logo offers a window into a time when computers and Logo had been in widespread use in schools for just a few years, while also delving into issues about informatics and education that remain relevant today.

An important event occurred in 1980—the publication of Seymour Papert's *Mindstorms* [13]. Educators throughout the world became excited by the intellectual and creative potential of Logo. Their enthusiasm fueled the Logo boom of the early 1980s. The book begins with an affirmation of the importance of making a personal connection with one's own learning and ends with an examination of the social context in which learning occurs.

In 1985, Logo Computer Systems corporation developed *LogoWriter*, which was a novel computer program in several ways. First, it included word processing capabilities. Second, the user interface was simplified and made more intuitive. *LogoWriter* also included, similarly as the earlier "sprite" Logos, multiple turtles that could take on different shapes.

Another innovation of the mid-eighties was Lego/Logo. M. Resnick and S. Ocko developed a system which interfaced Logo with motors, lights, and sensors that were incorporated into machines built out of Lego bricks and other elements.

Then more and more interesting programs appeared: Scratch, SNAP!, NetLogo, Comenius Logo, and so on. Scratch became a block-language metaphor and was widely used all over the world.

Many tasks for learning Logo have been developed, and many books have been published on Logo-inspired ideas to introduce children to programming. The ideas from Logo reoccur time after time; through the development of learning tools in many countries when new educational products are developed.

There have been many important events in connection with teaching Logo and algorithms all over the world (see Fig. 1). A detailed Logo Tree Project is provided by Boytchev [2].

3 Pascal as a Language to Introduce Algorithms

Pascal, the great programming language that served in education for many years, was designed in 1968 at ETH Zurich, and implemented and published in 1970. Pascal was designed by Niklaus Wirth with the goal to encourage good programming practices to novices by using structured programming and data structuring.

XLogoOnline		ABZ of ETH Zurich
Code.org		
Librelogo		
Turtle Academy		
Blocky		
SNAP!-BYOB	**2010**	B. Harvey, J. Mönig J. Hromkovič: *Algorithmic Adventures: From Knowledge to Magic*, 2009.
StarLogo TNT		
Computational Thinking		
Khan Academy		
Bebras		
Scratch		MIT Media Lab
	2000	
Netlogo		U. Wilensky
		B. Harvey: *CS Logo Style*, 1997.
CS Unplugged		Tim Bell
Programmable Bricks		
Logo Blocks		
Micro Worlds		M. Resnick: *Turtles, Termites and Traffic Jams: Exploration in Massively Parallel Microworlds*, 1994.
Python	**1990**	G. van Rossum
IOI		
EuroLogo conference		
(now Constructionism)		
Lego Logo		M. Resnick, S. Ocko
Object Pascal		
Logo Writer		
Logo Visualization	**1980**	S. Papert: *Mindstorms: Children, Computers, and Powerful Ideas*, 1980.
Constructionist approach		
		N. Wirth: *Algorithms + Data Structures = Programs*, 1976.
Pascal		N. Wirth
Logo	**1970**	S. Papert

Fig. 1. Influential contributions to school informatics education based on tools and approaches for learning algorithms and core concepts

In lectures, articles, and books, Wirth describes the history of (and road to) Pascal [18,19]. Structured programming and strict type checking were two of the innovative ideas proposed by programming researchers at this time.

Pascal had big influence on programming education in many countries. In Lithuania, Pascal was chosen as the language to communicate to big machines and express algorithms especially for secondary school students. It was a way

PROGRAMAVIMO KULTŪRA

TRYLIKTOJI PAMOKA

Skyrelį tvarko LTSR MA Matematikos ir kibernetikos instituto jaunesnioji moksliné bendradarbė Valentina DAGIENĖ

Programuotojai, rašantys neališkias, griozdiškas programas, mėgsta teisintis, kad programa skiriama kompiuteriui, o ne žmogui. Be abejo, kompiuteriui programos aiškumas nesvarbus — jis mechaniškai atlieka veiksmus ir nesidomi programos vaizdumu. Tačiau kad ir kaip atrodytų keista, didžiausias programų skaitytojas vis dėlto yra žmogus, o ne kompiuteris. Skaitydamas programas, žmogus susipažįsta su kitų programuotojų idėjomis ir patirtimi, mokosi pats sudarinėti programas. Dažnai tenka tobulinti ir pačių sukurtas programas. Visais tais atvejais reikia gilintis į programos esmę. Ką tik parašytą, dar šviežią atmintyje programą skaityti ne taip sunku. Tačiau ilgainiui ji pamirštama. Todėl programos turi būti rašomos aiškiai, vaizdžiai, suprantamai.

