Development of Algorithmic Thinking and Imagination: base of programming skills

EVA MILKOVÁ
Department of Informatics and Quantitative Methods
Faculty of Informatics and Management
University of Hradec Králové
Rokitanského 62
CZECH REPUBLIC
eva.milkova@uhk.cz

Abstract: This paper is based on rich experience gained in the area of computer science education and it could serve as an inspirational material directed to all educators developing students’ programming skills. The area of software development has undergone a rapid expansion and this trend is so far continuing. Each developer has to learn constantly and master new technology. However, the foundation a developer gains at the beginning of his/her career plays a crucial role. An essential part of studies at faculties preparing students in the area of computer science is the development of student’s ability to think algorithmically. Students must be able to create various algorithms solving given problems starting with easy ones and consecutively increase their algorithmic knowledge and shifts during studies till the level where they deeply understand much more complex algorithms. The aim of this paper is to introduce our approach that has proven to be successful in the optimization of teaching and learning a subject developing algorithmic thinking of beginners. This is followed by a discussion of the benefits of puzzles, solved within subjects, dealing with graph algorithms and enabling development of students’ logical thinking and imagination, i.e. skills needed for deeper understanding more complex algorithms.

Key-Words: Computer science education, algorithms, algorithmic thinking, graph theory and algorithm, puzzles

1 Introduction
An essential part of studies at faculties preparing students in the area of computer science is the development of student’s ability to think algorithmically. Students must be able to create various algorithms solving given problems starting with easy ones and consecutively increase their algorithmic knowledge and shifts during studies till the level where they deeply understand much more complex algorithms.

Education at secondary schools and colleges in the area of informatics is directed mainly to a user attitude in the Czech Republic. Only students attending optional subjects dealing with programming languages are familiar with creating algorithms. Thus a lot of students coming to universities are without any algorithmic knowledge at the beginning of their studies.

There are many different theoretical researches which deal with the question of how to consequently develop algorithmic thinking of students. Their basic aim is to improve the quality of teaching and students’ self-learning.

In this paper, as an inspiration, we introduce at first our approach to the development of algorithmic thinking of beginners within the subject Algorithms and Data Structures.

It is followed by a discussion concerning further development of algorithmic thinking by our students within subject Graph Theory and Combinatorial Optimization, where more complex algorithms on graphs have been explained. The principles that we apply in our teaching will be introduced as well as three puzzles developing students’ logical thinking and imagination. The puzzles of different level of difficulty presented herein prove both, the students’ ability to find out the appropriate graph-representation of given task and solve it using appropriate algorithm.

2 Development of Algorithmic Thinking
University departments that train students in computer-related disciplines still mostly teach the algorithm design jointly with teaching a certain programming language. Former textbooks such as
[4], [14] which were used at the Czech universities in the past dealt with structured programming languages. On the contrary, the recent trend is directed mostly towards object oriented languages, see e.g. [1], [12], [13]. However, at conferences there have been long discussions regarding what kind of programming is suitable for beginners. Protagonists of object oriented languages argue that students beginning with structured programming acquire habits that cause big problems for them when using object oriented languages.

This was the main reason why we decided at our faculty, more than ten years ago, to build the subject Algorithms and Data Structures and place it into our curricula before other subjects which deal with algorithmic and programming skills. The approach used in this subject lacks all the structures and constructs from the structured paradigm. Therefore the students do not acquire any habits which might complicate their entrance into the object oriented world.

Thanks to the fact, that our approach has proven to be successful in the optimization of teaching and learning a subject developing algorithmic thinking of beginners we have already introduced it in several papers (see e.g. [5], [7], [10]). We have mostly concentrated on the multimedia support used in this approach or on types of tests assessing students’ knowledge gained in the subject. Here let us remind the main approach to the subject and control of lessons when teaching the subject Algorithms and Data Structures. (c.f. [3], [5])

2.1 The subject Algorithms and Data Structures

One can imagine a brick-box, a nice and useful game for children. There are only several base elements available from which children are able to create incredible buildings. Our approach to the creation of algorithms is based exactly on this image.

