Informatics in Education, 2018, Vol. 17, No. 1, 61–75

DOI: 10.15388/infedu.2018.04

Analysing the Different Approaches in Mathematics and Informatics for Solving a Task and Educational Implications

Ingrid NAGYOVÁ
University of Ostrava, Department of Information and Communication Technologies, Faculty of Education
Dvořákova 7, 701 03 Ostrava, Czech Republic
e-mail: ingrid.nagyova@email.com

Received: August 2017

Abstract. The paper focuses on the parallels, which are rooted in the simultaneous development of mathematics and informatics. Both mathematics and informatics are based on problem-solving. However, the approaches to determining problems, solution techniques and interpretation of results are different.

The paper shows different approaches of mathematics and informatics for solving a simple problem from the informatics competition. It was presented for students, who would be future informatics teachers, and it has become the beginning of the discovery of unexpected relationships and rules' chain, the source of successive tasks, and various methods of their solution. The paper brings the results of the constructivist teaching of students in the form of a fictional interview of mathematician and informatician. Fictional cooperation of a mathematician and an informatician in analysing and solving problems will allow for a detailed analysis and comparison of both fields, which will lead to determining both common and different elements.

Keywords: mathematics, informatics, instruction, informatics task.

Introduction

Mathematics can be characterized from various perspectives – as a science, a technique, or even an art. It is extremely difficult to find an unambiguous definition of a human activity such as mathematics (Kuřina, 2011). We will focus on mathematics instruction, especially in elementary schools and high schools. Kuřina (2008, p. 14) argues that in school, *"correct solutions to mathematical problems should be presented and both mathematical techniques and theory should be developed"*. According to Kuřina (2008), solving mathematical tasks and problems using mathematical tools and techniques is the foundation of good school mathematics.

Paul Halmos, a Hungarian-American mathematician, also stresses the importance of problem-solving in mathematics instruction. In his book (1982, p. 524) Halmos is trying to find the core of mathematics, stating that all parts of mathematics (axioms, proofs, concepts, definitions, formulas, theories, and methods, among others) are equally important. *"It is nevertheless a tenable point of view",* says Halmos, *"that none of them is at the heart of the subject, that the mathematician's main reason for existence is to solve problems, and that, therefore, what mathematics really consists of, is problems and solutions."*

These findings can also be used in informatics instruction, which appears to be similar to mathematics instruction. This similarity was caused by the centuries of simultaneous development. Both fields were so similar that there were no clear boundaries between them (Modeste, 2016). Modeste shows how important this development and interaction of mathematics and informatics are from the didactic point of view.

As in mathematics instruction, the problem-solving process and its evaluation form the informatics instruction's core. Schubert and Schwill (2004) argue that informatics instruction should be based on an active student who learns the informatics concepts, focusing on problems, approach (Vygotsky, 1978), and practical use. Informatics instruction is based on real-life problems and processes, which are presented by the teacher, solved by students – using an active approach (analysis, specification, concretization) – and then evaluated from different perspectives (correctness, efficiency, and optimization, among others). Furthermore, Schubert and Schwill create a basic informatics instruction model, specifying the individual competencies and learning objectives of informatics instruction. They describe particular didactic situations which may occur during the instruction of informatics, focusing on informatics ideas and techniques applicable in individual didactic situations. They also stress the actual computer work's limited role in informatics classes, which is only useful when obtaining results of solved problems.

Mathematics and informatics are extremely similar, just as one would expect, considering their historical development. Both are sciences focused on solving problems, especially considering the instruction of mathematics and informatics, respectively. However, as far as the social perception of mathematics and the opinions and attitudes of students, the future teachers are concerned, the relationship between mathematics and informatics is not so definite.

The paper is aimed at finding the relations between the instruction of mathematics and informatics and determining both their common and different elements. It uses the example of both the mathematician and informatician's approach to solving a particular problem ('The Informatics Beaver' competition) to show how differently both fields approach something that they have in common, i.e., solving a problem.

Project of Solving a Simple Informatics Problem

Aspiring teachers, who study informatics at the Pedagogical Faculty of the University of Ostrava (Czech Republic), have rather distorted ideas about informatics, about their research subject, and their methodology. They study informatics because they find it

easy to master, expecting it would not take much effort to obtain the diploma. On the contrary, their relationship with mathematics can often be characterized as a phobia (fear) of mathematics.