Programavimo vadovėliuose vis daugiau dėmesio skiriama geram programavimo

mentarais galima paaiškinti ne tik kintamųjų vardus, bet ir atskiras programas dalis, nurodyti, ką vienas ar kitas sakinys atlieka ir panašiai. Komentarus galima įterpti visur tarp atskirų simbolių, žodžių, skaičių, vardų. Jie suskliaudžiami skliaustais (*ir*).

Komentarai padeda greitai ir lengvai skaityti programas. Tačiau jais nereikia piktnaudžiauti — komentarai turi būti jakoniški, griežti, trumpai nusakantys pagrindinius dalykus, neužgriozdinantys programos teksto.

Paminėsime dar vieną programavimo kultūros elementą — programų redagavimą. Redagavimu vadinamas programos teksto išdėstymas popieriaus lape. Nekyla abejonių, kad žmogui kur kas lengviau skaityti vaizdžiai išdėstytą programą. Be to, tokioje programoje būna mažiau klaidų (pavyzdžiui, sunkiau pamiršti žodį end, jei jis rašomas po jį atitinkančių žodžių begin), lengviau jas surasti ir pataisyti.

Kaip kuo geriau suredaguoti programą, neretai priklauso nuo paties programuotojo — svarbu tik, kad būtų aišku ir vaizdu. Visose pamokose mes stengėmės pateikti suredaguotas programas. Laikėmės kai kurių bendrų redagavimo taisyklių: sakinius, esančius klojo mėnesio pabaigoje prieauglį duos tik pirmoji pora, todėl turėsime tris poras, o dar po mėnesio prieauglį duos ir pradinė pora, ir pora, gimusi prieš du mėnesius. Todėl iš viso bus 5 poros.

Simbolių F(n) pažymėkime triušių porų skaičių, kurį turėsime po n mėnesių. Matome, kad n-ojo mėnesio pabaigoje turėsime tiek porų, kiek jų buvo prieš mėnesį, t.y. F(n−1) ir dar tiek naujų porų, kiek jų buvo prieš du mėnesius, t.y. (n−2)-ojo mėnesio pabaigoje. Kitaip sakant, gausime tokią priklausomybę:

F(n) = F(n−1) + F(n−2)

Pateiksime programą triušių porų skaičiui po n mėnesių spausdinti.

```
program fibonacci;
  var fn, (* F(n) *)
      fn1, (* F(n−1) *)
      fn2, (* F(n−2) *)
      n, (* mėnesių skaičius*)
      k: integer;
begin
  fn1:=1; (* F(−1) *)
  fn:=1; (* F(0) *)
  read (n);
  for k:=1 to n do
    begin
      fn2:=fn1;  (* praėjo *)
      fn1:=fn;   (* vienas *)
      fn:=fn1+fn2(*mėnuo*)
    end;
  write (fn)
end.
```

```
for s:=1 to n do
  begin
    write (s);
    for d:=1 to
      if s mod
        write ('
    writeln
  end
end.
```

Jis nėra efektyv
kadangi iš uždavini
nereiau, kad n būt
delis, tai skaičiavir
daug ir nėra reika
pektų tobulinti pro

KONTROLINIAI UŽDAVINIAI

13. Duota progra

```
program atspėk;
  var a, b, c,
      i, j: integer;
begin
  read (n);
  a:=1;  b:=1;
  for i:=0 to n do
    begin
      for j:=1 to b
        write ('*');
      writeln;
      c:=a+b;
      a:=b;  b:=c
    end
end.
```

Ką išspausdins k
atlikęs šią program
dinis duomuo yra
uždavinio sprendim
tas šia programa? (
14. Pradiniai du
natūrinių skaičių s
pabaigos požymis
Sudarykite program
siam ir didžiausiam
riui spausdinti. Sel
giantis nulis sekos
komas. (9 balai)
15. Pradiniai du
du natūriniai skaič
reiškiantys stačiaka

Fig. 2. Programming lessons were printed in the largest national daily newspaper from 1981 to 1983, from January to May, once a week. In the upper left corner appears the logo of the School—JPM: letters J, P, and M from the Lithuanian name of the school (Jaunųjų programuotojų mokykla).

to algorithmic thinking. Pascal was the backbone for the Young Programmers' School by Correspondence established in 1981 in Lithuania [9]. The goal of the school was to teach the main concepts of procedural programming and basic algorithms using Pascal notation. The first programming lessons were published in the largest daily newspaper over nearly half of a page a few times per month for a number of years, see Fig. 2.

All the teaching materials of the Young Programmers' School consisted of several chapters of lessons in a cyclic manner: (1) identifiers, variables, constants, assignment statements and sequence of statements, (2) conditional statements, (3) repetitions of actions, (4) programs and their execution by a computer, (5) logical values, (6) functions and procedures, (7) recursion, (8) discrete data types, (9) real numbers and records, (10) arrays, (11) programming style, (12) program design. Theoretical knowledge was delivered only as supplementary material for understanding informatics concepts and program design assignments.

