

Olympiads in Informatics

5

A large, semi-transparent graphic of a stethoscope is positioned in the center-right area of the page. The stethoscope has two blue earpieces and a silver chest piece. It sits atop a dark blue rectangular background that contains the IOI logo and text.

IOI
INTERNATIONAL OLYMPIAD IN INFORMATICS

ISSN 1822-7732

**INTERNATIONAL OLYMPIAD IN INFORMATICS
VILNIUS UNIVERSITY
INSTITUTE OF MATHEMATICS AND INFORMATICS**

OLYMPIADS IN INFORMATICS

Volume 5 2011

Selected papers of
the International Conference joint with
the XXIII International Olympiad in Informatics
Pattaya City, Thailand, July 22–29, 2011



OLYMPIADS IN INFORMATICS

ISSN 1822-7732

Editor-in-Chief

Valentina Dagienė

Vilnius University, Lithuania, valentina.dagiene@mii.vu.lt

Executive Editor

Richard Forster

British Informatics Olympiad, UK, forster@olympiad.org.uk

International Editorial Board

Gerald Futschek, Vienna University of Technology, Austria, futschek@ifs.tuwien.ac.at

Bruria Haberman, Holon Institute of Technology, Israel, habermanb@hit.ac.il

Marcin Kubica, Warsaw University, Poland, kubica@mimuw.edu.pl

Ville Leppänen, University of Turku, Finland, villelep@cs.utu.fi

Krassimir Manev, Sofia University, Bulgaria, manev@fmi.uni-sofia.bg

Rein Prank, University of Tartu, Estonia, rein.prank@ut.ee

Peter Taylor, University of Canberra, Australia, pjt013@gmail.com

Troy Vasiga, University of Waterloo, Canada, tmjvasiga@cs.uwaterloo.ca

Peter Waker, International Qualification Alliance, Republic of South Africa,
waker@interware.co.za

http://www.mii.lt/olympiads_in_informatics

Foreword

This volume of *Olympiads in Informatics* marks the journal's first lustrum – five years of research and reports on topics of interest to national and international informatics olympiads and contests. Pedagogical, sociological and technological. Each volume represents a year's work by the authors, reviewers and editors – a lustrum's work published in an olympiad of time. A lustrum was also the period in Ancient Rome when a census was taken of the populace and a ritual cleansing of the state performed. A time for looking at what had gone on before and what would be changing for the future.

In the introduction to the first volume we wrote “*... we communicate during ... national and international events, but too frequently these are conversations between small groups, if not individuals, and often such conversations are piecemeal and quickly forgotten ... We hope that this [journal] will be a benefit, not just to the IOI community, but to the wider community of educators in our field.*” Have we been a success? Certainly within the IOI community we now have a published body of literature (over 75 papers) covering a wide range of national and international issues connected with contests in informatics and computer science education research.

Recognition outside the community has perhaps been slower in coming – most papers are primarily authored by those within it – but gradually we are appearing in scientific databases and being referenced externally. Beginning with this volume, papers will be indexed in EBSCO and CEEOL.

The editorial board is acutely aware that to attract good papers, authors need to know that journal is academically recognised, yet at the same time is accreditation requires a journal of good research papers. We are all fortunate that we have academics within our community who are willing to submit high quality material and the editorial board continues to actively discuss and pursue means of moving the journal up through the journal rankings.

When we started the journal and associated conference we had originally intended to have a main topic each year. Firstly we focused on the experiences of running national olympiads. In our second year training and tasks were the focus. For the last few years we have not had a main topic for the year. Issues of national olympiads, training and tasks have continued strongly over those years, along with papers on the technologies that underlie our events and, more generally, teaching and curricula across the globe.

Our shared experiences in running national olympiads continue to be of interest and we have moved away from reports primarily about the structure of such contests. National reports continue however to be of interest – our problems are common problems and numerous papers have bemoaned, for example, the lack of state teaching for informatics.

The format for the journal has undergone some changes for this year's volume; our lustrum if you will. We feel it is important that serious research forms the foundation of

the journal and such papers form the primary section of the journal. We also understand the need for continuing to share our national experiences, potentially of less interest to those outside the community, so have introduced a country reports section. Finally, this volume presents some book or task solution reviews which we hope will be of interest to our readership. We would like to encourage to write short comments, opinions and challenges – it will be useful for everybody within community.

As always thanks are due to all those who have assisted with the current volume – authors, reviewers and editors. A lot of work goes, not only to the writing of the papers, but to an extended period of review and correction and, in several cases, translation. Peer reviewing all of the papers takes a significant amount of time and work and special thanks should be given to those otherwise unsung reviewing heroes: Ian W. Atha, Giorgio Casadei, Sébastien Combéfis, Gordon V. Cormack, Hugo Duenas, Walter Erquímigo, Gerald Futschek, Sari Haj Hussein, Rob Kolstad, Ville Leppänen, Marcin Kubica, Krassimir Manev, Timo Poranen, Rhein Prank, Miguel Revilla, Peter Taylor, Troy Vasiga, Willem van der Vegt, Tom Verhoeff and Peter Walker.

Last, but by no means least, particular thanks are due to the organisational committee for IOI'2011 in Thailand without whose assistance we would be unable to hold the conference. Their assistance, during what is an already busy period, is gratefully received.

Editors

Algorithmic Problem Solving and Novel Associations

David GINAT

*Tel-Aviv University, Science Education Department
Ramat Aviv, 699978 Tel-Aviv, Israel
e-mail: ginat@post.tau.ac.il*

Abstract. We elaborate on the essential role of novel associations between recognized task patterns and invoked algorithmic schemes, during algorithmic problem solving. We display three algorithmic tasks of different levels of difficulty, and characterize them by their required pattern-scheme associations. We display diverse student solutions to the tasks, which reflect different levels of competence; and suggest a series of considerations of which tutors should be aware upon selecting and posing algorithmic challenges to students.

Key words: problem solving, patterns, algorithmic schemes.

1. Introduction

The domain of algorithmic problem solving involves tasks of different levels of difficulty. The less challenging tasks involve straightforward analysis and familiar utilizations of algorithmic schemes (templates, or design patterns; Astrachan *et al.*, 1998; Linn and Clancy, 1992). The more challenging tasks require insightful observations and more involved utilizations of such schemes. A primary objective of a tutor who poses more challenging tasks is to examine whether problem solvers reach suitable observations, and associate them with corresponding compositions of algorithmic schemes. The corresponding compositions may derive from novel associations between observed task patterns and the composed schemes.

For example, one basic, familiar scheme is that of Binary Search, which is commonly employed on an ordered list of input values. Yet, it may be used in less familiar ways, such as searching in a range of successive integer values (e.g., a sequence of guesses, in which the answer for each guess is “got it”, “too low”, or “too high”). The less familiar ways may be obvious to an experienced problem solver, but may require some novel associations from an inexperienced one, who is only acquainted with Binary Search on a list.

In the design of challenging tasks, which are aimed for competitions of algorithmic problem solving, a task designer would usually like to examine contestants’ original associations, in less familiar or even novel ways. Such associations are essential components of competition tasks. The objective of this paper is to elaborate on this notion through some examples, and characterize differences between problem solvers who reach such associations and problem solvers who do not.

In the next section we display three tasks that require associations, which in our experience are less familiar to students. We usually pose these tasks during the beginning of our second stage (top 40 students) of our Olympiad activities. The tasks are used for both evaluating the students' problem solving abilities and teaching them implicit elements of problem solving. The students offer solutions that involve different levels of competence, which reflect various associations (or lack of associations) between observed task patterns and employed algorithmic schemes. We display for each task its diverse student solutions.

We conclude with a discussion section, in which we suggest a series of considerations that arise from the displayed tasks and the student solutions. Tutors' awareness of these considerations may assist teaching and learning in general, and the design, selection, and characterization of challenging tasks in particular.

2. Novel Associations in Algorithmic Tasks

In what follows, we present three algorithmic tasks, of different levels of difficulty, which require novel associations. In the solution of each task, one has to recognize patterns on which to capitalize, invoke a suitable algorithmic scheme, and employ it in a rather less familiar way. The difficulty of each task stems from the task's hidden patterns and the novel associations required for invoking and employing the suitable algorithmic schemes. The more difficult it is to recognize the patterns and yield the associations, the harder the task.

The first task involves a simple pattern and simple scheme utilization. The second involves a hidden pattern and a sub-component of a familiar (though slightly subtler) scheme. The third task also involves a hidden pattern and a feature of a familiar scheme, but the feature is an *implicit* characteristic of the scheme, rather than an explicit sub-component. Each task is presented in a separate subsection. After the presentation of each task, we first describe our experience with unsuitable allies that were followed by some of the students, and then present the suitable solutions, obtained by other students, which derive from novel associations.

2.1. Simple Patterns and Simple Schemes

The following task is known as the Longest Stuttering Subsequence Problem. One solution of the task is rather simple, and another is much more involved (Mirzaian, 1987b). We aimed at the simple solution. Although simple, it still requires some association, which in our experience is not straightforward to all problem solvers.

Longest Stuttering Subsequence. Given a string A, of length n , and a string B, of length m , where $m \leq n$; output an integer k , which indicates the largest stuttering of B in A. We say that B is *stuttered* in A j times, if A contains j (not necessarily adjacent) appearances of the 1st symbol of B, followed by j appearances of the 2nd symbol of B, . . . followed by j appearances of the last symbol of B.

For example, if A is 002332256233263 and B is 23, then the longest stuttering of B in A is 3. If B is 236, then the longest stuttering is 1.

It is clear that the output cannot exceed the value of n/m . In our experience, students approached this task in three different ways. They all noticed the stuttering characteristics that, “if there is no stuttering of size k , then there is no stuttering of size larger than k ”. But, they associated this observation with very different schemes.

One approach was based on backtracking, and involved very little capitalization on the task characteristics. The programs of these students were composed of two stages: their first stage calculated the number of appearances of each symbol of B in A, and determined an upper bound for the output (which may be smaller than n/m). Then, they scanned A, up to that bound, with the 1st symbol of B, followed by the 2nd symbol of B, and so on. If the scan reached a point where this bound could not be met by one of these symbols, then the program backtracked to an earlier symbol in A, and retried the scan with a smaller stuttering bound. Unfortunately, this “operational” approach is inefficient, and its implementation is error-prone.

The second approach was better related to the simple stuttering characteristic mentioned above, and involved linear search for the largest stuttering value. These students’ algorithm started either from 1, or (descended) from n/m . For every considered stuttering size, j , they checked its validity. Correct, but still inefficient.

The students of the third approach capitalized elegantly on the stuttering characteristic and associated the above recognized pattern with the scheme of Binary Search. Their solution obtained the largest stuttering value much more efficiently. They noticed that Binary Search on k , the stuttering metric, is relevant here, even though their familiarity with Binary Search mostly involved its application on an ordered list of arbitrary values. They demonstrate some novel associations.

The time complexity of the latter computation is $O(n \log(n/m))$. The more involved solution of this task (Mirzaian, 1987b) is based on the halving method, employed in the domain of VLSI (Mirzaian, 1987a), which was beyond the scope of our training. Its time complexity is $O(n)$.

In retrospect, we may notice that the students who followed the first approach, of backtracking, expressed an “operational” (“how to do”) perspective, with no underlying pattern. The students who followed the second approach did relate to the simple task pattern, but did not associate it with the most suitable algorithmic scheme. Such association was demonstrated only by the students of the third approach.

2.2. Hidden Patterns and Scheme Sub-Components

The second task that we display is more involved. It is related to the notions of list ordering and list inversions (Ginat and Garcia, 2005). Like the previous task, here too, we experienced several solution approaches, based of different levels of insight and associations.

Widest Inversion. Given a list of N distinct integers, output the width of the widest inversion; that is, the distance between the two unordered elements that are furthest from one another (e.g., for the input 2 5 1 9 4 7 the output will be 3 – the distance between 5 and 4.)

Students approached this task in four different ways. One way was based on the exhaustive computation of directly finding for each element e the furthest element to its right that is smaller than e . The other three solution approaches were more efficient, though not always correct.

Students felt that there is an efficient solution to this task, and some attempted a greedy solution. This solution was not based on any recognized pattern, but rather on a variant of a sub-component of Quick Sort, in which two pointers are “run” concurrently from the two ends of the list. One pointer is set to the left-end of the list and the other – to the right-end. They are advanced concurrently “inwards”, one step at a time, until an inversion is found, or until they meet; then, these pointers are advanced separately “outwards”, and the output is the widest inversion found in this process.

This two-stage scan – first “inwards” and then “outwards” – yields the correct result for the input 2 5 1 9 4 3. But, does it yield the correct output for all inputs? Not quite. The input: 7 2 4 17 6 5 13 10 18 19 falsifies this scheme, as the “inwards” scan stops when the pointers are at <17,13>, and the following “outwards” scan yield the inversion <17,10> as the widest inversion. Yet, this is not the widest inversion (the widest inversion is <7,5>). Some students who realized examples that falsify this approach tried to patch their solutions with some “local patches”, which did not really improve the situation.

The third approach we observed was based on relevant insight. Some students noticed that it may be beneficial to look at location differences instead of value differences, as the widest inversion is tied to the largest location difference between unordered elements. Thus, if we know the location of the lowest element in the original list, and the location of the second-lowest in the original list, and so on, then the task may actually be reduced to a task of finding the “largest drop” among these locations.

We exemplify the latter idea, of transforming the point of view into a “location differences” task, with the list 7 2 4 17 6 5 13 10 18 19 above. We create a list of element locations. The locations-list for the above list is: 2 3 6 5 1 8 7 4 9 10. (the location of 2 is 2, the location of 4 is 3, the location of 5 is 6, . . . the location of 19 is 10). Notice that the latter list is a permutation of the integers 1 . . . 10. We now have to calculate the “largest drop” in this permutation; that is, the maximal difference between two values such that the first among them is larger than the second. The largest drop is obtained from 6 and 1; and this indeed yields the widest inversion <7,5>.

The above insightful point of view may be tied to two algorithmic schemes – sorting (for creating the locations-list) and calculating the “largest drop”. The former is of time complexity $O(N \log N)$, and the latter – $O(N)$. Thus, the observation of the pattern of an equivalent task, of location differences, yielded a rather efficient solution of $O(N \log N)$.

Yet, one can still do better. Some students noticed the pattern that only some of the integers in the original list may be candidates for the left-end or the right-end of the widest inversion. For example, in the above list (7 . . . 19), the 2 cannot be the left-end of the widest inversion, since 7 is larger and is on its left. By the same reasoning, none of the integers 4, 6, 5, 13, and 10 are candidates for the left-end of the widest inversion. Similarly, the 13 cannot be the right-end of the widest inversion, as 10, which is smaller, is to its right. None of the integers 6, 17, and 7 are candidates for the right-end of the widest inversion as well.

Upon examining the candidates for the left-end and the right-end of the widest inversion, students noticed another pattern: the list of the left-end candidates and the list of the right-end candidates are increasing from left to right. The list of the left-end candidates is: 7 17 18 19. The list of the right-end candidates is: 2 4 5 10 18 19. (Notice that the same integer may appear in both lists.)

At this point, students who recognized the above two patterns associated their observations with the sub-component of Merge Sort in which two lists are merged into one, by scanning both lists concurrently. Thus, given the two increasing lists (of the left-end candidates and the right-end candidates), we may scan both lists concurrently from left to right, with two pointers, while finding for each left-end candidate its farthest right-end match. (After the pointer on the left-ends list will be advanced to the next candidate, the pointer on the right-ends list will be advanced as far as possible from its current location.)

The above solution may be implemented in $O(N)$ time, as both the construction of the lists of the left-end/right-end candidates, and the concurrent scanning, in a Merge Sort manner are of $O(N)$.

All in all, we may notice that the students who offered the first two solutions did not recognize any pattern on which to capitalize. Those in the second group turned to the “operational” Quick Sort idea, of “running” two pointers “inwards” from the two list ends, but this idea was not based on any relevant observation (and actually yielded an erroneous solution). The students of the latter two solutions did observe relevant patterns, before devising their solutions, and capitalized on these patterns. Some of them associated the original task with another task, which can be solved rather efficiently; and others noticed two insightful patterns and associated their observations with a sub-component of the familiar Merge Sort algorithm.

2.3. Hidden Patterns and Implicit Scheme Characteristics

The two previous tasks involved sequences. Their solutions involved explicit algorithmic schemes or sub-schemes. The task displayed in this section involves graphs. It is also related to a familiar algorithmic scheme, but the link between the task’s relevant pattern and this familiar scheme is not as explicit as it was in the previous two tasks. Here, it is implicit.

Non-Modulo-3 Cycle. Given an undirected graph of N nodes, where the degree of each node is at least 3, output the following: if the graph includes a cycle of length that is not a multiple of 3, then output such cycle; otherwise output “no such cycle”.

Students approached this task in two different ways. Both ways were based on the very familiar scheme of DFS (Depth First Search; Manber, 1989). But, there was a big difference between the two ways – one way was based on a most relevant pattern of this scheme, and the other was not.

The vast majority of the students did not recognize any pattern that derives from the task specifications. Being familiar with the notion of recognizing cycles in a graph by back-edges of the DFS, they invoked DFS, in which every cycle discovered by a back



Fig. 1. Three cycles with two back edges.

edge was checked. If a “non-modulo-3” cycle was found, then it was displayed; otherwise, their algorithm notified that there is no such cycle in the graph.

Unfortunately, this hasty, “operational” design is incorrect, as there are graph cycles that may not be examined. We illustrate it with the Fig. 1. If an edge “goes back” from node v to node t , and another edge “goes back” from node u , which is a descendant of v in the DFS tree, to a node s , which is on the tree path from t to v ; then the algorithm will not examine the cycle: $u - s - -t - v - -u$. (We use “ $-$ ” to denote a single edge, and “ $--$ ” to denote a path of one or more edges.)

Thus, the algorithm may notify “no such cycle” when there is actually such cycle. Some students noticed this difficulty, and tried to patch a correction, by starting a separate DFS from every one of the graph nodes. However, their patches, which still involved no capitalization on the task characteristics, were also erroneous.