When we lead our students’ first steps in the creation of algorithms we explain to them that it is like building interesting objects out of just a few basic elements. In the subject Algorithms and Data Structures it means that we start our teaching with basic algorithmic structures (basic elements from the brick-box) and typical algorithmic structures (a few parts made out of these elements) written in Czech meta-language and then we let students get into the secrets of making whole algorithms (building whole constructions) written in Czech meta-language as well.

Remark: The used Czech meta-language is nothing more than the Pascal programming language basic commands. We decided for this way of algorithms’ description because Pascal programming language was created by Nicklaus Wirth especially for educational purposes.

Thus as basic algorithmic structures are blocks of commands, incomplete and complete branching and loop construction are considered. As typical algorithmic structures, i.e. common structures typical for the whole group of problems, we consider e.g. determination of the number of elements with the given property, determination of the sum and the product of the elements, finding the first and last element with the given property, determination of the maximum and minimum of all elements or of elements with the given property.

At the lectures we explain all the structures of algorithms, at first only those which use single variables. We always try to use names of variables that describe their use. Obviously, in the beginning examples of algorithms are demonstrated graphically by developing diagrams. In the diagrams we use two types of shapes only: a rectangular for commands and a rhombus for conditions. The action of each algorithm is illustrated by a step-by-step procedure for suitable initial values.

After a thorough exercise of basic algorithms on problems using single variables (above all those dealing with unknown number of values, because these tasks often trouble the students) we proceed and explain the data structure one-dimensional array and later two-dimensional array as well.

During lessons students apply the acquired knowledge to a variety of tasks. After some time when students have prepared their solutions on paper, each task is illustrated by two or three students at the blackboard and their solutions are compared and discussed by all students. On the one hand this means that students are led to try to find more solutions to the given task and to be able to understand the efficiency of algorithms as well. On the other hand when incorrect solutions occur among the presented solutions the teacher has an opportunity to discuss with students where the problem is. Moreover, using the program ALGORITHMS mistakes in incorrect algorithms can be emphasized on suitable entrance dates together with the values of used variables.

Remark: The program ALGORITHMS is created in the Delphi environment. Using the program, students can place their solution of the given task written in Czech meta-language into the program and the program shows them step-by-step how their algorithm works and if it is correct or not. The
program also shows the actual values of used variables in each step of the algorithm’s process.

At our lessons we solve not only the whole tasks but we also let students complete prepared algorithms.

The whole area explained within the subject Algorithms and data Structure is introduced in the textbook [6], where more than 150 problem assignments, questions and exercises are presented. The accuracy of a solution can be verified with the help of the program ALGORITHMS which is enclosed, together with solutions of all the textbook’s given tasks, on the CD attached to the textbook.

3 Development of Imagination
After gaining deep insight into the creation of basic algorithmic constructions in the subject Algorithms and Data Structure and practicing the acquired knowledge within subjects dealing with programming languages, students’ logical and algorithmic thinking is deepened in the subject Graph Theory and Combinatorial Optimization. The aim of the subject is not only to develop and deepen students’ capacity for logical and algorithmic thinking, but also to develop student’s imagination. Well-prepared students should be able to describe various practical situations with the aid of graphs, solve the given problem expressed by the graph, and translate the solution back into the initial situation.

Here let us draw attention to the development of students’ algorithmic thinking and imagination using puzzles. We introduce three puzzles and their solutions based on the Breadth-First-Search Tree property.

3.1 Breadth-First Search Tree
Before we start to deal with puzzles let us remind the well-known Breadth-First-Search algorithm. We describe it using Czech meta-language (see the section 2.1) and as an edge colouring process. We also introduce the definition of the Breadth-First-Search Tree, its property and theorems following from it. (c.f. [9])

begin
choose the first vertex \( x \) in FIFO;
if there is an uncoloured edge \( \{x,y\} \) then
if the vertex \( y \) is uncoloured then
begin
search and colour blue both the vertex \( y \) and the edge \( \{x,y\} \);
insert the vertex \( y \) into FIFO;
end
else
search and colour the edge \( \{x,y\} \) red
else
delete the vertex \( x \) from FIFO;
end;
end.