Actually the Young Programmers' School laid the foundations to teach informatics in all upper secondary schools in Lithuania starting from 1986, and later, in 2005, in all lower secondary schools.

Informatics education in Lithuanian schools (grades 10–12) was based on Pascal, the language that fit perfectly for thinking about and developing algorithms. Pascal's advantage was its simple syntax and logical structure. Also, we can use Pascal to develop algorithms using pen and paper and later run them on a computer, which was an important aspect in the early eighties when schools had very limited access to machines. We taught algorithms to solve tasks, and we taught and learned how to think, both constructively and critically.

4 Programming, Children, and Bebras Tasks

Logo and Pascal have influenced informatics education in schools all over the world. A whole generation grew up programming with the Logo turtle as a visual tool and with Pascal for text-based algorithms. A group of informatics researchers in Lithuania developed many interesting tasks for learning programming and used them in school informatics curricula and after-school activities.

Researchers have shown that learning to program is a complex cognitive activity [16]. For example, the features of Logo as a programming language, like interactivity, modularity, extensibility, and flexibility of data types, are very powerful. Modular design requires children to organize their programs into conceptually independent units. This modularization process is the core to teaching informatics. First, it helps to approach challenging problems by breaking them down into smaller tasks, which can be solved independently by applying top-down or bottom-up problem-solving strategies. Second, thinking about problem instances in an abstract way helps identifying structural properties inherent to the problem, and therefore allows children to classify them into abstract problem classes. This results in a high intrinsic cognitive load.

Developing abilities to master modern technologies and skills for solving problems is among the most important capabilities of an educated future citizen of the information society, and these abilities can be directly connected with informatics education. Problem solving by means of programming does not lose its importance in a contemporary school equipped with modern information technologies, and it will remain a very important part of understanding the information processing and running of a computer.

Programming is an activity composed of several components: comprehension of the problem, choosing and encoding an algorithm, debugging, testing, and optimizing. Since many of the skills required for successful programming are similar to those required for effective problem solving, computer programming, particularly choosing one of several possible solutions and later debugging it within a short period of time, provides a fertile field for developing and practicing problem-solving skills in an environment that is engaging for young students [3].

When students begin to learn the basics of programming, they soon try to find a place where they can demonstrate their skills and projects, share their interests or compare themselves with others. This might explain the reasons why many students, soon after they have started learning programming, choose one of the areas where they are able to demonstrate their work immediately, e.g., creation

of web pages or computer graphics. For some areas, e.g., developing algorithms, it is not easy to find a practical demonstration. The most powerful means that endorse the students' motivation are competitions and contests.

Nowadays, England has become the country to mandate computer science classes for all children between the ages of 5 and 16. Their age will determine exactly what they will learn, with topics ranging from algorithms to debugging code and lessons in programming languages.

4.1 Worldwide Challenge on Informatics Concepts

Pasi Sahlberg, a world-renowned Finnish educator, indicated playful learning, games, and gamification as one of the success factors of Finnish education [15]. Playful learning activities can attract the children's attention to learn various subjects.

Olympiads and contests in informatics focus mainly on developing algorithms and programming. The programming course can be followed by the introduction of some basic concepts of algorithms, data structures, recursion, procedures, and fundamental methods for designing algorithms. These and other basic informatics concepts can be introduced to students by contests or other activities in an attractive way.

In 2003, the idea of the Bebras contest on informatics was proposed (the name Bebras—in English *beaver*—is connected with a hard-working, intelligent, goal-seeking and lively animal). It took almost a year to create sample tasks and to prepare the technology for implementation. The contest started with a few countries and focused on school students divided into a few age groups.

Later, many other activities have been developed under the Bebras umbrella: hands-on seminars for students and teachers, discussions for deepening informatics knowledge and teamwork on developing Bebras tasks, so that the contest idea was changed to a broader Bebras challenge on informatics and computational thinking. In the past years, the number of Bebras participants has been notably growing. In 2007, the Bebras contests took place in seven countries, with about 50 000 participants in total. In 2017, more than 2 million students from over 50 countries were involved in solving Bebras tasks world-wide. Slovenia had the strongest relative participation with nearly 30 000 contestants, whereas France had the highest total number of participants, over half a million.

The crucial point of the Bebras challenge are the tasks based on informatics concepts [1,14]. The challenge needs to have a challenging set of tasks. The developers of Bebras tasks are seeking to choose interesting tasks (problems) to motivate students to deal with computer science and to think deeper about what constitutes the core of informatics. Some agreements on task development criteria have to be settled. Initially, six task topics (types) were proposed: (1) Information comprehension, (2) algorithmic thinking, (3) using computer systems, (4) structures, patterns, and arrangements, (5) social, ethical, cultural, international, and legal issues, as well as (6) puzzles [6,17]. During the last few years, a two-dimensional system for categorizing tasks was elaborated [7]. The suggested categorization system incorporates both informatics concepts and computational thinking skills in the classification of tasks.