In order to solve the task correctly (and efficiently) one has to recognize a pattern that arises from the task description. In the previous two tasks that we presented, the recognized patterns did not derive from a particular algorithmic scheme that one would have considered for solving the task. Here, it is different. The relevant pattern derives from an *implicit* characteristic of the algorithmic scheme, DFS, which one would consider for the solution.

During DFS, the computation reaches a node which is the end of a branch; i.e., a node v from which the computation “returns up” in the DFS tree. A tree-edge leads to this node. Since the degree of this node, v , is at least 3, there are (at least) two back edges “going back” from this node to two nodes, u and w , in the DFS tree. These two back edges yield three cycles: 1. $v - w - -v$; 2. $v - u - -v$, and 3. $v - u - -w - v$, as may be seen in the Fig. 2.

It is easy to show that at least one of these cycles must be of a length that is not a multiple of 3. The output will be that cycle (and once it is found, the DFS will halt).



Fig. 2. Three cycles with two back edges from an end of a DFS branch.

Very few students noticed this pattern. Those who did recognize it managed to associate the task specification with the above implicit characteristic of DFS. These students demonstrated a novel association that combined an “assertional” perspective, of pattern recognition, with the natural “operational” view of DFS. Unfortunately, the majority of the students failed to do so, and only applied a limited, “operational” perspective of DFS.

3. Discussion

We presented different algorithmic tasks that require various kinds of novel associations in algorithmic problem solving. We displayed student solutions to these tasks, and revealed strong correlation between problem solving competence and the recognition and capitalization on novel associations. The required novel associations and the diverse student solutions illuminate a series of considerations of which tutors should be aware in designing and posing algorithmic challenges to students. We list and describe them below.

- **Operational and assertional perspectives.** These two perspectives encapsulate two different viewpoints of an algorithmic solution. The former focuses on “how” should a computation be performed, and what are the algorithmic schemes that describe the computation operations. The latter focuses on “what” are the characteristics underlying the computation. Both perspectives are essential. While the operational perspective is natural, the assertional perspective involves hidden patterns, which may not be easy to unfold. Problem solvers should be aware of both perspectives, and particularly seek characteristics on which to capitalize, and assertions that capture solution behaviours (Dijkstra *et al.*, 1989). Yet, our findings reveal that students do not always demonstrate the latter. The more competent students are well aware of seeking characteristics on which to capitalize, while the less competent ones do not seek, or only partly seek them.
- **Hastiness.** The less competent students do not recognize sufficient task characteristics. This phenomenon may derive from lower competence in unfolding hidden patterns; but may also due to the undesired problem solving discipline of hastily “jumping” to the composition of an algorithm, without conducting a suitable, thorough task analysis. This could be seen with the less competent students in all our examples – “jumping” into backtracking in the first task; “jumping” into the erroneous (Quick Sort based), greedy solution in the second task; and not seeking any DFS characteristic in the third task. The particular occurrence of hasty greedy solutions is described further in Ginat (2003).
- **Lack of rigor.** In algorithmic problem solving it is often the case that the problem solver is not asked for any assertional argument regarding the correctness of her/his solution. This may increase the possibility of erroneous solutions. Such an occurrence is particularly evident when students seek efficient solutions, which they are unable to argue as correct. A typical example is displayed in our findings, of the Quick Sort based solution to the second task.

- **Limited flexibility with familiar algorithmic schemes.** One of the key elements in algorithmic problem solving is flexible utilization of familiar algorithmic scheme. The natural way to employ familiar schemes is by using them in a way that is similar to the way(s) seen so far. But, sometimes the suitable way is not analogous to previous experiences. The tasks in the three examples of this paper illustrate this phenomenon. The first task required a less familiar utilization of Binary Search; the second involved a variant of a sub-component of Merge Sort; and the third task required an insightful observation and a corresponding flexible utilization of DFS. The distinction between more competent and less competent students is related to their demonstration (or no demonstration) of flexible utilization of familiar schemes.
- **Limited resources and heuristics.** Challenging algorithmic tasks may require problem solving resources and heuristics that are not always explicitly underlined to learners. The second task in this paper involved such elements. The elegant solution to this task involved the notion of “candidates”. This notion appears in algorithmic tasks, such as the Celebrity and the Majority Problems (Manber, 1989), but it is not made explicit as a useful tool for algorithmic problem solvers. So is the relevance of location characteristics rather than value characteristics in list-processing tasks. This was the case in the relatively efficient solution to the second task, which was transformed to a “location differences” task. One example of such an occurrence appears in the elegant solution to the game task of IOI ’96, of collecting numbers from the left-end and the right end of a given list (<http://olympiads.win.tue.nl/ioi/ioi96/contest/ioi96g.html>). The heuristic of reducing a given task into another task is explicitly demonstrated in advanced CS topics (e.g., NP-complete), mostly as a proof means. However, it may also be useful in algorithmic problem solving, as occurred here in the relatively efficient solution to the second task.
- **Absent novel associations.** Novel associations are related to all the above considerations. Upon algorithmic problem solving, a problem solver should apply both operational and assertional perspectives; employ careful task analysis; seek relevant, rigorous patterns on which capitalize; flexibly tie them to the task at hand, using explicit and implicit resources; and create novel associations that yield the suitable elegant solution. Less competent and more competent students differ by their employment of little, some, or much of the above, as could be seen in our findings.

The three tasks presented in the previous section yielded diverse student solutions, which were related to all the considerations described above. We referred in the paper to all these considerations, but primarily focused on the latter one, of novel associations between task patterns and familiar algorithmic schemes. We examined it through different levels of difficulty, and characterized these levels with the three titles of the sub-sections of the previous section. We believe that such characterization may be of benefit for task designers, in describing characteristics of challenges posed to students. Yet, there may be additional facets of characterization. The facet presented here may serve as an initial

facilitator for further ones, which may yield additional core characteristics of challenging algorithmic tasks.

References

- Astrachan, O., Berry, G., Cox, L., Mitchener, G. (1998). Design patterns: an essential component of CS curricula. In: *Proc. of the 29th SIGCSE Technical Symposium on CS Education*, ACM, 153–160.
- Dijkstra, E.W. et al. (2003). A debate on teaching computing science. *Communications of the ACM*, 32, 1397–1414.
- Ginat, D. (2003). The greedy trap and learning from mistakes. In: *Proc. of the 34th SIGCSE Technical Symposium on CS Education*, ACM, 11–15.
- Ginat, D., Garcia, D. (2005). Ordering patterns and list inversions. (*Online*) *Journal of Computer Science Education*, ISTE SIG Publications.
- Linn, M.C., Clancy, M.J. (1992). The case for case studies of programming problems. *Communications of the ACM*, 35(3), 121–132.
- Manber, U. (1989). *Introduction to Algorithms: A Creative Approach*. Addison-Wesley.
- Mirzaian, A. (1987). River routing in VLSI. *Journal of Computer and System Sciences*, 34(1), 43–54.
- Mirzaian, A. (1987). A halving technique for the longest stuttering sequence problem. *Information Processing Letters*, 26, 71–75.



D. Ginat – heads the Israel IOI project since 1997. He is the head of the Computer Science Group in the Science Education Department at Tel-Aviv University. His PhD is in the computer science domains of distributed algorithms and amortized analysis. His current research is in computer science and mathematics education, focusing on cognitive aspects of algorithmic thinking.

Survey on Informatics Competitions: Developing Tasks

Lasse HAKULINEN

*Department of Computer Science and Engineering, Aalto University School of Science
P.O.Box 15400, FI-00076 Aalto, Finland
e-mail: lasse.hakulinen@aalto.fi*

Abstract. Informatics competitions offer a motivating way to introduce informatics concepts to students and to find new talents. There are many different competitions in the field of informatics with different objectives. In spite of these differences, they all share the same need for high quality tasks. Tasks can be seen as the heart of the competition, so the effort put in developing new tasks should not be underestimated. In this survey, the development of tasks and different task types are discussed. The focus is on the Olympiads in Informatics competitions, but tasks in other competitions are discussed as well.

Key words: task development, competition tasks, informatics competitions, IOI, competitions and education.

1. Introduction

Motivating students to learn is important for all educators. Competitions offer a convenient way to bring informatics concepts to students in a different fashion than regular teaching in schools and universities. It could be said that the tasks are the heart of the competition. Therefore, designing tasks that support the goals of the competition is an important and demanding undertaking.

Nowadays, there are many different informatics competitions from small to worldwide events. Also, the types of tasks vary from tasks solved with pen and paper to complex problems dealing with large datasets and sophisticated algorithms. Many different types of events offer a wide range of possibilities for students to get involved with informatics.

Competitions can be a place for students to learn new concepts on informatics. They can also be a source of motivation for students interested in problem solving, programming and other aspects of informatics. Students who get interested in programming contests are usually looking for a place for training their skills and to gain some informatics education (Dagienė, 2010).

In this survey, the development of tasks and different task types are discussed. There are a lot of different competitions in the field of informatics and therefore the variety of tasks is wide. This paper describes the tasks used in different competitions, but focuses on

the ones used in Olympiads in Informatics and similar competitions. Also, the educational aspect of competitions is discussed.

Different informatics competitions are introduced in Section 2. Task types are described in Section 3 and the content of the tasks in Section 4. In Section 5, the process of developing new tasks is discussed. The usage of competitions in education and suggestions for task development are discussed in Section 6. Finally, the conclusions of competition tasks and their development are presented in Section 7.

2. Informatics Competitions

There are several informatics competitions held worldwide and the variety of the competitions is wide. Also, the fundamental purposes of the competitions vary. The purpose can be, for example, testing competitors' knowledge, enabling learning, finding especially talented competitors, or promoting informatics. Because the focus of the competitions varies, also the tasks used in different competitions have different characteristics. In addition, the target group and required preliminary knowledge on informatics can be different in each competition.

International Olympiad in Informatics (IOI Website, 2010) is a well known competition that has been organized annually since 1989. Naturally, the competition and tasks have evolved during the last two decades, but the high-level goal is still promoting computer science among the youth, and discovering and stimulating talented students (Verhoeff, 2009). Tasks in IOI concern algorithmic problem solving and they must be implemented in one of the few specified programming languages.

Another well known competition is the ACM International Collegiate Programming Contest (ICPC) where student teams solve programming tasks of varying difficulty (ACM, 2010). In ICPC, teams of three competitors can use one computer to solve several algorithmic programming tasks. The difficulty levels of the tasks vary so that the teams must prioritize the time used for each task.

There are also many other informatics competitions that all have their own characteristics and objectives. Following is a list containing some popular competitions and their characteristics:

- TopCoder: Several different types of competitions from short algorithmic tasks to marathon matches that can last for weeks (TopCoder Website, 2011).
- Google Code Jam: Algorithmic programming problems (Google Code Jam Website, 2011). Initially established in order to find top talents to work at Google (Big-mouth media, 2003).
- ICFP: Programming contest where participating teams can be of any size and they can use any programming languages (ICFP Website, 2011). Teams will have 72 hours time to complete the tasks.
- Imagine cup: Teams are supposed to show how technology can be used to solve the world's toughest problems (Imagine Cup Website, 2011).

Many of the competitions involve quite difficult tasks and are meant for advanced students or even professionals. However, there are also competitions that are meant for any

students interested in informatics regardless of their skill levels. One such competition is Bebras (Dagienė and Futschek, 2008), which is aimed for pupils from lower secondary to upper secondary level.

3. Task Types

3.1. Tasks with Programming

Many of the tasks in informatics competitions involve programming. In these tasks, competitors are asked to return a working program code for a certain problem. Typically, the tasks deal with large data sets that have to be manipulated using sophisticated data structures and algorithms. With programming tasks, it is also possible to give instant automatic feedback to the competitors and allow them to return the same task multiple times until they get the right solution.

3.2. Tasks without Programming

It is easy to see why programming tasks are a natural choice for task types in informatics competitions. However, there are significant advantages of using non-programming tasks that should not be overlooked. On the other hand, when using non-programming tasks, there are also obstacles that must be addressed different ways than in programming tasks.

One issue in organizing informatics competitions can be the resources, if the use of computers is required. When using tasks that do not require programming, it is possible to organize large scale competitions with pen-and-paper tasks. Burton (2010) describes how the Australian Informatics Competition (AIC) have only pen-and-paper tasks and hence it is accessible to a much broader audience than programming competitions. He says that the greatest difficulty with pen-and-paper informatics competitions is to retain the focus on algorithms and algorithmic thinking. In AIC, the tasks are either multiple-choice questions or the answer can be given as an integer.

Kubica and Radoszewsk (2010) suggest the use of tasks that require algorithmic thinking, but no programming in order to attract students who know nothing about programming or algorithms. They claim that offering tasks or puzzles requiring various levels of algorithmic thinking is a good way to popularize learning programming among young pupils. They provide a couple of problems that require algorithmic thinking but that are formulated in a purely mathematical manner. The problems are designed so that the desired solution is the one that minimizes the total time of inventing it and executing it by hand. The trivial solutions are usually more time consuming to perform. They also suggest that problems that require the knowledge of classical algorithms and advanced techniques should be excluded from the competitions.

Another interesting way to promote informatics among students is described by Bell *et al.* (1998). They introduce a set of activities called *Computer Science Unplugged* that are designed to teach school students about computer science concepts and awake their interest on computer science without using computers at all. They claim that a common

misconception is that computer science is only about programming. Computer Science Unplugged addresses that misconception by presenting the ideas and issues of computer science with activities without computers.

Computer Science Unplugged is not a competition in itself, but its activities can be combined with competition tasks. Voigt *et al.* (2010) combined the Computer Science Unplugged approach to competition-style programming problems. By using both approaches, they wanted to reinforce the concepts students learn using the unplugged approach, and on the other hand, to connect programming with computer science concepts. In the teaching sessions, they first discussed the topic with unplugged activities. After that, the students were given several small competition-style programming task dealing with the same topic. By testing the students' knowledge before and after programming tasks, they found out that the students' performance on the test after the programming activity was significantly improved. Also, students responded mostly positively to the combination of unplugged activities and programming tasks.

3.3. *Code Understanding*

One possibility for a task type that is not widely used in competitions, is using tasks that require code reading skills. In real life, programmers often have to read source code made by other people. In this kind of tasks, memorizing well known algorithms by heart would not be as beneficial as in the traditional programming tasks. The contestants have to really understand the given code in order to improve or analyze it. Code understanding tasks can be used in both, programming and non-programming, types of tasks.

Opmanis (2009) described the use of code understanding tasks in the *Ugale* competition. Competitors were given a code and the assignment was to find an error, construct the worst case counterexample, estimate the complexity, or implement a more efficient version of the same algorithm. Also the Syrian Olympiad in Informatics included tasks that had some initial code, which had to be improved (Idlbi, 2009). They used *Scratch* to motivate younger students to participate and included tasks with some initial game elements that needed to be improved.

3.4. *Subtasks*

With some of the tasks, it might be difficult to get a nice score distribution. It may be that the students who are close to solving the task get as few points as the students who do not have a clue about the right solution. Often it is desirable to give some kind of reward to the competitors who understand the concepts of the tasks even if they are not able to provide completely correct answer.

Vegt (2009) explains how subtasks are used in the Dutch Olympiad in Informatics to get a nice score distribution and to reward contestants for what they were able to solve. Using subtasks also opens up new opportunities. For example, Vegt says that when using subtasks, it is easier to slip in a more theoretical question as a subtask.

4. Content of the Tasks

Different task types mentioned in the previous section should be considered when designing new tasks for a competition. In an ideal case, the task type and the content of the task would support each other so that the goals of the competition will be met most efficiently.

When developing tasks for competitions, the wide range of concepts covered in informatics should be borne in mind. Naturally, all competitions have their own goals and the content of the tasks should be build to meet the goals. Nevertheless, when considering the whole spectrum of informatics competitions, it would be desirable to cover all the major concepts of informatics. In real life, some concepts appear in tasks more often than others.

Some task topics that are common in many informatics competitions are, for example: algorithms, programming, problem solving, logic and graphs. Verhoeff (1990) points out that it is important to cover a wide range of informatics aspects in the task set for two reasons. Firstly, it is unfair if "specialists" in certain field have an advantage, because one aspect appears in many of the tasks. Secondly, competitors can work on the types of problems that they like the most. The latter is especially important for the weaker competitors. When the range of topics is wide, weaker contestants have enough topics to choose from allowing them to focus on the tasks that they are most comfortable with.

Constructing a suitable set of tasks for a competition can be a demanding assignment. Verhoeff *et al.* (2006) present a very detailed list of informatics concepts to be used in IOI tasks in their proposal for an IOI syllabus. Opmanis (2009) provides a different type of list of topics used in the *Ugale* competition. The list by Opmanis contains typical topics but also some unusual categories such as a category called *Dominoes* that has tasks using a Dominoes game as a setting for combinatorial tasks.

There are some topics that are rarely used in competitions even though they are an important part of informatics. Tru and Ivanov (2008) state that software testing is a major area of software development that is mostly neglected in the informatics competitions. They examine the main modes of operation of software testers and discuss their suitability as the basis for competition tasks. They discuss the suitability of black-box and white-box testing, as well as static and dynamic testing techniques. They analyze a set of tasks involving testing and conclude that software testing can be a basis for a good competition task.

Criminal activity and misuse of computers is nowadays a big problem worldwide. Therefore, also ethics should be an important part of informatics education. Futschek and Dagienė (2009) address this issue by stating that proper behavior of computers should be part of all Bebras task sets. As an example, they present a task dealing with junk mail where you are asked for the correct way of acting when receiving an e-mail requesting you to forward it to your friends.

4.1. Games and Puzzles

The content of the task can also have a big influence on the motivation towards completing it. Carefully selected topic can make the understanding of the problem easier and help the competitors to focus on the core problem of the task rather than on a long task description.

Using games is one way to make the competition tasks more attractive for students. Ninka (2009) discusses the use of reactive and game tasks in informatics competitions. He points out that when using a game, there is no need for a story, and therefore students save some time getting into the actual problem. Also, students have some prior experience of games, which helps them to get motivated. He gives an example of the students' high motivation towards game tasks, where students continued pondering a game used in a competition while waiting for their flight and managed to improve their initial solution. Ninka claims that students are more biased to discuss game tasks after the contest with the aim to discover what they may have missed during the contest.

