ACTA DIDACTICA NAPOCENSIA

Volume 7, Number 1, 2014

KEY CONCEPTS IN INFORMATICS: ALGORITHM

Péter Szlávi, László Zsakó

Abstract: "*The system of key concepts contains the most important key concepts related to the development tasks of knowledge areas and their vertical hierarchy as well as the links of basic key concepts of different knowledge areas.*" (Vass 2011) One of the most important of these concepts is the algorithm. In everyday life, when learning or working we continually make and execute algorithms, design series of activities and information flow processes, which make this world fully comprehensible only for those who are familiar with the basics of these activities. We believe that the algorithm is the most important concept in informatics. If it is overshadowed in IT education, you will get a distorted picture, which might lead IT education astray. In this article we concentrate on what an average student is supposed to learn about the concept of algorithm and what scheduling is to be applied to achieve our aims in years 1-4., 5-6., 7-8. and 9-10.

Key words: key concept, data, primary and secondary school.

1. Areas of Informatics

Information technology is difficult to define since it is one of the youngest science areas and knowledge areas. One reason is its young age whereas the other one is its extremely rapid development, and its impact on other fields of knowledge.

Though Hungarian primary and secondary education last for 12 years altogether, we do not deal with every age group in this article but we focus on all students who are supposed to learn informatics in years 1–10. Regarding 11th and 12th graders, only those study this concept on a deeper level that are planning to go into an ICT career and participate in programming contests. The latter group has not been included in this survey; just like those who took part in programming contests at a younger age. All in all, we concentrate on what an average student is supposed to learn about the concept of algorithm and what scheduling is to be applied to achieve our aims.

To define the key concepts of informatics, first you should overview the possible curricula of informatics as a field of knowledge (Zsakó 1996, Szlávi & Zsakó 2005, Zsakó 2007).

A. **Algorithmization, data modelling, programming** (At school – and even in your everyday life – you continually execute algorithms, work with data structures e.g. fill in forms and questionnaires, design series of activities and information flow processes, which implies that this world is fully comprehensible only for those who are familiar with the basics of these activities.)

B. **Programming tools** (They include language and other tools which you must be familiar with to be able to implement and try algorithms and data models.)

C. **Performing application tasks with computer** (It means the solubility of everyday problems by using IT tools: video and graphic editing, word processing, spreadsheets, database management and presentation.)

D. **Managing application systems** (Here you should separate the knowledge of applications from the ability to manage rapidly aging devices; although teaching how to use them, of course, should take place parallelly.)

E. **Computer-aided problem solving** (In this topic you should start from a problem emerged – e.g. organising a school trip. First you should approach it as an organisational task, then you should choose the tools and devices for the subtasks – not necessarily IT tools –, and then create a new tool if necessary.)

Received 13 February 2014.

F. **Infocommunication** (You should be aware of the social impact of ICT technologies and adapt to the changes; you should use ICT tools properly.)

G. **Media informatics** (There appeared media heavily infiltrated with IT tools that require IT expertise for proper use; the electronic counterparts of the traditional media open up new opportunities and new media appear; all these might raise cognitive processes and entertainment to another level.)

H. **Informatics tools**: Principles of their operation and application (A wide variety of hardware and software tools are available, the appropriate use of which every user must acquire.)

I. **The mathematical basis for informatics** (The mathematical foundations necessary for IT skills are not included in the mathematics curriculum or they are not discussed where they are needed – mathematics at secondary school, for instance, does not deal with the matrix, but spreadsheet skills demand the formulation of this concept at least in an informal way – which is alright but it implies that it is the ICT subject that should discuss this issue as well as other applications of mathematics.)

J. **Informatics and society** (It is worth learning about the history of informatics – as it is part of the culture –, dealing with its possible development and its current impact on society, data security, data protection and the ethical issues of applying information technology.)

2. ICT competences

The key concepts of informatics cannot be separated from ICT competences; they must comply with them: (Zsakó 2007, Zsakó & al. 2008)

- Algorithmic thinking
- Data modelling
- Modelling the real world
- Problem solving
- Communication skills
- Application skills
- Team work, collaboration, interoperability
- Creative skills
- Orientation and information skills
- Systemic thinking

3. Key Concepts in Informatics

The key concepts of informatics are partly based on more general concepts of other fields (e.g. sign – data are described with signs, signs are used when you communicate, and they are used to describe documents). On the other hand, these concepts appear in other subjects as well (e.g. model, problem) (Vass 2011).