The causes of mathematics' phobia, or even fear of learning, are described by Papert (1980). In his monograph he precisely defines the terms mathophobia and fear of learning (Papert, 1980, p. 38). At the same time, he attempts to offer teaching methods in order to overcome the fear of mathematics and learning, which often prevails in the adult population.

Wolfram (2010) is also concerned with mathematics education, and he finds out, like that of Papert, the way leads through informatics and computers, "*There is a fantastic way to do that in the modern world. It's called programming.*"

Five years ago, we started to solve a simple task from The Informatics Beaver competition: There are some cars in the garage and each car can drive out from the garage through the top or bottom exit. We looked for the right leaving order for cars (see later – Task 1 in next chapter). Its solution was very simple for students. Suddenly, however, the question came and changed the task and process of its solution: How many solutions does the task have at all? Nobody in the lecture room knew the answer. We started to solve the problem together. Someone shouted out, "That's easy. In the first step, top and bottom car will leave the garage. In the second step, again and so on until the garage is empty. It means 2^n leaving orders for n cars." Nobody objected, and we ended this solution.

Next year, we tried to continue solving the problem. We created a program, exactly according to the algorithm proposed by the students a year before. Taking advantage of the recursive function was necessary. We had to learn a lot together before the students were capable of solving and painting all leaving orders.

What showed up? Each leaving order was repeated twice. Why? We had forgotten that the two different processes of driving out the car from the garage generate one leaving order – the colour of leaving order is not affected by driving out the last car through the top or bottom exit.

Then I thought: There can be also two cars simultaneously driven out of the production line in the factory. We build a special gateway at the upper exit through which there can be driven exactly two cars simultaneously (see later – Task 4). When we try to examine these sequences, we obtain the Fibonacci sequence. We explore the process of creating new sequences together with our students, and we modify the previous program.

This was only a short illustration, how we developed the solution of a simple task in many directions. The scope for searching the solutions of given tasks, which was offered to our students, enabled us to ask more and more questions. Our search for answers and solutions to the tasks led us to the discovery of new and unexpected rules and relations.

In our paper, we present results of the teaching in the form of fictive dialogue between mathematician and informatician. Even though we programmed, analysed the process of solving the tasks, and so on, the students thought that we did not study informatics, but mathematics. Therefore, we were trying to strictly distinguish the work of mathematician from the work of informatician. We summarize our results in conclusion and try to examine both differences and common analogies in both points of view.

Fictional Cooperation between Mathematics and Informatics

The Informatics Beaver competition is held in a number of countries. It is intended for pupils of the primary and secondary school. In 2009, intended for pupils 10 to 12 years of age, the following task was included in the Benjamin category of the Czech version of the competition.

Task 1:

Cars are parked in a narrow garage (see Fig. 1).

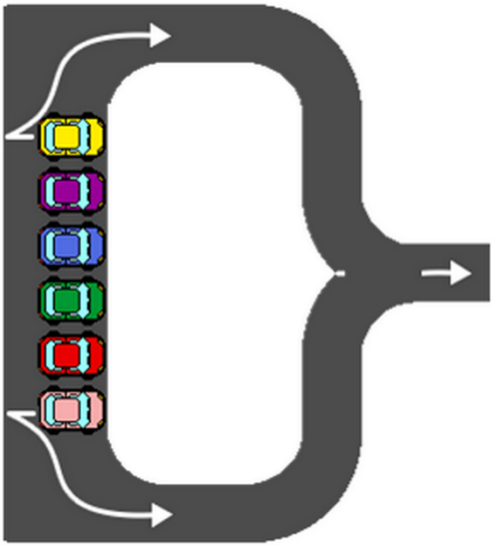

Fig. 1. Cars parked in a narrow garage.

Only one of the cars at the end of the line can leave the garage. In what order could the cars leave the garage?

Choose one of the following answers:

A. 

B. 

C. 

D. 

As far as informatics is concerned, the task shows that the order (algorithm) of cars leaving the garage can be determined in a number of ways, e.g., as a sequence of coloured cars. When solving the problem, one needs to realize that the order of the leaving cars (hereinafter referred to as leaving order) must be read from right to left – the car which left the garage first is the rightmost. Therefore, we can rule out answer D – the green car cannot leave the garage first. Similarly, answers B (the red car cannot leave through the bottom exit before the pink car) and C (the purple car cannot leave through the top exit before the yellow car) can also be ruled out. Therefore, answer A is correct. It is an informatics task to verify the correctness of an algorithm that is (non-traditionally) based on the order in which coloured cars leave a garage.

