Olympiads in Informatics, 2016, Vol. 10, 61–72

DOI: 10.15388/ioi.2016.04

The Place of the Dynamic Programming Concept in the Progression of Contestants' Thinking

Ágnes ERDŐSNÉ NÉMETH[1,2], László ZSAKÓ[3]

[1]*Batthyány High School, Nagykanizsa, Hungary*
[2]*Doctoral School, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary*
[3]*Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary*
e-mail: erdosne@blg.hu, zsako@caesar.elte.hu

Abstract. The special problem-solving strategies have been receiving a lot of attention lately, whether it is teaching computational thinking for all or computer science for competitors. A didactically interesting question is how problem solving can be developed in children's minds, what steps and tasks lead through from understanding the idea to its professional usage. In this paper we present and explain how and in what forms the given problem-solving strategies, especially the dynamic programming concept, appear in children's informatics studies: from CS unplugged activities through Bebras tasks and national CS competitions to efficient coding at the IOI.

Keywords: dynamic programming concept, teaching informatics in primary and secondary schools, preparing for contests.

1. Overview

There are a lot of problem-solving strategies which every contestant has to be familiar with. In this paper, we want to focus on one of the most important ones, specifically dynamic programming, because "[Until] you understand dynamic programming, it seems like magic" (Skiena, 2008).

The dynamic programming name came from Bellmann in the 1950's (Dreyfus, 2002). He wanted to find a name for the multistage decision making process, in which the general case is very hard to solve thus one must use the divide & conquer concept. It leads back to special cases, when the solution is expressed more easily – not in terms of the unknown function, but in terms of an action or decision.

If we want to speak about dynamic programming, we also have to examine other corresponding concepts, like recursion, memoization and divide & conquer.

The concept of recursion consists of a recursive function and a recursive implementation; it is a top-down approach. Usually the runtime of the algorithm is exponential and this technique usually fails already in a small sample size. The memoization is a top-down approach as well, it is more effective than recursion. The core idea is the

same: recursive function and recursive implementation, but storing the calculated results, thus without calling the recursive step again when the actual state previously appeared.

When we are experienced enough to implement a recursive approach correctly, there is a special technique for dramatically reducing the runtimes of certain algorithms from exponential to polynomial or from factorial to exponential, at the expense of higher memory usage. This is done by solving sub-problems and storing the results using a dynamic programming concept. The concept is about a recursive function, but the implementation is to build the values from bottom up without calling the recursive step repeatedly.

The simplest type of dynamic programming is when an array is filled – namely a table – cell-by-cell in a predetermined order, implementing the recursive function from bottom up. We will call it a *basic DP*.

Some people use the term dynamic programming only for those recursive problems, which involve optimization, but the technique of completing a table to solve any other type of problem is almost the same when it comes to implementing the solution. We will call it *classic DP*.

The dynamic programming as problem-solving strategy is the implementation of the following five steps (Horváth, 2004):

1. Analysing the (optimal) solution's structure.
2. Dividing it into subproblems and components:
   a. Dependence of components must be acyclic.
   b. Every subproblem (optimal) solution should be a (recursive) expression of components' (optimal) solution.
3. Expressing each subproblem's (optimal) solution as a (recursive) function of components' (optimal) solution.

These three steps are the planning of recursive algorithm. DP comes from the next two steps:

4. Calculating subproblems' (optimal) solutions from bottom up (completing a table).
5. Calculating an (optimal) solution from the previously calculated and stored information.

If you are familiar with the basic DP techniques then you could continue with some advanced techniques (Steinhardt, 2008).

The first advanced type is to keep track of all possible transition states. In this case DP means filling out a table row-by-row, with the values depending on the row number and values in the previous row.

Second advanced type is the dynamic greedy type.

The third advanced type is a steady state convergence, only for more experienced students. In this case the recursive equation must be repeatedly applied, then values will converge exponentially to the correct values.

All of the DP types have a place in different stages and in different ages of contestants' studies.