Algorithms, data, data structures, data representation, and abstraction are the most important areas of informatics for schools. Creating an interesting task (problem) that is based on informatics concepts requires a lot of intellectual effort. Collaboration with different cultures and researchers, as well as peer-review and discussions, are very important for developing tasks which promote and introduce core concepts of informatics.

4.2 Bebras Tasks

An international Bebras workshop on creating tasks (problems) based on informatics concepts for all age groups is organized annually in different countries. The main goals of the workshops are to develop a set of tasks for the upcoming challenge, to discuss them and to come to an agreement among the countries with different (or without) curricula and traditions of teaching informatics in general education. All submitted tasks, including graphics, are developed under the Creative Commons Attribution-ShareAlike international license.

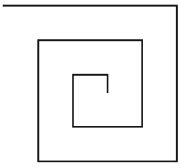
Several types of tasks (problems) have been used in the challenge: interactive (dynamic) tasks, open-ended tasks, and multiple-choice tasks are the most common groups. Motivation and attraction of students to take part in the challenge and to think deeper about informatics concepts are based on context-rich and powerful tasks, whose development and introduction are undoubtedly very challenging for researchers as well as for teachers.

Multiple-choice tasks have four non-trivial and well-defined answer choices with only one correct solution. For dynamic tasks, interactivity is defined as a two-way transfer of information between a user and a machine. Thus an interactive task provides a specification of the problem, and, in solving it, a student needs to interact directly with the computer: drag and drop objects, click spots on pictures, manipulate objects using a keyboard, select list elements, etc. Many countries have established networks and teams of researchers, teachers, and educators for creating and discussing tasks. They usually come with new proposals every year. Below, three examples of Bebras tasks are provided and discussed.

Task Example 1: Drawing a Spiral. Ada has drawn a rectangular spiral with the help of a dynamic object, a turtle, and two commands:

forward 10 — the turtle moves forward drawing a line which is 10 steps (dots) long;
left 90 — the turtle turns left making an angle of 90°.

Which of the following numbers expresses the length of the whole spiral in the number of dots?



A 550
B 170
C 300
D 250

This example is a task from a decade ago. It introduces two commands: move forward and rotate. Additionally, the task requires calculations based on some understanding of what is happening.

Task Example 2: Beaver Code. Beaver Barbara has two rubber stamps. With the first one, she can produce a little flower and with the second one a little sun. Being a clever girl she thinks of a way to write her own name with these two stamps by encoding each letter with a sequence of flowers and suns as follows:

Letter	B	A	R	E	Y
Code	✿	✷✷	✷✿✷	✷✿✿✿	✷✿✿✷

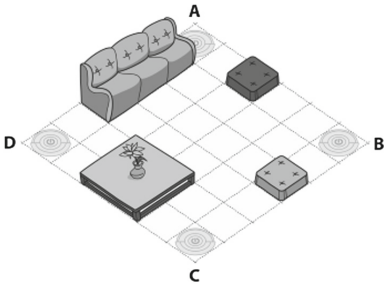
The name "Barbara" would be encoded as: ✿✷✷✷✿✷✿✷✷✷✿✷✷✷.

She then goes on to also write down the names of her friends. But, unfortunately, they got all mixed up (see the right column below). Drag the corresponding sun-flower-codes to the names of Barbara's four friends.

Abby		✷✷✷✿✷✷✿✿✷✷✷
Arya		✿✷✷✷✿✷✷✿✷✷✿✷
Barry		✷✷✿✿✷✿✿✿✷
Ray		✷✿✷✷✷✷✿✿✷

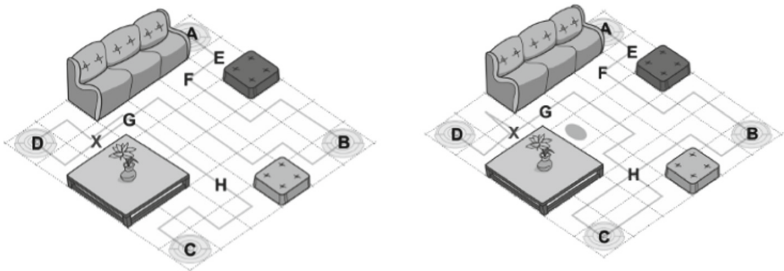
This task was developed by a group of educators from Switzerland in 2016 (U. Hauser, J. Hromkovič, T. Kohn, I. Kosírová, B. Steffen, Ch. Datzko). Often in informatics, instead of storing data in a simple and straightforward way, we can devise a scheme to store it more efficiently, using less space. Letters have different frequencies in texts (depending on the given language), and we can use these frequencies to improve our encoding. Specifically, we encode frequent letters with smaller codes. In this task, B takes one symbol, A takes two, and the other letters take more. However, you have to make sure that the code is unambiguous. Huffman encoding is a widely used algorithm.