Let's imagine that a mathematician and an informatician met and started discussing the solution possibilities. The mathematician would say, "Wow, Informatician, I had no idea that informatics dealt with such problems. I always thought that an informatician sat at a computer and wrote the program which is hard to be understood anyone else. This, however, is a nice task suitable for anyone. I am thinking, we do know one solution, but how many solutions are there? Have you thought about it? I have."

This is how the cooperation between the mathematician and informatician started. Let's take a look at what can be discovered – both from the mathematician and the informatician's point of view.

Note: This is where the mathematician and the informatician's cooperation on the task regarding cars leaving a garage begins. Their dialogue and sharing of results not only enhances and develops the simple task but also takes it into unexpected directions. We believe that they, too, benefit from the cooperation. At the same time, we would like to ask the readers to look at the task from school mathematics and informatics' point of view. In science circles, their opinions would be more critical, and they would probably use different methods. We intend to show how a pupil-mathematician and pupil-informatician can cooperate in primary and secondary school, respectively, how they approach research, and what methods they use.

Task 2:

There are cars of different colours in a garage. How many different leaving orders are there?

Mathematician: As far as mathematics is concerned, this is a combinatorial problem – the different colours of the cars make this task a simple. The mathematician will start by defining the term "leaving order" i.e. the order in which the cars leave the garage based on their colour. The rightmost element represents the first car that left the garage. We are interested in the overall number of leaving orders.

Furthermore, we will give a definition the term "top car", i.e. the car closest to the top exit, and it can use this exit to leave the garage. The term "bottom car" can be defined in a similar manner.

There is only one leaving order for one car in the garage – it is not important whether the car leaves the garage through the top or the bottom exit. There are two different leaving orders for two cars in the garage – the top car will leave first, followed by the bottom car, or the bottom car will leave first, followed by the top car.

The following consideration applies to n of coloured cars ($n \geq 1$): At the beginning, there are two different options to leave the garage – either the top or the bottom car can first drive out of the garage. There will be left $n-1$ cars in the garage. Again, there are two different options to leave the garage. We can continue like this until there will be only one car left in the garage. As far as the options to leave the garage is concerned, it is not important which exit the last car will use.

Statement 1:

There are 2^{n-1} leaving orders for n cars of different colours ($n \geq 1$).

The proof of Statement 1 can be found in the above consideration.

Therefore, there are $2^{6-1} = 2^5 = 32$ different leaving orders for six cars of different colours.

Informatician: The informatician will approach the problem differently. They can use the mathematician's knowledge and write a program which will generate the result 2^{n-1} for the preset ($n \geq 1$).

If the informatician wants more information, he will use a suitable program to generate the coloured leaving order for the cars in the garage. In order to do so, they need to define the data structure of the program. The cars in the garage will be represented by the string of letters `ord = 'ABCDEF'` (the length of the string corresponds to the number of cars in the garage) where A (`ord[0]` – the first element of the `ord` string) represents the top (yellow) car, B represents the purple car … and F (`ord[-1]` – the last element of the `ord` string) represents the bottom (pink) car. The process of the car leaving the garage is represented by removing the one character either from the left or the right side of the `ord` string – if the top car leaves, the A character is removed; if the bottom car leaves, the F character is removed. The currently generated order is marked `actual`; the generated orders will be saved into the `result` array.

The possible leaving orders can also be generated using a backtracking algorithm (Wirth, 1975). If the yellow car leaves the garage first, we write it to the `actual` order and recursively search all the possibilities to leave the garage for the remaining cars. Then we return the yellow car to the garage and make the pink car leave the garage first through the bottom exit. Again, we write it to the `actual` order and recursively search all the possibilities to leave the garage for the remaining cars. Algorithm 1 describes the proposed procedure. The variable `car` represents the car that is currently driving out of the garage through the top or bottom exit (top car or bottom car).

```
def generate_order(ord, actual):
    if garage is not empty:
        for car in (top car, bottom car):
            let car leave garage, modify ord
            write car to actual
            generate_order (ord, actual)
            if garage is empty:
                write actual to result
            return car to garage
```

(*Algorithm 1*)