Applying the Breadth-First-Search it is evident that the blue coloured edges form a spanning tree \( T \).

Definition
Let \( G \) be a connected undirected graph, let \( v \) be a vertex of \( G \), and let \( T \) be its spanning tree gained by the Breadth-First-Search of \( G \) with the initial vertex \( v \). An appropriate rooted tree \( (T,v) \) let us call a Breadth-First-Search Tree (BFS Tree shortly) with the root \( v \), the edges of \( G \) that do not appear in BFS Tree let us call non-tree edges and the components of the forest \( T' = (T,v) - v \) let us call \( (T,v) \)-subtrees.

Theorem
Let \( G \) be a connected undirected graph, let \( v \) be a vertex of \( G \), and let \((T,v)\) be a BFS Tree with the root \( v \). Then the end-vertices of each non-tree edge of \( G \) belong either to the same level or to the adjacent levels of \((T,v)\).

Statement 1
Let \( G \) be a connected undirected graph, let \( v \) be a vertex of \( G \), and let \((T,v)\) be a BFS Tree with the root \( v \). Then the length of the shortest path from the vertex \( v \) to a vertex \( y \) in \( G \) is equal \( h(y) \).

Statement 2
Let \( G \) be a connected undirected graph, let \( v \) be a vertex of \( G \), and let \((T,v)\) be a BFS Tree with the root \( v \). There is a circle of odd length in \( G \) if and only if there is a non-tree edge having both end-vertices in the same level of \((T,v)\).

Statement 3
Let \( G \) be a connected undirected graph, let \( v \) be a vertex of \( G \), and let \((T,v)\) be a BFS Tree with the root \( v \). There is a circle containing the vertex \( v \) in \( G \) if and only if there is a non-tree edge having its end-vertices in different \((T,v)\)-subtrees.

Statement 4
Let \( G \) be a connected undirected graph, let \( v \) be a vertex of \( G \), and let \((T,v)\) be a BFS Tree with the
root \(v\). There is a circle containing an edge \(\{v, w\}\) in \(G\) if and only if there is a non-tree edge having one end-vertex in the \((T, v)\)-subtree with the root \(w\) and the other end-vertex in another \((T, v)\)-subtree.

Moreover, using BFS and the property of BFS Tree we are also able to formulate the other statements, obvious necessary conditions as e.g.

Statement 5

Let \(G\) be a connected undirected graph, let \(v\) be a vertex of \(G\), and let \((T, v)\) be a BFS Tree with the root \(v\). If there is a non-tree edge having its end-vertices at once in the same level of \((T, v)\) and in different \((T, v)\)-subtrees, then there is an odd circle containing the given vertex \(v\).

Remark: The contrary statement if there is an odd circle containing the given vertex \(v\) in the given graph \(G\), then there is a non-tree edge having its end-vertices at once in the same level of \((T, v)\) and in different \((T, v)\)-subtrees, need not be true (see the following section).

3.2 Puzzles

Our approach to the development of algorithmic thinking and imagination within the subject Graph Theory and Combinatorial Optimization can be characterized by the following basic principles that we apply in our teaching (see [8]).

- When starting an explanation of new subject matter, a particular problem with a real life example or puzzle is introduced as a motivation and suitable graph-representation of a problem is discussed.
- If possible, each concept and problem is examined from more than one point of view and various approaches to the given problem solution are discussed with respect to the already explained subject matter.
- In addition to words visualization of the particular issue as well as it is possible is done.
- The explained topic is thoroughly practiced and students’ own examples describing the topic are discussed.
- Using the constructed knowledge and suitable modification of the problem solution, we proceed to new subject matter.

In this paragraph let us introduce three puzzles of different difficulties. Each of them can be successfully solved using algorithms based on BFS Tree property. A level of complexity to find out the appropriate graph-representation of each puzzle will be obvious.

**Puzzle 1**

Let us have a look at the Fig. 1. There are two types of cells (fields): white and black. The task is to find a way to move from the point \(S\) (Start) to the point \(P\) (Post) using the smallest number of steps possible keeping the following rules:

- One step means to go on one cell.
- Go either horizontally or vertically.
- Do not enter nor go through black cells.