Ninka also points out how the number of students who are fond of algorithms and programming has reduced. On the other hand, he says that the situation is quite different when students have to program a game. This motivation towards games could be used to promote informatics competitions and also to get students interested in algorithms.

Vegt (2006) explains how games, algorithmic thinking and competitive programming are combined in CodeCup. CodeCup is an annual game programming competition that is a side event for the Dutch Informatics Olympiad (NIO). In CodeCup, competitors are supposed to write a program that can play a certain game. Vegt says that often the games chosen for the competition do not have a known perfect solution. That way they encourage competitors to develop their own original ideas, rather than implementing known algorithms.

5. Task Development

It is very important for a competition to have a decent set of tasks in order to be successful. Especially when there are so many different informatics competitions held each year, it may not be easy to find new and genuine ideas for interesting and educational tasks. So, how to develop good task and what should be borne in mind when designing new tasks?

5.1. Features of a Good Task

Burton and Hiron (2008) define several features of a good informatics olympiad task. However, they remind that the desired features may vary depending on the goals and target group of the competition. Similar features are also discussed by Diks *et al.* (2007). Following is a summary of the features defined by Burton and Hiron:

- The problem statement should be short and easy to understand.
- The algorithms that solve the task should not directly resemble a classic algorithm or a known problem. However, it can be a modification of a classic algorithm.
- There should be several different valid solutions of varying difficulty and efficiency in order to allow weaker students to gain partial marks for the task.
- The official solution should be reasonably concise.
- In most cases (depending on the difficulty of the task), the official solution should also be the best known solution for the task.

- For the experienced students, it is nice to have tasks where it is not obvious in what category the desired algorithm belongs to.

Tasks do not need to be difficult or complex in order to be useful in competitions. Dagienė and Futschek (2008) discuss criteria for good tasks in the Bebras International Contest on Informatics and Computer Literacy. The tasks are not programming tasks, but they are solved using a computer. The two task types in Bebras competitions are interactive and multiple-choice questions. The tasks are generally fairly short because one of the criteria of a good Bebras task is that it can be solved in 3 minutes. The main idea behind Bebras is not to ask for already learned facts but to give problems that allow students to learn something about concepts that might be new for them (Futschek and Dagienė, 2009).

5.2. Sources of Inspiration for New Tasks

Burton and Hiron (2008) discuss different techniques that can be used when searching for new ideas for tasks and refine these ideas into problems suitable for an informatics olympiad. Perhaps the most difficult part of creating a new task is getting the initial idea of the task. They say that one of the most common techniques is to look around and take inspiration from things that you see in real life. They also suggest an alternative approach where you could find somebody unfamiliar with computer science and ask them for ideas. They also point out that one of the disadvantages of looking around for inspiration is that you do not have a solution in mind. Therefore, the task creation process can be very time consuming when you have to also find the solutions for all the possible task candidates. Third source of inspiration they introduce, is to use and modify a problem one faces in a daily work, e.g., a small piece of a complex research problem or an unconventional algorithm. An advantage of this method is that often the solution is in the research paper or you have solved it yourself.

Tasks do not need to be generated from scratch every time. Burton and Hiron also suggest combining old tasks when designing new tasks. They illustrate how to find the characteristics of old tasks and find a combinations of characteristics that have not yet been used when designing a new task. Another point of view they pointed out when designing new task is to start with a standard algorithm and set a task that modifies it in some way.

Maybe one of the most exotic ideas by Burton and Hiron is to start by drawing random things on a blank piece of paper. By adding objects and relations you might start to get vague ideas of where you are heading and what could be the problem statement of the task. They also propose searching ideas from other disciplines such as mathematics or from games and puzzles. They state two ways games and puzzles are useful. First, they provide ready-made tasks such as playing a game optimally or solving a puzzle using the smallest number of moves. Second, they can supply interesting sets of objects and rules that can act as starting points for other tasks.

Also Pankov (2008) suggests the use of real things around us to get inspiration for new tasks. He claims that using a real situation or task is more preferable than starting

with an effective algorithm and composing a task with a corresponding subject. In his opinion, that is because using real life situations can yield original tasks with natural, short and elegant formulations and also give less advantage to experienced participants.

Pankov (2010) also says that natural sciences contain many interesting laws and facts that could be used as a base for competition tasks in informatics. He offers three types of tasks that can be set if the properties of an object are given: combinatory, optimization and interaction. As an example, he shows how the conservation laws can act as a basis for a competition task where competitors must find the smallest possible velocity of merged pointwise objects.

Many of the data structures used in competition tasks relate closely to real life. This way students do not necessarily need to be familiar with the data structures beforehand. Graphs are widely used in tasks of all difficulty levels. Graphs are an important origin of tasks as they are modeling real life and therefore the tasks become easy to understand also for younger students (Manev, 2008).

5.3. Task Preparation Process

Getting the initial idea or creating the first prototype of the task is just the beginning when preparing a new task for a competition. Diks *et al.* (2008) propose best practices that should be applied in a task preparation process for any programming contest. They present the task preparation process used in the Polish Olympiad in Informatics (POI) and say that a rigorous implementation of the process is the key to assure good quality of tasks and to ensure that mistakes in the tasks are discovered before the contest, when it would be too late to correct them. Also Verhoeff (1990) gives detailed guidelines for the process of constructing a task set for a competition. He focuses on "ACM-style" competitions and suggests that each task should have one person who is responsible for the task.

Diks *et al.* (2008) suggest a list of aspects that should be taken into account when judging the appropriateness of a task. The listing is very similar to the features of a good olympiad task by Burton and Hiron (2008). In addition, Diks *et al.* emphasize the significance of task analysis and verification in the preparation face. For instance, the task analysis report should contain a set of source codes of solutions in all competition languages with model, suboptimal and wrong algorithms. Similar process is suggested by Verhoeff (1990), who points out that in addition to the correct solution, the solutions that are considered too inefficient should be implemented during the task preparation process. This way the time limits and the input data can be fixed so that undesired solutions will fail in the tests even if they produce the correct output.

It is also important to take advantage of the previous competitions. Wang *et al.* (2010) discuss the methods used to assure the task quality in the China National Olympiad in Informatics (CNOI). In CNOI, after each competition there is a session where the tasks and their solutions are discussed. Competitors are encouraged to join in and ask questions about the tasks. Wang *et al.* say that the session helps to discover the imperfection of the tasks, which usually leads to improvements in the future. Also Verhoeff (1990) suggests having an evaluation after the competition in order to improve the arrangements for the following competitions.

The quantity of needed tasks varies among the competition organizers. Kolstad (2009) explains how the constant need for new tasks in the USA Computing Olympiad (USACO) was addressed. USACO organizes annually six internet-based programming competitions in three divisions so they typically need 75 new tasks of different difficulty each year. They tackled the issue by creating a web-based system, *probgate*, where the task creating community can upload and edit their tasks. This has speeded up the process of developing large amounts of new tasks that meet the quality criteria set for them. However, Kolstad states that USACO's way of creating tasks emphasizes tasks throughput and not detailed analysis of tasks or extremely thorough black-box testing.

Dagienė and Futschek (2008) discuss the development of new tasks for the Bebras competition. They point out that attraction, invention, tricks and surprise should be desirable features of each problem presented to competitors. They also remind that the problems have to be selected carefully bearing in mind the different aspects of each problem, i.e., how to interpret task's attractiveness to students.

5.4. Choosing the Task Type

Choosing the most suitable task type is also one thing that should be considered when designing new tasks. The purpose of the competition and the content of the tasks often derives the design towards some task types. For example, programming tasks are a natural way to test programming skills. However, one should keep an open mind for different task types as well.

Opmanis (2009) describes different task types used in the *Ugale* competition. The task in the *Ugale* competitions differ from the task used in most of the informatics competitions. For example, there is no constraint that written program must run in particular time and space limits during program execution. The contestants can also use any software during the contest. Some of the tasks are supposed to be solved with a computer and some of them are pen-and-paper type of tasks. One task can also be both. Opmanis states that it is really challenging to develop a task, which could be solved either by using pure mathematic skills without a computer, as well as by writing a correct computer program that gives the answer.

6. Discussion

In many cases, informatics competitions are closely related to education. They are not necessarily part of the formal curricula, but many of the competitions are aimed for students. Therefore, the potential of exploiting competitions in education should not be underestimated. The educational point of view should be borne in mind also when designing the task set for a competition. If the goal is to encourage many students to join, it should be made easy for students to participate even if they are not the top talents. On the other hand, competitions could be integrated to the traditional education so that they are not seen just as a separate activity for a small marginal group.

6.1. Competitions and Education

The motivation level of students is clearly a significant factor in learning, regardless of the discipline in hand. Informatics competitions and competition-type assignments can be one way to motivate students to learn informatics. Verhoeff (1997) gives a historic overview of competitions and education. He states that as the society and cultures constantly change, the educational practices must change as well.

Verhoeff brings out that there are conflicting opinions about bringing competitions into education. One view is that competitions are a part of every culture and therefore students should be introduced to them, because they will need the competitive skills later in life. The other view is that competition can be seen as opposed to collaboration and therefore should be avoided in education. However, Verhoeff reminds that different views of competitions should be distinguished. The focus of the competition can be, for example, in defeating other contestant, or in some external entity. The latter can, in fact, encourage students to teamwork.

Also Lawrence (2010) points out that competitions may be highly beneficial for some students, while for others it can be a negative factor. He also describes the use of competitive programming in several computer science courses. He says that pedagogical results from the courses have shown benefits, but it is critical to make sure that the competition is presented in a proper way. Students must have the opportunity to excel also without the competition setting because not all students are motivated by it. However, competitive programming assignments can provide an ongoing feedback throughout the assignment and encourage friendly competition with the instructor and other students.

Dagienė and Skupienė (2010) also discuss how competitive programming can be used in informatics education. They point out that problem solving is one of the most important parts of teaching of cognitive skills. They say that programming provides a challenging environment for learning problem solving and therefore teaching programming should be brought back to the secondary education curricula. Dagienė and Skupiene state that competitive learning is an important source of motivation for students to improve their programming skills.

Programming competitions do not necessarily have to be separate from the more traditional forms of informatics education. In National University of Singapore (NUS), competitive programming have been integrated to the curricula (Halim and Halim, 2010). They offer a special module called *Competitive Programming* where students can put their theoretical knowledge in use in a competition style setting. However, the module is not available for all students since there are pre-requisites that the students must satisfy.

Vasiga *et al.* (2008) ponder what do the tasks in olympiads actually measure? They state that the principal focus of olympiad tasks should be problem solving. They suggest criteria for analyzing tasks and claim that the goal of focusing on the problem solving skills is often hard to attain due to detailed information processing, mystery and esoteric prior knowledge of algorithms. Also Kiryukhin (2010) discussed the content of the tasks in informatics olympiads and their relationship to national informatics education. He compared the informatics olympiads and the requirements of the Russian State School

Education Standard (SSES) in Informatics. He concluded that there is a large difference between SSES and the content of the informatics olympiads, which reduces the possibilities of major portion of the students to participate and succeed in the competition.

Combining informatics competitions and education is not a trivial task. Often the competitions might be seen as a separate activity for a small group of informatics enthusiasts. If the aim is to get majority of the students to participate, collaboration between competition organizers and educators is crucial. In order to get many of the students to join in, the competition tasks and the content of the school curricula should not differ too much.

6.2. Suggestions for Task Development

Task development can be time-consuming and the demand for new unique tasks is big. Each task type has its advantages and disadvantages that should be considered. Programming tasks are well suited for complex algorithms and large data sets. They can also be graded automatically and instantly. However, they require competitors to learn and use some of the supported programming languages. On the other hand, non-programming tasks can lower the bar for participation and encourage different people to join in. Nevertheless, with non-programming tasks the data sets have to be smaller and keeping the focus on algorithms might be challenging.

Tasks that require code reading skills could be used more widely. They offer a great potential to test different kind of topics, for example:

- Finding an error in existing code can be used to test debugging and testing skills.
- Improving a given solution can be used to test code understanding.
- Algorithm analysis can be tested by requiring an input, which results to the worst case running time of the algorithm.

It should be made sure that the tasks measure the objectives that are set for them. For example, does the task measure some preliminary knowledge the contestants should have, or is it testing their inventiveness and problem solving skills? Typically, it is not an intention to measure only the facts and methods that are memorized in advance. Often there is a classical algorithm that can be used in the solution even if the focus is on problem solving. However, if the task is designed well, it requires problem solving to figure out how to use the known algorithm.

Games can be one way to tackle the challenge of creating tasks that focus more on problem solving than memorizing classical algorithms. Each game has a unique set of rules. These rules provide a unique basis that might help designing the task in a way that finding a known algorithm that solves the problem is not obvious. Games can also help the competitors to get interested in algorithms.

7. Conclusions

Problem solving is clearly an important skill for all students. Different types of competitions can be used to offer students opportunities to practise and develop their problem

solving skills. Competitions offer a great potential to enrich informatics education. However, integrating competitions and education is not an easy task and it should be done keeping all types of learners in mind. In this survey, several different types of tasks that have been used in informatics competitions were discussed. The tasks vary from pen-and-paper types of tasks to demanding programming tasks involving large data sets and complex algorithms. This variability in task types surely offers challenges for many different types of learners to benefit from the competitions.

It is a richness that we have so many different types of competitions involving informatics. There is no need to include all the task types in one competition as the goals of the competitions differ as well. However, it should be borne in mind that the topics of the tasks would cover the goals set for the competition. For some topics it is easier to create new tasks than for others. Therefore, attention should be paid also to the rarely used topics such as testing and ethics.

Also the task type should be tailored for the participating competitors. Short non-programming tasks can be used to attract new students to the competitions. Tasks can be formed in a way that there is no need for preliminary knowledge, which lowers the bar for participating. On the other hand, complex programming tasks with sophisticated algorithms and data structures can be used to find the top talents and to push the skillful competitors even further in their competence.

Although the difficulty levels and the types of tasks can vary significantly, the basic concepts of a good task stay mostly the same. First of all, the task should be easy to understand and unambiguous. Usually the competitions strive for tasks that do not require memorizing existing algorithms or solution patterns. The difficulty level should be suitable for the target group and in a competition there should be tasks that are easy and hard enough for each contestant. An ideal task is also interesting for the students and so exiting that it inspires students to study the subject even further.

The development of tasks is an essential aspect when organizing a competition. The tasks must be different from the previous tasks and naturally it is difficult and time consuming to come up with a lot of unique tasks. Many different methods for task development were presented in this survey. Combining these methods and task types could be a useful way to tackle the challenge of creating fresh and interesting questions for future competitions.

References

- ACM (2010). *Fact Sheet*, 1st edn.
<http://iiu.edu.my/acmicpc/upload/ACM-ICPC-Factsheet.pdf>. Accessed 12.11.2010.
- Bell, T., Witten, I., Fellows, M. (1998). *Computer Science Unplugged: Off-Line Activities and Games for All Ages*. Citeseer.
- Bigmouth media (2003). Google Launches Code Jam 2003. <http://www.bigmouthmedia.com/live/articles/google-launches-code-jam-2003.asp/1305/>. Accessed 17.02.2011.
- Burton, B. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.
- Burton, B., Hiron, M. (2008). Creating informatics olympiad tasks: exploring the black art. *Olympiads in Informatics*, 2, 26–33.
- Dagienė, V. (2010). Sustaining Informatics Education by Contests. *Teaching Fundamentals Concepts of Informatics*, 1–12.

- Dagienė, V., Futschek, G. (2008). Bebras international contest on informatics and computer literacy: criteria for good tasks. *Informatics Education-Supporting Computational Thinking*, 19–30.
- Dagienė, V., Skupienė, J. (2010). Olympiads in informatics: competitive learning of programming for secondary school students. In: Verdú, E., Lorenzo, R., Revilla, M., Requeras, L. (Eds.), *A New Learning Paradigm: Competition Supported by Technology*, 107–126. Sello Editorial.
- Diks, K., Kubica, M., Radoszewski, J., Stencel, K. (2008). A proposal for a task preparation process. *Olympiads in Informatics*, 2, 64–74.
- Diks, K., Kubica, M., Stencel, K. (2007). Polish olympiad in informatics – 14 years of experience. *Olympiads in Informatics*, 1, 50–56.
- Futschek, G., Dagienė, V. (2009). A contest on informatics and computer fluency attracts school students to learn basic technology concepts. In: *Proceedings 9th WCCE 2009, Education and Technology for a Better World*.
- Google Code Jam Website (2011). <http://code.google.com/codejam/>. Accessed 17.02.2011.
- Halim, S., Halim, F. (2010). Competitive programming in National University of Singapore. In: Verdú, E., Lorenzo, R., Revilla, M., Requeras, L. (Eds.), *A New Learning Paradigm: Competition Supported by Technology*, 173–206. Sello Editorial.
- ICFP Website (2011). The ICFP Programming Contest. <http://icfpcontest.org/>. Accessed 17.02.2011.
- Idlibi, A. (2009). Taking kids into programming (contests) with scratch. *Olympiads in Informatics*, 3, 17–25.
- Imagine Cup Website (2011). <http://www.imaginecup.com/>. Accessed 17.02.2011.
- IOI Website (2010). International Olympiad in Informatics. <http://www.ioinformatics.org>. Accessed 12.11.2010.
- Kiryukhin, V.M. (2010). Mutual influence of the national educational standard and olympiad in informatics contests. *Olympiads in Informatics*, 4, 15–29.
- Kolstad, R. (2009). Infrastructure for contest task development. *Olympiads in Informatics*, 3, 38–59.
- Kubica, M., Radoszewski, J. (2010). Algorithms without programming. *Olympiads in Informatics*, 4, 52–66.
- Lawrence, R. (2010). Motivating students using competitive programming. In: Verdú, E., Lorenzo, R., Revilla, M., Requeras, L. (Eds.), *A New Learning Paradigm: Competition Supported by Technology*, 157–172. Sello Editorial.
- Manev, K. (2008). Tasks on graphs. *Olympiads in Informatics*, 2, 90–104.
- 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 "Ugale" – finding new task types. *Olympiads in Informatics*, 3, 80–100.
- Pankov, P. (2008). Naturalness in tasks for olympiads in informatics. *Olympiads in Informatics*, 2, 115–121.
- Pankov, P. (2010). Real processes as sources for tasks in informatics. *Olympiads in Informatics*, 4, 95–103.
- TopCoder Website (2011). <http://www.topcoder.com>. Accessed 16.02.2011.
- Truu, A., Ivanov, H. (2008). On using testing-related tasks in the IOI. *Olympiads in Informatics*, 2, 171–180.
- Vasiga, T., Cormack, G., Kemkes, G. (2008). What do Olympiad Tasks Measure? *Olympiads in Informatics*, 2, 181–191.
- Vegt, W. (2006). The CodeCup, an annual game programming competition. In: *Perspectives on Computer Science Competitions for (High School) Students*. <http://www.bwinf.de/competition-workshop/papers.html>.
- Vegt, W. (2009). Using subtasks. *Olympiads in Informatics*, 3, 144–148.
- Verhoeff, T. (1990). *Guidelines for Producing a Programming Contest Problem Set*.
- Verhoeff, T. (1997). The role of competitions in education. In: *Future World: Educating for the 21st Century, a Conference and Exhibition at IOI*.
- Verhoeff, T. (2009). 20 years of ioi competition tasks. *Olympiads in Informatics*, 3, 149–166.
- Verhoeff, T., Horváth, G., Diks, K., Cormack, G. (2006). A proposal for an IOI syllabus. *Teaching Mathematics and Computer Science*, 4(1), 193–216.
- Voigt, J., Bell, T., Aspvall, B. (2010). Competition-style programming problems for computer science unplugged activities. In: Verdú, E., Lorenzo, R., Revilla, M., Requeras, L. (Eds.), *A New Learning Paradigm: Competition Supported by Technology*, 207–234. Sello Editorial.
- Wang, H., Yin, B., Liu, R., Tang, W., Hu, W. (2010). Selection mechanism and task creation of Chinese national olympiad in informatics. *Olympiads in Informatics*, 4, 142–150.