Based on the above algorithm, a program can be created, and the correctness of the algorithm can be proved. It is obvious that each leaving order is included twice in the `result` array. The last car and it's driving out poses a problem as it is not important whether it leaves through the top or the bottom exit. The proposed algorithm requires a simple modification of condition (**if** the garage **is** empty): The current leaving order will only be saved into the `result` array if `car` leaves the garage through the top exit, i.e., if `car` is the top car. See the modified condition:

if garage **is** empty **and car is** top car:
write **actual** to **result**

A program in Python language based on the above algorithm may look as follows:

```
result = []

def generate_order(ord, actual):
    if len(ord) > 0:
        actual = actual + ord[0]
        ord = ord[1:]
        generate_order(ord, actual)
        if len(ord) == 0:
            result.append(actual)
        ord = actual[-1] + ord
        actual = actual[:-1]
        actual = actual + ord[-1]
        ord = ord[:-1]
        generate_order(ord, actual)

ord = 'ABCDEF'
generate_order(ord, '')
print(result)
```

How will the informatician continue to think? If it is clear that the algorithm is correct, the informatician becomes interested in its effectiveness. Since it is a recursive algorithm, its effectiveness is related to the number of times the `generate_order` recursive function call. The function is called for the first time for the input string `ord` of the *n* length. The function is called twice in each of the following recursion levels – for the top car and the bottom car until there is no car left in the garage. The depth of the nested recursive function calls corresponds to the *n* length of the `ord` string. These considerations can be used to formulate Statement 2.

Statement 2:

The number of times the `generate_order` function calls itself for the `ord` string of the n length corresponds to the maximum number of nodes in a complete binary tree with the depth n, i.e., the `generate_order` function is called $2^{n+1} - 1$ times.

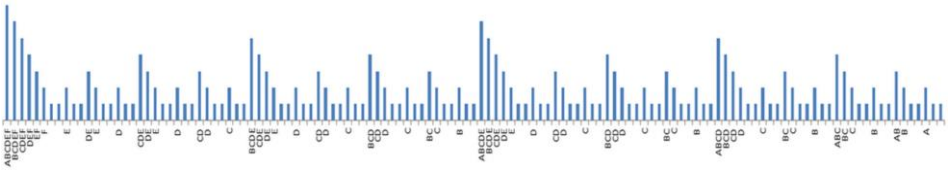

Fig. 2. Sequence of the `generate_order` function calls for six cars in the garage.

The number of times the function calls itself for the six cars in the garage is $2^{6+1} - 1 = 2^7 - 1 = 128 - 1 = 127$. The sequence of the `generate_order` function calls for the input `ord = 'ABCDEF'` string can be seen in Fig. 2 (all 127 function calls are included). The cars' current order in the garage can be seen in the bottom part of the figure. The sequence of function calls corresponds to the binary tree traversal method – preorder traversal. Moreover, the figure also shows that half of the `generate_order` function calls are realized for the `ord` string of the 0 length, i.e. for an empty garage.

The above algorithm analysis and calculation process is a typical informatician approach. When solving a problem, the mathematician works with algorithms, creates them, verifies their logical correctness, and considers their application possibilities. Their requirements and space and time complexity are not important to the mathematician.

After a while, the mathematician realizes that the number of leaving orders is related to the colours of the cars, to the different colours. As a result, the mathematician begins to look at the problem from a different angle.

Task 3:

How many leaving orders are there if we admit that cars in the garage are not different colours, i.e., in the garage at the beginning there are groups of cars coloured the same colour?

Mathematician: How to find relationships for different colours of cars in the garage? The mathematician will focus on individual situations and try to find their solutions. If all the cars in the garage have the same colour, there is only one leaving order. Other possible situations can be seen in Fig. 3. Both variants work with two colours – yellow cars (hereinafter marked with the letter A) and pink cars (hereinafter marked with the letter B).

- In Variant I (left), the cars are parked by colour alternately – yellow, pink, yellow, pink, and so on.
- In Variant II (right), the cars of one colour group are parked on top, and the cars of the other colour group are parked beneath them. There does not need to be the same number of cars of a particular colour – there can be, for example, two yellow cars and four pink cars beneath them or only one yellow car and five pink cars.

The solution to Variant I is based on the assumption that if the top yellow car drives out, the pink car must drive out in the next step, using either the top or the bottom exit.

