# Algorithmic and Logical Thinking Development: base of programming skills

EVA MILKOVÁ
ANNA HŮLKOVÁ
Department of Informatics
Faculty of Science
University of Hradec Králové
Rokitanského 62
CZECH REPUBLIC
eva.milkova@uhk.cz, anna.hulkova@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' algorithmic thinking and programming skills. 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 and logical games, solved within subjects, dealing with graph algorithms and enabling further development of students' algorithmic thinking as well as logical thinking and imagination, i.e. skills needed for deeper understanding more complex algorithms.

*Key-Words:* Computer science education, logical games, multimedia applications, 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*. (Remark: Thanks to the fact, that our approach has proven to be successful, we have already introduced it at conferences, cf. [6], [8], [13]).

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 some puzzles and logical games developing students' logical thinking and imagination.

## 2 Algorithmic Thinking Development

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 [5], [18] 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. [2], [4], [14], and [16]. However, at conferences there have been still 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.

To avoid the mentioned possible problems, our approach that we have been using for many years in the subject *Algorithms and Data Structures* is based on an imagination of a brick-box, where only several base elements are available from which children are able to create incredible buildings, i.e. 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) and then we let students get into the secrets of making whole algorithms (building whole constructions).

### 2.1 Lectures and lessons

We do not use any programming language in the subject *Algorithms and Data Structures*, students write algorithms on paper in Czech meta-language. The used Czech meta-language is nothing more than the Pascal programming language basic commands. (Remark: We have decided for Pascal programming language because it was created by Nicklaus Wirth especially for *educational purposes*, see [18].)

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 vales*, 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.*

### 2.2 Self-study and the feedback

There is an important question. How can students get feedback for their solutions written on paper in the Czech meta-language when studying at home? There are a lot of tasks that we give our students to solve. They solve not only the whole tasks but we also let them complete prepared algorithms and determine values of variables similarly as you can see in the following examples.

*Example* 1 (single variables)

Complete the algorithm which calculates and writes out the following value of sum:

If $x \leq y$ then it is the sum of numbers $x, x + 1, \ldots, y$, and if $x > y$ then it is the sum of numbers $x, x - 1, \ldots, y$.

```
begin
   read(x);
   read(y);
   sum := .....;
   number := x;
   if x ..... y then
     for number := ..... to ..... do
        sum := sum + .........
   else
     while number ..... y do
     begin
        sum := sum + .........;
        number := number - 1;
     end;
   write("The sum of integers from
",x," to ",y," is equal to", sum,
".");
end.
```

*Example* 2 (one-dimensional array)

Complete the algorithm solving the following task. In the sequence of *n* integers saved in the array **a** (in items a[1] , ..., a[n]) determine the first minimum value and then sum all integers behind the found minimum value.

```
begin
   minimum := a[1];
   sum := ..........;
   for i from 2 to n do
   begin
     sum := sum + ..........;
     if a[i] ... min then
     begin
        minimum := ..........;
        sum := ..........;
     end;
   end;
end.
```

*Example 3* (single variables)

Determine what value will appear in variables *x*, y, and *z* after carrying out the following algorithm

```
begin
   x := 1;
   y := 3;
   z := x * y;
   while z < 8 do
     z := z + 2;
   if x > y then
     y := x + y;
end.
```

x = .....  y = .....  z = .....