L. Hakulinen is a doctoral student at the Department of Computer Science and Engineering, Aalto University School of Science. He received his master's degree in computer science in 2010. Currently he works at the Learning + Technology Group (LeTech) and his research focuses on using serious games in computer science education.

Contest Environment Using Wireless Networks: A Case Study from Japan

Kentaro IMAJO

*Graduate School of Informatics, Kyoto University
36-1 Yoshida-Honmachi, Sakyo-ku, 606-8501 Kyoto, Japan
e-mail: imajo@iip.ist.i.kyoto-u.ac.jp*

Abstract. The Japanese Committee for the IOI (JCIOI) had held the second-round contests of the Japanese Olympiad in Informatics (JOI) in which nearly 60 contestants participated. The JCIOI did not employ networks in the second-round contests to avoid high cost. However, contestants sometimes make simple mistakes (e.g., format errors), so the JCIOI adopted wireless networks to respond to submitted source codes in the last second-round contest. This paper describes how we built the contest environment using wireless networks. In addition, we also discuss the judging system used in the second-round and final-round contests.

Key words: wireless network, programming competition, judging system.

1. Introduction

The Japanese Committee for the IOI (JCIOI) (The Japanese Committee 2006) has been holding contests to select delegates to compete in the IOIs. The contests consist of three rounds, which are the first-round, the second-round and the final-round contests.

The JCIOI had stopped sending delegates to the IOIs between 1998 and 2005 and resumed sending them in 2006. All of the contests since 2006 have been computer-based, and their tasks are IOI-like. Contestants compete over the Internet in the first-round contests. In the second-round and the final-round contests, contestants are gathered on one site to compete on allocated computers. In the last first-round contest, held in December 2010, 714 contestants participated. Likewise, 61 contestants participated in the last second-round contest, held in February 2011, and 18 of them were invited to the final-round contest. Although it was originally scheduled for March 2011, it was postponed because of the Great East Japan Earthquake on March 11, 2011. Four of them will be selected for delegates to compete the IOI 2011.

In the second-round contests in the years between 2006 and 2009, the JCIOI manually collected the programs written by the contestants by using USB flash memories at the end of the contest. In the final-round contests in the years between 2006 and 2009, the JCIOI prepared wired networks and offered the judging system PC²(California State University, Sacramento 2000) in order to select four delegates a year to compete in the next IOI. The JCIOI was not able to prepare wired networks in the second-round contests due to cost. However, in the second-round contests, one out of ten contestants on average

submitted at least one invalid program that either did not compile or produced an incorrect format. Those contestants left the contest without knowing that they had submitted invalid programs.

The author has developed a judging system named Imo Judge (Kentaro IMAJO 2008) for private contests (e.g., the 1st Imos Contest (Osaka University Competitive Programming Club 2008), JAG Practice Contest for ACM/ICPC Asia Regional Contest (ACM-ICPC Japanese Alumni Group 2010)) since 2008. The author heard about the above incidents with the IOI preparation contests and suggested to the JCIOI to adopt this judging system for grading programs in those contests. As a result, it was officially employed in the final-round contest of JOI in 2010 for the first time.

There are no wired networks in the facility where the second-round contests were held, and it is very costly to set up wired networks. Thus, the JCIOI have manually collected the programs written by the contestants until 2009. In order to prevent contestants from submitting invalid programs, we decided to adopt wireless networks in the second-round contests. We prepared wireless networks, a web server and a web-based uploading system in the second-round contest in 2010 for an experiment on wireless networks. The contestants competed on computers connected to the web server wirelessly, and they submitted programs through their web browser using the web-based uploading system. The system provided web pages including a web form to upload programs with an authentication.

In the second-round contest in 2010, we manually collected programs using USB flash memories as a backup alternative. Since there was no trouble with wireless networks and the system in 2010, we decided to offer a web-based judging system with wireless networks in the second-round contest in 2011. In addition to the functions of the web-based uploading system, it provided functions to execute the submitted programs and show the partial results during the contest.

In this paper, we describe how we built wireless networks for IOI-like contests in Section 2, and we describe our motivation in building such networks in Section 3. Furthermore, we describe a judging system that we used in Section 4. We conclude this paper in the Section 5.

2. Experiment

We prepared one web server, one wired router, two wireless routers and more than 60 laptop computers for the second-round contests in 2010 and 2011; see Fig. 1. The web server hosted a web service and a database service. Some of the laptop computers functioned as judge servers, whereas the others were used by the contestants. Every laptop computer was wirelessly connected to the web server through the wireless routers.

2.1. Wireless Networks

We used AirPort Extreme (Apple Inc. 2003) for wireless routers. AirPort Extreme has a function to construct a wireless guest network. Although laptop computers in the guest

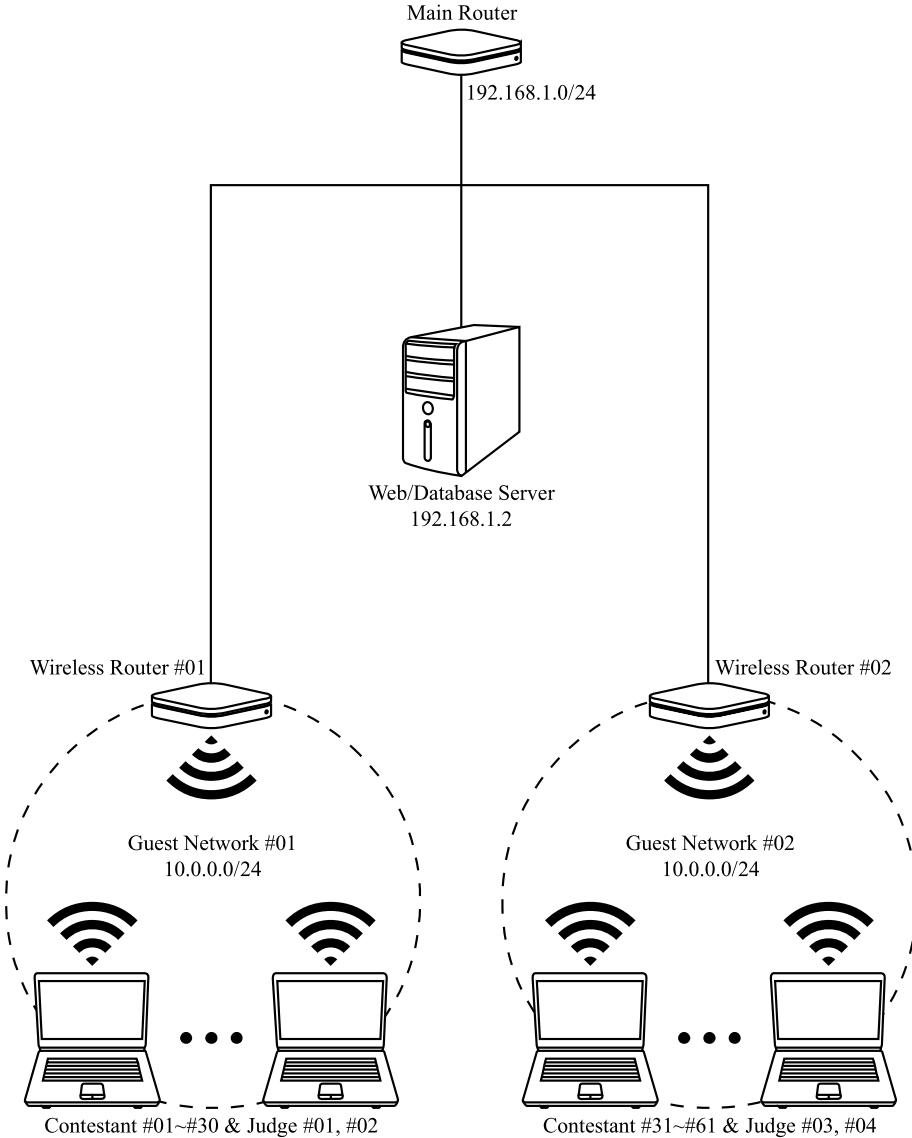


Fig. 1. Network layout in the second-round contest of JOI in 2011.

network can access computers in the Wide Area Network (WAN), they cannot communicate with other laptop computers in the guest network. In addition, they cannot be accessed passively at all since a guest network is protected by a Network Address Translation (NAT; Wikipedia 2001) firewall that modifies the IP addresses of the laptops to a single IP address, thus hiding them from outsiders. In the setup of Fig. 1, the laptops in each guest network can only communicate with the web server. Thus we used guest networks.

The wireless routers and the server were placed at one corner of the room, whereas the laptop computers were distributed around the room. However, one can use a different arrangement.

2.2. Judge Servers

We operated an IOI-like web service in 2011, which is a part of the Imo Judge System. The web service receives programs and displays the execution results on site.

The Imo Judge System consists of a web service and a judge service. The two services communicate through a database service that was installed on the web server. The judge service was installed on judge servers within the same environment of the contestants' computers. Access to the judge servers was restricted because the judge servers belonged to guest networks, and the judge service in the judge servers requires pulling submission data to grade from the web server. Therefore, it periodically checks the judge queue of the database service in the web server, and it grades each submission data and records its result to the database. The number of judge servers was at most four in the second-round contest in 2011.

3. Results and Discussion

This section describes why we relied on wireless networks and discusses the results of using them in the second-round contests in 2010 and 2011.

3.1. Durability and Security

This time we relied on AirPort Extreme wireless routers because they can officially manage 50 wireless connections at the same time according to Apple Inc., although ordinary wireless routers can manage about 20 at the most. In fact, each wireless router was able to manage more than 30 wireless connections simultaneously in the second-round contests in 2010 and 2011. Furthermore, we examined the durability by accessing the wireless twice a second from 60 laptop computers simultaneously and for a few minutes. We noticed that the networks were able to completely withstand the load under these circumstances.

Owing to guest networks, contestants cannot interfere with each other at all, and judge servers cannot be accessed passively. Hence, we only need to be mindful of the security of access to the web server and the execution on the judge servers.

3.2. Expenses

Since a wireless router can manage many users at the same time, the number of wireless routers that are required to build the wireless networks is small. The additional expenditure for the second-round contests was the price of those wireless routers at the most. The payment for the personnel who constructed these networks was minimal.

Compared with wired networks, wireless networks require no cabling. Wired networks on the other hand, not only require cables, but also cable wrappers. One should also take into account the difference in the number of staff needed to layout each type of network.

4. Judging System

A judging system must be capable of handling any program safely. Imo Judge prevents various hazardous programs from causing fatal problems. This section describes how Imo Judge handles some of those programs.

4.1. Protection of I/O Files

Imo Judge naturally executes programs with the privilege of a normal user. To grade programs for interactive tasks, we must restrict both reading-ahead and rewriting. Hence, I/O files must be hidden from programs. Data from standard input are given through a pipe of a process that has the root privilege in order to circumvent reading-ahead. Likewise, data from standard output are recorded through a pipe of a process that has the root privilege in order to circumvent rewriting.

4.2. Resource Limitations

Resource limitations are inevitable for judging systems on account of stability. UNIX-like operating systems can easily limit resource consumption using a function *setrlimit* (Linux man page 2008a). This function prevents programs from consuming excessive resources (e.g., memory, output file size, number of processes). Moreover, they can provide information of resource consumption (e.g., memory consumption, CPU time consumption) using another function *wait4* (Linux man page 2008b).

4.3. File Writing Pollution

Although a normal user cannot write in most of the directories except for its own home directory, the directories */tmp* and */var/tmp* can be written by anyone including normal users. Therefore, it is necessary to clean up those directories before executing any program; otherwise the programs can produce cache files and read them beyond their transaction.

4.4. Attacks in Compilation

It is possible to make a compiler run out of control using a *template*, which is one of the functions of C++. In addition, it is also possible to make a compiler output many warnings using *templates*. Thus, a judging system must assume that a compiler may not finish compilation or a compiler may output excessive warning messages.

5. Conclusion and Future Plan

We have demonstrated that contests with 60 participants can be managed safely with wireless networks. The truth is that such contests can be held anywhere with sufficient power supplies, and with no special facilities or heavy investment. Moreover, the setup was quite simple. Even if something goes wrong, it is easy to identify the source of the problem and fix it by replacing a wireless router.

We will release the Imo Judge System as an open source software so that anyone can hold contests without the knowledge of Linux, server administration or databases.

Acknowledgements. I appreciate the JCIOI for providing me with the precious opportunity of conducting this experiment. I would also like to thank to Takuya AKIBA whose helpful suggestions improved the security of the Imo Judge System. I am thankful for comments from anonymous referees and Marco Cuturi. Finally, I appreciate the feedback offered by Professor Seiichi Tani and Mayumi Morimoto.

References

- The Japanese Committee for the IOI (2006). *The Japanese Committee for the IOI*.
Available at: <http://www.ioi-jp.org/index-e.html>.
- California State University, Sacramento (2000). *CSUS Programming Contest Control (PC\2) Home Page*.
Available at: <http://www.ecs.csus.edu/pc2/>.
- Kentaro IMAJO (2008). *Imo Judge*.
Available at: <http://judge.imoz.jp/>.
- Osaka University Competitive Programming Club (2008). *The 1st Imos Contest*.
Available at: <http://oucpc.imoz.jp/contest/imos01/>.
- ACM-ICPC Japanese Alumni Group (2010). *ICPC Tokyo/Asia Regional Preliminary Contest*.
Available at: <http://judge.imoz.jp/?cid=11>.
- Wikipedia (2001). *Network Address Translation – Wikipedia, the free encyclopedia*.
Available at: http://en.wikipedia.org/wiki/Network_address_translation.
- Apple Inc. (2003). *AirPort Extreme*.
Available at: <http://www.apple.com/airportextreme/>.
- Linux man page (2008a). *setrlimit(2) – Linux Manual page*.
Available at:
<http://www.kernel.org/doc/man-pages/online/pages/man2/setrlimit.2.html>.
- Linux man page (2008b). *wait4(2) – Linux Manual page*.
Available at:
<http://www.kernel.org/doc/man-pages/online/pages/man2/wait4.2.html>.



Kentaro IMAJO participated in the IOI 2006 and won a bronze medal. He has been involved in contests of JOI as a coach since 2007. He participated in IOI 2009 as a member of Host Science Committee. He is currently a first-year master's student in Graduate School of Informatics at Kyoto University.

Codility. Application of Olympiad-Style Code Assessment to Pre-Hire Screening of Programmers

Grzegorz JAKACKI¹, Marcin KUBICA², Tomasz WALEŃ^{1,2}

¹ Codility Ltd., London, UK

² Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland

e-mail: {jakacki,walen}@codility.com, {kubica,walen}@mimuw.edu.pl

Abstract. Over past 20 years programming contests have grown substantial expertise in code assessment, yet it is rarely applied outside the contest settings. This paper presents a case of successful adoption of Olympiad-style code assessment technology in Codility, a commercial web service intended for pre-hire screening of software developers. We discuss correctness and soundness of the automated pre-hire screening, and characterise the class of programming tasks best suited for such application.

Key words: programmers recruitment, pre-hire screening, program evaluation, time complexity assessment, model-checking.

History

In 2005 one of the authors joined Exoweb.net, an outsourced software development business in Beijing, China. At that time Exoweb, a company with a workforce not exceeding 10 software developers, managed to secure a new software development contract generating an immediate demand for another 10 software developers (a demand which only continued to grow for many subsequent months). It has to be noted that due to the character of the contract Exoweb was very cautious about the overall skill levels of prospective employees. Based on previous experiences and the existing literature on the subject (most notably Spolsky, 2000; Weinberg, 1998), Exoweb management designed a recruitment process in which each candidate had to undergo three successful independent interviews with senior software engineers. Moreover, Exoweb management decided that adherence to item 11 of so-called *Joel Test* was crucial, i.e., that each candidate had to convincingly demonstrate the ability to write correct code in some imperative language (the choice of the language was left to the candidate).

As soon as Exoweb started advertising the job positions, it became apparent that the interviews with senior engineers were the main bottleneck of the recruitment process. There were only three software engineers in the company able to conduct the interviews. The hires-to-candidates ratio was around 1 : 50 and the time spent in interviews with one candidate was 20–180 min with an average of approximately 100 min, yielding the approximate amount of 80 hours of senior engineer’s time per one hire.