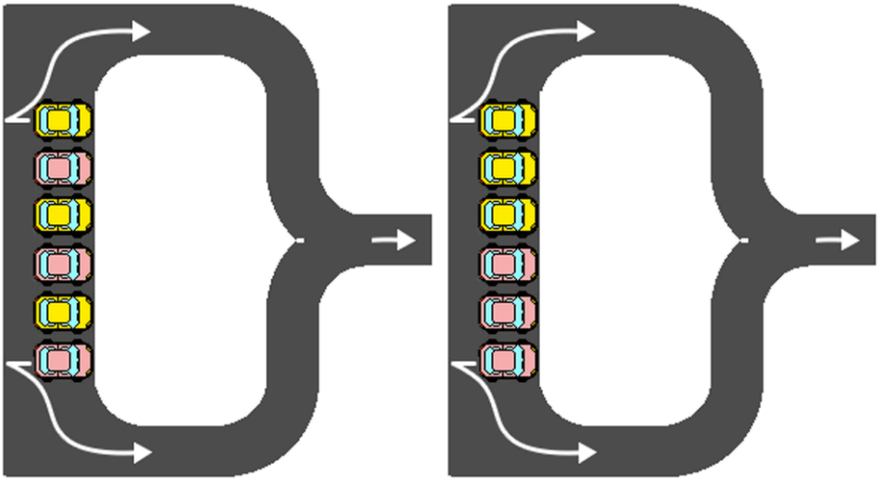

Fig. 3. Colour variations of the cars in the garage – Variant I (left), Variant II (right).

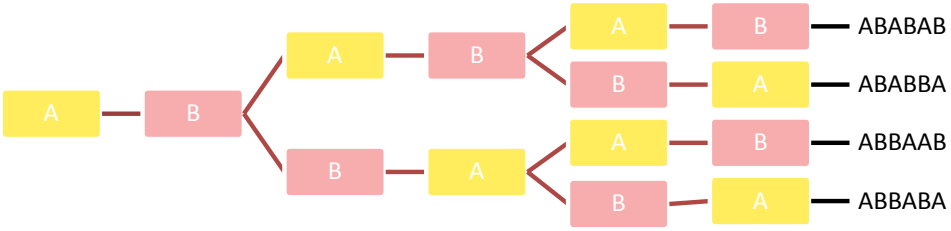

Fig. 4. Leaving orders for Variant I, the yellow car leaves first.

And then, either a yellow or a pink car may leave the garage as the third. However, this fact clearly determines the colour of the fourth car – if the third car was yellow, the fourth one must be pink, or vice versa. The same can be said about the fifth and sixth cars. The situation with the yellow car being the first one to leave the garage can be seen in Fig. 4. If the pink car leaves the garage first, the situation would be similar – symmetrically, there would be another four leaving orders (BABABA, BABAAB, BAABBA, and BAABAB). With six cars in the garage, there are eight different solutions to Variant I.

Statement 3:

Variant I can be defined as cars in a garage whose colours regularly alternate in the ABABAB pattern. In Variant I, the number of leaving orders for n of cars in the garage ($n \geq 1$) can be calculated using the $2^{\lfloor \frac{n}{2} \rfloor}$ relation, where $\lfloor a \rfloor$ represents the lower integer of the number a. In other words, $\left\lfloor \frac{n}{2} \right\rfloor$ corresponds to integer division of n by the number 2.

The proof of Statement 3 uses the fact that the colour of every second car is unequivocally determined by the previous car leaving the garage. If n is odd, the colour of the first car in the leaving order is unequivocally determined.

