Olympiads in Informatics, 2011, Vol. 5, 71–81
© 2011 *Vilnius University*

Stimulating Students' Creativity with Tasks Solved Using Precomputation and Visualization [*]

Tomasz KULCZYŃSKI, Jakub ŁĄCKI, Jakub RADOSZEWSKI
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
e-mail: t.kulczynski@students.mimuw.edu.pl, {j.lacki,jrad}@mimuw.edu.pl

Abstract. We analyze types of tasks in which the (easiest) solutions require performing experiments. Good visualization of the results of such precomputation helps to make the key observations required to solve the task. We also discuss the interconnections between such task types and experimental research, including the issue of computer-aided proofs.

Key words: programming contests, precomputation, visualization, task development.

1. Introduction

Solving a task at a programming contest usually requires inventing an algorithm, implementing it using a chosen programming language and testing the program properly. In the first of these phases, the contestant usually performs several observations – scientific discoveries – some of which she combines to create the actual algorithm. There are several ways in which the aforementioned observations can be made: this could be purely mathematical reasoning, playing with sample data, or analyzing the results of some computer experiments. This paper attempts to analyze and classify types of problems, in which precomputation and good visualization of experimental data provides the key to the solution.

So far there has been a number of publications considering *new* types of problems, not known to the broad IOI community or not used at the IOI competition. Particular examples were mentioned by Kemkes *et al.* (2006) – open-ended tasks, Truu and Ivanov (2007) – testing-related tasks, Burton (2010), Dagienė (2006, 2010), Kubica and Radoszewski (2010) – algorithmic puzzles solved without using a computer; additionally, Burton (2008) and Opmanis (2009) each presented several novel problem types. In this paper we consider a type of problems which is already present in the IOI and national olympiads in informatics, which we claim to be underrepresented in comparison to its connections with computer science research and building actual computer software (more discussion can be found in the Conclusions). Thus our approach is somewhat similar to the one used by Ninka (2009), who considered reactive and game tasks at programming contests. Ribeiro and Guerreiro (2007) discussed a related topic of applications of graphical user interfaces and computer graphics in programming contests tasks; we, however,

[*]The authors thank an anonymous referee for helpful suggestions on the presentation of the paper.

consider graphical techniques applied not in the problem statement or the interface for the contestant's program, but in the process of creating the algorithm used in the solution.

Below we consider several examples of tasks solved using precomputation and visualization. For each of them we describe the tool (usually a single auxiliary program) used to obtain the experimental data, and we show how the analysis of this data leads to a solution of the initial problem. There are cases when the observations made during this initial part of solving the problem require a proof, however, notably, in some (not necessarily trivial) cases one can utilize the experimental data to design a formally correct solution even without the necessity of any additional proof. The level of complexity of the visualization varies from outputting elements of a number sequence to drawing more complicated arrays and figures in a plane.

2. Generating Number Sequences

In this section we show three problems which consist in computing elements of number sequences. We could be asked to compute the nth element of a sequence for a given n, or the last index of an element of a sequence which does not exceed a given threshold c, or the number of elements of the sequence from a given interval $[a, b]$ etc. Here we can utilize the simplest precomputation and textual visualization technique, i.e., generating and outputting consecutive elements of a sequence, and some variants of this technique.

2.1. *Finiteness of a Sequence: Right-Truncatable Primes*

Problem. We consider right-truncatable primes, i.e., prime numbers for which every non-empty prefix is also a prime number (Sloane, the sequence A024770). More precisely, these are primes in which repeatedly deleting the least significant digit gives a prime at every step until a single digit prime remains. For example, 293 is a right-truncatable prime, while 223 is not despite being prime itself. We are asked to find the number of right-truncatable primes in a given interval $[a, b],\ b \leqslant 10^{18}$.

This problem was used at the 5th Polish Junior Olympiad in Informatics (V OIG).

Precomputation. We need to find a method of generating consecutive right-truncatable primes. Let us note that every right-truncatable prime having at least 2 digits is an extension of a right-truncatable prime by a single least significant digit. This yields a recursive algorithm which can be implemented using a queue. We start by inserting all single-digit primes into the queue (see Fig. 1a). Then in every step we extract the first element p from the queue and check if any of the numbers $10p + i$, for $i = 1, 3, 7, 9$, is prime (see Fig. 1bc). If so, we obtain another right-truncatable prime and put it to the end of the queue.