- algorithm
- data
- document
- communication
- model
- task
- problem
- project
- program
- software

- hardware.

Please remember that this overview of key concepts does not mean that it is exactly what should be taught at school. This article is about what teachers need to know so as to teach well! There are often functions related to the concepts: concepts are the ones we work with, while functions are the activities carried out on them. Consequently, it might be interesting to overview the activities.

4. A key concept in informatics: Algorithm

In everyday life, when learning or working we continually make and execute algorithms, design series of activities and information flow processes, which make this world fully comprehensible only for those who are familiar with the basics of these activities.

In the beginning algorithmization is not about computer implementation. Just think of a classical, several thousand year old algorithm which defines the greatest common divisor of two integers: it is the Euclidean algorithm The executor of an algorithm, i.e. the processor is often the human being who creates and interprets the algorithm. (Moreover, when working on a new problem, even an experienced programmer tries walking in the computer's shoes, thus testing the operability of the solution.) Everyone performs algorithms every day. In addition, the majority of people also formulate algorithms for themselves and for others (eg. transport, assembling things etc.).

Not until you have done this, can you leave the execution of a precisely formulated algorithm to an automaton i.e. to a computer. Basically, it requires a different mindset, as an intelligent algorithm-executor sensibly executes our algorithms, controls them, and sometimes even corrects the mistakes we have made. An automaton, on the other hand, does not think but executes – if it can – what is included in its program.

We can conclude that the majority of people create algorithms for others (and for themselves); whereas implementing an algorithm is a common task for everyone.

What has it got to do with programming? Donald Knuth gives a possible answer: "*If you want to learn something, teach it. You are successful if people understand. They may say they understand even if they don't. The ultimate test if you are doing well is to teach it to a machine!*" (D. Knuth, 14th January 2012, ETH, Swiss Olympiad in Informatics)

4.1. Areas to be developed in years 1–4

For this age group an algorithm is a sequence of activities which can be executed by its creator – it is so even if the executor is actually a Lego robot (Pásztor & al. 2010, Petre & al. 2004) or a Logo turtle. The essence of both lies in the separability of algorithms and data: data are in the "memory" of the executor, and can be described as state components of the executor, as a finite automaton, while the algorithm can be read from another place. (Kalaš 2010)

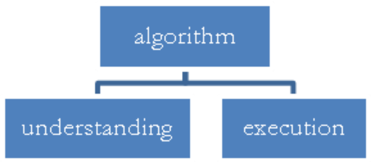


Figure 1. *Algorithm concept for years 1–4*

The most elementary level of algorithmic thinking is recognising algorithms and the problems that can be solved by using them.

You can define what an algorithm is. It is a procedure that can be divided into steps; it is executable (and there is an executor belonging to it); it has a fixed order of executing operations – it is quite difficult to understand (Dagdilelis & al. 2004); during the steps something happens to something; the steps are either elementary actions (that the executor understands) or they are algorithms themselves (which implies further specification).

Understanding has two stages:

- You understand what you should do,
- You understand why you should do that and not something else.

The question ”why” belongs to a higher level of thinking, to the analysis of algorithms, which means recognising, rejecting, varying and selecting alternatives.

Likewise, the ability to understand a known algorithm (i.e. one that has already been explained) and the ability to understand an absolutely new algorithm that has never been seen before also represent different levels of knowledge. The latter is the third level of algorithmic thinking.

If you received instructions from someone independent of you (especially in a dependency relation) step by step, you would have little scope for consideration but carry out the algorithms, whereas if it were you to formulate instruction, it would be completely different.

The ability to implement algorithms is a level higher than the ability to understand them. In this case it is not enough to simply understand a process but you have to monitor the interim stages during the implementation (keep in mind, register etc.) i.e. you should not pay attention to what is going on but to what you should do next depending on certain factors.