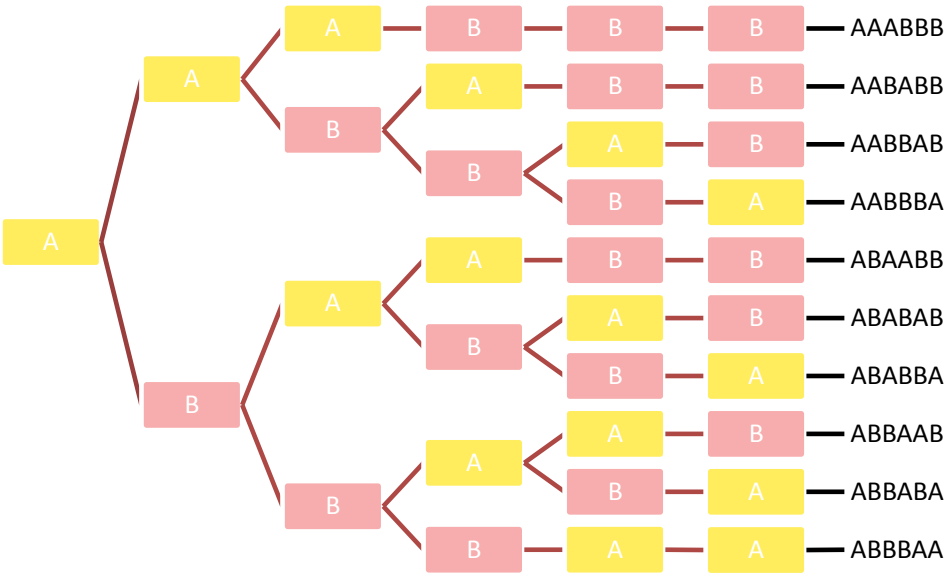

Fig. 5. Leaving orders for Variant II, the yellow car leaves first.

In Variant II, the number of leaving orders is higher – there are as many as 20 leaving orders for the six cars in the garage in Fig. 3. Fig. 5 shows 10 leaving orders for six cars, assuming that a yellow car left the garage first. Symmetrically, a pink car leaving the garage first would result in another 10 leaving orders.

Examining other leaving orders can help the mathematician reveal other important regularities applicable not only when solving a problem, but also at a more general level. Since the mathematician examines particular situations (Variants I & II) in detail, they can find regularities valid only for those situations. At a general level, they learn that if leaves of a complete binary tree of depth n are the leaving orders for `n` ($n \geq 1$) cars of different colours, the binary tree representing leaving orders for cars with colours that allow repetition (there are groups of cars of the same colour in the garage) is considerably pruned. In Variant I, the tree is pruned at the second level – the colour of the second car in the leaving order is unequivocally determined by the colour of the car which left the garage first. In Variant II, there is no pruning until the Level 4 – in case the first three cars in the leaving order were of the same colour. This simple consideration proves that there are fewer leaving orders in Variant I than in Variant II.

Informatician: The input `ord` string of *Algorithm 1* contains information about the cars' colour, which is encoded in the individual letters – A, B, C, D, and so on. If we allow the cars in the garage to be of different colours, the values in the `ord` string can be changed. In Variant I, `ord = 'ABABAB'`; in Variant II, `ord = 'AAABBB'`. *Algorithm 1* requires a simple modification: When writing the current leaving order `actual` into the `result` array, one needs to make sure that the `actual` order is not already in the `result` array, i.e., it has not already been generated by the cars leaving the garage in a different order – see *Algorithm 2*.

```
def generate_order (ord, actual):
        if garage is not empty:
                for car in (top car, bottom car):
                        let car leave garage, modify ord
                        write car to actual                    (Algorithm 2)
                        generate_order (ord, actual)
                        if garage is empty and car is top car:
                                if actual is not in result:
                                        write actual to result
                        return car to garage
```

It is not difficult to write a program based on the above algorithm. Compared to the original program, the only function that needs to be added is checking the uniqueness of the individual leaving orders in the `result` array, which corresponds to element search in this array. This problem can be solved using the informatics resources.

The optimization of the algorithm with regard to the minimization of the recursive `generate_order` function call is more demanding, with respect to traversing through the complete binary tree of leaving orders. However, the mathematician finds out that it is not necessary to traverse through the complete binary tree. The presented optimization requires a fundamental modification, which will not be included here.

Using a computer which gives results quickly, the informatician can deal with Task 3 and its variants I and II differently.

As far as Variant I is concerned, in particular cases, the number of leaving orders can easily be verified. As we have seen, there are eight leaving orders for the input strings ‘ABABAB’. The same can be said for the input string ‘ABABABA’. However, there are twice as many (16) leaving orders for the input string ‘ABABABAB’.

Table 1

Number of leaving orders for Variant II

0 ‘’						
1 ‘A‘	1 ‘B‘					
1 ‘AA‘	2 ‘AB‘	1 ‘BB‘				
1 ‘AAA‘	3 ‘AAB‘	3 ‘ABB‘	1 ‘BBB‘			
1 ‘AAAA‘	4 ‘AAAB‘	6 ‘AABB‘	4 ‘ABBB‘	1 ‘BBBB‘		
1 ‘AAAAA‘	5 ‘AAAAB‘	10 ‘AAABB‘	10 ‘AABBB‘	5 ‘ABBBB‘	1 ‘BBBBB‘	
1 ‘AAAAAA‘	6 ‘AAAAAB‘	15 ‘AAAABB‘	20 ‘AAABBB‘	15 ‘AABBBB‘	6 ‘ABBBBB‘	1 ‘BBBBBB‘
1 ‘AAAAAAA‘	7 ‘AAAAAAB‘	21 ‘AAAAABB‘	35 ‘AAAABBB‘	35 ‘AAABBBB‘	21 ‘AABBBBB‘	7 ‘ABBBBBB‘	1 ‘BBBBBBB‘