*Example* 4 (one-dimensional array)

There are *n* integers saved in the array **a** (see the Table 1). Determine the values in the array **a** after finishing the following algorithm. Write them to the table.

```
begin
   n:=6;
   x:=a[1];
   i := 2;
   while i ≤ n - 1 do
   begin
     if a[i] > x then
     begin
        a[1]:= a[i];
        a[i]:= x;
     end;
     i := i + 1;
   end;
end.
```

| a[1] | a[2] | a[3] | a[4] | a[5] | a[6] |
|---|---|---|---|---|---|
| 11 | 8 | 19 | 7 | 16 | 17 |
| | | | | | |

Table 1 Table of integers saved in the array **a**

*Example* 5 (two-dimensional array)

Integers are saved in the two-dimensional array **a**. Determine the values in the array **a** after finishing the following algorithm. Write them to the table **Final position** (see the Table 2).

```
begin
  m := 4;
  for i := 1 to m do
     for j := 1 to m - i + 1 do
        a[i,j] := (i + j) mod 2;
end.
```

**Starting position**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 2 | 5 | 8 |
| 2 | 3 | 9 | 6 | 7 |
| 3 | 8 | 7 | 5 | 3 |
| 4 | 4 | 2 | 5 | 8 |

**Final position**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

Table 2 Table of integers saved in the array **a**

The answer to the question given at the beginning of this section is: Students can practise their knowledge using program ALGORITHMS developed by our student within his thesis [16] 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. In this way students can place prepared algorithms given in tasks (cf. examples 3-5 above) into the program and *see the final values of requested variables*.

Moreover, using the program on lessons *mistakes in incorrect algorithms can be emphasized on suitable entrance dates together with the values of used variables* (cf. text in the section 2.1 above: 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.).

The program ALGORITHMS is user friendly and its functions are arranged to be intuitive and at the

same time to remind professional editors and debuggers of well-known programming languages, which also facilitates the subsequent transition to them. Because many users are beginners the program is free of many unnecessary features which would rather complicate its use at this level.

With the ability to be localized into different languages the program can theoretically be used in the user´s own mother tongue, including the possibility to define own keys of used meta-language. [11]

The design of the program is shown on Fig. 1.

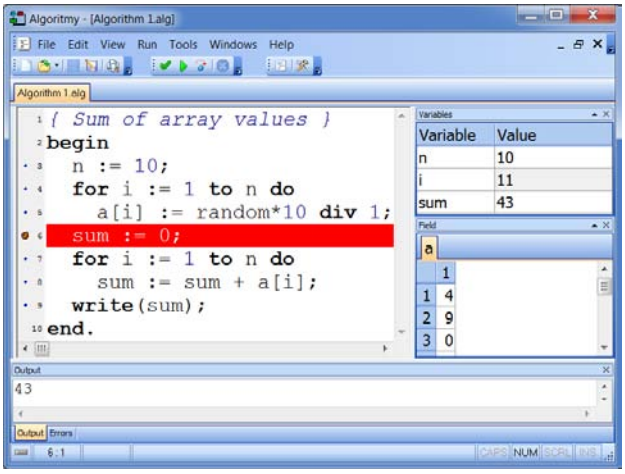


Fig. 1 Program ALGORITHMS - main window with an algorithm

## 3 Development of Logical Thinking

Logical thinking is an important foundation skill. Albrecht in his book [1] says that *the basis of all logical thinking is sequential thought. This process involves taking the important ideas, facts, and conclusions involved in a problem and arranging them in a chain-like progression that takes on a meaning in and of itself. To think logically is to think in steps.*

Let us add that *sequential thought can be enhanced through the development of algorithmic thinking* and that algorithmic thinking can be deeply enhanced in the subjects dealing with combinatorial optimization.

Thus after gaining deep insight into the creation of basic algorithmic constructions in the subject *Algorithms and Data Structure* and practising 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. [9]

Our approach can be characterized by the following basic principles that we apply in our teaching.

- 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 the following sections let us focus on the role of suitable puzzles and logical games used in education of the discussed subject at first.

On several puzzles of different level of difficulties we discuss a possibility how to enhance the students' ability to find out the appropriate graph-representation of given task (i.e. how to develop their *logical thinking* and *imagination*) and together solve it using appropriate algorithm (i.e. how to develop their *algorithmic thinking*). Using logical games we can also practise and discuss various topics in an enjoyable way. Two logical games will be introduced.