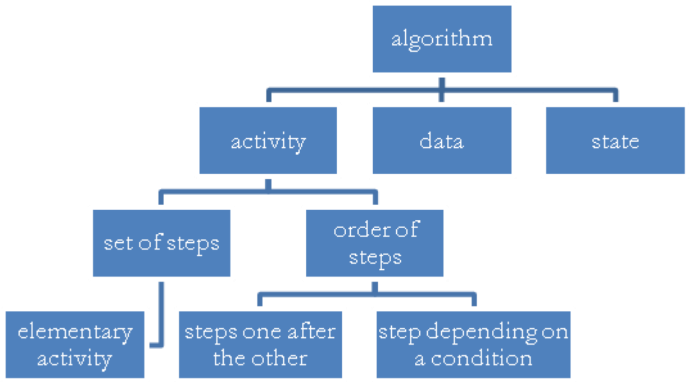


Figure 2. *Concept of algorithm structure for years 1–4*

4.2. Areas to be developed in years 5-6

Although you could have formulated it for the previous age group, this is the age when it is more important to understand that a computer algorithm executed by yourselves or by a computer – though similar – are fundamentally different from one another. The algorithm that you execute is executed by an intelligent executor, automatically correcting the mistakes made at the time of formulation during the execution. The computer, however, does not correct anything but will execute what you prescribed for it. Philosophically, it is more important perhaps that computer algorithms are methods and can be used dozens of times for solving problems, whereas you yourselves execute them only finitely often (or just once).

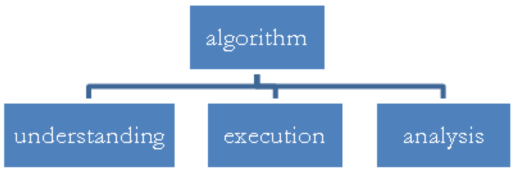


Figure 3. *Algorithm concept for years 5–6*

Analysing algorithms is partly about recognising the basic rules of constructing algorithms:

- Each basic step is to be executed (in the given order).
- You are to choose one of the basic steps and execute it.
- A basic step is to be executed several times, repeatedly.

Since the steps of algorithms themselves can also be algorithms, which can be labelled (actually, they should be labelled so that they could be quoted), the abstract notion of procedure can be formed.

On the other hand, the analysis of algorithms means that you understand what their parts are for, what your goal was when making them and how the whole problemsolving process was broken down to smaller subtasks.

In addition to the above mentioned, algorithm reading ability also belongs here. It means that you are able to understand a complex activity formed in an algorithm language by someone else: you can see the aims of the parts, their relations to other parts and you are also able to explain them. (Verbalization may not be identical with understanding; in fact, it is probably a new, higher level.)

Those that are able to understand and implement algorithms are not necessarily able to make new ones as that process requires some surplus. First you should consider (Hvorecky 1983, Hromkovic 2009)

- what you know;
- what you would like to know;
- what will happen;
- what data you will have to work with;
- and how you can break down your tasks into smaller subtasks.

It partly means swapping reading activities for writing activities, but there is much more to that. At lower competence levels you have guidelines, which help your thinking, but in this case you need to find out everything yourself.

Realising algorithms is a real ICT task: an algorithm should be described with a tool in a way that the result could be executed by an automaton (e.g. a computer). It also implies that you should learn to use the tool (practically the programming language) you write the algorithm in. (Saeli & al. 2011)

Various tools also require a specific way of thinking that you should acquire:

- How do you envisage the execution of programs?
- How are programs constructed?

Like for 1–4 graders, in this age group acting and trying out algorithms without using a computer significantly contribute to students' comprehension. Obviously, this is why Logo language nearly dominates in this age group because in this language it is easy to link algorithms to end products. "Parts" of algorithms, socalled procedures can also be linked to these end products. (The more elaborate description of this topic would lead us from the algorithm concept to the related concept of programming languages, and to the role of the first programming language.) It may also be useful to link the algorithm concept to other knowledge areas (Kátai & al. 2011).

On the other hand, you can rarely produce flawless things at once. Therefore, you might need to see whether the algorithm created is correct or not, and to recognize and correct the mistakes if there are any. The latter could seemingly mean a simple analysis of the algorithm, but in ICT you have more possibilities. There exist such complex systems (programming environments) that you can use to make this activity more efficient than sheer thinking although, of course, it does not substitute thinking.

Testing is a "not liked" duty which requires much more thinking than you would expect. (Another side comment: choosing an algorithm world and a related programming language where faulty programs are also interesting – where you can boast of a mistake could turn testing and debugging into a positive activity.)

The purpose of testing is to give a schedule or scores to the discretion of the correctness of the code. In order to achieve this goal, you should find a method on what data are needed to get all the items of the code into motion. It is obvious to try and rely on the analytical ability mentioned at Point 3, though it is not the only system. The apprehensive knowledge of the task (and not the solution!) could help you find relevant cases. After you detect an error, your analytical ability will also be required for debugging.