The interviewers observed that an easily identifiable reason of many failed interviews was the inability to write bug-free code. In an attempt to reduce the number of interviews, Exoweb management decided to administer a written programming to each candidate prior to interviews. Soon the test has been moved from paper to computer and augmented with an automated checker (Exobench) developed in-house in the spirit of checkers employed by IOI and ACM CPC. Exobench was implemented as a command-line tool and non-technical admin staff has been trained to operate it without assistance of engineering staff.

It turned out that approximately 9 in 10 candidates were being reliably rejected without wasting any senior engineer's time in the interviews. Moreover, the admin staff able to operate the screening software was much cheaper (with salaries lower by a factor of 3–5) and much easier to hire than technical interviewers. The total amount of senior engineer's work per one hire dropped from 80 hours to 12 hours. The process has been subsequently executed for 30 months without major modifications, covering 2.5 thousands applicants, with automated pre-screening efficiency of 1 : 10 and the interview rejection rate of 1 : 5.

In 2008, given the successful validation at Exoweb, the idea of automated pre-hire screening has been re-implemented as an on-line service and offered under the trade name Codility.

It is worth mentioning, that the founders of Codility are all former IOI contestants and medalists, and they are (currently or have been previously) associated with Polish Olympiad in Informatics (POI). The two sources of experience, the industry and the Olympiad (Diks *et al.*, 2007), have been augmented to create this innovative and successful enterprise.

Tasks and Task Preparation

The purpose of pre-hire screening differs much from the goals of programming contests. In both cases the focus is on differentiating participants according to their skills. In case of programming contests the goal is to select top contestants and fairly distribute points among all the other contestants. On the other hand, the goal of pre-hire screening is to eliminate all the applicants not possessing sufficient programming skills. Hence, the tasks used at Codility are generally much more elementary (cf. Verhoeff *et al.*, 2006; Diks *et al.*, 2008). Also, the assessment result is more of a binary nature (reject or send for interviews), than in IOI. It is quite natural, since the final outcome is also binary – the goal is to select all the applicants that can be of interest for the employee. It is not important to differentiate those that are eliminated, and those who pass will undergo further selection procedures anyway.

The variety of programming languages, that can be used by applicants is much wider than in IOI or other programming contests. The reason is, that applicants can hardly be expected to train focusing on Codility settings. To the contrary, employees expect Codility to asses applicants' programming skills, regardless of the particular programming languages they are using. Hence, Codility system supports most popular programming languages, including: C, C++, C#, Java, JavaScript, Pascal, Python, Perl, PHP and Ruby.

That poses challenges on both task preparation and evaluation. The latter ones include comparable assessment of solutions written in fully compilable languages (such as C and C++), byte-code level compilable languages (such as Java and Python) and scripting languages (such as PHP).

Task preparation consists of similar elements as in programming contests (cf. Diks *et al.*, 2008): task formulation, model solutions, alternative slower solutions, example wrong solutions tests and result checker. Some details however, are different.

Typical task formulation is much shorter and simpler than in IOI. Narrative story is not present. Textual problem description is usually accompanied by an example, input data constraints (sometimes implicit), formal problem definition and interface specification. The input-output interface slightly differs from interfaces used at IOI or other Olympiads in Informatics. The task solution comprises a function written to a specific interface, which significantly reduces the burden of parsing/unparsing the data – the input data is given as function arguments, the output data is passed via return value. In IOI terminology, the tasks are of batch nature. Interactive and open-data tasks are not supported, and actually – not needed. For each supported programming language, an interface specification and the formulation are automatically augmented to form the whole task statement.

As we have already mentioned, tasks used by Codility are much easier, than those that can be seen at programming contests. Still, similarly to IOI, there usually exists a trivial, but suboptimal solution. The ability to produce such a solution proves only elementary programming skills, while producing the optimal solution requires some algorithmic skills. An example task formulation can be found in Appendix A.

The number of supported programming languages is significant, and what more important, evolves all the time. Developing all required solutions in all the programming languages would be a burden. Moreover, the number of tasks that are ready to be used and are maintained exceeds a hundred and is growing, and extending tasks analyses whenever a support for a new programming language is included would be simply unfeasible. Hence, a more generic and flexible framework had to be developed. Python has been chosen as a “front-end” programming language – solutions written in (a carefully chosen subset of) Python are automatically translated to all the other supported programming languages. Contrary to IOI practice, the test cases are not pre-determined and are replaced by a testing module (written in Python) which (among other checking procedures) runs solution against dynamically generated test cases. With a support of dedicated libraries encapsulating particular execution environments, programs written in any supported programming language can be run against the Python tests in an uniform manner. Time limits are automatically calibrated for all the supported languages independently, based on running times of model solutions automatically translated into target languages.

Tests are prepared according to similar outlines, as in Olympiads in Informatics (Diks *et al.*, 2008). They include:

- correctness tests of various sizes,
- border-cases tests,
- efficiency tests.

Compared to IOI and other Olympiads in Informatics, there is more stress on correctness and border-cases, and time complexity is a bit less important. Due to the fact that test-case execution is not a pre-determined set of test-runs, but rather a function in Python test module, arbitrary evaluation logic can be easily implemented, including execution of multiple test-runs under one test-case to emulate IOI test grouping.

Checker as a Web Service

Codility operates as a website (codility.com) offering pre-screening service to software recruiters. As such, the service has two main user types: candidates and recruiters. Generally users of each type interact with the system through a separate interface.

From the very beginning Exobench (and later Codility) were designed to minimize the need for technical staff supervision in the very first line of screening of candidates. Contrary to IOI, no technical supervision was intended during the test and tests were meant to be applied by admin staff without extensive technical training. Furthermore, the nature of the recruitment process called for tests that are less “ruthless” than in IOI, in particular some tolerance to simple, avoidable mistakes (e.g., wrong name of the function, missing return statement) was desired. These factors led to introduction of the candidate’s interface. Initially, it was a CLI command, enabling the candidate to compile the solution and run it against a small pre-defined subset of test-cases (usually just the example test-case discussed in the problem formulation). Ability to compile the code and execute a simple test with one command gave candidates a quick way to run a sanity check, at the same time ensuring that they have a chance to fix simple mistakes before submitting the solution. In such a setup it was also easy to assure that the execution of a solution via candidate’s and recruiter’s interfaces take place in the same controlled environment, minimizing the discrepancies due to different compiler options, different ways of calling the solution function, different seeds of pseudo-random generators etc. Candidate’s interface bears some similarity to the contestant’s interface present in ACM CPC. This contest enables contestants to submit solutions for evaluation during the competition as well as obtain the information about correctness of the submission. The main difference is that Exobench/Codility by design limits the number of test-cases executed as a part of sanity check to reduce the turn-around time and to prevent the technique (well-known among CPC contestants) of extracting information about test-cases through multiple submissions. Contrary to ACM CPC, Codility does not penalize candidates for multiple sanity check submissions.

When the system has been re-implemented as web service, the candidate’s interface took form of a micro-IDE consisting of:

- task description pane,
- code editor,
- control buttons, including the “VERIFY” button, enabling the candidate to run the compilation and sanity check,
- countdown timer.

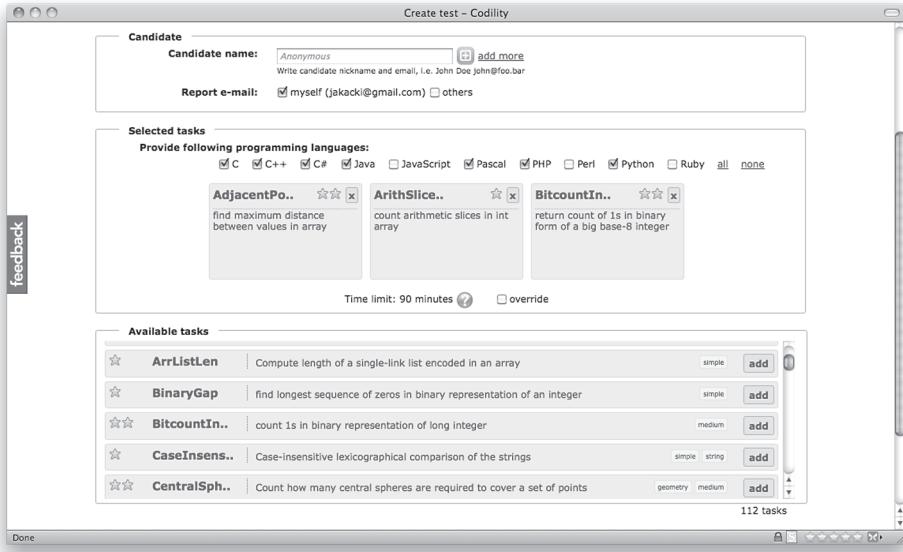


Fig. 1. Codility recruiter's interface for building new tests.

Subsequently more features were added, in particular ability to add candidate's own tests to the sanity check run.

The recruiter's interface of Codility contains several screens and is mostly focused on building, replicating and administering the tests, as well as reviewing the candidate's ranking and detailed results of tests.

Evaluation Technology

Codility was initially based on the evaluation model evolved by IOI and ACM CPC, i.e., execution of test-runs in the sandboxed environment. Codility augmented this model with several technologies, two of which are discussed below.

Time Complexity Assessment

Codility test report consists of a numeric score, measuring the quality of the solution (much like the score in IOI) together with a detailed report, including the test results for each test-run. The test-run results constitute documentation of solution shortcomings and provide additional insight for a technical recruiter. Codility team has observed however, that compared to IOI, the software engineer looking at the test-run results has a limited knowledge about the nature of the test-runs. Moreover, it has been identified, that a single most important piece of information that the recruiting engineer is trying to infer from the test-run results is the assessment of the performance scalability of the solution. This observation led to the development of proprietary technology for assessment of asymptotic

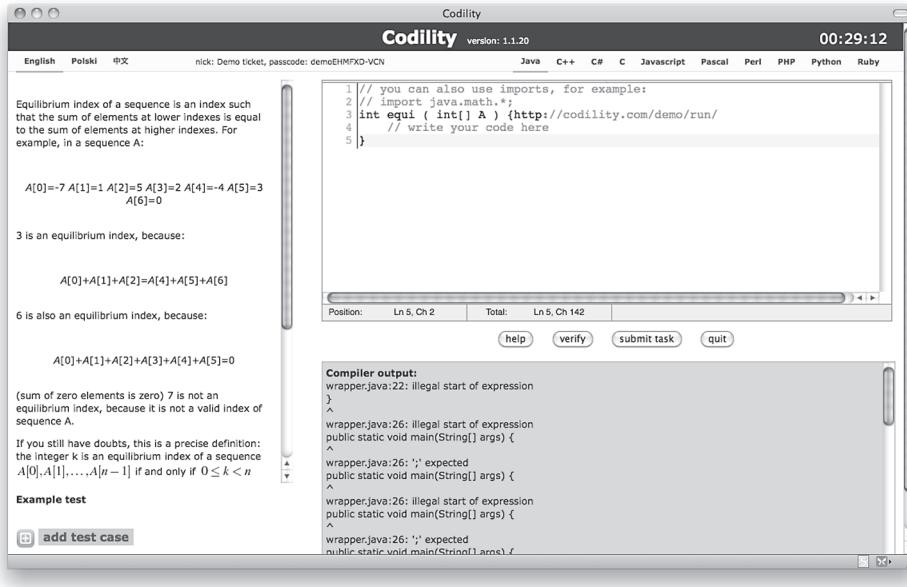


Fig. 2. Candidate's interface (micro-IDE). Task description shown in the left pane; code editor and error console in the right panes.

Example test

WRONG ANSWER (got 7, which is not valid array index)

add test case

write here your own test data	remove
<div style="border: 1px solid black; padding: 2px;">[1,2,3]</div>	
NO RUNTIME ERRORS (returned value: 3)	
write here your own test data	remove
<div style="border: 1px solid black; padding: 2px;">[-7, 1, 5, 2, -4, 3, 0]</div>	
NO RUNTIME ERRORS (returned value: 7)	

Fig. 3. Part of web interface for adding candidate's own testcases.

computational complexity of the solutions. The assessment engine attempts to derive a simple closed form of the function bounding the running time. The results are presented as an expression in O-notation.

Model Checking

Test-run based evaluation is definitely a proven way of assessing the functional correctness of solutions, however in general the equivalence of the model solution and the tested solution cannot be established computationally (which is rather fundamental corollary of Rice's theorem). Any finite set of test-runs is doomed to contain "holes". Imperfections of a particular solution may manifest themselves just in these uncovered holes, remaining undiscovered by the particular test-run set. Codility has conducted a research to find out how often solutions are "overscored" due to insufficient test-run coverage. The discussion of this research is beyond the scope of this paper, however it led to another interesting augmentation of checking technology. Codility checker employs a hybrid of random test generation, enumerative test generation, symbolic interpretation and model checking to proactively "cover the holes", i.e., to generate the test-runs aimed at derailing particular overscored solutions.

Scalability

Unlike IOI, part of Codility checking has to happen in real time. Candidates' interface enables execution of sanity checks, including compilation and execution of certain test-runs. To be of any use, the feedback from these checks has to be presented to candidates quickly, ideally within seconds. This requirement calls for an architecture which can provide enough computing power to run the checkers in case of demand spikes. Codility is implemented as highly scalable distributed application within Amazon EC2 computing cloud. Elastic architecture enables on-the-fly addition of computing power if the performance metrics drop below threshold levels. Beyond standard performance metrics (e.g., load average) carefully chosen set of application-specific metrics is constantly monitored, including (a) the number of queued checker jobs, (b) min/avg/max checking time within last 5 minutes, (c) number of ongoing tests. To assure service quality for customers in diverse geographies, Codility servers are distributed over data-centers in US, Ireland and Singapore.

Correctness and Soundness

Risking a gross oversimplification we may say that evaluation systems, like IOI checkers or Codility, attempt to assess a group consisting of good programmers and poor programmers. The goal is to obtain a concrete division of the group into winners and losers, so that the set of winners most closely matches the set of good programmers, and the set of losers most closely matches the set of poor programmers. The notion of being good or poor programmer is intuitive rather than well-defined and the difficulty of building an evaluation systems lies in approximating these fuzzy concept with concrete divisions into winners and losers.

Codility team has identified two desired properties of an evaluation system (named after similar properties of formal systems in mathematical logic):

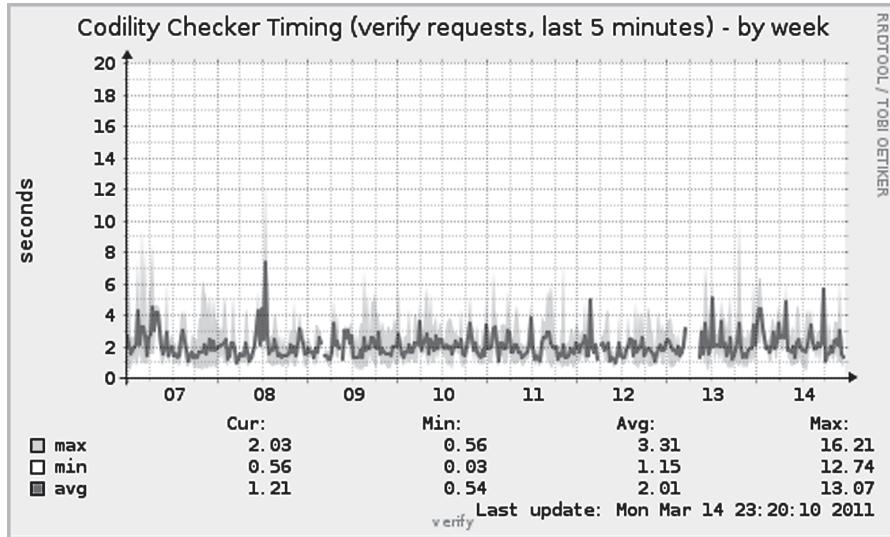


Fig. 4. Min/avg/max turnaround times of “VERIFY” checks tracked by Munin on Codility production servers.

- **correctness** – all individuals evaluated as winners are good programmers (all bad programmers are evaluated as losers),
- **soundness** – all good programmers are evaluated as winners (all individuals evaluated as losers are poor programmers).

Furthermore, Codility team has observed that different properties are more desired in a contest settings and in a pre-hire screening settings.

Correctness is crucial for contests. It would be detrimental to the contest reputation if the winners are proven to be inferior programmers. Correctness is also critical to reaching the goal of regional contests, aiming to maximize the chances of the regional representatives. Soundness, on the other hand, is good to have, but less crucial. It is an accepted fact, that a programming champion may not succeed in a particular competition due to bad luck, bad day, etc. Undoubtedly evaluation system which can provide soundness is desired, but occasional lack of soundness is acceptable.

In pre-hire screening correctness is not considered crucial. Due to the nature of the pre-screening process (high-pass filter before more elaborate in-person interviews), it is accepted that occasionally a poor programmer sneaks through the evaluation system, due to sheer luck, pre-existing knowledge of test problems or cheating (e.g., impersonation by a smarter colleague in an on-line test). As long as such incidents are infrequent, the screening efficiency is not seriously affected. Codility surveyed professional recruiters about their perceived percentage of candidates who cheat in on-line pre-screening tests. The answers varied, but never exceeded 10%. With Codility screening efficiency reaching 90%, even the most pessimistic scenario yields a screening ratio exceeding 80%.