The section ends with brief description of a multimedia program that is not only a substantial help to students in their self-study but it also helps teacher explain all needed concepts and the process of particular algorithms on lectures and seminars.

### 3.1 Puzzles

In this section we introduce at first two puzzles, chosen from the Czech semi-monthly magazine *Hádanka a Křížovka* (*Riddle and Crossword puzzle* in English), suitable to be solved in topic of graph theory dealing with isomorphism.

Isomorphism is an important basic graph theory concept explained in any textbook dealing with graph theory. Let us remind ourselves of its definition [1]:

Two graphs $G = (V, E)$ and $G^* = (V^*, E^*)$ are called isomorphic if a bijection $f: V \to V^*$ exists such that $\{x, y\} \in E$ if and only if $\{f(x), f(y)\} \in E^*$ holds for all $x, y \in V, x \neq y$.

A simple explanation of isomorphism is that two graphs are isomorphic if they have the same "structure" and differ only by the names of their vertices and edges. A nice motivation suitable to the concept is given in the following puzzle.

*Detective office*

Two detectives investigated the same group of people and used graph-representation for the relation between each pair of people who know each other. The first detective represented the people by letters, the other detective by numbers (see Fig. 1 and Fig. 2). Our task is to find out the connection between their graph-representations.

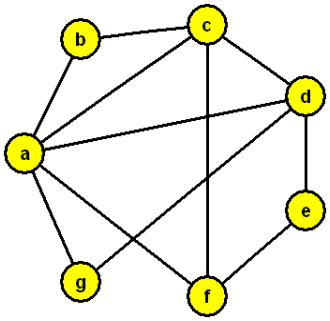


Fig. 1 The first graph-representation

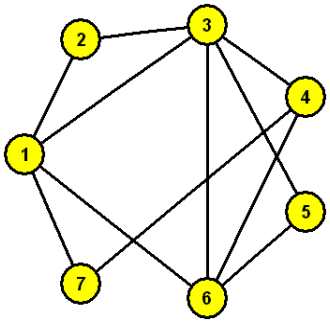


Fig. 2 The other graph-representation

To solve this simple puzzle, an isomorphism must be found between the two graphs illustrated in the figures above. The puzzle doesn't demand a graphical interpretation of the given task because it is set directly in the graphs. The solution can be found quiet easily considering the degrees of vertices.

The following puzzle is more complex (cf. [9] [12]) particularly in regards to finding out an appropriate graph representation of the task.

*Cities*

Try to place the names of cities Atlanta, Berlin, Caracas, Dallas, Lima, London, Metz, Nairobi, New York, Paris, Quito, Riga, Rome, Oslo and Tokyo into the frames of the given map (Fig. 3) so that no city shares any letter in its name with any cities in its adjacent frames (horizontal or vertical).

To solve this puzzles using graph theory it is necessary first to make a graph-representation of the map and also of the relation between two cities that do not contain the same letter in their names (see Fig. 4 and Fig. 5), and then to find an isomorphism between the graph representing the map and a subgraph of the graph representing the relation.