The time complexity of this algorithm is $O(K\sqrt{M})$, where K is the number of right-truncatable primes generated and M is the greatest of them. We can run such an algorithm for K and M which are "reasonably small" and output the elements of the sequence computed. It appears that the algorithm terminates with an empty queue, i.e., there are only 83 right-truncatable primes, and the largest of them is 73 939 133.

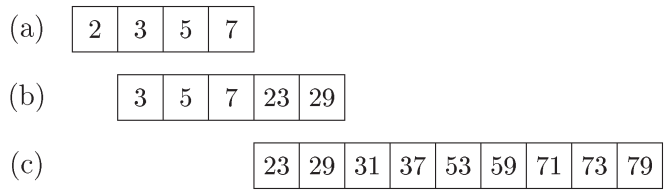

Fig. 1. The queue with right-truncatable primes: (a) the initial queue; (b) the queue after 2 is processed; (c) the queue after all single-digit right-truncatable primes are processed.

Solution. It suffices to hardcode all 83 right-truncatable primes into the solution. Such a solution works in constant time. Another option is using the above described precomputation tool as the solution. With a reasonable implementation it also works sufficiently fast.

2.2. *Sparsity of a Sequence: Antiprimes*

Problem. An antiprime number (also referred to as a highly composite number) is a positive integer for which the number of divisors increases to a record (see Sloane, the sequence A002182). In other words, m is antiprime if $d(m) > d(k)$ for all $k < m$, where $d(m)$ is the number of divisors of m. A first few elements of this sequence are 1, 2, 4, 6, 12, 24, 36, 48, 60, 120, 180. Our task is to write a program that computes the greatest antiprime which does not exceed a given threshold $c,\ c \leqslant 2\,000\,000\,000$.

This problem was used at the first stage (problems solved by students at home) of the 8th Polish Olympiad in Informatics (Rytter, 2001).

Precomputation. We would like to use the same approach as before. The basic algorithm consists in iterating over consecutive positive integers and counting their factors. This yields $O(M\sqrt{M})$ time, where M is the greatest integer considered. Using Eratosthenes' sieve, one can improve this algorithm to $O(M \log M)$ time, see the following section.

Running such a program for $M = 10^6$, we obtain 38 antiprimes. From this we guess that the distribution of antiprimes is very sparse. Hoping to be able to preprocess all antiprimes in the interval $[1, 2 \cdot 10^9]$, we run the precomputation tool for $M = 2 \cdot 10^9$ and wait for a couple of minutes (hours or days, if the less efficient preprocessing algorithm was used). Thus we obtain exactly 68 antiprimes, which is a number of the expected order of magnitude.

Solution. Again, the solution simply iterates over a hardcoded array of integers in constant time. This time, however, the preprocessing time was much greater than previously. If c was of a larger magnitude, e.g., 10^{18}, the described approach would not end up successfully. In this case an efficient solution can be obtained by limiting oneself only to a small (but sufficient) number of candidates for antiprimes, i.e., integers of the form

$$2^{a_2} \cdot 3^{a_3} \cdot 5^{a_5} \cdot 7^{a_7} \cdot \ldots, \quad \text{where} \quad a_i \geqslant a_j \text{ for } i \leqslant j.$$

2.3. *Utilizing Asymptotics of a Sequence: EKG Sequence*

Problem. Consider an integer sequence defined as follows: $a_1 = 1$, $a_2 = 2$ and a_n for $n \geqslant 3$ is the smallest positive integer not used previously which shares a factor with a_{n-1} (Sloane, the sequence A064413). The first 10 elements of this sequence are: 1, 2, 4, 6, 3, 9, 12, 8, 10, 5. We are to compute the nth element of this sequence, for a given $n \leqslant 1\,000\,000$.

This problem is studied by Lagarias *et al.* (2002) and Hofman and Pilipczuk (2008), it was also used at East Central North America 2003 regional team programming contest.

Precomputation and Visualization. The experimental tool will work in a similar way as previously. Obviously, its output cannot be simply hardcoded into the final solution, so this time we need more complex observations.

The algorithm could be as follows. We store a dictionary of already used elements, and for every $n = 3, 4, \ldots$ when computing a_n we iterate over consecutive positive integers, omitting the previously used ones. To check the common-factor condition we apply Euclid's gcd algorithm. The time complexity of such approach is $O(N \cdot M \cdot \log M)$, where N is the greatest index considered and M is the maximum value of an element of the sequence computed.