Soundness, on the other hand, seems very important in pre-hire screening. Programming contests see abundance of talent and generally there is always enough candidates to put somebody on the podium, but in recruitment the talent is scarce. In organizations that

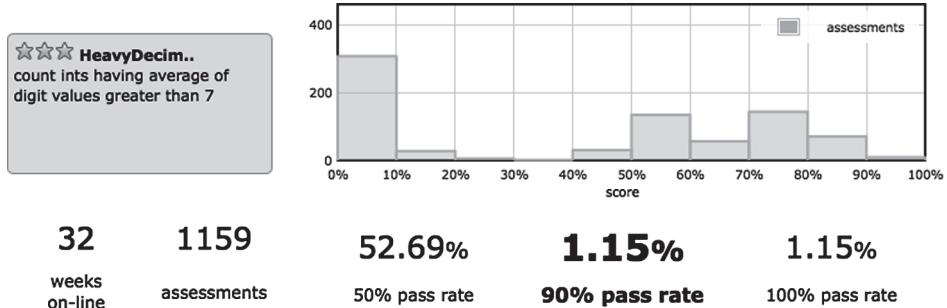


Fig. 5. Fragment of Codility webpage showing distributions of scores for a particular problem. Two peaks are visible around popular suboptimal solutions.

have exhausted peer recommendations, recruiters sift through candidates from the job market, most of whom are unfit for the job. At Exoweb, a case study described earlier in this paper, roughly 50 candidates needed to be pre-screened and subsequently 5 of them interviewed for one of them to be hired. With these numbers, a campaign aiming to hire 10 persons needs to pre-screen 500 and interview 50. An evaluation system that evaluates one additional poor programmer as winner per each ten winners, necessitates 5 more interviews. On the other hand, a system that evaluates one good programmer as a looser per each ten good programmers, necessitates over 50 more candidates to be pre-screened. As long as the automated pre-screening is cheap, time occurs to be a crucial factor here – 5 interviews with poor programmers will take up to 5 hours of senior engineers’ time, but generating 50 new reasonably matching leads from a job ads in 5 hours is expensive and hardly attainable. (Certain on-line channels in very large job markets, like China, are capable of generating large amounts of leads in short time, however the prospective gain is usually wrecked by drastically high pre-screening rejection rate.) In the recruitment setting soundness is more desired than correctness.

Scoring

Codility scoring system generally does not differ from the one used by IOI. Candidates solve between 1 to 6 problems (with average of 2.24 problems per person), the total score is the sum of scores obtained for individual problems. Contrary to IOI, the problems are written and timed with assumption, that a competent programmer should be able to obtain the score close to 100%. Generally score distributions differ substantially from problem to problem, with numbers of candidates scoring 100% ranging from as high as 48% to as low as 1.15%. 6.9% of all paid Codility evaluations achieved top score, 11.7% achieved score exceeding 90%. Distributions are generally biased towards lower end with peaks around scores corresponding to suboptimal solutions.

Test name	Stats
big1	 <ul style="list-style-type: none"> OK: 81 Runtime error: 39 Wrong answer: 45 Time-limit exceeded: 478
big2	 <ul style="list-style-type: none"> OK: 10 Runtime error: 42 Wrong answer: 30 Time-limit exceeded: 561
example	 <ul style="list-style-type: none"> OK: 498 Runtime error: 42 Wrong answer: 87 Time-limit exceeded: 21
extreme_equals	 <ul style="list-style-type: none"> OK: 487 Runtime error: 57 Wrong answer: 89 Time-limit exceeded: 15
medium1	 <ul style="list-style-type: none"> OK: 416 Runtime error: 69 Wrong answer: 106 Time-limit exceeded: 57
medium2	 <ul style="list-style-type: none"> OK: 280 Runtime error: 63 Wrong answer: 90 Time-limit exceeded: 215
medium3	 <ul style="list-style-type: none"> OK: 227 Runtime error: 64 Wrong answer: 74 Time-limit exceeded: 283

Fig. 6. Part of Codility analysis of failure reasons for particular test-cases.

Conclusions

Program evaluation methods, developed by IOI and other programming contests, can be successfully applied in pre-hire screening. Codility founders have validated this claim, first in an enclosed setting of one large hiring campaign, later in the market. Today Codility is a profitable business catering to customers like Nokia, Siemens or Barnes&Noble. Codility-funded research led to development of two innovative technologies supporting code evaluation, namely automated time complexity assessment and reactive tests augmentation. Hopefully the innovation in this space will lead to further advances in evaluation technology employed by programming competitions.

References

- Diks, K., Kubica, M., Stencel, K. (2007). Polish olympiad in informatics – 14 years of experience. *Olympiads in Informatics*, 1, 50–56.
- Diks, K., Kubica, M., Radoszewski, J., Stencel, K. (2008). A Proposal for a task preparation process. *Olympiads in Informatics*, 2, 64–75.
- Spolsky, J. (2000). *The Guerilla Guide to Interviewing*. Joel Spolsky's blog 2000.
<http://www.joelonsoftware.com/articles/fog0000000073.html>.
- The Joel Test*. <http://www.joelonsoftware.com/articles/fog0000000043.html>
- Verhoeff, T., Horváth, G., Diks, K., Cormack, G. (2006). A proposal for an IOI syllabus. *Teaching Mathematics and Computer Science*, 9(1).
- Weinberg, G.M. (1998). *The Psychology of Computer Programming*, Dorset House.

Appendix A

Example Task Description

Equilibrium index of a sequence is an index such that the sum of elements at lower indexes is equal to the sum of elements at higher indexes. For example, in a sequence A :

$$A[0] = -7, A[1] = 1, A[2] = 5, A[3] = 2, A[4] = -4, A[5] = 3, A[6] = 0.$$

3 is an equilibrium index, because:

$$A[0] + A[1] + A[2] = A[4] + A[5] + A[6].$$

6 is also an equilibrium index, because:

$$A[0] + A[1] + A[2] + A[3] + A[4] + A[5] = 0.$$

(sum of zero elements is zero) 7 is not an equilibrium index, because it is not a valid index of sequence A .

If you still have doubts, this is a precise definition: the integer k is an equilibrium index of a sequence $A[0], A[1], \dots, A[n - 1]$, if and only if, $0 \leq k < n$ and:

$$\sum_{i=0}^{k-1} A[i] = \sum_{i=k+1}^{n-1} A[i].$$

Assume the sum of zero elements is equal zero. Write a function:

```
int equi(int[] A);
```

that given a sequence, returns its equilibrium index (any) or -1 if no equilibrium indexes exist. Assume that the sequence may be very long.



G. Jakacki (1975), M.Sc. in computer science, CTO&CEO at Codility Ltd., part-time lecturer at Warsaw University, formerly software developer and technical team leader at Exoweb.net and Synopsys Inc., contributor to Polish Olympiad in Informatics. His interests focus on building and motivating software development teams, practical applications of software theory, semantics of programming languages.



M. Kubica (1971), PhD in computer science, assistant professor at Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, scientific secretary of Polish Olympiad in Informatics, CEOI'2010 chairman and former chairman of Scientific Committees of IOI'2005 in Nowy Sącz, CEOI'2004 in Rzeszów, BOI'2001 in Sopot, and BOI'2008 in Gdynia, Poland. His research interests focus on combinatorial and text algorithms.



T. Waleń (1978) PhD in computer science, principal developer and founder of Codility, is a software practitioner and theoretician, lecturer at University of Warsaw. Designed and implemented evaluation solutions used at major programming contests worldwide and was one of principal contributors in USOS, the nationwide education management system. Involved in teaching advanced subjects in computer science, as well as a research on combinatorial algorithms.

Preparing for the IOI through Developmental Teaching

Vladimir M. KIRYUKHIN, Marina S. TSVETKOVA

*Dept. of Informatics and Control Processes, National Research Nuclear University "MEPhI"
31 Kashirskoe Shosse, Moscow 115409, Russian Federation*

*Academy of Improvement of Professional Skill and Professional Retraining of Educators
8 Golovinskoe Shosse, Moscow 125212, Russian Federation
e-mail: vkiryukhin@nmg.ru, tsvetkova@lbz.ru, msym@lianet.ru*

Abstract. The upbringing of future winners of the IOI is a difficult task with many aspects. The best teachers and coaches in many countries are struggling to solve it. Up to now, ready-made recipes have not been offered. The experience gained in some countries could help to identify patterns of working with talented children, which have a significant impact on their success, combining the development of talent with the specifics of the child's age. This article describes the methods of preparation of Russian students for informatics olympiads, including the IOI, which have proved their efficiency for many years. Particular emphasis in the preparation of students rests on the development of their talent, not on solving many competition tasks. On this basis, the most appropriate ways of training gifted students, as well as the used information resources and modern information technologies, are described.

Key words: informatics, computer science, secondary school education, informatics olympiads, IOI, preparation for informatics olympiads, methods of work with talented children, reproductive teaching, developmental teaching, anticipatory teaching, Internet resources for informatics olympiads, self-study work of gifted students.

1. Introduction

There is no doubt that a significant success in the IOI can be achieved only by students who are gifted by nature. Along with the necessary knowledge and the skills required for solving non-standard tasks, such students have important personal qualities – self-improvement abilities, self-confidence, striving for success, etc.

Since the IOI is an objective estimation of the degree of gifts possessed by its participants, all of the above should be voiced in the development of talent through preparation for the IOI. Clearly, in this case the process of personal development must be organized in a way, which organically combines mastering of knowledge and cognitive methods, skills formation and development of creative activity; defined by the content of the IOI. From the other side, it is necessary to include, in all topics of the IOI content, elements that promote the development of personal qualities such as concentration on goals, perseverance to achieve results, responsibility and self estimation, mutual help and support in team-

work, friendliness, empathy to the opponent, self confidence, focus on high achievement, an adequate level of claims, etc.

Russian specialists have accumulated considerable experience in the identification and development of gifted students in informatics and information technologies within the Russian Olympiad in Informatics (RusOI). They use specific methods that encourage creativity, based on a system of tasks, which require problem-oriented, retrieval, heuristics, research and project-oriented working, both as individuals and in a team. These methods have highly informative and motivating potential, which corresponds to a higher level of cognitive activity and interests of gifted students.

And more, the recent forms of organization for working with gifted children, especially in the training for informatics olympiads, has undergone strong renewal under the rapid development of education and information technologies and scientific knowledge. A particularly strong influence on this has been the emergence of the Internet on the life of schools, children and families, because the students get real access to resources that help to demonstrate their talent. In this paper we consider in details how this affects the preparation for programming contests, including the IOI.

In Section 2 different traditional models for organization of students' preparation are considered. Section 3 is devoted to creative development of talents in the process of preparation. Different methods and forms for preparation for the IOI are presented in Section 4 and Section 5, and some conclusion – in Section 6.

2. Traditional Models of Preparation for the IOI

Recently, widespread opinion has been that the more competition tasks a students have solved in the preparation process, the more chances they have to succeed at the IOI. Moreover, the process of preparation is structured in such way that, first, the student solves the proposed tasks, and then, after evaluation of the solution, looks for mistakes with the help of a coach or the available resources, including Internet resources. When the students are not able to produce solution, the coach explains to them one possible algorithm, and the student memorizes that algorithm.

This approach, founded on reproductive teaching, is popular due to habit. Its basic disadvantage is the absence of a developmental aspect, as it is constructed on a reproduction of patterns. Because of their extraordinarily ability, gifted child cram into memory patterns for various difficult tasks and the techniques to solve them, subsequently reproducing the stored solutions (Fig. 1). We will call such a model the “Reproduction of patterns with support of memory”. In this case, development of theoretical knowledge and technological abilities is subordinated to features of the tasks and aimed at accumulation, not on discovering the underlying problem-oriented situation.

Another way to apply the above mentioned model is the participation in as many different programming contests as possible, which are currently carried out often enough, especially on the Internet, with subsequent analysis of unsolved or partially solved tasks. Certainly, participation in programming contests is necessary, and it has a lot of positive

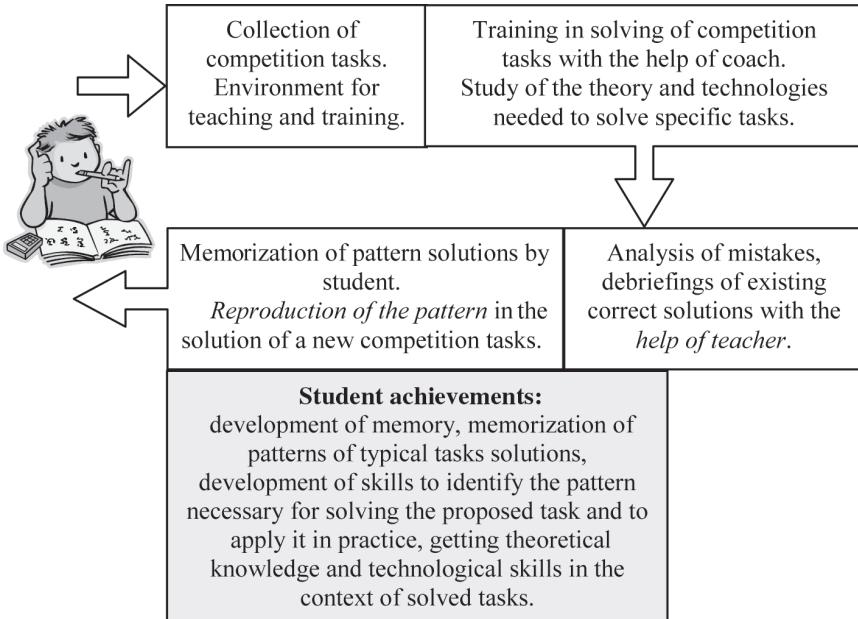


Fig. 1. The model “Reproduction of patterns with support of memory”.

aspects: it teaches the mobilization of forces, persistence, and forms the ability to sustain long time pressure. But competitive activities are only implementation and demonstrate only the already achieved capabilities of students. In this case, considering participation in programming contests as preparation for the IOI would be not quite correct. In this case a gifted students fall into the training environment imposed by the specific competition, where they could not choose new trajectories or discover more useful topics and tasks. New knowledge is perceived in the context of the proposed tasks. It is formed haphazardly and is not fixed as a private intellectual achievement of the student.

It is possible to emphasise positive influences of the reproductive approach to memory development and acquisition of experience in specific field. Its disadvantages are the lack of focus on creativity and the search for non-standard solutions, on novelty of ideas, on raising the desire to seek own solutions and striving for self-discovery of truth (possibly well known for the teacher, but new for the student).

Clearly, the reproductive approach aims not at developing of talent but at intensive training on the basis of another's solutions or solution patterns. It displaces the emphasis of teaching from the development of talent to the intensive practice and memorization. This leads to insufficient study of the theoretical bases of trained topics, and as a consequence – decreases the potential for the generation of new ideas.

When in a competition task set there is a task which the students have not solved earlier, they do not find in their memory a pattern for reproduction, become stressed and lose belief in their forces; as a result – refuse to think out an original solution. This precisely shows how the violation of rules of development of talent leads to decrease of creative activity and volition of the student, so necessary during the competition. Results of partic-

ipants prepared in such a way at the IOI are often rigidly differentiated: 80–100% of the points if the pattern of a solution is known or 0–30% if the pattern is not known. Moreover, such students demonstrate best result in their last school year, which is quite natural.

The model of reproduction of patterns with a support of memory justifies itself only in a case when a gifted student, for some reason, has not been involved in olympiad movement early and his preparation is limited to last 2–3 years (grades 9–11) of specialized school. It demands intensive work and endurance, does not give steady results on competitions and leads to stress.

Further development of the above model is based on the formation and phase expansion of the so-called in Russian pedagogy *the zone of nearest development* of the student, which is an area of further in-depth and phase learning topics in informatics required for olympiad preparation and interesting to student. Such approach gives students the support to move towards the new knowledge and skills independently, to the discovery of new ideas and solutions without the help of a coach, that is, to create a product of their creativity.

In this context, solving competition tasks has to play the role of mediator, leading the talent on a trajectory of development. Analysis of another's solution can be a valuable pedagogical tool only if it is based on a partial student's solution and preliminary independent analysis and search for mistakes. Such analysis of a difficult task solution becomes not a pattern, but a support in teaching. If, after an analysis of a correct solution, the students can find their own original solution or can create a task with similar solution and develop tests for it, then it is possible to say that reproductive teaching is built in competently in the development system and provides good results (Piaget, 1983). Such methodology is based on the productive activity of the student and called the Model of productive teaching “Stages of development” (Fig. 2).

Certainly, using this model, it is possible to reach a certain level of qualification, and even to show good results in some competitions. However, it does not guarantee success at the IOI, especially when the goal is the gold medals. The results of the IOI participants prepared using this model usually range from 30% to 60% of the points for complex tasks and grow gradually with the maturity of the student. The model justifies itself in a case when the gifted students are involved in regular preparation, from 7th to 11th grade, studying informatics at school with obligatory regular additional preparation in a system of elective courses in informatics and mathematics.

“Stages of development” is the traditional model of preparation in Russia today. It requires regular attended classes, work with a coach in a group, home self-preparation and use of resources on the Internet. It should be emphasized that it is a teaching model with elements of development, but it is still not a development teaching system acting in harmony with the growth of gifted students and not even an individual plan for harmonious development embedded in the educational environment at school.

To achieve stable progress in the development and manifestation of the talent for solving complex tasks, it is necessary to enrich the “Stages of development” model with developing approaches and, first of all, to move from a rigid binding to competition tasks only, to someone else's solution and to an age threshold of complexity in informatics

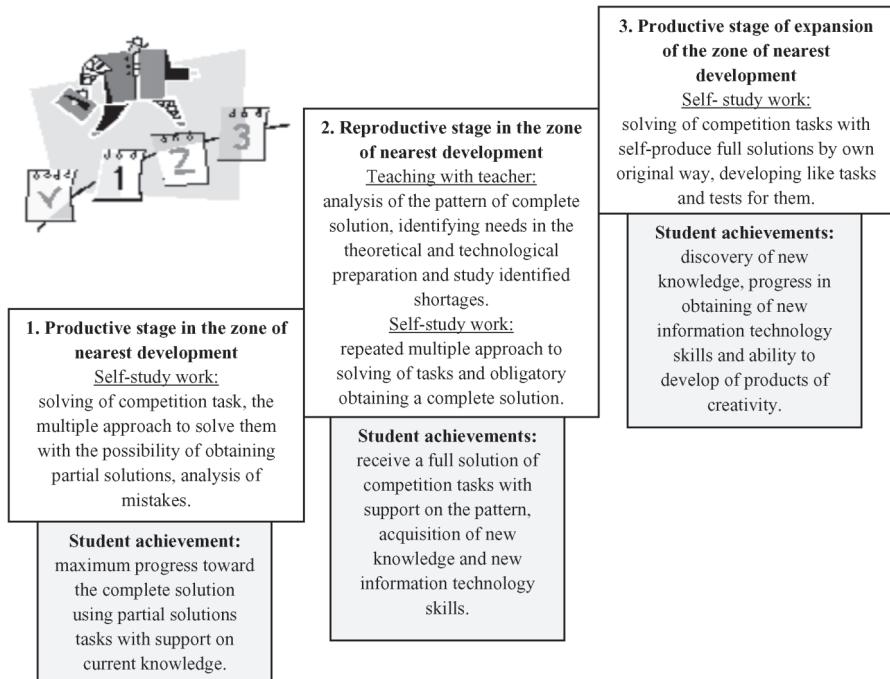


Fig. 2. Model of productive teaching “Stages of development”.

teaching at school. It is important to refocus on the discovery of new knowledge by the students themselves through a wide range of courses in topics that help the development of talent. It is necessary to ask the students to search for original solutions to difficult tasks and to encourage them to compose their own tasks on the basis of patterns. It is also important to expand preparation, applying informatics in educational projects of school, which help students' development with a combination of new knowledge and skills, placing them in an area of the nearest development. In other words, it is necessary to provide the formation and expansion of an individual student's development horizon in the frame of the olympiad preparation system prevailing today.