2. Place of Dynamic Programming in Algorithms Textbooks

There are a lot of textbooks about algorithms. They discuss all algorithms sequentially and directly as they are written for university students. The structure of these books and the place of dynamic programming in them is different.

In the textbook of Skiena (2008) the DP takes place after data structures, sorting and selecting algorithms, graphs, combinatorics, backtrack and parallel programming. In the chapter of dynamic programming, he compares algorithms using cache vs. computation at first. After this, he discusses the problem of string matching, longest increasing subsequence and partition problem. The limitation of DP is presented via travelling salesperson's problem and there are some words about correctness and efficiency of DP. Just after the DP concept comes the recursion and memoization, the concepts of the top-down structures.

In the textbook of Kleinberg and Tardos (2006) there are basic algorithms, graphs and greedy algorithms first. To process sets which can be divided into independent parts they recommend divide & conquer concept. Just after this method, they speak about DP: the recursion, the memoization and iteration over sub-problems are the parts of this chapter.

In the textbook of Dasgupta *et al.* (2006), dynamic programming is presented after arithmetics, primes, cryptography, hashing, divide & conquer, graphs and greedy algorithms via examples: travelling salesman's problem, longest increasing subsequence and knapsack.

In the textbook of Sedgewick and Wayne (2011) there is no chapter dedicated to DP, but the concept appears in some points of the book.

In the textbook of Gupta (2009) DP comes after introduction to algorithms, divide & conquer method and greedy method.

Bebis (2007) has lot of chapters about sorting and searching methods, DP comes after that.

There are not two textbooks, in which the place of DP is the same in the sequence of algorithms. Sometimes it comes before graphs (Gupta, 2009), often it comes after graphs. Occasionally it is before recursion (Kleinberg and Tardos, 2006), but in the most common order its place is after recursion and memoization. Sometimes it is placed before the greedy algorithms, sometimes after. Textbooks do not usually stress the differences between dynamic programming and greedy approach, nor warn they about the danger of accidentally using one instead of the other.

These textbooks are almost useless for primary and secondary school pupils because of their advanced mathematics and informatics contents. Some parts of them may be useful in upper secondary, just for contestants preparing for IOI.

3. New Way for Contestants Learning DP in Upper Secondary School

Last year there was a paper in IOIJournal about the critical analysis of textbooks and new way of teaching DP (Forišek, 2015) for upper secondary school students preparing for national olympiads and IOI. He started with the Fibonacci sequence – something is

well-known by children from math studies – implementing it with a recursive function and making this function more efficient with memoization. He compares iterative and dynamic solution, then introduces DP bottom-up. He demonstrates that the exponential solution longest common subsequence problem with a top-down approach turns polynomial ($o(n^2)$) with a bottom-up approach. It is a very good structure if students want to prepare for olympiads all at once.

There is a book about competitive programming, which was written for contestants preparing for ACM ICPC and IOI (Halim and Halim, 2014). It is not a real textbook, it teaches the effective type detection of tasks and the correct, error-free coding, not the concepts behind the algorithms. According to it contestants' main goal 'should be to honing [their] ability to recognize a problem as DP, finding the recursive formula for such a problem, coding the problem, and doing all of this quickly'.

After data structures and problem-solving strategies like searching – iterative & recursive – divide & conquer concept and greedy algorithms comes dynamic programming through an example: UVA 11450 wedding shopping. The chapter begins with the repetition of recursion, backtrack, optimization and counting problems. It shows, that this task's solution: greedy method failed with wrong answer (WA), divide & conquer method failed with WA (because of non-independent parts), complete search failed with time limit exceeded (TLE). The dynamic programming method with top-down (memoization) and with bottom-up approach is working. After this very detailed analysis he gives six more example with analysis and many task recommendation to become familiar with this method.

We think these methods work effectively for students in upper-secondary-school-age (grade 11–13) preparing purposefully for national and international olympiads.