We run this algorithm, e.g., for $N = 1\,000$. Analyzing such a prefix of the sequence, first of all we note that all its elements are rather small: the greatest of them is $1\,563$. More involved observations can be made using a visualization of the computed elements, see Fig. 2. From this graph we can conjecture that $a_n \sim n$, $a_n \sim \frac{3}{2}n$, or $a_n \sim \frac{1}{2}n$. Surprisingly enough, these observations are sufficient to construct an efficient algorithm for computing the elements of the EKG sequence.

Solution. Assume that we wish to compute a_n for all $n \leqslant N$. We will show how to utilize the observations to improve the effectiveness of computing a_n from a_{n-1}. This

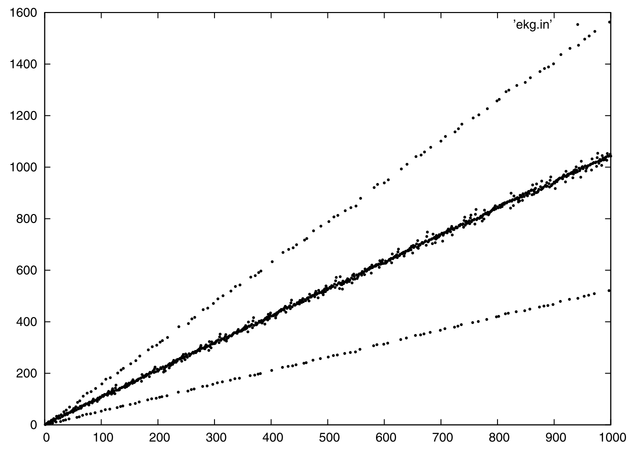

Fig. 2. A graph of the first 1 000 elements of the EKG sequence.

time we iterate over all prime factors of a_{n-1}. If we knew, for every prime number, what is its smallest multiplicity not present in the prefix of the sequence, we would choose the prime factor of a_{n-1} with the smallest such multiplicity.

Here we use the observation related to the asymptotics of a_n: we assume that all elements of the EKG sequence do not exceed $2N$. Hence, to find all distinct prime factors of any a_n, we can use Eratosthenes' sieve, which after initialization in $O(N \log \log N)$ time allows factorization of integers in $O(\log N)$ time.

For each prime number in the range $1 \ldots 2N$ we store the smallest of its multiplicities which was not yet used in the sequence. After we compute a term a_n, we update such values of all prime factors p of a_n. Note that updating the value for p can take several steps, each of which can be performed in constant time if we use a Boolean array to store elements already present in the sequence. The total number of such steps is $O(N \log N)$, which yields the time complexity of the whole algorithm.

Let us recall that the observation we made about asymptotics of the EKG sequence is only a conjecture. However, we do not need to prove it at all. Indeed, it suffices to write a program utilizing this observation, checking if we do not exceed the $2N$ range when searching for the requested term of the sequence. If this condition holds for the input data, the output of the program is correct. And this is the case here, the conjecture holds for $N \leqslant 1\,000\,000$.

3. A Squirrel in a Plane

In this task our goal is to simulate a process, which takes place on grid points in a two dimensional infinite plane. This task is almost impossible to solve without any visualization. Moreover, making a good visualization requires using a drawing utility.

Problem. Consider an infinite plane with several nuts placed in its grid points and a squirrel, which collects them. It starts in the origin facing north, and in each unit of time it performs a move in the following way:

- If there is a nut in its current location then it picks it up, turns 90° right and walks 1 unit straight ahead.
- Otherwise, the squirrel drops a nut in its current location (we assume that the squirrel has a sufficient number of nuts each time), turns 90° left and also walks 1 unit straight ahead.

Given an initial arrangement of n nuts in the plane (their coordinates x_i, y_i are integers and satisfy $-2 \leqslant x_i, y_i \leqslant 2$), we are to find the number of nuts that will lie on the ground after t units of time ($n \leqslant 7$, $t \leqslant 10^9$).

This problem was used at a training camp for Polish contestants in 2007.

Visualization. There seems to be no apparent way of solving the problem. Hence, we will try to find some particular pattern of the squirrel's moves (i.e., a cycle or a big number of steps resulting in a small change in positions of the squirrel and nuts). In order to do this, we simulate the moves of the squirrel for a few different inputs. It suffices to write a program that works for much smaller values of t and then visualize the moves using a graphics tool. For each tested input, the picture looks similar: first, the squirrel moves around the origin for some number of steps (about a few thousand steps in most cases), then it begins to go in one direction by repeating a sequence of moves, see Fig. 3 and 4.