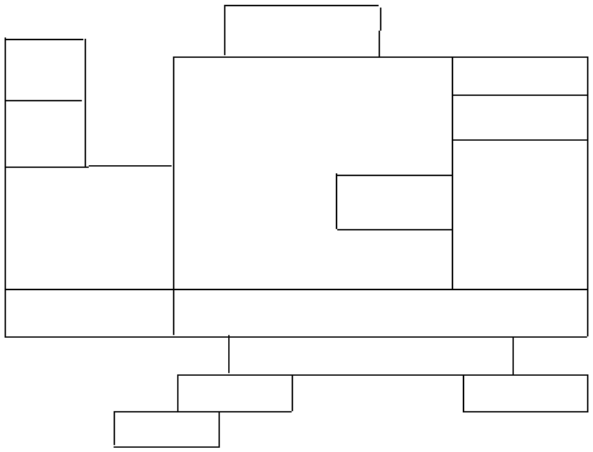

Fig. 3 Map of the puzzle *Cities*

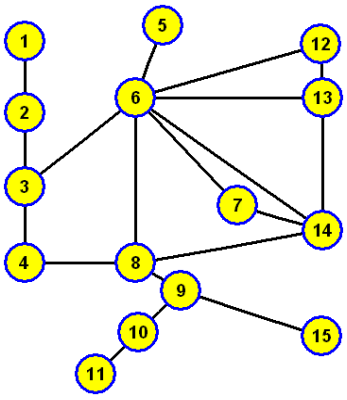


Fig. 4 Graph-representation of the map given in the puzzle *Cities*

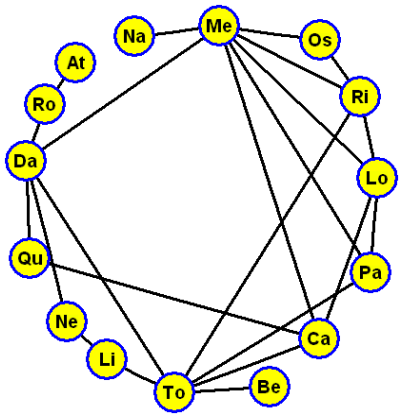


Fig. 5 Graph-representation of the relation given in the puzzle *Cities*

The map and also the determined relation between two cities have obvious graph-representation for everyone experienced in graph theory.

However, to consider the given rule as the relation "be adjacent" defined as "city *x* is adjacent with city *y* if there is no same letter in their names" and find out the appropriate graph-representation, i.e. a graph, whose vertices represent the cities and edges corresponding with the relation "be adjacent", cause students mostly difficulty because the puzzle *Cities* is introduced in the subject in one of the first lessons. The puzzle serves as a very useful first step into the development of students' ability to „see" graph-representation of a task.

Considering the vertices with the biggest degree and their neighbors (vertex 6 with the degree of 7 must correspondent with vertex *Me*, the only vertex with a degree larger or equal to 7) there is no problem changing the view of the graph given in Fig. 5 to form another view (see Fig. 6) from which the solution (see Fig. 7), i.e. subgraph isomorphic to the graph on the Fig.4, is quite clear.