Task Example 3: Robot Cleaner (proposed by V. Dagienė, Lithuania). A robot washes a square-tiled floor by using the following commands: F—move forward one tile (takes 1 min), R—turn 90° right (performed instantly), W—wash a tile (takes 1 min). The robot starts and ends on one of the four corner tiles (A, B, C, D) — but not necessarily on the same tile.



What is the minimum number of minutes the robot needs for cleaning all the available tiles on the floor?

For a solution strategy, the main issue is to minimize the number of tiles which are passed twice. There is a tile X which must be passed twice in every chosen way (as seen in the two figures below). Tiles on the "bottleneck" to corner tiles A, B, C, and D are described by letters E, F, G, H, respectively. Tiles E and F are at the A bottleneck, tile G at the D bottleneck and H at the C bottleneck. Corner B does not contain any bottleneck tile. If a corner tile is neither the starting nor the finishing point, the robot must pass through bottleneck tiles of this corner twice. If the starting and finishing corner are different, bottleneck tiles of these two corners need to be passed through only once. Thus, we can try to arrange the path such that the robot goes through as few bottleneck tiles as possible. If the robot starts and finishes at the same corner tile, it must pass all of the bottleneck tiles twice. Since the bottleneck at A contains 2 tiles, namely E and F, a good strategy is to choose the point A as a starting (or finishing) tile. Then, we have to decide between finishing points C and D (in both cases we avoid going twice through another bottleneck point, G or H).



The trajectory in the picture to the right above shows that, for moving from A to D, the robot needs an odd number of moves so that, after using 28 moves forward, at least one tile is not passed yet, so it is longer than from A to C (trace in the left picture above). The minimum time for cleaning is thus 27 (washing) + 28 (moving) minutes = 55 min by going from A to C.

This is a special case of the traveling salesman problem—finding the shortest path when we want to visit every graph node, which transforms to the extended Hamiltonian path problem if there is a path in the graph to visit every node exactly once. There is no efficient algorithm to solve either of these two problems. For small cases, students can investigate the situation and come up with a strategy for solving the problem in a systematic way.

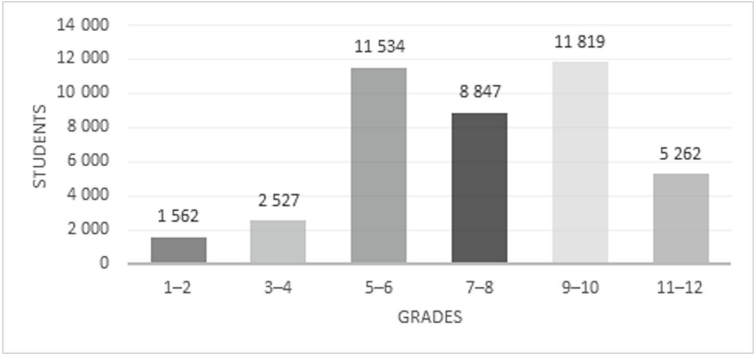


Fig. 3. Number of participants distributed by age groups (school grades)