3. Creative Development of Students in Olympiad Preparation Environment

In Russia, for the development of creative abilities of students, the theory of *developmental teaching* offered in the 30 years of the twentieth century (Vygotsky, 1991) is widely used. According to this theory, the leading aspect of methods for development of creative ability of children is the shift from a teaching process to a process of learning and development. Another aspect of applying this theory, reflecting the orientation of modern society to the knowledge processes, is the assimilation of new scientific and technological achievements, particularly in the field of informatics and information technologies, and their introduction into the context of the lives and studies of gifted students.

In preparing students for the IOI attention should also be given to practical skills. From participants of the IOI it is required not only to generate ideas and to develop algorithms for the solution of a competition task, but also to implement it correctly in the given programming environment. This presupposes for IOI participants not only the knowledge of one programming language, but also ability to work with corresponding operating system, compiler, debugger and the other software available during the competition. It is no less important for the IOI participants to possess the technology for testing programs, including the development of tests and auxiliary programs for the testing of their solutions, because tests development could be quite complicated itself and also requires a creative approach.

As experience has shown, work with gifted students in the process of preparation for the IOI determines the basic principles of *cooperative pedagogy* that characterizes the work of teachers and coaches. Such pedagogy assumes the refusal of intellectual slavery and is implemented on the experience both of the student and the teacher.植 another's ideas, even of great people, is still not a key to independent thinking. To build up a student's inner world, to express it through words and good deeds, that is the essence of creative activity of the student, and the role of teachers or coaches is to help as much as possible for resolving this problem.

Such support from a teacher or coach allows the student to find original ways for solving tasks, often not known by professionals. Such small achievements of gifted students also bring their unique contribution to the work of adults, opening up for them new aspects and ways to approach competitive tasks. And this is the invaluable effect of cooperative pedagogy for coach and student: each contributes to the development of creative potential of the other.

Applying the area of the nearest development concept in preparation for the IOI allows reaching the nearest border of developmental teaching – a complexity threshold. The first steps in the manifestation of student's talent can be regarded as overcoming this threshold. In combination with additional (individual or in small groups) teaching, advancing teaching of gifted students in the area of their cognitive interests forms a development horizon of a particular gifted student, which is called the “mental horizon” (Dewey, 1990) – the upper bound of developmental teaching during the given age period. Naturally, the development horizon is extended in the process of the student's growth and increasing of their individual achievements in the area of the nearest development. Thus, movement toward the development horizon is going on at an individual pace on complexity thresholds, each of which has their own area of the nearest development. Advancement to the individual development horizon with a teacher or coach is fully implemented in solving of competition tasks. Competition tasks in this sense are advancing tasks, and various programming contests represent the complexity thresholds for the student. Undoubtedly, they make a basis of anticipatory teaching model which could be called “Development horizon” (Fig. 3).

This model of training gifted students, based on anticipatory teaching, should be applied from early childhood. This will allow the building of the development horizon of a gifted student, constantly expanding the area of the nearest development, solving difficult tasks, and even creative and research tasks, challenging unique student abilities to

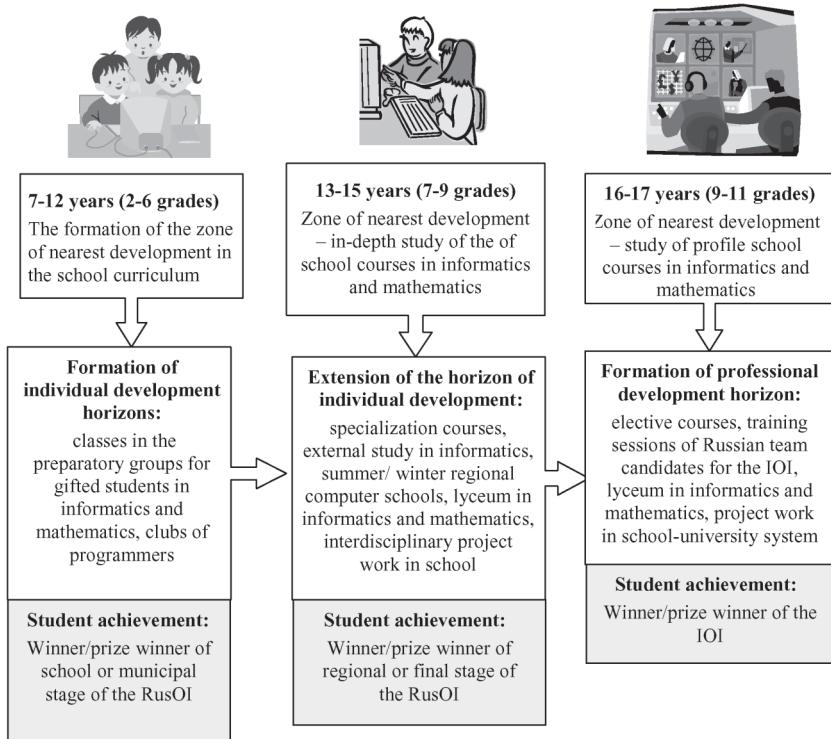


Fig. 3. Anticipatory teaching model “Development horizon”.

apply the knowledge in practice. It is possible not only to solve competition tasks but also to work with hypotheses, to explore different methods of getting knowledge, to feel the necessity for the generation of ideas, for mastering unique technologies of working with information.

It is important to note that the identification of gifted students in the domain of informatics should begin as early as possible, and for that purpose contests in primary school can be used, which also must be focused on the development of algorithmic thinking, mathematical ingenuity, and the desire to use computer as an assistant in searching for task solutions and implementing ideas. In this case, their development horizon is the desire to become a winner of contest for their age group.

For enthusiastic students of grades 5–6 the dominating areas of nearest development are the areas of school course and out of class study group, and the development horizon is an additional group of teaching upon students' request. This teaching is carried out by a teacher with the use of the tasks from the school and municipal stage informatics olympiad. Development horizon in this case is the achievement of a winner diploma at the school stage of the RusOI.

For students of grades 7–9 the area of the nearest development dominates, which is determined by electives courses in informatics for small groups of students and these are carried out by school teachers. These elective courses are supplemented by school

informatics courses, and the development horizon is an individual teaching with tasks from the municipal and regional stages of the informatics olympiad. This teaching is carried out by the regional coach. Development horizon in this case is the achievement of the winner diploma at the municipal and regional stages of the RusOI.

For students of grades 9–11 the horizon of development dominates, and it is complemented by the area of the nearest development: elective courses and profiled high-school courses in informatics, carried out by school teachers. Horizon development is the individual self-preparation, using tasks from the final stage of the RusOI and the IOI with the support of regional coaches, as well as regular participation in specialized team practice session. The development horizon in this case is the achievement of the winner diploma at the RusOI final stage and motivation to participate in the IOI.

As seen from the description of this environment, it guarantees successful promotion of a gifted student to the highest results. Success is ensured by the quality of school education, possibility for profiled education in informatics, cooperation between schools and universities in the individual preparation of talented students, as well as access to resources for self-preparation.

Practical usage of the development horizon model has allowed the defining of five methodical supports, based on this model, which are characteristic for training gifted students in their preparation to the IOI:

The support on individual culture consist of speed of thinking and reaction, speed of reading, speed of typing, and so forth. All these qualities could be developed from an early age, which certainly allows students to show talent with a higher effect. It is easy to check the level of development of these qualities because they indirectly affect speed of execution of any creative tasks.

The support on heuristic knowledge is the ability of a student to approach complex topics by solving tasks and to apply knowledge in practice. For estimation of achievement of this support, the brainstorm method for search of original solution of complex tasks is proposed. In this case it is important that a students find alone the idea of the solution of a problem, formulate and prove hypothesis, using their experience of discovery. The higher achievement here is the usage of all knowledge for composing competition tasks and approaches to their solution.

The support on technology consists of an excellent knowledge of the competition environment and ability to use it for implementing ideas of solution in practice, which also requires skilful use of the computer. In order to implement their discovery in practice, the students should be able to competently use a given programming system, to implement the algorithm in a program, to search for and eliminate syntax errors in the program, to tune the program and to achieve the best possible result. Here important qualities such as the ability to work with the task statement, to construct a formal description of the task, to search or choose ideas for the solution and a method for appropriate implementation first as an algorithm and then as a program. In addition, an important aspect of technological effectiveness is the ability to debug/test the solution.

The support on systematic is based on the usage in teaching of such tasks which require comprehensive application of all three above-mentioned supports. This requires

a specially selected system of competition tasks sets for a particular student. Thus, it is possible to use already available complete sets of IOI competition tasks. To assess achievement of support on systematic it is necessary to verify a student's ability to analyze the obtained task solutions, to achieve systematically smooth and high results on each set of tasks, and also to develop independently similar competition tasks with the full system of description including not only the task statements, but also formats of input and output data, descriptions of algorithms corresponding to each level of complexity for the task and set of tests.

The support on creative activity is based on the development of the students' desire to share their creative experience with others. It is possible to form such qualities on the basis of student performances after competition with analyses of tasks solutions and preparing lectures on topics which they have thought up creating their own tasks. But most important is involving the students in scientific research where their talent could be used in solving real research problems. Early entrance of the gifted students in the scientific community, participation in research or other scientific work will enable them to see themselves in science and profession, find horizons for practically using their talent in various fields of knowledge. Omission of such work with student leads to deformation of their talent, concentrating only on competitions, solving problems in the name of gaining points, which restricts the personal development of the gifted students and usage of their talents in the profession. To assess the level of student creative activity it is possible to use the traditional forms for the assessment of achievements of young scientist.

4. The Modern Forms of Preparation for the IOI

Forms of preparation for the IOI are always influenced by objective factors: information resources accessible to the school and the family; professional skills of teachers and coaches; level of development and distributions among teachers and coaches of modern educational techniques for training gifted students, and especially the new information technologies.

Along with traditional forms of preparation for the IOI there are now actively developing network forms. Development of network forms of training began with the creation of network olympiad communities all over the country. These network communities had no territorial borders and were closely connected with the student's teams that regularly participate in ICPC, from Russia and the world. Additional to the process are the regional communities of teachers and coaches, which also began to appear on the Internet. It is obvious that the inherited dissociation of training of gifted students will pass gradually away.

Openness and accessibility of the Internet immediately stepped up the work on development of methodical digital collections for support of the olympiad communities. For a short time various websites appeared that contain materials useful for training contestants at different levels. Now there are many of those websites that are used actively by the students in preparing for the IOI.

Another new form which begun to actively develop recently is the organization of on-line programming contests. Each student can participate in those contests; it does not matter where they study or reside. This gave, from one side, an impulse for development of new forms of self-preparation and, on the other side, created the necessary conditions for all contestants to demonstrate objectively their abilities and afford themselves to the olympiad community.

An important consequence of the appearance of on-line programming contests is the formation on the Internet of a distributed portfolio of each student. Such a portfolio is a personal folder of the students with their competition tasks solutions and their rating in the databases of participants in the various programming contests, including the IOI. It allows the creation of new and unique approaches for selection of gifted students and presenting them to community of coaches, irrespective of their residing place.

Now the network forms of training gifted students and their preparations for the IOI continue to develop. These also include distance teaching (schools with remote access, remote teachers, distant centres of additional education, etc.), the creation of Internet resources that contain a large number of competition task and real time evaluation systems, and other services such as video web sessions. This will eliminate the unequal conditions for tutorship, will allow students to use video resources on the Internet with lectures given by leading teachers and famous scientists and also to work with them in a remote mode. Development of such forms will continue, because expanding the number of contestants leads to shortage of qualified teachers and coaches, but the appropriate expanding of the number of specialized schools and communities of qualified teachers and coaches, obviously, is not possible.

Considering the development of forms for preparation for the IOI it should be noted that the appearance of new forms does not deny all existing forms, especially those that prove themselves in practice. Moreover, such forms could be integrated with new ones, providing an evolutionary transfer from one form to another.

5. Self-Preparation for the IOI – Individual Horizon of Development

For any forms of training gifted students in their preparation for the IOI, self-preparation remains one of the most important components of success. It does not matter how talented or gifted is the student by nature, only an intensive self-preparation and devotion will allow them to go to the top of the informatics olympiads. The most important component of teachers or coaches work here is to identify an individual learning trajectory for each gifted student and organize their self-preparation.

Development of Internet technologies and their widespread use at schools provides students with excellent possibilities for self-study work in their preparation for the IOI. This concerns the choice of school for further studying, the choice of education institution for extracurricular activities or coach, and the choice of methodical and/or teaching materials, proposed in printed or electronic form. This also applies to teachers or coaches who have the opportunity to identify talented students and organizations working with them in a remote mode by using the possibilities of the Internet.

The basis of self-preparation for the IOI and constructing individual trajectories of such preparation consists of the following methodological and didactic materials on informatics olympiads:

- IOI Syllabus (Verhoeff *et al.*, 2006) and Complex Curriculum of Olympiad in Informatics Preparation (Kiryukhin, 2007, 2009) that define the content of preparation;
- materials for theoretical preparation, presented in printing or/and electronic forms, including video lectures;
- collections of competition tasks of all levels of complexity and topics, with short methodical instructions for their solutions (Kiryukhin and Okulov, 2007; Kiryukhin, 2008, 2009, 2011);
- websites with competition tasks collections and the possibility of automatic on-line evaluation of submitted tasks solutions;
- websites providing regular on-line programming contests.

Informatics olympiad contests are the basis for the development of trajectories for individual preparation for the IOI. This concerns both theoretical preparation and building of individual strategies on tasks solving that allow the covering all topics and didactical units of contents.

Despite the availability of sufficient number of digital educational resources on the Internet, students should focus on working with books. Reading books deepens and expands knowledge acquired by students in the classroom, promotes mastery of the methods of solving olympiad tasks and the application of knowledge in complex and non-standard situations. A culture of working with a scientific book thoughtfully and recording the facts, necessary for solving a task, in a notebook is an essential part of the general culture of students' self-preparation.

Now in Russia there are many published books that can be of great benefit to students, preparing themselves for the IOI as well as to teachers and coaches working with these students. First of all we have to mention the series on informatics olympiads by the publishing house "BINOM" (BKL, 2006) and by the publishing house "Prosveschenie" (Prosveschenie, 2010).

Among students the opinion circulates that books have already passed away and all necessary information for olympiad preparation can be found on the Internet. But this is not true, because the book remains the only source where the information is systematically presented, methodically supported and the reliability of information contained therein is carefully verified at all stages of preparing book by a publishing house. Moreover, necessary and useful information is contained in the book in a compact form. A book is always available, easy to work with, and does not require long search on the Internet.

Internet resources with collections of competition tasks are very popular among students, teachers and coaches, as many programming contests in informatics summarize their results immediately after the event and publish them on the Internet. From one side it is good to have all this information available, but from the other side, it seriously complicates the process of students' preparation for informatics olympiads. It is important not to solve tasks in accidental order but to build the most effective trajectories of tasks, covering the modern contents of informatics olympiads.

It is important to note, that IOI competition tasks are specific. Most of them aim to evaluate the creative potential of students supposing a lack of professional knowledge and skills. This is achieved through the multilevel character of such tasks, but not in terms of task statement, but in terms of variety of possible algorithms for solving them. In this case, the size of the possible input and run time limit defined in the task statement determine its complexity. Evaluation of a solution takes into account not only the total but also any partial (incomplete) solutions, i.e., solutions able to solve the task, inside the time limit, on smaller than specified in the task statement size of the input. So, competition tasks with the same statements as in the IOI, but with smaller sizes of the input, could be solved by beginners. That is why an important aspect of self-preparation for the IOI is the ability of the students to analyze their own solution and to decide whether it is partial or complete.

A very important role in the process of evaluating of solutions belongs to the set of test cases because each test or group of tests tries to prove correctness and efficiency of the solution and has its own “weight” in the evaluation criteria. That is why contestants have to be ready to generate their own test cases during the contest. Generating appropriate test cases is neither easier nor less interesting than creating the solution of a competitive task. The opposite, it is quite a difficult problem, which the IOI participants have to solve during the competition also. And corresponding abilities have to be formed in preparation time.

Ten years ago the task evaluation process was a great challenge both for the students and teachers or coaches. The introduction of automated evaluation systems has drastically changed the situation for the better. Now, there are many websites on the Internet with the functionality to evaluate the solutions of students online. Developing Internet resources with collections of competition tasks and the possibility of automatic evaluation of task solution is difficult enough and a time-consuming problem. But it is important that the number of such resources is increasing. Work in this direction is very important, because increasing the enrollment of students by the olympiad movement leads to the problem of a shortage of qualified teachers and coaches working with them, and to endlessly increase the number of qualified teachers, coaches and special schools is not possible.

An important component in self-preparation of students for the IOI is the participation of students in the online programming contests that take place regularly all over the world. Participating in these competitions, students gain experience, can monitor their level of preparation and can correct the trajectory of their further preparation. Something more, they could periodically compare their results with the results of their rivals from other countries, receiving in such way an impulse to develop and search for more effective forms of self-preparation for the IOI. On the other hand, teachers or coaches that train gifted students have the possibility to observe the progress of the best students from other countries and regions. As a result, this contributes to creation of new and unique approaches for selection of gifted students and introducing them in the community, regardless of their country and residence.