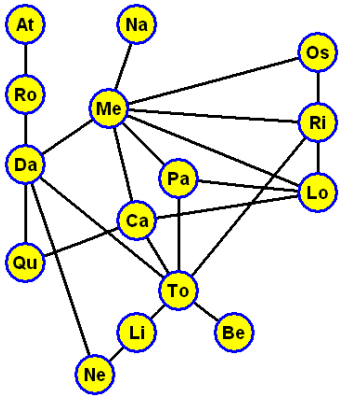


Fig. 6 Another picture of Fig. 5

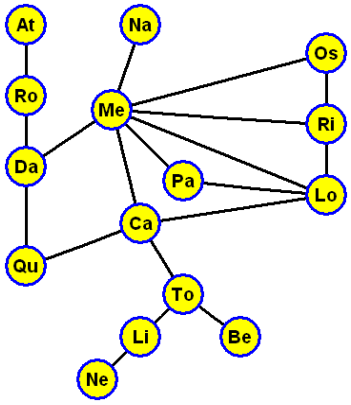


Fig. 7 Solution of the puzzle *Cities*

Before we start to deal with the others puzzles let us briefly 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, appropriate theorem and statement discussed thereinafter in this paper.

```
begin
initially all vertices and edges of
the given connected undirected graph
G, with n vertices and m edges, are
uncoloured. Choose any single vertex,
insert it into FIFO, colour it blue
and search it.
while FIFO is not empty do
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*

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*), where *h*(*y*) is the level of (*T*, *v*) where the vertex *y* lies (supposing *h*(*v*)= 0).

There are more statements following from the above mentioned theorem (an overview can be seen e.g. in [9]) however for the aim of this paper the introduced statement is sufficient.

Both following puzzles of different difficulties can be successfully solved using BFS algorithm including the previous statement. A level of complexity to find out the appropriate graph-representation of each puzzle is obvious.

*Puzzle* 1

Let us have a look at the Fig. 2. 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.

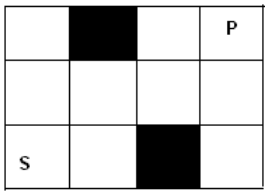


Fig. 2 Picture to the given puzzle 1

A graph representation to the task (see Fig. 4) can be easily done in the following way.

Let us complete the Fig. 2 by numbers and letters to get Fig. 3.

Then each cell is represented by the vertex Pc, where P∈{A, B, C, D} and c∈{1, 2, 3} and an edge is between each pair of vertices where the step defined by the above rules is possible.

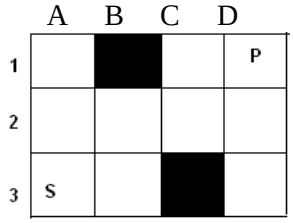


Fig. 3 The Fig. 2 completed by numbers and letters

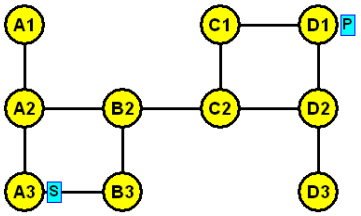


Fig. 4 Graph representation to the puzzle 1

*Puzzle* 2

Let us have a look at the Fig. 5. 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.
- 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.)

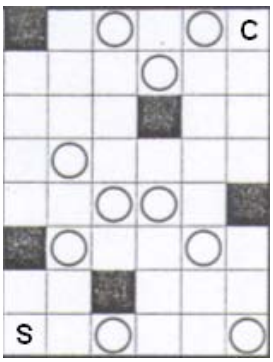


Fig. 5 Picture to the given puzzle 2

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. 5 by numbers and letters in the same way as in the puzzle 1 and imagine that each cell is represented

either by the vertex $Pc^2$, or by the vertex $Pc^3$, where $P \in \{A, B, C, D, E, F\}$ and $c \in \{1, 2, \ldots, 8\}$. The upper index determines the used speed.

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

*Solution to the puzzles* 1 *and puzzle* 2

Using the Breadth-First Search to solve both tasks with regard to the *Statement* appropriate BFS Trees are obtained. 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 (one can even see all four shortest paths of the length 5).

Here let us illustrate on Fig. 6 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 $F1^i$ (the cell **C**, which can be achieved either as the vertex $F1^2$ or as the vertex $F1^3$), is perceivable.

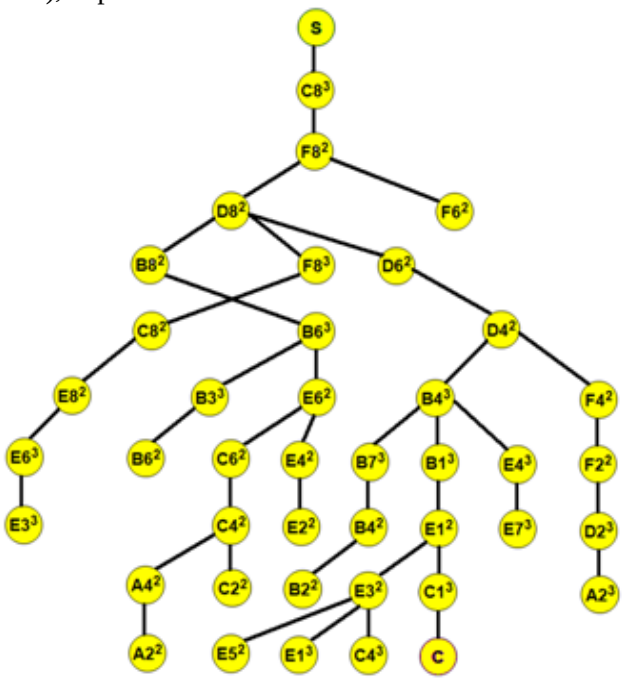


Fig. 6 BFS Tree to the puzzle 2

The first puzzle is really easy and students are able to solve it themselves on the lessons. However to solve second puzzle is much more difficult in regards on graph representation of the puzzle and solution itself. The second puzzle is therefore solved together with students on lectures or during lessons.