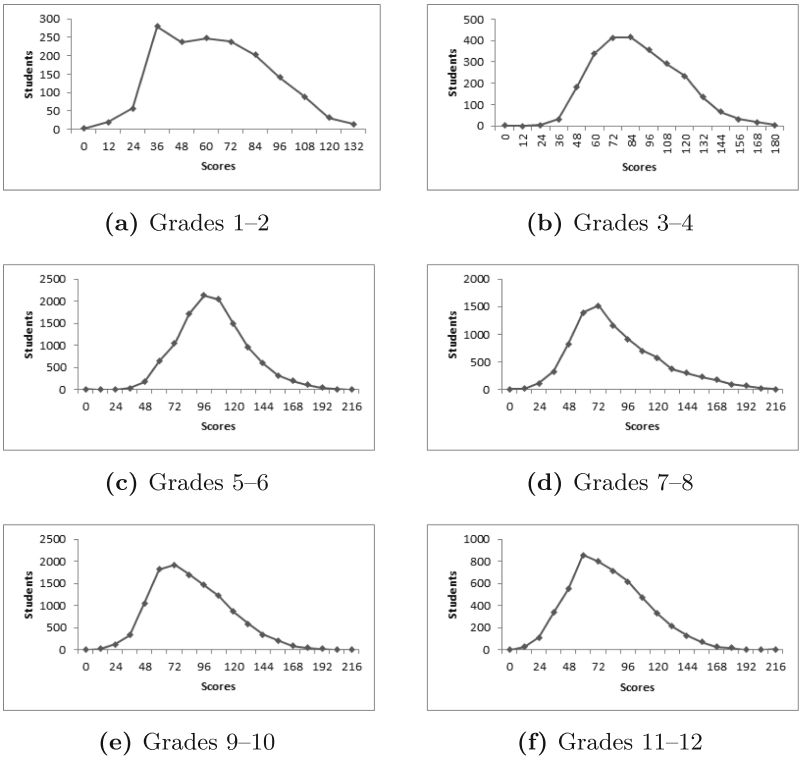
4.3 Solving Tasks: A Lithuanian Case

In November 2017, over 2 million school students from 40 countries took part in the Bebras challenge. Countries in the Southern Hemisphere are going to run the contest later in March.

As a case study, we will discuss data about the Lithuanian participants in the 2017 challenge. From an organizational viewpoint, most countries are running the contest and supplementary activities in a similar way. The countries choose tasks on their own from a Bebras task pool—there is a large amount of reviewed and well documented tasks for various age groups. However, we have noticed that many countries have agreed to use the same (or almost the same) set of tasks. The set of tasks chosen in Lithuania is overlapping with the selection in many countries: Austria, Germany, the Netherlands, Switzerland, Finland, Hungary, Sweden, etc. (all age groups except grades 1 and 2). Lithuania had over 41 000 participants in the 2017 autumn Bebras challenge. The distribution of participants by grades is presented in Fig. 3.

The Bebras community has reflected on the involvement of girls in informatics education. An investigation on gender issues was done during previous years, for example, in the UK and Lithuania [8]. The Bebras challenge can be seen to be an event that attracts girls' attention to informatics education: worldwide, more than 40% of participants are girls (we cannot estimate the number exactly because some students have not indicated their gender). There is evidence from several countries that relatively more girls participate at a younger age and more boys take part in the higher grades [12].

The data shown in Fig. 4 breaks down the scores per age group in Lithuania Bebras 2017. Lithuania has all six age groups covered now—the youngest grades 1 and 2 joined the contest last year. Most countries have organized the challenge based on 4 or 5 age groups in primary schools as well as lower and upper secondary schools. Scores are grouped and the number of students scoring a range of marks is shown for each age group. As we can see, the figures show a normal



(a) Grades 1–2 **(b)** Grades 3–4

(c) Grades 5–6 **(d)** Grades 7–8

(e) Grades 9–10 **(f)** Grades 11–12

Fig. 4. Distribution of scores from tasks in all age groups in Lithuania (grades from 1 to 12)

distribution of scores in solving tasks across all age groups (similar results were obtained in the 2015 challenge in both Lithuania and the UK [8]).

We selected 67 different tasks distributed among six age groups. Some tasks were the same for 2–3 age groups. Tasks are categorized according to informatics concepts and computational thinking skills. For this study, we are interested in tasks related to algorithm concepts and algorithmic thinking. Usually, tasks cover more than one concept of informatics, but then we can usually estimate what the main concept behind the task is. 37 tasks were chosen for detailed investigation in this study. Eight tasks were solved by more than half of the students and six tasks were very hard—less than 10% solved them. This year, the chosen algorithm tasks were really challenging for the students—nobody obtained full scores. For the hardest tasks like "Pizzeria Biberia", "Arabot's walks" and "Book-Sharing Club", the most common format is a long text with several detailed pictures. We have observed that students, especially in grades 7–10, which were the youngest students to whom these tasks were given, do not like reading long explanations and usually skip such tasks.

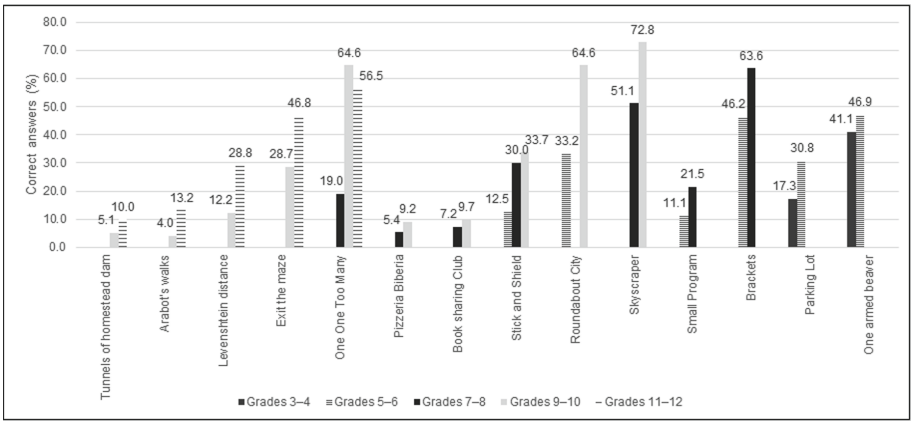
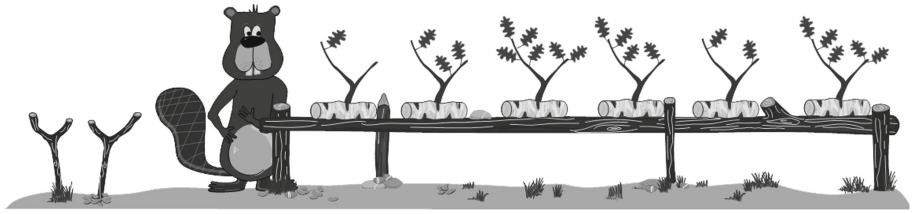