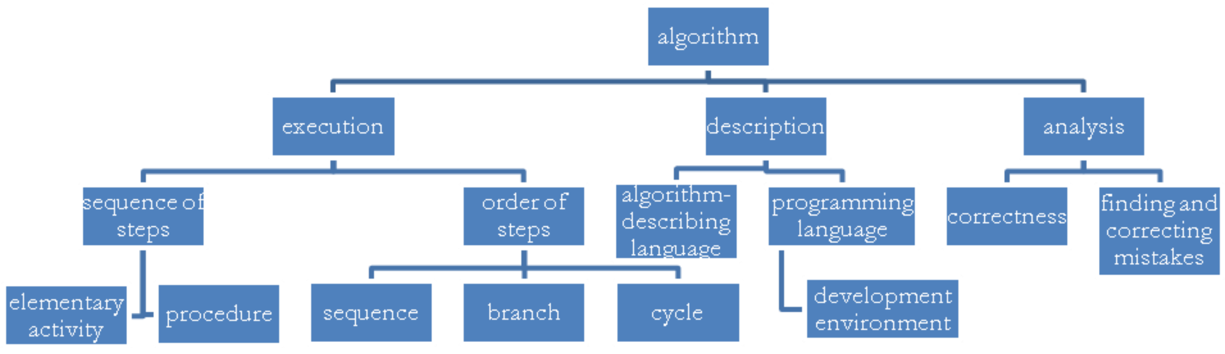


Figure 4. *Concept of algorithm execution and description for years 5–6*

4.3. Areas to be developed in years 7–8

In this age group there might happen a huge conceptual transformation: algorithms turn from a sequence of activities into calculation. In previous age groups algorithms work with objects of the real world, and their functioning is affected by signals or perhaps by parameters coming from the outer world. Now you use data to describe objects of the real world (in tight connection with the evolution of the data concept), which are loaded into the computer, there they are somehow represented and, using them, the algorithm calculates other data, which should be delivered to the user, or perhaps help him/her interpret the data received.

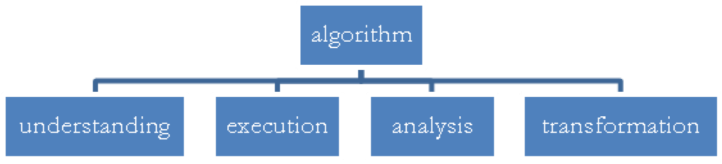


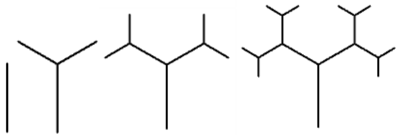
Figure 5. *Algorithm concept for years 7–8*

Understanding algorithms written by others is also a relatively easy task. Writing your own algorithm is not much more complicated, either – when compared to the task of modifying and improving an algorithm produced by someone else.

In the latter case it is not only the execution that you have to figure out, but you need to understand the way of thinking of the writer of the original algorithm and why he/she created the algorithm exactly that way etc.

You should also understand where you can enter the different world of thoughts of another person, what you can modify and what interventions will be effective etc. It may be much more difficult than writing an algorithm of your own, starting from scratches. You often receive inaccurately designed algorithms (or programs in the ICT world), which do not meet your expectations. Therefore, you must be able to change them so that they become useful and serve your purposes.

An algorithmic world linked to a visual world (e.g. Logo) makes understanding the concept of recursion specifically easy. The example below shows that a tree consists of a trunk and two smaller trees.



```
learn tree :pieces :length
  forward :length
  if : pieces >1 [left 60
          tree : pieces -1 : length /2 right 120
          tree : pieces -1 : length /2 left 60]
  back :length
end
```

Figure 6. *Algorithm concept for years 7–8, algorithm description with picture*

This is the point where the need to follow the implementation step by step may separate from the algorithm concept. The example above can be explained as follows: you draw a tree trunk, and then you make someone draw two trees with one height less in the right direction at the end of the trunk. This step may also provide a good transition to object-oriented approach later.