Now we make the assumption that the same holds for every other input. To construct a solution, we still need to know what is the repeated sequence of moves and when the squirrel starts to repeat it. By looking at the pictures we have made, we conjecture that the sequence is always the same. Hence, we search for it by simulating the squirrel's moves for one initial configuration.

We check possible lengths of the sequence in an increasing order. Given a candidate length l, one can easily check whether (after some initial movements) the squirrel repeats a pattern of l steps. We discover that the repeating sequence consists of 104 moves, in which 12 new nuts are put on the ground.

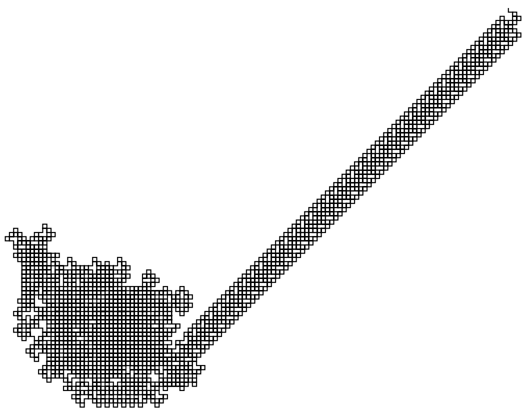
Fig. 3. The first 14 000 moves with initially empty plane.

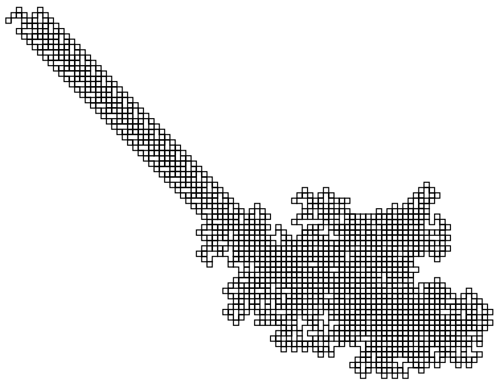
Fig. 4. The first 11 000 moves with some other initial configuration.

The same precomputation tool can be used to check how many moves are made before the first occurrence of the considered sequence. It turns out that the number of moves varies, but the squirrel always moves close to the origin. Basing on the results of experiments, we assume that when the squirrel reaches the distance of 200 units from the origin (in the maximum metric), it has already begun to repeat the sequence of moves.

Solution. We write a program that simulates the moves of the squirrel. The simulation stops after x steps, if *any* of the following holds:

- the squirrel has reached the distance of 200 units from the origin at least once and $t - x$ is divisible by 104,
- $t = x$.

We output the number of nuts on the ground after x moves increased by $\frac{t-x}{104} \cdot 12$.

As we do not have a proof, we cannot be certain if this solution always produces correct answers or if it works efficiently enough (i.e., what is the number of simulation steps we run in the worst case). Instead of a formal proof, we can check, for all possible initial configurations of nuts (there are only 726 206 such configurations, avoiding rotations and reflections one can reduce this number significantly), if each of them eventually has the same repeating sequence of 104 moves and if this sequence starts repeating fast enough. It takes a few minutes to verify this, and afterwards we are certain of the correctness of the solution.

4. Playing Games

Visualization is a common approach to a majority of problems related to deterministic games. In the examples below, the data obtained by precomputation lets us obtain a correct solution. However, to be fully convinced of its correctness one needs to formally prove the utilized observations.

4.1. *Two-Dimensional Precomputation: Number Game*

Problem. Consider a two player game. The first player receives a pair of positive, relatively prime integers (i, j). In one turn a player chooses one of the numbers (which has to be greater than 1), subtracts 1 from it and then divides both numbers by their greatest common divisor. The player who cannot make a move, that is, gets a $(1, 1)$ pair, loses the game.

The task is to determine if a given position is winning.

Visualization. With a simple dynamic programming, we obtain a Boolean matrix $a[i, j]$, such that $a[i, j] = 1$ iff a position (i, j) is winning.

We immediately see that $a[i, j] = 0$ iff i and j have the same parity. Once we have this hypothesis, we can prove it easily.