## 3.2 Logical games

The logical games are welcomed to diversify the lessons. We use the games, which solve two or three students together. For inspiration let us introduce two examples here.

*Game* 1 - Sprouts

Planar graphs, their planar representations respectively, can be practised in an enjoyable way using the game Sprouts.

Sprouts is a simple game developed in 1967 by Michael Paterson and John Conway. The game is described by Baird and Schweitzer in the following way [3]:

Sprouts is played by two players connecting spots with lines on a playing surface. The playing surface (piece of paper or computer screen) begins with *n* spots for the players to choose between. Players take turns connecting spots and adding a new spot along the drawn lines with the following constraints:

- The line must not touch or cross itself or any other line.
- The new spot cannot be on an endpoint of the line, and thus splits the line into two parts.
- No spot can have more than three lines connected to it; note that when a new spot is created, it starts with two lines already connected to it.

Eventually, there are no legal moves remaining, and the player who makes the last move wins.

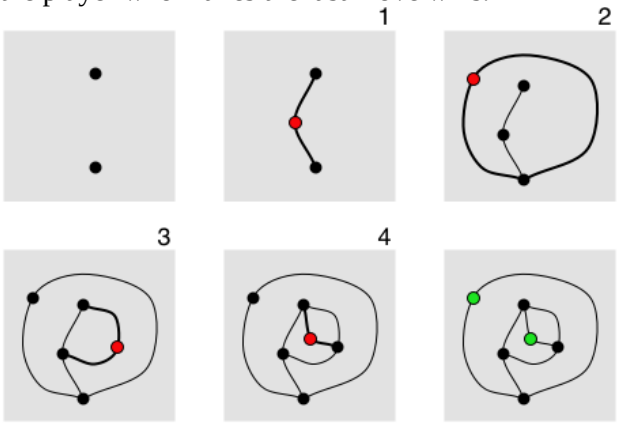


Fig. 7 The game Sprouts beginning with two spots

In the lesson two students solve the game on the blackboard and some particular solutions are discussed by the others. For example, in the Fig. 7 (see [20]) a possible process of the game Sprouts with two starting spots is presented. This illustration of the game finishes with the last picture (with regard to the constraints no other step is possible) and discussion can start.

For example we discuss with students the question *how to change the process described in the Fig. 7 to continue at least one step*. Such a possibility is shown in the Fig. 8.

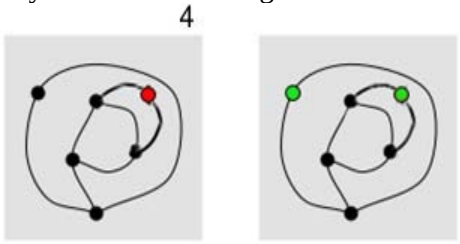


Fig.8 Part of another solution that can continue

With regard to this change we emphasize that *joining green vertices on the last picture we get planar graph in both figures,* but *planar representation is gained only in the Fig. 8.*

*Game* 2 - HYCLE

The 3-D *computer* game HYCLE (**H**amiltonian c**YCLE**) is designed for one player and its goal is to find the Hamiltonian circle in the given graph: "Visit every vertex exactly once and end up where you started". [21] Player can move the graph to improve his vision of vertices and edges. This feature can make the game easier, but also can confuse the player if he/she is not carful enough.

HYCLE is a good tool not only for practising Hamiltonian graphs, but also for the development of spatial imagination. The game has fifteen levels and the difficulty of the given graph increases at each level (see Fig. 9).

Students can try to play this game themselves at home.


Fig. 9 Level 4 of HYCLE [21]

It is well known that to determine whether a given graph is or is not Hamiltonian belongs to the NP problems. However, in lessons students practise this knowledge on small graphs only using the following simple rules to find out the result.

- If a given graph has *n* vertices then a Hamiltonian circle has exactly *n* edges.