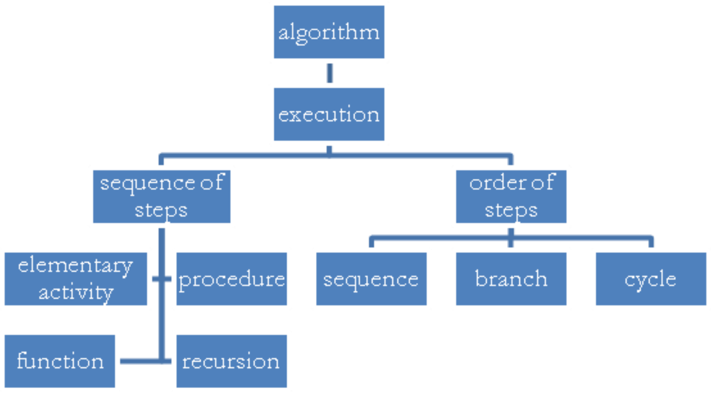


Figure 7. *Concept of execution of the algorithm description for years 7–8*

4.4. Areas to be developed in years 9–10

You cannot design algorithms smoothly unless you acquire some systematic method for making algorithms (and their related data models). You need to find a method to shortlist the infinite set of algorithms to a handy set with relatively few items. Obviously, you cannot do this without a largescale restriction of tasks; or only when you allow giving a small number of algorithm schemes instead of giving a small number of algorithms. The creative process in this case is selecting and combining suitable algorithm schemes and adapting them to concrete activities. The establishment of a scheme system requires analogical and abstract thinking. The distinction between essential and irrelevant features – abstraction; recognizing similarities – recognizing analogies.

Encoding can be made much less tiresome if you formulate the rules that you use for writing a code in a given language from an algorithm which contains the substantive description of the solution. Such rules can be defined to any language, and their “transplantation” can be made mechanical to 90%. (Menyhárt & al. 2012)

The work you should invest does not grow linearly with the increase in the size of algorithms. Sooner or later you will reach a stage when the solution of the problem cannot be seen as a single step. It is when abstraction and the systematic planning of algorithms could play a decisive role. (Harangozó & al. 1996)

It is when you may want to consider setting subobjectives and designing a series of activities for the particular subobjectives. (It is quite common in everyday life when several people are working together. In programming, however, it is not always necessary though it is frequent during the development of large software systems.). Of course, it is a serious task to see that by meeting the subobjectives and with a good synthesis, the task can be accomplished.

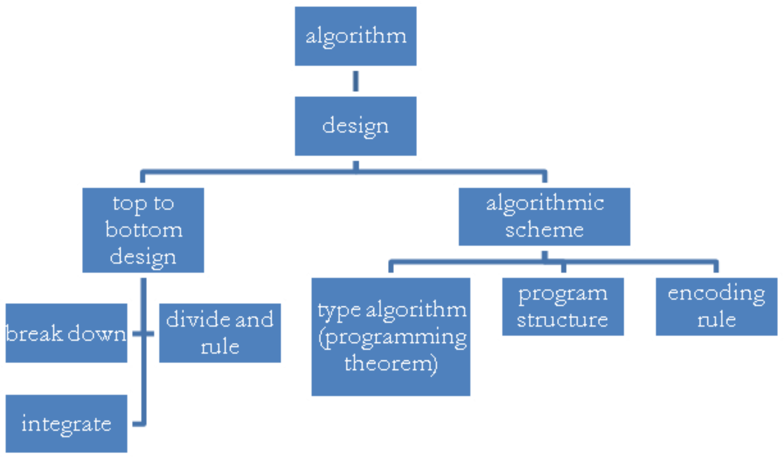


Figure 8. *Algorithm design in years 9–10*

You cannot always be satisfied with the algorithm you have created. Perhaps you cannot find an algorithm for a given task (which may also mean that you have come across a problem that cannot be solved algorithmically), or it could mean that the ICT solution takes such a long time that exceeds reasonable limits (has an exponential number of steps). To recognize this, first you need information about the running time of programs and the number of steps of the used algorithms.

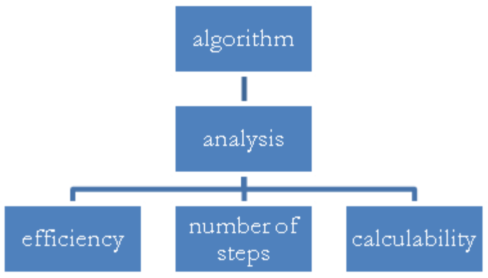


Figure 9. *Algorithm analysis in years 9–10*

5. Why can the identification of key concepts play an important role in designing the teaching-learning process?

We believe that the algorithm is the most important concept in informatics. If it is overshadowed in IT education, you will get a distorted picture, which might lead IT education astray.