Fig. 5. Percentage of students that solved correctly the algorithms tasks that were given to more than one age group

As mentioned above, some tasks were given to 2–3 age groups. In the algorithmic concepts category, there were 13 tasks chosen for two age groups and two tasks for three age groups. We observed that students in the older age groups are doing better than expected. For most of the tasks, the differences in solutions between age groups are quite large, except for "Stick and Shield" (30% solved by grades 7–8 and 33.7% by grades 9–10) and "One-Armed Beaver" (41.1% solved by grades 3–4 and 46.9% by grades 7–8), see Fig. 5.

Students had the possibility to comment on the tasks in the Lithuanian contest management system. The task "One-Armed Beaver" (proposed by Christian and Susanne Datzko, Switzerland), given to students in grades 3–4 and grades 5–6, raised most excitement and got the most positive feedback.

One-Armed Beaver. The more leaves a branch has, the tastier it is. The beaver intends to sort the branches based on their taste using the temporary storage beside him. Please help the beaver sort the branches by taste so that the tastiest branch is closest to him.



Variables are a core concept in informatics: This task focuses on swapping the value of two variables. Swapping two pieces of wood in two places by using a temporary new storage—this is about variables and their values. Also a sorting algorithm is needed for this task. For sorting a list while having a limited amount of memory, we use a sorting algorithm such as selection sort: Going from the

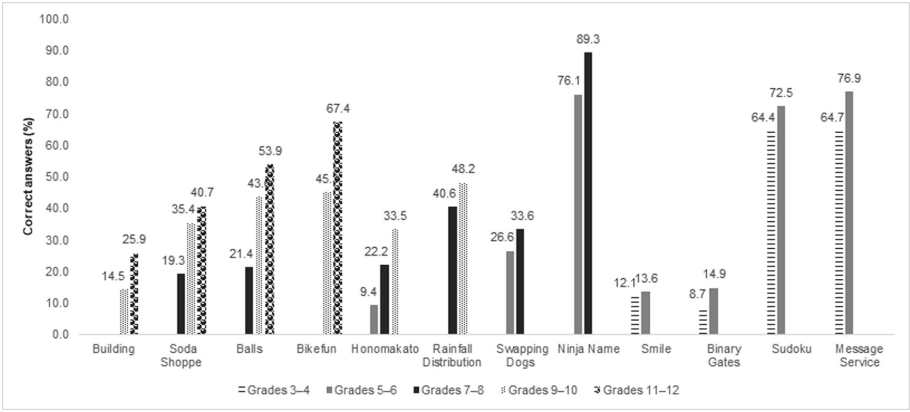


Fig. 6. Percentage of students that solved correctly the tasks related to data structures and data representation that were given for more than one age group

first to the last element, swap the current element with the smallest one in the remaining list using a temporary variable.

The statistical data (see Fig. 5) of the task "One Too Many" (Janez Demšar, Slovenia) shows that the juniors solved it better than the oldest students (64.6% of students in grades 9–10 solved this task correctly and the result is better than the 56.5% for grades 11–12). Core concepts in this task include handling a text editor and how to follow a sequence of instructions. The task is also about swapping values of two variables by using a temporary variable, but in a more complex situation than in "One-Armed Beaver" above.

The concept of data, data structures, and data representation is a separate category within the categorization of Bebras tasks. However, that category is very tightly related to algorithms and computational thinking. Observing the results in this category, we have noticed that there is not a big difference between scores gained by adjacent grades for the two oldest age groups solving a given task; when 3 groups solved the same task, the youngest group has much lower results for each task (see Fig. 6). This indicates that data structures and data representation concepts are more easily accepted by different age groups than algorithmic concepts.

The participants of the Bebras challenge liked the task "Ninja Name": It is short and clearly formulated. Understanding a string is a core concept of this task, and it involves coding as an attractive element as well. Letter replacements in strings are often used in informatics tasks. These tasks serve to provide an easier visualization of what is happening when elements are replaced within complex rows of data such as strings of signs.