Examining the number of leaving orders in Variant II may lead to interesting findings. There is no leaving order for an empty input string. There is only one leaving order when one car is in the garage. Similarly, only one leaving order exists if all the cars parked in the garage are of the same colour. If the cars in the garage are of two different colours, the number of leaving orders differs with respect to the number of cars and the ratio cars of one colour to the other. Informatik inserted data into Table 1. The top row of each table cell represents the number of leaving orders while the bottom row represents the value of the `ord` input string.

The result is the well-known Pascal's triangle. The proof is simple, based on the Pascal's triangle's definition. The program is allowed to obtain this result, because the program calculates the results for a variety of input strings in a short period.

The task can be further developed.

Task 4:

We install a turnstile at the top exit, which would let exactly two cars go through at once. This device would influence the process of finding the leaving orders. How many leaving orders are in this special garage? All the cars in the garage are of different colours.

The informatician will begin to modify their algorithm and find that it is not difficult – two cars must leave through the top exit at the same time; as far as the bottom exit is concerned, the algorithm will remain unchanged. The modification will show in the cycle header – for the top exit, two elements of the `ord` input string must be selected at once:

for **car** in (two top cars, bottom car):

The informatician will have both the correct solution and the number of leaving orders immediately.

The mathematician, on the other hand, must again approach the problem by analysing particular situations.

- If there are two cars in the garage, they can leave at once using the top exit, or they can leave one after another using the bottom exit. There are two solutions for $n = 2$.
- The options for the situation with three cars in the garage are as follows: The first two cars leave through the top exit, and the third car leaves through the bottom exit; or one car leaves through the bottom exit and the remaining two cars leave through the top exit; or all three cars leave through the bottom exit. Therefore, there are three solutions for $n = 3$.
- Similarly, it can be proved that there are five solutions for $n = 4$.

After solving the problem, the mathematician will not only create the Fibonacci sequence (Nagyová, 2016), but they can also discover rules for constructing all the solutions for a given number *n* of cars in the garage if the solutions for $n - 1$ and $n - 2$ are known.

Discussion

Mathematics and informatics instruction is based on good mathematical/informatics problems and the process of their solution. Based on this claim, it might seem that not only mathematics and informatics instructions are very similar, but also both the disciplines are very close to each other in their approaches. The presented fictional interaction points to the different approaches that the mathematician and informatician would probably take.

The questions asked by the mathematician are different from those asked by the informatician. The mathematician tries to look at the problem from different angles, by asking the following questions: If one solution is already known, are there more solutions? If there is a solution for cars of one colour, can solutions for different colour distribution also be found? What is the relation between the number of solutions for n, $n-1$ and $n-2$ of cars in the garage? The mathematician tries to find relations between phenomena, draws conclusions, and proves the statements.

On the other hand, the informatician approaches the problem from a data representation and data structure perspective. They try to find the best way to organize the data so that an algorithm for a given data structure would be simple and feasible in real or minimum time. The informatician always wants the created algorithm to be correct. At the same time, they monitor the algorithm regarding the amount of time required and memory used.

The mathematician focuses on finding solutions to particular situations: a particular number of cars, a particular colour of cars, and so on. They try to analyse those problems in detail and find not only their solutions but also regularities. As far as didactics of mathematics is concerned, this is the recommended process – the process of building mathematical knowledge can be divided into the following levels: motivation, isolated models, generalization and generic model, abstraction and abstract knowledge, and crystallization (Hejný, 2012).

On the other hand, the informatician designs the data structure and algorithm so that it would be able to solve the entire group of similar problems. From the very beginning, they disregard particular situations and try to look at the problem comprehensively. Unlike the informatician, the mathematician has in-depth knowledge of particular isolated situations. The informatician disregards the individual problems and often does not even know solutions to them. This may result in errors in the informatician's solution of problems. Moreover, detecting such errors may be extremely complicated for them. These errors are often logical errors in the algorithm which constructs the result.