The algorithm concept can fully evolve in primary and secondary education after stages described in this article for students that want to have more profound knowledge on algorithmization and programming. It is not only the group of students that study computer science in tertiary education, but a wider pool as the skill of writing programs can be useful and important for mathematics and science students as well as for future researchers.

References

[1] Dagdilelis, V., Satratzemi, M., Evangelidis, G. (2004), Introducing Secondary Education Students to Algorithms and Programming, *Education and Information Technologies*, 9:2, pp. 159–173.

[2] Harangozó, É., Szlávi, P., Zsakó, L. (1996), Joining Programming Theorems, a Practical Approach to Program Building, *Annales Universitatis Scientiarum Budapestinensis. Sectio Computatorica*.

[3] Hromkovic, J. (2009), Algorithmic Adventures – From Knowledge to Magic, *Springer*.

[4] Hvorecky, J., Kelemen, J. (1983), Algoritmizácia, elementárny úvod, *ALFA*, Bratislava.

[5] Kalaš, I. (2010), Recognizing the potential of ICT in early childhood education, *UNESCO Institute for Information Technologies in Education*.

[6] Kátai, Z., Kovács, L. I., Kása, Z., Márton, Gy., Fogarasi, K., Fogarasi, F. (2011), Cultivating algorithmic thinking: an important issue for both technical and HUMAN sciences, *Teaching Mathematics and Computer Science*, 9/1, pp. 107–116.

[7] Menyhárt, L., Harangozó, É. (2012), Hogyan bírjuk gondolkodásra hallgatóinkat, miközben segítjük is munkájukat? Dokumentáció alapú programfejlesztés, *INFODIDACT 2012 – 5. Informatika Szakmódszertani Konferencia*, Zamárdi, Hungary, 22–23 November 2012.

[8] Pásztor, A., Pap-Szigeti, R., Lakatos Török, E. (2010), Effects of Using Model Robots in the Education of Programming, *Informatics in Education 2010*, Vol. 9, No. 1, pp. 133–140, Institute of Mathematics and Informatics, Vilnius.

[9] Petre, M., Price, B. (2004), Using Robotics to Motivate 'Back Door' Learning, *Education and Information Technologies*, 9:2, pp. 147–158.

[10] Saeli, M., Perrenet, J., Jochems, W. M. G. (2011), Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective, *Informatics in Education 2011*, Vol. 10, No. 1, pp. 73–88, Vilnius University.

[11] Szlávi, P., Zsakó, L. (2011), Az informatika általános kulcsfogalmai: adat. *INFODIDACT 2011 – 4. Informatika Szakmódszertani Konferencia*, Szombathely, Hungary, 31 March – 1 April 2011.

[12] Szlávi, P., Zsakó, L. (2005), Informatics as a Particular Field of Education, *Teaching Mathematics and Computer Science*, 3/1, pp. 151–162.

[13] Szlávi, P., Zsakó, L. (2010), Informatikai kompetenciák: Algoritmikus gondolkodás, *INFODIDACT 2010 – 3. Informatika Szakmódszertani Konferencia*, Szombathely, Hungary, 22–23 April 2010.

[14] Szlávi, P., Zsakó, L. (2012), ICT competences – Algorithmic thinking, *Acta Didactica Napocensia*, Vol. 5, No. 2, pp. 49-58.

[15] Vass, V. (ed.) (2011), NAT-hoz illeszkedő kulcsfogalom-rendszer, kulcskompetencia-térkép, *Nemzeti Tankönyvkiadó*.

[16] Walter Gander (2012), Informatics in Schools? ECSS 2012, 20-21 November 2012, Barcelona, http://www.inf.ethz.ch/personal/gander/talks/GanderECSS2012.pdf

[17] Zsakó, L. (2007), *Fejezetek az informatika szakmódszertanából*, Habilitációs értekezés, Debrecen.

[18] Zsakó, L. (1996), Teaching Informatics in Hungary, *The IOI'96 NewsLetter*, No 2, pp. 5–6, No 3, pp. 5–6, No 4, pp. 5–6.

[19] Zsakó, L., Kátai, Z., Nyakóné Juhász, K. (2008), ICT methodology. *Teaching Mathematics and Computer Science – Infodidact*, pp. 3–24.

Authors

Péter Szlávi, Eötvös Loránd University, Budapest, Hungary, e-mail: szlavip@elte.hu

László Zsakó, Eötvös Loránd University, Budapest, Hungary, e-mail: zsako@caesar.elte.hu