5 Conclusions and Challenges

We need to work on balancing continuity and innovativeness in informatics education. Is continuity boring? The sun comes up every morning, we take the same route to work every day, and through the window we see the same landscape. We often do not even notice these things but there is a place for them in our minds and so we feel well and peaceful. A similar understanding of continuity can be applied to learning informatics at schools for students—lessons, everyday problem solving, focus on core concepts, and a few contests at various levels per year as motivational tools can be a comfortable continuity.

I ask myself, what has been the most important phenomenon during all those years of introducing informatics in schools? Probably—the relationship among people. While sitting for long hours at your workplace, in different meetings, no longer knowing if you are thinking in the right direction, not knowing how to move on, you suddenly receive a message from a person, then from another one, and then from yet another one. You feel connected and see many others who search, doubt, discover, and rejoice. We must not stop dreaming, searching and communicating: about everyday life events and informatics education among them.

Involving students in the recognition of informatics as a science discipline should be our target, and we should try to achieve it more successfully. Well-organized activities with interesting, playful, and exciting tasks will introduce students to the essence of the computer science world and help them to understand core concepts of informatics.

Acknowledgements. I want to acknowledge my doctoral students Gabrielė Stupurienė and Lina Vinikiene for taking care of the Bebras challenge in Lithuania and sharing their ideas on how to improve Bebras tasks based on informatics concepts.

References

1. Bellettini, C., Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A., Torelli, M.: How challenging are Bebras tasks? An IRT analysis based on the performance of Italian students. In: Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, pp. 27–32. ACM (2015)
2. Boytchev, P.: Logo Tree Project (2014). http://www.elica.net/download/papers/LogoTreeProject.pdf. Accessed 5 Feb 2018
3. Casey, P.J.: Computer programming: a medium for teaching problem solving. In: Computers in the Schools, vol. XIII, pp. 41–51. The Haworth Press Inc., New York (1997)
4. Cohen, A., Haberman, B.: Computer science: a language of technology. ACM SIGCSE Bull. **39**(4), 65–69 (2007)
5. Dagienė, V.: The Road of Informatics. TEV, Vilnius (2006)
6. Dagienė, V., Futschek, G.: Bebras international contest on informatics and computer literacy: criteria for good tasks. In: Mittermeir, R.T., Sysło, M.M. (eds.) ISSEP 2008. LNCS, vol. 5090, pp. 19–30. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69924-8_2

7. Dagienė, V., Sentance, S., Stupurienė, G.: Developing a two-dimensional categorization system for educational tasks in informatics. Informatica **28**(1), 23–24 (2017)
8. Dagienė, V., Sentance, S.: It's computational thinking! Bebras tasks in the curriculum. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 28–39. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_3
9. Dagys, V., Dagienė, V., Grigas, G.: Teaching algorithms and programming by distance: quarter century's activity in Lithuania. In: Proceedings of the 2nd International Conference on Informatics in Secondary Schools: Evolution and Perspectives (ISSEP 2006), pp. 402–412. Institute of Mathematics and Informatics, Vilnius (2006)
10. Ershov, A.P.: Programming, the second literacy. Microprocess. Microprogr. **8**(1), 1–9 (1981)
11. Hromkovič, J.: Contributing to general education by teaching informatics. In: Mittermeir, R.T. (ed.) ISSEP 2006. LNCS, vol. 4226, pp. 25–37. Springer, Heidelberg (2006). https://doi.org/10.1007/11915355_3
12. Hubwieser, P., Hubwieser, E., Graswald, D.: How to attract the girls: gender-specific performance and motivation in the Bebras challenge. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 40–52. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_4
13. Papert, S.: Mindstorms: Children, Computers, and Powerful Ideas. Basic Books Inc., New York (1980)
14. Pohl, W., Hein, H.W.: Aspects of quality in the presentation of informatics challenge tasks. In: ISSEP 2015. LNCS, vol. 9378, pp. 21–32. Springer, Cham (2015)
15. Sahlberg, P.: Finnish lessons 2.0: what can the world learn from educational change in Finland? Teachers College, Columbia University (2015)
16. Sweller, J.: Cognitive load theory. In: Psychology of Learning and Motivation, vol. 55, pp. 37–76. Academic Press (2011)
17. Vaníček, J.: Bebras informatics contest: criteria for good tasks revised. In: Gülbahar, Y., Karataş, E. (eds.) ISSEP 2014. LNCS, vol. 8730, pp. 17–28. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09958-3_3
18. Wirth, N.: Recollections about the development of Pascal. In: Bergin, T.J., Gibson, R.G. (eds.) History of Programming Languages II, pp. 97–120. Addison-Wesley (1996)
19. Wirth, N.: Pascal and its successors. In: Broy, M., Denert, E. (eds.) Software Pioneers, pp. 108–119. Springer, Heidelberg (2002). https://doi.org/10.1007/978-3-642-59412-0_8