If both numbers are of the same parity, subtracting one from either of them leads to a position, in which the numbers have different parity. It follows that the greatest common

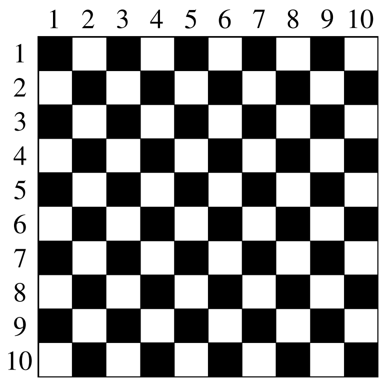

Fig. 5. Number game: a matrix describing winning positions. A cell $a[i, j]$ is white iff a position (i, j) is winning.

divisor is odd, so we end up in a position (i', j') where $i' \not\equiv j' \pmod 2$. On the other hand, given a position with an even and an odd number, one can subtract 1 from the even number. As a result, both numbers will become odd, so the opponent's position will be losing.

This yields a simple way of checking whether a given strategy is winning as well as playing the game optimally.

4.2. *One-Dimensional Precomputation: Rectangle Game*

Problem. The board in this game is a $x \times y$ ($1 \leqslant x, y \leqslant 2 \cdot 10^9$) rectangle composed of unit squares. There are two players. In one move a player makes a single horizontal or vertical cut that produces two rectangles of integer dimensions. After each cut the rectangle with smaller area is discarded (or an arbitrary one in case of a tie). The player who gets a 1×1 board loses the game. The task is to write a set of functions that will implement an optimal strategy of the first player, but for the sake of simplicity we will only describe how to check whether a given position is winning.

This problem was used at the 17th International Olympiad in Informatics (Radoszewski, 2005).

Analysis. A straightforward visualization would be a Boolean matrix, such that $a[i, j] = 1$ iff an $i \times j$ rectangle is a winning position. However, in this problem some initial analysis can simplify the task significantly. We observe that the game is equivalent to a Nim game with two piles containing x and y stones, with the restriction that a player is allowed to take at most half of all stones from one pile. Hence, we can use Sprague-Grundy theorem (see Ferguson, 2005) and consider each pile separately.

The position in the game with one pile is simply the current number of stones. For each position p, we compute its *nimber*. The nimber is equal to 0 for losing positions. For winning positions it is the smallest integer which is not a nimber of any position that can be reached from p in one move. By Sprague-Grundy theorem, given the nimbers of both piles, a position in a two dimensional game is losing if and only if the bitwise xor of those nimbers is equal to zero. Since there are only two piles, this is equivalent to equality checking.

Precomputation. Having reduced the problem to one dimension, we find the nimbers of a few positions:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Nimber	0	1	0	2	1	3	0	4	2	5	1	6	3	7	0	8	4

We now conjecture that the nimber of position $2n$ is n and that nimbers of positions $n, n+1, \ldots, 2n-1$ form a permutation of integers from 0 to $n-1$. We prove both of these claims inductively.

The basis holds trivially. Now assume that the nimbers of positions $n, n+1, \ldots, 2n-1$ are distinct integers in the range $[0, n-1]$. Given a pile with $2n$ stones, the reachable positions are exactly $n, n+1, \ldots, 2n-1$. We know that the smallest number which is not a nimber of those positions is n. Hence, the nimber of position with $2n$ stones is n. If there are $2n+1$ stones on a pile, then every position with $n+1, n+2, \ldots, 2n$ stones is reachable. The nimbers of those positions form a set of n distinct integers from the range $[0, n]$. It is easy to observe that the smallest number excluded from this set is the nimber of the position n. This completes the proof.

A simple formula for nimbers follows from the proof:

$$\text{nimber}(n) = \begin{cases} 0, & \text{if } n = 1, \\ n/2, & \text{if } n \text{ is even}, \\ \text{nimber}((n-1)/2), & \text{if } n \text{ is odd}. \end{cases}$$

Solution. Using the aforementioned formula, one can compute nimbers for an $x \times y$ board in $O(\log x + \log y)$ time. By comparing the nimbers for x and y, we check if the position is winning.

5. Conclusions

In this paper we present several examples of problems which can be solved using precomputation and visualization techniques. In some of these tasks it suffices to perform a simple preprocessing of terms of a number sequence, but in other cases one requires a more complex approach to visualization, including the usage of drawing tools. In each case we put an emphasis on obtaining a provably correct solution. We claim that randomly guessing a solution without any justification is not a good practice and should not be popularized among contestants.

As we already mentioned, experimental approach to problem solving is highly related to computer science research. The results of such experiments can be used to better understand the approached problem and state good conjectures. As a basic example, most of the known properties of the aforementioned EKG sequence were first conjectured using graphical visualization of the experimental data, and their proofs were discovered only after (see Lagarias *et al.*, 2002; Hofman and Pilipczuk, 2008). In such cases computer science serves as an experimental tool for mathematics. Notably, sometimes the connection between the two branches is even closer: it happens that the results of computer experiments form a part of the proof of the conjectured property. Again, the EKG sequence provides a good example: the proof of the fact that each prime number p appears in the EKG sequence in a fragment $2p, p, 3p$ works only for $p > 25\,000$, while for all the remaining primes this conjecture was verified experimentally (Hofman and Pilipczuk, 2008). A much more famous example is the proof of the fact that every planar graph can be colored with only 4 colors. The proof is composed of a large number of cases which were verified by a computer. Finally, the ability to analyse large amounts of data and conclude about the structure of the data is very useful for software engineers. In conclusion, we claim that problems requiring good precomputation and an analysis of its results are a valuable part of programming contests.

References

Burton, B.A. (2008). Breaking the routine: events to complement informatics olympiad training. *Olympiads in Informatics*, 2, 5–15.

Burton, b.a. (2010). encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.

Dagienė, V. (2006). Information technology contests – introduction to computer science in an attractive way. *Informatics in Education*, 5(1), 37–46.

Dagienė, V. (2010). Sustaining informatics education by contests. In: *4th International Conference on Informatics in Secondary Schools – Evolution and Perspectives*, *LNCS*, 5941, 1–12.

Ferguson, T.S. (2005). *Impartial Combinatorial Games*, class notes.
`www.math.ucla.edu/~tom/Game_Theory/comb.pdf`.

Hofman, P., Pilipczuk, M. (2008). A few new facts about the ekg sequence. *Journal of Integer Sequences*, 11.

Kemkes, G., Cormack, G., Munro, I., Vasiga, T. (2006). New task types at the canadian computing competition. *Olympiads in Informatics*, 1, 79–89.

Kubica, M., Radoszewski, J. (2010). Algorithms without programming. *Olympiads in Informatics*, 4, 52–66.

Lagarias, J.C., Rains, E.M., Sloane, N.J.A. (2002). The EKG sequence. *Experiment. Math.*, 11 (3), 437–446.

Ninka, I. (2009). The role of reactive and game tasks in competitions. *Olympiads in Informatics*, 3, 74–79.

Opmanis, M. (2009). Team competition in mathematics and informatics "ugāle" – finding new task types. *Olympiads in Informatics*, 3, 80–100.

Ribeiro, P., Guerreiro, P. (2007). Increasing the appeal of programming contests with tasks involving graphical user interfaces and computer graphics. *Olympiads in Informatics*, 1, 149–164.

Radoszewski, J. (2005). Rectangle game. In: Kubica, M. (Ed.), *IOI'2005. Tasks and Solutions*, 43–49.

Rytter, W. (2001). Liczby antypierwsze (Antiprime numbers). In: Diks, K., Waleń, T. (Eds.), *VIII Olimpiada Informatyczna (VIII Polish Olympiad in Informatics)*, 41–44 in Polish.

Sloane, N.J.A. Sequences A024770 (right-truncatable primes), A002182 (highly composite numbers), and A064413 (EKG sequence). In: *The On-Line Encyclopedia of Integer Sequences*. `oeis.org`.

Truu, A., Ivanov, H. (2007). On using testing-related tasks in the IOI. *Olympiads in Informatics*, 2, 171–180.

T. Kulczyński (1988), student at Faculty of Mathematics, Informatics and Mechanics of University of Warsaw, winner of IOI'2007 and medalist of multiple international informatics olympiads, deputy leader of the Polish team at the IOI 2008–2010, former member of the HSC of BOI'2008 in Gdynia, Poland.

J. Łącki (1986), PhD student at Faculty of Mathematics, Informatics and Mechanics of University of Warsaw, former member of the HSC of BOI'2008 in Gdynia, Poland, organizer of training camps for finalists of POI.

Jakub Radoszewski (1984), PhD student at Faculty of Mathematics, Informatics and Mechanics of University of Warsaw, chairman of the jury of Polish Olympiad in Informatics, Polish team leader at IOI 2008–2010, former member of the HSC of IOI'2005 in Nowy Sącz, CEOI'2004 in Rzeszów, and BOI'2008 in Gdynia, Poland. His research interests focus on text algorithms and combinatorics.