4. Teaching Dynamic Programming in Primary and Secondary School

If the students know structured programming concept (Floyd), they are familiar with the top-down concept too, because of stepwise refinement; and they are familiar with the concept of bottom-up, because concrete objects and functions. So the pupils knows top-down and bottom-up paradigms as soon as they begin to implement a computer program.

We think teaching dynamic programming ideally begins in upper primary school in mathematics and informatics lessons. Implementation of DP on computers is possible when children are familiar with the basic data structures (integer, boolean, array), basic algorithms (sequence, iteration, selection, searching, procedures and functions) and the concept of recursion.

4.1. *CSUnlugged*

There is not any activity yet that covers dynamic programming in the CSUnplugged repository, but there is an intention to make a good one from the change-making problem.

4.2. *BEBRAS*

In the Bebras competitions there are tasks about DP every year. There are other tasks, in which the recursion is the best solution. There are many in which the greedy gives wrong answer and the dynamic programming concept must be used for the right solution.

On Bebras competitions the DP problems appear in a wide range of age groups and difficulties also. It proves that the concept of DP comes much earlier than at the end of secondary school and it is understandable for everybody, not only for contestants: it is part of computational thinking skills.

The easiest task of DP is from 2013 for grade 5–8: the Pairs without Crossing (Kreuzungsfreie Pärchen). The brute force algorithm is working, but takes long time to try every possible connection pairs, the dynamic concept make it easy. In 2011 the problem Earn coins (Münzen verdienen) is a special case of the classic knapsack problem. This one was one of the hard problems for the grade 5–6, middle for grade 7–8 and easy for grade 9–10. In 2014 another classic DP problem appeared, the Expensive Bridges (Teure Brücken) which one was hard problem for the grade 7–8, middle for grade 9–10 and easy for grade 11–13.

There are more problems connected to DP for secondary-school-age students: the Jumping Puddles (Pützenspringen), the game ROOK, in 2010 the task Pinecone (Tannenzapfen), in 2014 the Best translation (Beste übersetzung) and in 2015 the Fireworks2 (das Feuerwerk2).

4.3. *Tasks for Contestants of Upper-Primary-School-Age*

In the grade 5–6 we could start with LOGO programming. It provides a strong foundation in the basic programming structures, like sequence, iteration and selection. Through the drawings, it also visualizes the concept of recursion well. They can imagine and implement binary tree (Fig. 1), the Sierpinski-triangle (Fig. 2), the Koch-curve (Fig. 3).

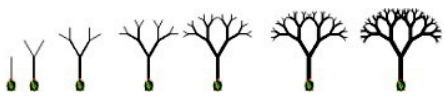

Fig. 1. Binary tree.

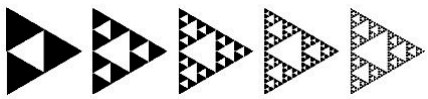

Fig. 2. Sierpinski-triangle.

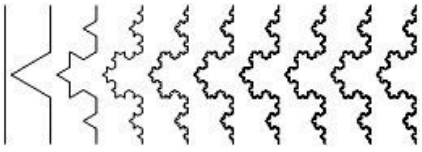

Fig. 3. Koch-curve.

In Hungary and many other countries, they meet table filling and the thought of recursion in mathematics. They calculate total number of possible paths in a grid of characters, from the top-left corner to the right-bottom corner to spell a given word (Fig. 4). They calculate total number of possible paths, even if there are empty squares in the grid (Fig. 5).

D	Y	N	A	M	I	C	P
Y	N	A	M	I	C	P	R
N	A	M	I	C	P	R	O
A	M	I	C	P	R	O	G
M	I	C	P	R	O	G	R
I	C	P	R	O	G	R	A
C	P	R	O	G	R	A	M

1	1	1	1	1	1	1	1
1	2	3	4	5	6	7	8
1	3	6	+				

Fig. 4. Spelling a given word.

D	Y	N	A	M	I	C	P
Y	N	A	M	I	C	P	R
N	A		I	C	P	R	O
A	M	I	C	P	R	O	G
M	I	C	P	R	O	G	R
I	C	P	R	O	G	R	A
C	P	R	O	G	R	A	M

1	1	1	1	1	1	1	1
1	2	3	4	5	6	7	8
1	3	0	+				
1	4	4					

Fig. 5. Spelling a given word with a pit.