- If a vertex *v* has degree *k* then Hamiltonian circle has to contain exactly two edges incident with the vertex *v*.
- When constructing a Hamiltonian circle in a graph with *n* vertices no circle containing less than *n* vertices is allowed to be created (closed) during the process.
- Once a constructed Hamiltonian circle contains two edges incident with a vertex *v* the remaining edges incident with the vertex *v* are excluded.

To diversify lessons we let students to play the following modification of the above described game HYCLE.

*Our game is designed for two students*. They take turns in including edges to a created Hamiltonian circle. Player who makes a mistake with regard to the given four rules lose, but only if the other player point it out. If a player makes a mistake and his partner doesn't see it, both players lose (Hamiltonian circle cannot be created). *The best result is achieved when the Hamiltonian circle is found and thus both players are winners*.

### 3.3 Multimedia program GrAlg

In the section 2.2 we have already discussed how useful and important can be role of a multimedia program for self preparation of students to a subject.

It is known that multimedia applications have substantially influenced education. They give teachers an excellent chance to demonstrate and visualize the subject matter more clearly and comprehensibly, as well as also enabling them to prepare study material for students which optimizes their study habits.

We would like to emphasize that along with large software products dealing with a wide spectrum of objects developed by a team of professionals very important role play also various programs dealing with objects appropriate to course subject matter created on a script given by the teacher with regard to students needs similar as our program ALGORITHMS (cf. [19]).

In the subjects dealing with graph theory and combinatorial optimization there is no problem in illustrating the needed concepts using graphs. However, it is very important to prepare suitable illustrative graphs and have the possibility to use

colours to emphasize characteristics of the explained concepts.

In the subject *Graph Theory and Combinatorial Optimization* the essential tool used not only by students but also by teacher is the program GrAlg (**Gr**aph **Alg**orithms) created in the Delphi environment by our student within his thesis [15].

The main purpose of the application is the easy creation and modification of graphs and the possibility to *emphasize with colours basic graph-concepts and graph algorithms* on graphs created within the program.

Its big advantage is also the possibility to run programs visualizing all of the subject *Graph Theory and Combinatorial Optimization* explained algorithms on graphs in a way from which the whole process and used data structures can clearly be seen (see Fig. 10).

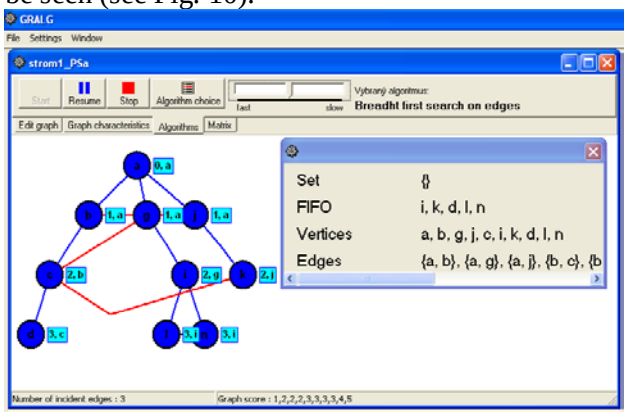


Fig. 10 Program GrAlg – visualization of the Breadth-First Search algorithm

Thanks to the fact that the program allows opening more than one window so that two (or more) objects or algorithms can be compared at once teachers can explain discussed problems from more points of view and show mutual relations among used concepts and algorithms.