On the other hand, the informatician's approach to problem-solving makes it possible to solve problems which the mathematician cannot solve in their complexness (they are able to solve only particular cases). Acquiring and processing large amounts of data in a short period enables the informatician to make conclusions, giving them an advantage over the mathematician who would need much more time (e.g., discovering the Pascal's triangle in Task 3, Variant II).

Conclusion

Both mathematics and informatics deal with problem-solving. The presented differences in approach are also manifested in their didactics and approaches to instruction.

The mathematician works with particular situations and tries to find relations between them. One approach, in constructivist mathematics instruction, is of building mathematical knowledge through separated models which students can understand and find relations between them. The discovered relations help students generalize the context and create generic models. General abstraction leads to crystallization of mathematical knowledge (Hejný, 2012).

The informatician analyses the problem to create a data structure and subsequent specification of the problem-solving process (algorithm), which they then translate into a programming language. The resulting program can solve not only the assigned task but also a group of similar problems. As it can be seen in the informatics instruction model (Schubert and Schwill, 2011), it uses the same approach. By analysing a real-life problem, the students learn to specify an informatics problem with specified input parameters and specified outputs. Then they make a problem-solving plan (algorithm), which they practically implement and verify both the correctness of the results and the effectiveness and quality of the computation.

The mathematician deals in detail with a particular problem, while the informatician focuses on the problem-solving techniques. In both cases, it is the problem-solving process, not the result that is important – regardless of whether it is the process of solving one particular problem (which becomes a separate model) or the process of creating and implementing a particular algorithm which solves a group of problems. Both the fields are closely interconnected and have the potential to influence each other. Like any other field of human activity, mathematics cannot make do without informatics. Informatics and its approach to problem-solving are based on mathematics and uses a number of its concepts.

Mathematical and informatics problems can be found everywhere. Both the mathematician and the informatician need to be able to look for them and ask the right questions. If they are able to do so, their dialogue and cooperation may lead to solutions (both mathematical and informatics), which can then lead to unexpected discoveries (recursion, the Pascal's triangle, the Fibonacci sequence, and so on.). This process of discovering can be observed in how the original trivial task (Task 1) changed during the dialogue. The potential and motivational nature of those "unexpected contexts" that are presented to students is enormous in its consequences.

References

Halmos, P.R. (1980) The heart of mathematics. *The American Mathematical Monthly*, 87(7), 519–524.

Hejný, M. (2012). Exploring the cognitive dimension of teaching mathematics through scheme-oriented approach to education. *Orbis Scholae*, 6(2), 41–55.
http://www.orbisscholae.cz/archiv/2012/2012_2_03.pdf

IBobor – Súťaž: Archív Úloh 2009/2010. (2009). Benjamin. http://ibobor.sk/sutaz_demo/
Kuřina, F. (2011). *Matematika a Řešení Úloh*. University of South Bohemia, České Budějovice.
Kuřina, F. (2008). Může být školská matematika matematikou dobrou? *Pokroky Matematiky, Fyziky a Astronomie*, 53 (4), 322–335.
Modeste, S. (2016). Impact of informatics on mathematics and its teaching. In: Gadducci F., Tavosanis M. (Eds), *History and Philosophy of Computing. HaPoS 2015*. Springer, Cham, 243–255.
Nagyová, I. (2016). Konstruktivismus v didaktické přípravě učitelů. *Didinfo 2016*. UMB, Banská Bystrica. http://didinfo.umb.sk/public/filestore/documents/richtext/202/zbornik_2016_final.pdf
Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books, Inc.
Schubert, S., Schwill, A. (2004). *Didaktik der Informatik*. Heidelberg: Spektrum.
Vygotsky, L.S. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, Cambridge.
Wolfram, C. (2010). *Teaching Kids Real Math with Computers*. New York: TED Conferences, LLC. https://www.ted.com/talks/conrad_wolfram_teaching_kids_real_math_with_computers
Wirth, N. (1975). *Algorithms + Data Stuctures = Programs*. New Jersey: Prentice-Hall.

I. Nagyová obtained PhD degree in the field of informatics education in 2009 at Constantine the Philosopher University in Nitra (Slovakia). She has been working as a researcher and teacher at the Faculty of Education of the University of Ostrava, Czech Republic since 2003. She received her master degree in mathematics. She was working as a programmer for ten years and involving in the development and processing of psychodiagnostic tests. Her main research interest is the constructivism and constructivist teaching. She is focusing on algorithms and creative programming and deals with programming instruction. She tries to create and search examples of correct implementation of constructivism in teaching and apply them in teaching algorithms and programming.