Pupils of grade 7–8 meet simple recursive sequences and functions, like the Fibonacci sequence. In mathematics lessons, they meet combinatorial problems (permutation, variation and combination without repetition) without naming them. They come across problems, like longest/shortest path in a directed/undirected graphs and coloring problems on very small graphs, coin changing problem for small numbers, shaking hands/sitting in a row/around a table, they calculate extreme values in Diophantine problems. These problems are solved with table filling or with recursive expressions. The formulas, like n! or $\binom{n}{k}$ are not formulated.

They can solve problems like these:

How many different, 10 cm high towers can build from 2 cm high blue, 2 cm high yellow and 1 cm high red building blocks?

Fig. 6. $a_1 = 1$, $a_2 = 3$, $a_3 = 5$, $a_4 = 11$ – Counting with drawing systematically.

If they draw it systematically (seeing the blocks from bottom), they can guess the recursive expression $a_n = a_{n-1} + 2*a_{n-2}$ *and they check it empirically. The solutions are: 1, 3, 5, 11, 21, 43, 85, 171, 341, 683.*

*How many different covering exist on a 2*8 table with 1*2 size dominos?*

Fig. 7. $a_1 = 1$, $a_2 = 2$, $a_3 = 3$, $a_4 = 5$ – Counting with drawing systematically.

If they draw it systematically (seeing the blocks from the right), they can guess the recursive expression $a_n = a_{n-1} + a_{n-2}$ *and they check it empirically. The solutions are:1, 2, 3, 5, 8, 13, 21, 34. The solution can be built for any given N with a table filling.*

In primary schools the children meet problems with large numbers also, like 1000 points, 2016 numbered cards, 10 000 people. In such cases, the obvious idea is – instead of the original task – to examine a simplest problem. What happens if the number of points is 2, 3, 4, ..., the number of cards is 5, 6, 7,…, the number of people is 2, 3, 4,…? If any regularity is noticed, they try to verify it empirically and apply it on the original problem with large numbers. On primary-school-level, formulating and using the hypotheses is enough to solve such problems.

On the other hand, these problems can be implemented on the computer, as a recursive expression or with table filling for larger numbers. The next classical problems are the first appearance of dynamic programming approach for them:

Robot – *A robot starts from the top left corner (1,1) of a MxN grid. At each step the robot can make one of the two choices: move one cell to the right or move one cell down. How many possible paths are there for the robot to reach the right-bottom corner of the grid?*

The robot problem can occur in many variations at grade 7 and 8:

- *The question is the same, but there are cells in a grid, on which the robot can't step on (traps).*
- *The question is the same, but there are cells in a grid, on which the robot have to step on (mandatory fields).*
- *In every cell there are given number of pearls and the question is, what is the maximum number of pearls the robot can collect on its way.*
- *Mix of traps and pearls.*

Staircase – *You are standing in front of a staircase, which has N stairs. Your goal is to reach the top. If you are standing on the* i^{th} *step, you can hop to* $(i+1)^{st}$ *or* $(i+2)^{nd}$ *or* $(i+3)^{rd}$ *step. Given N, calculate the count of total possible paths for you to reach* N^{th} *stair!*

Coin Change – *You want to make change for given N cents and you have infinite supply of each of* S_1, S_2, .. , S_m *valued coins. How many ways can you make the change?*

Subset sum – *Detect, if any of the subset from a given set of N non-negative integers sums up to a given value S!*

Dice Throw Problem – *Given N dice, each with m faces, numbered from 1 to m. Find the number of ways to get sum X! (X is the summation of values shown by the dices.)*

Flooring – *How many different coverings exist on a 1*N floor with 1*1 and 1*2 parquet pieces? How many different covering exist on 2*N floor with 1*2 parquet pieces?*

Towers *– How many different, N meters high towers can be built from 2 meters high blue, 2 meters high yellow and 1 meter high red blocks?*

The previous dynamic programming problems should be solved at the primary-school-level as the analogous math problems: children formulate and use the hypothetical recursive expressions and implement them with table filling, without extensive argumentation. Mostly they cannot calculate directly the answer, but they can give a recursive formula and the direction of filling the table, and this way they can solve the problem. They can solve basic DP problems, without optimization.

On informatics contest the children use basic data structures (integer, boolean, one and two-dimensional array of integers, simple strings) and basic algorithms can be applied for various problems in various wording. Choosing, selecting, counting, searching, summarizing, selecting maximum/minimum, sorting, separating into two groups, prime testing are these basic algorithms. Stages of solving tasks are understanding the problem, choosing the right data structure, selecting right algorithm, implementing and testing it. In addition to the conservative tasks there are ad-hoc problems where children can apply basic algorithms creatively. The basic DP problems appear in the regional and national rounds of competitions.

4.4. *Tasks in Lower-Secondary-School-Age*

In Hungary and many other countries the children continue to learn combinatorial problems in grades 9–10 in mathematics lessons, they group and formalise these problems. During these years, they also meet the idea of mathematical induction, so they can prove the previously discovered recursive formulae. They learn about sequences and sometimes they give the explicit formulae for recursive expressions. They know the formula of n! and $\binom{n}{k}$, they also learn Pascal-triangle. But they do not necessarily know the relationship between the binomial coeffitients and the Pascal-triangle.

They can discover and solve more complex recursive expressions, sometimes these functions call each other, like:

*How many different covering exist on a 2*N table with 1*2 and 1*1 size dominos?*

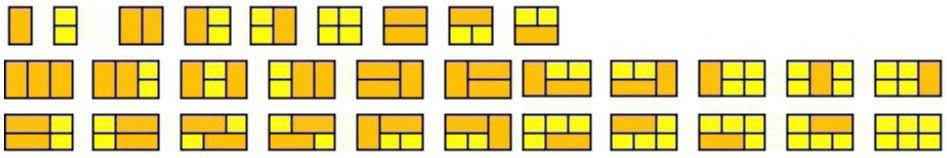

Fig. 8. $a_1 = 2$, $a_2 = 7$, $a_3 = 22$, $a_4 = 71$ – Counting with drawing systematically.

Drawing it systematically is not enough to formulate the recursive expression. One must analyze possible cases. The result comes with two expressions in simultaneous recursion: $d_n = a_{n-1} + f_{n-1}$, $f_n = a_{n-1} + d_{n-1}$, $a_n = 2*a_{n-1} + a_{n-2} + f_{n-1} + d_{n-1}$. *After simplification:* $a_n = 3*a_{n-1} + a_{n-2} - a_{n-3}$.

In this age they learn the basics of graph theory in mathematics, they learn types of graphs (trees, binary trees, relational matrix), storage of graphs (vertex matrix, edge matrix, edge list) and algorithms of graphs (breadth first traversal, depth first traversal, relations) in informatics. They meet recursion again, divide and conquer, backtrack, and greedy algorithms too, on basic level.

They learn all classic dynamic programming problems, as follows:

Partition problem – *Divide the set of numbers into two groups, where sum of each group is same!*

Longest Increasing Subsequence – *Find the length of the longest subsequence of a given sequence, such that all the elements are sorted in increasing order.*

Knapsack Problem – *A thief robbing a store can carry a maximal weight of W in his knapsack. There are N items and* i^{th} *item weighs* w_i *and is worth* v_i *dollars. What items should the thief take?*

Contiguous subsequence with maximum value – *Find the contiguous array with the maximum sum in a given an array, containing both positive and negative integers!*

Minimal number of coins for change – *What is the minimal number of coins, to make change for a given amount T only coins of values* $v_1, v_2, \ldots, v_n$ *can be used?*

These examples can be solved by using recursion with optimization, the right order of filling the table must be given usually it is not evident. There are a number of variations of these problems, sometimes the difference is merely wording.

Next type of DP problems is game strategy in various formats, like:

Optimal Strategy – *Consider a row of N coins of values* $v_1, \ldots, v_N$, *where N is even. We play a game against an opponent by alternating turns. In each turn, a player removes either the first or last coin from the row and receives the value of the coin. Determine the maximum possible amount of money we can definitely win if we move first?*

Basic data structures may be supplemented with string and real. There are a number of dynamic programming problems with strings, too:

Longest Common Subsequence – *Find the longest common subsequence of two strings, where the elements are letters from the two strings and they should be in the same order!*

Longest Common Substring – *Find the longest common substring of two strings!*

Edit Distance – *Given two strings and a set of operations Change (C), insert (I) and delete (D). Find minimum number of operations required to transform one string into another!*

On informatics contest classic dynamic tasks can be in every round, so the contestants have to be ready to solve it.

4.5. *National Olympiads*

In grades 11–12 contestants prepare for national Olympiads. They know a lot of algorithms, in these years they learn to use advanced data structures, like set, priority queue, stack, and they implement previously learned algorithms with these data types. While in previous years they were asked for the existence of the solution or the number of steps in the solution, now they also have to decrypt the entire path traversal in dynamic programming problems.

Sometimes there are strict memory limits and there is no space for the whole table. In this case, only the values of those cells have to be stored in memory, which are essential to the calculation of the next row. During the decryption process the rebuilding of the table may be as hard and interesting to implement as the original problem.

They learn about combinatorial and geometrical problems.

Within the concept of divide and conquer, recursion or dynamic programming more complicated expressions should be optimized.

There are advanced DP problems: all possible transitions and dynamic greedy type.

Example of complicated problems:

> **Balanced Partition** – *There is a set of N integers each in the range 0 ... K. Partition these integers into two subsets such that you minimize* $|S1 - S2|$*, where* $S1$ *and* $S2$ *denote the sums of the elements in each of the two subsets!*

The dynamic programming concept among strings also leads complicated problems, like:

> **Shortest Palindrome** – *Form a shortest palindrome by appending characters at the start of the given string.*
>
> **Palindrome Min Cut** – *Find the minimum number of cuts required to separate the given string into a set of palindromes.*
>
> **Longest Palindromic Substring** – *Find the longest palindromic substring of a given string!*
>
> **Longest Palindromic Subsequence** – *Find the longest palindromic subsequence of a given string!*

The contestants in the upper secondary age want to be computer scientists or engineers, they do not only know the basic algorithms but they can cope with complex tasks.

4.6. *Regional and International Olympiads*

When you want to prepare for Regional or International Olympiads you have to know everything about dynamic programming which is included in the textbooks. You have to solve a huge number of tasks. On Olympiads the solution of tasks are some kind of creative mixture of known algorithms.

Some examples:

> **Interval-Scheduling Problem** *(Greedy and DP Approach) N k-tuples processes are given with start & end times. Select as many processes as possible such that*

(I) no two selected processes intersect and (II) at most one process is selected from each k-tuple!

intersec Complete all tasks given the deadline, so that no task overlap!

Box Stacking – *You are given a set of N types of rectangular 3-D boxes, where the i^th box has height h(i), width w(i) and depth d(i). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. You can rotate a box, any of the side can be its base. It is also allowable to use multiple instances of the same type of box.*

Counting Boolean Parenthesizations – *You are given a boolean expression consisting of a string of the symbols 'true', 'false', 'and', 'or' and 'xor'. Count the number of ways to parenthesize the expression such that it will evaluate to true. For example, there are 2 ways to parenthesize 'true and false xor true' such that it evaluates to true. The order is defined only by parentheses.*

There are many tasks for practice on the online preparing and contest sites, like usaco.org, codeforces.com, codechef.com, uva.onlinejudge.org, spoj.pl. If you have met each type of concept detailed above, the textbook of Halim is a good choice for preparing for IOI.

5. Conclusions

Some antecedents of the dynamic programming concept for example the concept of recursion, might come up in earlier mathematics and informatics studies. If you are aware of this, introducing DP as a new problem-solving strategy is much easier.

We think, if you want to teach the technique of DP you have to start from a simple recursion then through memoization and table filling you could end with a real DP for optimization problems. You could start the whole process in the primary school age and circularly, returning to it in higher and higher levels your students would be familiar with this hard concept.

In the upper primary school, the basic DP comes up: a recursive expression implemented with table filling. At the beginning of secondary school, the classic DP programs continue the sequence: recursive expressions with optimization and at implementation the right order of table filling need to be thought of. In upper secondary school, the children can be familiar used with advanced types of DP: all of the previous problems with the retrieval of the way, how the optimal solution is built up, dynamic greedy type and the type, when you have to keep track of all possible transition states. Preparing for Olympiads, the students need everything from textbooks and the combination of other types of approaches.

Finally, it would not be magic for the contestants, just a useful problem-solving strategy.

References

Bebras–International Contest on Informatics and Computer Fluency (2007–2015). http://bebras.org http://www.beaver-comp.org.uk/; http://informatik-biber.de/archiv/

Bebis, G. (2007). *CS477/677 Analysis of Algorithms*. http://www.cse.unr.edu/~bebis/CS477/

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2001). *Introduction to Algorithms*. MIT Press, 2nd edition.

Dagienė, V., Stupurienė, G. (2016). Bebras – a sustainable community building model for the concept based learning of informatics and computational thinking. *Informatics in Education*, 15(1), 25–44.

Dasgupta, S., Papadimitriou, C.H., Vazirani, U.V. (2006). *Algorithms*. McGraw-Hill.

Dreyfus, S. (2002). Richard Bellman on the birth of dynamic programming. *Operations Research, INFORM.* 50(1), 48–51.

Erdős, G. (2010). A rekurzív módszer. In: *Magas Szintű Matematikai Tehetséggondozás Konferencia, ZALAMAT.* 20–32.

Forišek, M. (2015). Towards a better way to teach dynamic programming. *Olympiads in Informatics,* 9, 45–55.

Gupta, N. (2009). *Introduction to Algorithms*. http://www.curriki.org/oer/Introduction-to-Algorithms/

Halim, S., Halim, F. (2014). *Competitive Programming 3. The New Lower Bound of Programming Contests*. http://cpbook.net/

Horváth, Gy. (2004). A programozási versenyek szerepe az oktatásban. In: *INFOÉRA Konferencia.* http://www.infoera.hu/infoera2004/eaok/horvathgyula.pdf

Kleinberg, J., Tardos, É. (2006). *Algorithm Design*. Addison-Wesley.

Sedgewick, R., Wayne, K. (2011). *Algorithms*, Fourth Edition. *Addison-Wesley.*

Skiena, S.S. (2008). *The Algorithm Design Manual*. Springer-Verlag, 2nd edition.

Steinhardt, J. (2008). *Advanced Dynamic Programming Techniques*. https://activities.tjhsst.edu/sct/lectures/0708/dpadvanced.pdf

Á. Erdősné Németh teaches mathematics and informatics at Batthyány Lajos High School in Nagykanizsa. A lot of her students are in the final rounds of the national programming competitions, some on CEOI and IOI. She is a Ph.D. student in the Doctoral School of Faculty of Informatics, *Eötvös Loránd* University in Hungary. Her current research interest is teaching computer science for talented pupils in primary and secondary school.

L. Zsakó Dr. is a professor at Department of Media & Educational Informatics, Faculty of Informatics, Eötvös Loránd University in Hungary. Since 1990 he has been involved in organizing of programming competitions in Hungary, including the CEOI. He has been a deputy leader for the Hungarian team at International Olympiads in Informatics since 1989. His research interest includes teaching algorithms and data structures; didactics of informatics; methodology of programming in education; teaching programming languages; talent management. He has authored more than 68 vocational and textbooks, some 200 technical papers and conference presentations.