Moreover, the possibility to save each created graph in bmp format allows teachers easy insertion of needed graphs into the study material and thus saves their time when preparing text material and presentations (cf. the previous sections).

## 4 Conclusion

The paper is intended as an inspiration for all educators developing students' algorithmic and logical thinking, the base of 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.

*Acknowledgment*

This research has been partially supported by the Specific Research Project of the University of Hradec Králové No. 2109.

*References:*

[1] Albrecht, K.: *Brain Building: Easy Games to Develop Your Problem Solving Skills*, Prentice Hall, 1984.

[2] Arlow J, Neustadt I.: *UML a unifikovaný proces vývoje aplikací*. Computer Press, 2003.

[3] Baird, L., Schweitzer, D.: *Complexity of the Game of Sprouts.*, in Hamid R. Arabnia; George A. Gravvanis & Ashu M. G. Solo, ed., 'FCS', CSREA Press, 2010, pp. 177-182.

[4] Herout, P.: *Učebnice jazyka Java*. KOPP: České Budějovice, 2000.

[5] Jinoch J, Műller K, Vogel J.: *Programování v Pascalu*. SNTL: Praha, 1986.

[6] Milková, E.: Algorithms: The Base of Programming Skills. In: *ITI 2007 Proceedings of the 29th International Conferences on INFORMATION TECHNOLOGY INTER-FACES*, University of Zagreb, Croatia 2007, pp. 765 – 770.

[7] E. Milková at al., *Algoritmy – základní konstrukce v příkladech a jejich vizualizace* (in English: Algorithms – basic constructions and their visualization), Gaudeamus, Hradec Králové, 2010, s. 98.

[8] Milková, E.: Multimedia Applications and their Benefit for Teaching and Learning at Universities, *WSEAS TRANSACTIONS on Information Science & Applications*, Issue 6, Vol. 5, June 2008, pp. 869-879.

[9] Milková, E., Constructing Knowledge in Graph Theory and Combinatorial Optimization, *WSEAS TRANSACTIONS on MATHEMATICS*, vol. 8, no. 8, 2009, pp. 424-434.

[10] Milková, E.: BFS Tree and x-y Shortest Paths Tree. In: *Applied Computer Science, Proceedings of International Conference on Applied Computer Science* (ACS), WSEAS Press, Malta, September 15-17, 2010, pp. 391–395.

[11] Milková, E.: Multimedia Tools for the Development of Algorithmic Thinking. *Recent Patents on Computer Science*, Vol. 4, No. 2, 2011, pp. 98- 107.

[12] Milková, E., Development of logical thinking in enjoyable way. In: *Efficiency and Responsibility in Education 2011*, Czech

University of Life Sciences Prague, 2011, pp. 221-229.
[13] Milková, E.: Development of Algorithmic Thinking and Imagination: base of programming skills, In: *Proceedings of 16th WSEAS International Conference on Communications and Computers* (Part of CSCC'12), WSEAS Press, Kos Island, Greece, July 14-17, 2012, pp. 68–72.
[14] Pecinovský R.: *Myslíme objektově v jazyku Java 5.0*. Grada: Praha, 2004.
[15] Šitina, J.: *Grafové algoritmy – vizualizace*, Hradec Králové: thesis, 2010.
[16] Virius M.: *Java pro zelenáče*. Neocortex: Praha, 2001.
[17] Voborník, P.: *Programovací jazyk pro podporu výuky algoritmů*, Hradec Králové: thesis, 2006.
[18] Wirth N.: *Algoritmy a datové štruktúry údajov*. Alfa: Bratislava, 1989.
[19] XUE-YING MA, BIN-KUI SHENG: Designing Test Engine for Computer-Aided Software Testing Tools, *WSEAS TRANSACTIONS on COMPUTERS*, vol. 10, no. 5, 2011, pp. 135-145.
[20] http://en.wikipedia.org/wiki/Sprouts_(game)
[21] http://hry.czin.eu/48443-Hycle.html