![Fig. 1 Picture to the given puzzle 1](image1)

A graph representation to the task (see Fig. 3) can be easily done in the following way. Let us complete the Fig. 1 by numbers and letters to get Fig. 2. Then each cell is represented by the vertex \(P_c\), where \(P \in \{A, B, C, D\}\) and \(c \in \{1, 2, 3\}\) and an edge is between each pair of vertices where the step defined by the above rules is possible.

![Fig. 2 Fig. 1 completed by numbers and letters](image2)

**Puzzle 2**

Let us have a look at the Fig. 4. There are three types of cells (fields): white, black and circle. The task is to find a way how to move from the point \(S\) to the point \(C\) using the smallest number of steps as possible keeping the following rules:

- One step means to go on two (by the speed 2) or three (by the speed 3) cells.
- Go either horizontally or vertically.

![Fig. 3 Graph representation to the puzzle 1](image3)
On S your speed is 2. As soon as you enter a circle, change the speed to 3 and as soon as you enter another circle, change the speed to 2 etc. Do not enter nor go through black cells. (Note: You can enter the same cells more time.)

In this case a graph-representation of the task is not obvious immediately. It can be done in mind (the whole graph would be too large) in the following way: Let us complete the picture on Fig. 4 by numbers and letters in the same way as in the puzzle 1 and imagine that each cell is represented either by the vertex $\text{Pc}^2$, or by the vertex $\text{Pc}^3$, where $\text{P} \in \{A, B, C, D, E, F\}$ and $c \in \{1, 2, ..., 8\}$. The upper index determines the used speed.

In this way a directed graph $G$ is obtained. Its vertices are $\text{Pc}^i$, $\text{P} \in \{A, B, C, D, E, F\}$, $c \in \{1, 2, ..., 8\}$, $i \in \{2, 3\}$, and there is the directed edge from the vertex $\text{Xyz}$ to the vertex $\text{Uvw}$ in the graph $G$ if and only if there exists a step from the vertex $\text{Xyz}$ to the vertex $\text{Uvw}$ defined by the above rules.

Solution to the puzzles 1 and puzzle 2

Both tasks can be solved using the Breadth-First Search with regard to the Statement 1.

A BFS Tree appropriate to the puzzle 1, i.e. a BFS Tree appropriate to the Breadth-First-Search of the graph on Fig. 3 starting in the vertex $A3$ and a solution, i.e. the shortest path from the root $A3$ to the vertex $D1$ in the gained BFS tree, is obvious.

Here let us illustrate on Fig. 5 the needed part of the BFS tree appropriate to the puzzle 2 from which a solution, the shortest path from the vertex $A8^2$ (the cell S) to the vertex $F1^3$ (the cell C, which can be achieved either as the vertex $F1^3$ or as the vertex $F1^2$), is perceivable.

Puzzle 3

In the graph on Fig. 6 find the shortest path from the vertex B to the vertex K containing odd numbers of vertices.

Fig. 4 Picture to the given puzzle 2

Fig. 5 BFS Tree to the puzzle 2

Fig. 6 The given graph appropriate to puzzle 3
This puzzle is valuable to introduce thanks to the fact that all five above mentioned statements can be emphasized on its solution (we will present it at the conference). Namely, because the shortest path between vertices B and K is even, the path (B, 29, 30, K), we are looking for the shortest odd circle containing both vertices in a BFS Tree with the root in one of the two given vertices (either B or K). This puzzle can serve as useful example confirming that the contrary statement to the statement 5 need not be true.

4 Conclusion
In this paper we introduced our approach to the development of algorithmic thinking of beginners within the subject Algorithms and Data Structures and then three puzzles of different level of difficulty contributing to the development of students’ algorithmic thinking and their imagination. The paper is intended as an inspiration for all educators developing students’ programming skills. Sometimes it is not easy to find an appropriate content and approach to teaching a subject and each inspiration could be valuable to know and possibly to apply.

References: