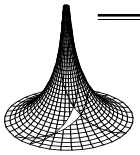

9/1 (2011), 13–25
tmcs@math.klte.hu
http://tmcs.math.klte.hu

Teaching
Mathematics and
Computer Science

How to teach computer programming if our goal is the International Olympiad in Informatics

Sándor Király

Abstract. Nowadays if a student in Hungary (age between 17–20 years old) wants to be the member of the Hungarian selected team (has four members) to participate in the International Olympiad in Informatics (IOI), first, he has to qualify himself in the first fifteen of the National Secondary School Competition (OKTV) in the programming category after the III. round. Then he should be in the first four place after the sixth round of the Selection Competition. Being successful is necessary that the student wants to start studying computer programming at least in the 9^{th} school year and he needs a teacher who prepares him. In the last nine years three students of the author have participated in five Olympics and two of them won gold and bronze medals. This article wants to demonstrate the methods that a teacher needs to use to teach students in 9^{th} school year for computer programming, to be the member of the Hungarian National Team after three or four years.

Key words and phrases: algorithm, Olympics, computer programming, talented student, differentiation, motivation.

ZDM Subject Classification: B20, B50, B60, C70, D50, P50, Q20, Q30.

1. Introduction

Comenius in his work of Didactica Magna promulgates the art of teaching for all. Qualification to the IOI, a student should only be the master the art of computer programming. But only students who are really talented can do this. How can we find these students? In most cases a teacher gets one or more groups of students and he has to teach them for programming at a sort of level. This

is the basic task. Among them the teacher should find those who can be the members of the team. A teacher should teach each student who wants to study computer programming. The 9^{th} or 10^{th} year is the latest time when the preparing should start. Students from 11^{th} to 13^{th} school year can apply for OKTV but before this competition it is advisable for students to participate in the Nemes Tihamér Competition as well and gain experience. If somebody does well in this competition or will be the member of the team of CEOI (Central–European Olympiad in Informatics) he will be closer to the Olympiad team.

If we accept the Renzoulli's modell [5] of talent as a take-off of defining talent it will become clear quite early which students are creative and who are genius enough. But their responsibility toward the task will turn out only after weeks or months. With hard work and a lot of practises computer programming can be learned at the level of Olympics.

2. What to teach and how to teach

Knowing the requirements of OKTV and the Selection Competition we can define six parts of topic that students have to learn about computer programming during years:

- basic structures, basic algorithm, simple data structures
- knowledge of Pascal or C (C++) (the official languages of Olympics)
- recursion, backtracking
- graph search algorithm and their adaptation, other graph algorithms
- greedy algorithms, dynamic programming
- other algorithms: combinatory and geometrical algorithms, interactive exercises

The requirements of advanced level final examination end with the recursion. The other algorithms can be found in the requirements of OKTV and Selection Competition.

2.1. Let's start with the basis

Before starting the real programming, it could be useful if we represent some useful algorithms that students use to solve their problems in their everyday life. (e.g.: phoning by using activate mobile, phoning by using deactivated mobile.)

At this point pupils meet with basic control structures, such as sequence, conditions and cycles. Understanding and using them are the essentials of computer programming so teaching them for students is key importance. As students have to analyze algorithm presented in pseudocode in the first round of competitions, use pseudocode to present our own algorithm.

Nowadays it's not worth making algorithm without computer for a long. After knowing the basic structures we can code our algorithms in the selected programming language. Choose Pascal or C (C++) as they are the official languages of IOI. It's all the same which of them we select. Pascal is good for beginners, and after 1-2 years talented pupils can learn C (C++) language easily.

After introducing the construction of a program coded in a given language, the assignment, operators, expressions, the introduction of cycles and conditions are the following steps. Then, after demonstrating the developing environment the real programming follows. The routine exercises help students to use basic structures of computer programming at skill level. For example: writing out numbers from 1 to 10, from 10 to 1, only odd numbers from 1 to 10. Or writing out characters from A to Z, or from A to z. Determining the greatest number between two numbers read from the keyboard, determining the greatest number between three numbers read from the keyboard, or can be a triangle constructed knowing the long of its three sides, decide if a read number is prime or not, decide if read numbers are primes or not, reading for the final character, Euclidean algorithm, etc. Ring the changes on these exercises so that students will be forced to use the previously learnt structures, operators or perhaps built-in functions.

Introducing the execution of a program step by step by using the developing environment helps students to understand control structures. They can see how sequences and other control structures work. Debugging of a program makes clear of working commands, when and how variables get values and why variables have different types. In most cases it's worth changing the layout of our programs to demonstrate these things for students. For example an if-then-else construction can be written this way:

```
if a>b then c:=a Else c:=b;
```

But writing this way is better:

```
if a>b then
   c:=a
Else
   c:=b;
```

In this period of the preparation the teacher makes the computer program, students only reproduce his program. They are just typing what teacher is typing. They are rather passive recipient. However, "What if we change this ..." type of questions helps the active getting of knowledge. Finishing the coding of a program teacher can question students permanently to turn out that students really have understood the algorithm. Questions and answers could be useful for the other students as well. It's worth mistyping the program purposely, faulting syntax errors and after compiling the program teacher can represent the most common error messages for students. So they meet the most common errors at that time when teacher can help them.

At the beginning few students can write a program independently. But later more and more students can write a program. The more exercise they solve the more solutions they analyze the sooner they will be able to work alone. In this period students have own ideas and they try to code their own algorithm. In this section it will turned out who are really good at computer programming. At this time the intellectual ability of students will be seen: attention, logical thinking, abstraction and memory. In this period teacher should use differentiation. Those who finished the coding can get a new exercise, similar to the previous one. They only need to modify the previous program a little.
It is important to get on the studying principle of Pólya [4]:

- the principle of active studying,
- the principle of phase following each-other,
- the principle of the best motivation.

For self-supporting work we should give students exercises similar to the ones they have solved already in the school or later at home and they know everything to be able to solve these, and we should give exercises for more talented students which require more creativity.

In the period of preparation it is determined how students make the layout of their programs, whether they use "talking" variables or comments. Teacher should care of the readable, the layout of programs and the observance of them.

Demonstrating of spectacular programs of old students may help keeping up the interest and motivate students as well.

After knowing arrays and using text files teacher can teach a programming theorems such as theorem of searching, theorem of selection, theorem of selection of extreme, theorem of union and segment, sorting etc.) These theorems are basic algorithms as well. Students need to know them at preparedness level. The 70%–80% of exercises in the second round of competitions can be solved by using these

algorithms in 9–10$^{\text{th}}$ class. If students learn how to debug their programs they can work faster and easier. In this period programs are getting more complicated so they need to learn how they can use the facilities of the developing environment.

The first round of OKTV and Nemes Tihamér goes by without computer so this round requires the ability of analysing of students. Not only programming but analysing algorithm presented in pseudo-code is part of the preparation too so teacher should pay attention for that, however, decoding an algorithm or finding mistakes in an algorithm is easier then writing a program.

If students can use programming theorems perfectly they are able to solve exercises of previous competitions. [8], [9] Though, there is no execution time limit of these programs, students should aim to find the more efficient algorithms. Students and teacher have to analyze the running time of the variants. Filter algorithms that are not efficient and that use wrong programming structures. Before finalizing the coded program pupils need to test it. Testing is elementary.

Students should understand that solving a problem requires the following steps:

- understanding the task
- making model if it is necessary
- finding and formulating coherencies
- planning the algorithm
- proving its correctness
- analyzing efficiency
- estimating the running time and memory requirement
- coding
- testing
- finalizing

There are several exercises which require the knowing and the usage of record. If students don't use it their programs won't be efficient. When they solve problems they have to learn the writing and the usage of own functions, procedures.

At this point teacher should differentiate as most talented students can solve more and more and harder and harder exercises as opposed to the other students.

It's worth using the Internet. Students can send their solutions via by Internet (e-mail or ftp site), teacher checks it and can send back his answers before the lesson. If a student sticks in and cannot continue his work he can question the teacher by sending an e-mail.

According to the author's experience from a group, consists of 15 students, only four or five students will be able to achieve the next period. If they are more it is better for the teacher. These students require to be prepared in extra lessons and in different way. It may happen that 4–5 students solve not the same problem in lessons. Let more talented students progress faster solving more difficult problems.

2.2. Recursion and its applications

Understanding recursion and its usage are not easy for either talented students. It is elementary for students to know recursion as it is used in backtracking and graph search algorithms and in some exercises of dynamic programming. So after introduction of the theory of recursion students should solve only very simple exercises. The easiest may be rewriting a simple for cycle to a recursive procedure. For example: write numbers from 1 to 10 by using a recursive procedure. If students analyze recursive algorithms and try to code these ones they may understand the recursion better. Simple exercises of recursion:

- recursive version of programming theorems
- writing a function that returns the factorial of a number
- writing a function that raises a number to a power
- writing the Euclidean algorithm as a recursive function
- writing a recursive function that convert a number into the binary number system

Analyzing the recursive, drawing algorithms in LOGO language also helps the understanding. Programs with proper layout also help the understanding if we debug them.
We can set the layout of the recursive function this way:

```
function pow(a,b:byte):longint;
begin
if a=b then pow:=a else if a>b then pow:=pow(a-b,b) else
pow:=pow(a,b-a);
end;
```

Debugging this function of this layout shows better the execution order of commands.

```
function pow(a,b:byte):longint;
begin
if a=b then
```

```
      pow:=a
   else
      if a>b then
         pow:=pow(a-b,b)
      else
         pow:=pow(a,b-a);
   end;
```

Students should understand what values will be assigned to the local variables, when the control comes back to the following command right after the recursive calling. During debugging, this can be demonstrated quite well if the program has the proper layout.

The following two excercies are more difficult although their solutions are similar.

1) Read in value $N(< 10)$ and by using a recursive procedure display to the screen the all N^*2 bracket rounded expression, where the number of the open brackets are higher or equal than the close brackets in each position and in the solution the number of open bracket is equal to the number of close bracket. E.g.: if $N = 3$, the solutions: ((())) (()()) ()(()) (())() ()()()
2) For a given $N(< 30)$ define the all series of $2^*N + 1$ of which the first and the last item are zero, and the difference between any item next to each other is 1 or -1 and each item is a positive integer number.
   E.g.: $N = 3$

$$0, 1, 2, 3, 2, 1, 0$$
$$0, 1, 2, 1, 2, 1, 0$$
$$0, 1, 2, 1, 0, 1, 0$$
$$0, 1, 0, 1, 2, 1, 0$$
$$0, 1, 0, 1, 0, 1, 0$$

After the review of backtracking we can start with the basic exercises. The easiest one is when a program has to display all the words consisting of letters of a given word and these words have the same length as the given word. In one version letters can be used once, in the other version letters can be used more than once. This is a simple backtracking problem as the condition of the recursive calling is very simple. Basic exercises are the following:

- Eight queen problem: we need to put eight queens in the chess table that they are not allowed to hit each other.

- Knapsack problem: given a set of items, each with a cost and a value, determine the number of each item to be included in a collection so that the total cost is minimal but the total value is as large as possible. (Later we had better solve this problem by the help of dynamic programming.)

Based on the knapsack problem it is very easy to make similar backtracking exercises. If students understand the schema of backtracking they will be able to solve these kinds of problems. Backtracking algorithms are required the fence painting problem (OKTV exercise in 1996) [10] or the second exercise of the Selection Competition in 2009. [11]
Nowadays, backtracking algorithms are rare, but teaching it is very useful as we can solve several exercise by making the backtracking algorithms faster.
As exercises in OKTV have time limit, students need to learn quicksort algorithm. For didactics reason students should understand this algorithm in this period. (They can use earlier mainly C programmer.)

2.3. Graph algorithms

The most common exercises require the knowledge of the depth-first search (DFS) algorithms and its applications. In 9–10 class knowing and using DFS are enough but later students need to know the topological ordering and searching of strongly related components. For a student in the 9^{th} school year knowing and using Dijkstra-algorithm, Floyd-Warshall algorithm, searching spanning tree in a graph and searching circles in a graph are too early. Later they have to know them at a preparedness level as these algorithms are very common in the Selection Competition.

Graph can be stored in a chained list, so before teaching graph algorithms students need to learn the complex structures such as chained list, queue and stack.

For didactics reason students should learn the breadth-first algorithm. We use this if we don't need the route in the graph. Typically it occurs in OKTV with mainly the usage of condition graphs.

During one year we can get up to this point of teaching a student in the 9^{th} school year who started to study computer programming in the beginning of the year.

It is worth considering teaching greedy algorithms before graphs by the help of introducing some simple examples. In the second round they occur quite often.

2.4. Dynamic, greedy or perhaps backtracking?

Teaching dynamic programming may be the hardest task for a teacher. Only after knowing greedy algorithms and backtracking should students start to study this. It has two types that may occur: recursion with memorizing and filling-table algorithms. The last one eliminates the recursion totally.

Start to teach dynamic programming very carefully and with very base exercises. Let's see an example:

We can construct a stable tower with cubes and we cannot put a larger cube on to a smaller one, and we cannot put a heavier cube on to a lighter one. Write a program that for a given N piece of cubes calculates the highest tower used these cubes.

A solution can be the following:

For any p, q cube pair Put(p, q) is true and only true if we can put cube q on to the cube p, that is $p <> q$ and $M(p) >= M(q)$ and $S(p) >= S(q)$, if M and S mean the height of the cubes and the weight of the cubes. Magas(p) is the height of the highest tower can be built on to the cube p. Magas$(p) = 1$, if we cannot put any cube on cube p. Otherwise Magas$(p) = M(p) +$ Max{Magas$(q_1), \ldots,$ Magas(q_k)}, where $q_1, \ldots, q_k$ are all the cubes we can put on to the cube p. Notice the any $p_1, p_2, \ldots, p_k$ cubes, where cubes can be put on to each other, we cannot put p_1 on to p_k, so we can calculate Magas(p) according to the recursive definition can be calculated. The run time of this program can be exponential based on N. The reason of it is the fact that the algorithm always calculate the value of Magas(p) to the same cube if we can put cube p on to the cube q. (Simple backtracking.) We can make this algorithm faster if we have already calculated the value of Magas(p) then we store this value, we memorize this value. So if we calculate with this formula: Magas$(p) = M(p)+$Max{Magas$(q_1), \ldots,$ Magas(q_k)} then we need recursive calling if we haven't calculated the value of Magas(q_i) yet. So we store the cube q_i for each cube p if we got the maximum value for p. We use an array to store these values. The running time of this program is $O(N^2)$.

It is worth solving the problems by using dynamic programming which we solved earlier as backtracking problems. For example the running time of knapsack problem is $O(n^*m)$ if we solve it by using dynamic programming. The point of the solution [2]:

opt(x, i) means the optimal value, where x is the capacity of the knapsack, and the set of things is $\{1, \ldots, i\}$. Then opt$(x, i) =$ max(opt$(x, i-1)$, opt$(x -$ weight$[i]) +$ value$[i]$) is the solution where the weight$[i]$ is the weight of the item

of i. value[i] is the value of the item i. (Naturally $\mathrm{opt}(x,i)=0$, if $x=0$ or $i=0$, and $\mathrm{opt}(x,i)=\mathrm{opt}(x,i-1)$, if $i>1$ and weight[i] $> x$)).

The money exchange is very similar to the knapsack problem. We have N piece of coins and we have to change M amount using N coins that N should be minimal. If x means the amount and i means the number of coins, $N[i]$ is the first i pieces of coin. A following formula gives the solution [2]:

$\mathrm{opt}(x,i)=\min(\mathrm{opt}(x,i-1),1)+\mathrm{opt}(X-N[i],i-1))$. (Of course if $i=0$ and $x>0$, than $\mathrm{opt}(x,i)=N+1$, $\mathrm{opt}(x,i)=0$, if $x=0$ and $\mathrm{opt}(x,i)=\mathrm{opt}(x,i-1)$, if $x<N[i]$.)

In this period students should go on step by step. Finding out these formulas is very hard at the beginning. Teacher should collect very similar exercises for students. For example the following exercises are similar to the coin change problem: The 4^{th} exercise of the Selection Competition in 2007 [12], fair sharing exercise in CEOI in 1995. [13]

Demonstrating some exercises of dynamic programming we have to decide if it requires greedy algorithm or dynamic algorithm. As there are strict time limit in the Selection Competition if a student solves an exercise as a dynamic algorithm he loses a lot of points. Proving the correctness of an algorithm is very important.

The main difference between dynamic and greedy algorithms is:

- Greedy algorithms mostly require to sort input data and to calculate the optimum for a local item.
- In dynamic case: we have to sort input data (if it is necessary) and for each item of the sorted items we calculate the local optimum and the program consists of the storing the selected local optimums and of using the previously selected optimums.

Exercises which are similar to each other help student to memorize algorithms, formulas. After solving similar problems students will remember the exercise he solved earlier when he participates in the Selection Competition. "There is no new thing under the Sun" – hope students. If there is, they use their creativity and their routines.

2.5. Other algorithms

The last years geometrical exercise occurred quite often. For solving these problems students need to know the following basic algorithms [1]:

- linking points to be closed, no intersection polygon

- determining the position of points
- searching line segment intersection
- white-black paring in flat
- the fares pair of point of a set of points

Up to now interactive task occurred only in the selection competitions. Most of them can be solved by using former algorithms, but the students need to know how they can use included libraries and units. In the Selection Competition students will not have time to test using of units. Solving 6 or 7 exercises of previous competitions (or some task of CEOI) is enough for the proper preparing.

3. Holiday

After the competition period students require to have a rest. The author's students solve only one task a week from April until the beginning of June and they have a full rest for one month. In this period it is good for students to learn another programming language or write programs that are solutions of noncompetition tasks. They prefer visual programming or web programming.

A holiday is very useful during the preparation as students don't have to study other subjects they have enough time. They need to do only one thing that they like: programming

The author's students have a lesson on one time per week. Before the lesson students get exercises by e-mail. They have to understand the problems and try to find a suitable algorithm for these problems. They come to the lesson to be prepared and they have ideas for the solutions of the problems. In the lesson we discuss their ideas with the official or the teacher's solutions which are elaborate.

Students code algorithms at home and send them via by Internet. Mistakes they made will be discussed in the following lessons. This is the end of a solution of a problem. Students can memorize these problems and solutions better than dealing with a problem only in one day.

It is very useful if older students give lessons and introduce some problems and their solutions.

If students solve all the exercises of OKTV and all the tasks of selection competitions they are ready to solve tasks of IOI. In most cases official solutions are available which help students quite a lot.

During the school year we can organize test competitions lasting 2–3 hours before rounds awards such as chocolate. It is also useful if students participate in on-line contests such as Croatian Open Competition in informatics. [14]

4. Differentiation

When teacher consider the meaning of differentiation, its responsibility or the regular tasks belonging to differentiation, they disclaim it. However, the whole differentiation is the part of the profession. Of course, it shouldn't appear in each lesson and for each student. [3] Teaching a student at this high level is not possible without differentiation. Teacher needs to use differentiation for students if they have almost the same abilities as they will not solve problems at the same time. One of them makes more mistakes others make less. Each student requires different preparation. Of course it is not totally different. Teacher needs to realize the weakness of students and eliminate these weaknesses.

5. Summary

The main aim of this paper was to demonstrate what to teach and how to teach computer programming for students who want to be the members of the Hungarian Olympic Team. In order to be able to solve complex task, students have to become familiar with the necessary algorithms. This article has summarized these algorithms. Moreover, students need to learn how they can analyze their algorithms. It can be seen that students have to work a lot to reach their goals. Being talented is not enough. In competitions students have no time to find out new algorithm. They have to use and modify what they have already learnt and used a lot.

This article has given methods how to teach computer programming for competitors. At the beginning the cycles and conditions are hard to understand for students. If they can understand and use them we can go on teaching algorithms.

When students start to work alone and are able to code programs the differentiation and the motivation are elementary. If they are familiar with recursion and backtracking, the understanding and the using of the graph algorithms will be trivial for them. The last topic which is hard to understand is dynamic programming. They need to solve very similar exercises to be able to find out formulas for the solutions.

Giving a solution for a problem is not enough. Students, if they want to be successful, have to learn that after proving the right and efficient of their solutions they have to code their algorithms without mistakes and they have to test their programs. Only one mistake, only one inefficient solution and they may be unsuccessful. Teachers can help them to be successful and to be the members of the Olympic Team.

References

[1] Gy. Horváth, *IOI-CEOI felkészítő: Geometriai algoritmusok*, 2006.

[2] Gy. Horváth, *IOI-CEOI felkészítő: Dinamikus programozási feladatok*, 2005.

[3] J. Ollé and J. Szivák, *Mód-Szer-Tár*, Okker Kiadó, 2006.

[4] Gy. Pólya, *A gondolkodás iskolája*, Gondolat Kiadó, Budapest, 2007.

[5] J. S. Renzulli, *What Makes Giftedness?*, 1978.

[6] L. Tóth, *A tehetségesek tanítása*, Kossuth Egyetemi Kiadó, Debrecen, 2000.

[7] L. Tóth, Tehetségdefiníciók, *Tehetség,* **XVI**, no. 2, 3–4.

[8] L. Zsakó and Gy. Horváth (eds.), *Programozási versenyfeladatok tára I. II. III.*, NJSZT, 2007.

[9] `http://nemes.inf.elte.hu`.

[10] `http://nemes.inf.elte.hu/1996/nt96-2f3.doc`.

[11] `http://tehetseg.inf.elte.hu/valogatok/2009/f1.doc`.

[12] `http://tehetseg.inf.elte.hu/valogatok/2007/f2.doc`.

[13] `http://www.infoera.hu/infoera2004/eaok/horvathgyula.pdf`.

[14] `http://www.hsin.hr/coci/`.

SÁNDOR KIRÁLY
FAISKOLA U. 24/B
3300 EGER
HUNGARY

E-mail: `ksanyi@nejanet.hu`

(Received June, 2010)