6. Conclusion

There is no doubt that success at the IOI is strongly determined by the existence in each country of a system for identifying talented young people and developing their abilities, both within the schools and within the system for additional education. The common approach, based on solving as many competition tasks as possible and participating in as many competitions as possible, is most suitable for the university students, because participation in competitions and regular solving of olympiad tasks is a good supplement to lectures and practical training at the universities. For the participants of the IOI to learn informatics in accordance with the university curriculum is almost impossible, and the goal of school is quite different. Therefore, it is necessary to teach the students to solve competition tasks not just from the positions of studying the theoretical knowledge used in the IOI. We need to focus on the development the students' ingenuity, the desire to make discoveries, and generate non-standard ideas, which can only be developed and not trained. There is an important role to play here for the techniques of preparation for the IOI, based on early (starting in primary school) development of gifted students, and on the modern education and information technologies.

Techniques for the development of gifted children have appeared at the beginning of the last century but with the development of information and education technologies they have been significantly updated and took on new forms for their implementation. They are especially important for informatics olympiad movement and, in particular, in preparation for the IOI. The main goal now is to adapt traditional techniques to the modern conditions of the development of society and education. Any experience in this direction is important and deserves our attention.

Of course, very talented children are born in each country and traditional teaching methods allow them to achieve high results at the IOI. For these results to be stable, it is necessary to use in preparation for the IOI developmental teaching methods and forms, which were discussed in this article. These techniques have proved themselves well in the preparation of Russian students for the IOI, and the authors hope that they may be useful to the teachers and coaches from other countries too.

References

- Piaget, J. (1983). Piaget's theory. In: Mussen, P. (Ed). *Handbook of Child Psychology*, 4th edn., Vol. 1. New York, Wiley.
- Dewey John (1900). *The School and Society. Educators*. Moscow (in Russian, Школа и общество).
- Vygotsky, L.S. (1991). *Imagination and Creativity in Children*. Prosvetenie, Moscow (in Russian, Благодарение и творчество в детском возрасте).
- Verhoeff, T., Horvath, G., Diks, K., Cormack, G. (2006). A proposal for an IOI Syllabus. *Teaching Mathematics and Computer Science*, 4(1), 193–216.
<http://www.win.tue.nl/wstomv/publications/ioi-syllabus-proposal.pdf>.
- Kiryukhin, V.M. (2007). The modern contents of the Russian national olympiads in informatics. *Olympiads in Informatics*, 1, 90–104.
- Kiryukhin, V., Okulov, S. (2007). *Methods of Problem Solving in Informatics. International Olympiads*. LBZ (BINOM. Knowledge Lab), Moscow (in Russian, Методика решения задач по информатике. Международные олимпиады).
<http://www.lbz.ru/katalog/products/literatura-dlja-shkol/informatika/olimpijskie-vysoty/metodika-reshenija-zadach-po-informatike-699>.

- Kiryukhin, V. (2008). *Informatics. Russian Olympiads*. Issue 1, Prosveschenie, Moscow (in Russian, Информатика. Всероссийские олимпиады. Выпуск 1). http://www.prosv.ru/book.aspx?ob_no=209&d_no=13828<ype=9577&subject=1467.
- Kiryukhin, V. (2009). *Informatics. Russian Olympiads*. Issue 2, Prosveschenie, Moscow (in Russian, Информатика. Всероссийские олимпиады. Выпуск 2). http://www.prosv.ru/book.aspx?ob_no=209&d_no=24868<ype=21818&subject=20708
- Kiryukhin, V. (2009). *Informatics. International Olympiads*. Issue 1, Prosveschenie, Moscow (in Russian, Информатика. Международные олимпиады. Выпуск 1). http://www.prosv.ru/book.aspx?ob_no=209&d_no=22548<ype=21818&subject=20708.
- Kiryukhin, V. (2010). Influence of the state school education standard on results of performance of the Russian pupils at the IOI. *Olympiads in Informatics*, 4, 15–29.
- Kiryukhin, V., Tsvetkova, M. (2010). Strategy for ICT skills teachers and informatics olympiad coaches development. *Olympiads in Informatics*, 4, 30–51.
- Kiryukhin, V. (2011). *Informatics. Russian Olympiads*. Issue 3, Prosveschenie, Moscow (in Russian, Информатика. Всероссийские олимпиады. Выпуск 3).
- Kiryukhin, V. (2011). *Method of carrying out and preparation for participation in the Olympiad in Informatics. All-Russian Olympiad*. LBZ (BINOM. Knowledge Lab), Moscow (in Russian, Методика проведения и подготовки к участию в олимпиадах по информатике. Всероссийская олимпиада школьников). <http://www.lbz.ru/katalog/products/literatura-dlya-shkol/informatika/olimpijskie-vysoty/metodika-provedenija-i-podgotovki-k>.
- BKL (2006). *Publishing Houses “BINOM. Knowledge Laboratory”* (in Russian). <http://lbz.ru/>
[katalog/products/literatura-dlya-shkol/informatika/olimpijskie-vysoty](http://lbz.ru/katalog/products/literatura-dlya-shkol/informatika/olimpijskie-vysoty).
- Prosveschenie (2010). *Publishing Houses “Prosveschenie”* (in Russian). http://www.prosv.ru/about.aspx?ob_no=228&d_no=11327.



V.M. Kiryukhin is professor of the Russian Academy of Natural Sciences. He is the chairman of the federal methodical commission on informatics which is responsible in Russia for carrying out the national Olympiads in informatics. He is the author of many papers and books in Russia on development of Olympiad movements in informatics and preparations for the Olympiads in informatics. He is the exclusive representative who took part at all IOI from 1989 as a member of the IOI International Committee (1989–1992, 1999–2002) and the Russian team leader. He received the IOI Distinguished Service Award at IOI 2003, the IOI Distinguished Service Award at IOI 2008 as one of the founders of the IOI making his long term distinguished service to the IOI from 1989 to 2008 and the medal “20 Years since the First International Olympiad in Informatics” at the IOI 2009.



M.S. Tsvetkova, professor of the Russian Academy of Natural Sciences, PhD in pedagogic science, prize-winner of competition “The Teacher of Year of Moscow” (1998), main expert of state projects of school education informatisation in the Ministry of Education of the Russian Federation (2001–2005), the expert of the World bank project “Informatisation of Education System”. Since 2002 she is a member of the Central methodical commission of the Russian Olympiad in informatics, the pedagogic coach of the Russian team on the IOI. She is the author of many papers and books in Russia on the informatisation of education and methods of development of talented students.

Teaching the Concept of Online Algorithms

Dennis KOMM

*Department of Computer Science
ETH Zurich, Switzerland
e-mail: dennis.komm@inf.ethz.ch*

Abstract. The term *algorithm* is well-defined: It is the formal description of a strategy, which always has to produce a solution for a specific problem. It is probably the most basic concept of computer science and accordingly both the creation and implementation of algorithms are among the most important things students have to learn when dealing with the subject. The situation is usually as follows. Given an input x for some problem P , we are interested in designing an algorithm A that reads x , performs calculations depending on x and creates some output $A(x)$ of some predefined form. Conversely, in many practical applications, this model is actually not accurate. Here, at one specific point in time, only parts of the input are known to the algorithm whereas parts of the output are already needed. We call an algorithm handling such a situation an *online algorithm*. Among the numerous examples for such situations are parts of any operating system, or routing and scheduling problems.

To the best of our knowledge, teaching the concept of online scenarios and algorithms to secondary school students only received little attention in the past. However, our experiences show that it catches the students' attention when taught in a comprehensive and motivated way. In this paper, we want to propose a strategy of how to introduce online algorithms by explaining the major ideas to students and we further want to share our experiences.

Key words: teaching, secondary school, online algorithms, competitive analysis.

1. Introduction

We expect the reader to be familiar with the basic definitions and notations of *online algorithms* and *competitive analysis* (introduced by Sleator and Tarjan (1985)). For further reading, we point to the standard literature (Borodin and El-Yaniv, 1998; Hromkovič, 2006; Irani and Karlin, 1997; Sleator and Tarjan, 1985). The material is suited to be presented in a lesson using slides that takes approximately two hours.

Our main goal is to explain secondary school students the concept of online scenarios and to make sure they understand that these situations frequently occur in the real world, for instance, when users interact with an operating system. Our aim is to furthermore use online scenarios as a framework to introduce the students to the following ideas, techniques, and concepts.

1. Worst-case analysis, that is, “an algorithm is only as good as it performs on the hardest instances”.
2. This analysis can be performed by means of an imaginary adversary¹ who tries to harm the algorithm as much as possible.
3. Randomization is a powerful tool in algorithm design.
4. It is important get a good intuition of the problem at hand to be able to make statements about it (to prove theorems).

In the presentation, we try to omit formal definitions and proofs whenever possible. Furthermore, we want to keep the notation as simple and straightforward as we possibly can. According to the amount of knowledge the students already possess, it might be a good idea to go into more detail at some points.

Please note that all results presented here have already been known. Especially those about *job shop scheduling with unit length tasks*, which we have chosen as a means to communicate the basic concepts introduced above, are due to Hromkovič *et al.* (2009), Hromkovič (2006, 2009).

2. A Sample Lesson

We now propose a concrete way of how to teach the concepts and ideas of online algorithms. We gave a talk with this material to a group of Swiss secondary school students and received very good feedback from both their teachers and the students themselves.

2.1. Introducing Online Problems

We first show the class something familiar by describing the principle of operation of an algorithm. Even if they never thought about a formal definition, they agree that the high-level scheme may be depicted as shown in Fig. 1. In our case, this is consistent with what the students experienced in class until this point, no matter how advanced their programming skills are. An algorithm A reads an input x , modifies it, and outputs a solution $A(x)$ afterwards. As a next step, we introduce a problem we want to solve by means of such an algorithm. We do not even need to talk about computer programs at this point, but only about an unambiguous and clear strategy a human can follow.

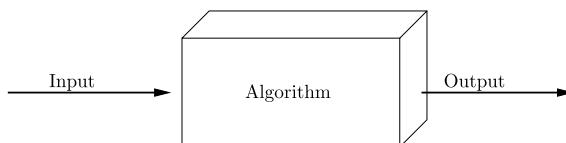


Fig. 1. The classical view of an algorithm.

¹Here, we consider so-called *oblivious adversaries*.

Suppose we sit in front of a monitoring screen in a police central and want to route a number of k police cars through a city. At any time point, an emergency call might come in. Using GPS, we know the position of every car at this point and we are searching for a policy to decide which police car to radio and send to the corresponding crime scene.

Actually, this problem is the well-known k -server problem in disguise (Borodin and El-Yaniv, 1998), introduced by Manasse *et al.* (1990). To simplify the discussion, we assume that the police men are busy for a while after arriving at the crime scene and that we, therefore, are not allowed to move the same car twice. It is obvious to the students that this is a problem of practical relevance. Next, we do not talk about any problems that might appear in this scenario, but just propose a sample instance as shown in Fig. 2. There are three police cars stationed at the depicted positions. At 12 o'clock, an emergency call comes in. The crime spot is located between the positions of car 1 and car 2. It does not really seem to matter at this point which of the two cars we send. We therefore decide to take car 2. However, ten minutes later, car 2 arrives at the crime scene, but just one minute after that, a second crime is reported and it is located at the old position of car 2.

We can now clearly state that it would have been a better strategy to send car 1 instead of car 2 and we address the key point of online computation by directly asking the students why we made a bad decision. The answer is both trivial and important: "Because we did not know this before."

By introducing the problem of online scenarios in the above way, it is immediately clear that the restriction of not knowing the whole input in advance is actually very natural. We then point out that algorithm engineers have to deal with this problem in many applications. Some examples are

- operating systems (e.g., the well-studied *paging* problem; Borodin and El-Yaniv, 1998),
- routing problems (as shown in the example), and
- scheduling problems (which will be discussed in the following).

Before we introduce the problem we deal with in the main part of the lesson, we need to supply the basic tools and models we need in what follows. Therefore, the first question we ask is

How do we measure how good an online algorithm performs?

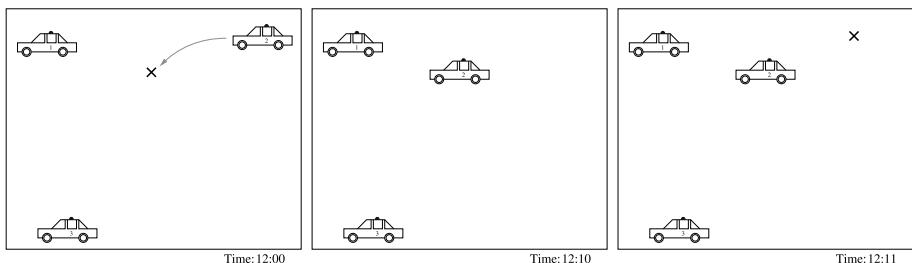


Fig. 2. An instance of the above problem to demonstrate the issues that come up when only knowing parts of the input.

By $\text{cost}(A(x))$, we denote the costs of algorithm A on an input x . For the above problem, these costs are, e. g., the overall distance the cars have to drive (or, formulated differently, the overall time they need to arrive at a crime scene). Suppose that, at the end of the day, we browse through the log files of all emergency calls that appeared today. We now have the knowledge to compute a solution that is optimal with respect to the above measure. The costs of this solution are denoted by $\text{Opt}(x)$. Compared to this solution, many of our decisions might have been bad, but we simply did not have another chance. We immediately had to send a police car when an emergency call came in. To evaluate our online strategy A on the input x , we want to look at the ratio

$$\frac{\text{cost}(A(x))}{\text{Opt}(x)},$$

and call this the competitive ratio of A on x . Furthermore, if we can guarantee that, for any possible instance x , we have

$$\frac{\text{cost}(A(x))}{\text{Opt}(x)} \leq d,$$

for some fixed d , we call A *d-competitive*. If the students already know the concept of approximation (offline) algorithms, they will surely recognize the analogy, but this is not important for understanding the concept in general. We now try to establish some intuition for *competitive analysis* in an informal way. The following question may be posed as an exercise to the students.

As we have already seen, the above problem seems to be somewhat hard for online algorithms, but how bad can it really be?

A possible idea to give an answer to this question is shown in Fig. 3. Suppose the police cars are positioned in a circle. We label them clockwise from 1 to k starting at the car on top. The first request (emergency call) is exactly between the two cars 1 and 2. Suppose we again choose to send car 2. After that, the next request appears and it is located just at car 2's starting point. The nearest car is car 3, but right after we told the corresponding driver to move there, another crime is committed at the starting point of this car. The requests continue in this fashion until the k th request is made at the starting point of car k (Fig. 3a). On the other hand, if we would have known the sequence of requests in advance, we would have moved car 1 instead of 2 at the beginning. Afterwards, no other movement would have been necessary (Fig. 3b).

Summing up, until this point, we have shown that the classical definition of algorithms as shown in Fig. 1 is not accurate for online scenarios due to our inability to predict the future. However, it is crucial to note that the basic properties of algorithms (which separate them from arbitrary programs) are *not* violated: Online algorithms halt on any finite input (which might be a prefix of a potentially infinite instance) and they create an output of some well-defined form.

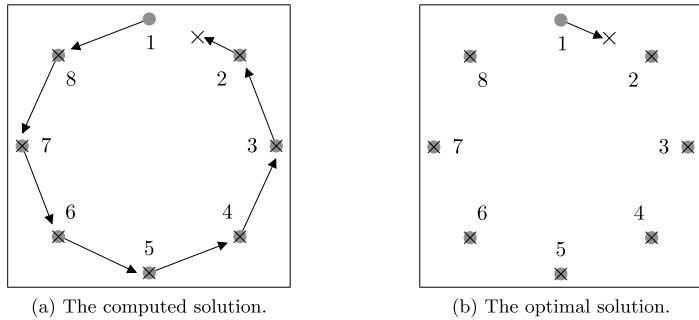


Fig. 3. A hard instance for the above problem. For the ease of presentation, police cars are marked by grey circles, requests are marked by black daggers.

2.2. Worst-Case Analysis by Means of an Adversary

As a next step, we want to elaborate the above idea of constructing worst-case input instances to talk about an online algorithm's performance. To do this, we ask the following question.

How do we know that an online algorithm A does not have a specific performance, i. e., is not able to achieve a certain competitive ratio?

Observe that, in the above informal definition of competitiveness, we clearly stated that an online algorithm has to have a competitive ratio of at most d on *all* possible instances to be called d -competitive. We had good experiences with explaining the students the following analogy.

Suppose I claim to have built a robot that beats every human player in tennis. Of course, you do not believe me, but I insist on it. I am convinced that my robot is better than anyone who challenges him. To disprove the claim, you might ask every single inhabitant of earth to play a match against the robot and you are positive that, eventually, someone will be found who wins. However, with almost seven billion people living on earth, it might take quite a while until someone is found who does not lose. What could you do to speed things up?

It does not take very long until the first student proposes: “We just let your robot play against Roger Federer.” This is just the answer we expected². We want to do the exact same thing with our algorithm A . We think of the best adversary there might possibly be. If he always has a strategy that makes A perform bad with respect to some measurement (that is, if he is able to construct a hard input instance where A can provably never be d -competitive), it directly follows that A is never better in general (is clearly not d -competitive), because we simply cannot guarantee that this input never appears in practice no matter how unlikely it is.

Accordingly, we introduce the adversary ADV that has the following properties:

- ADV knows A and

²Surely, there are equivalent or even better examples, but this one suits Switzerland.

- thus is able to predict any of its steps;
- ADV tries to make A perform as bad as possible.

Observe that we have actually requested the students to act as ADV for the police car problem when they were asked to construct an instance as shown in Fig. 3.

2.3. Job Shop Scheduling with 2 Unit Length Tasks

We are now ready to describe the problem we want to investigate in the following. The ideas how to present the material are taken from the book “Algorithmic Adventures” by Hromkovič (2009). Here, we want to use the things we have taught until this point to show how randomization helps to deal with the shortcomings of online computation. To this end, it is sufficient to informally define the problem as follows.

Suppose we are dealing with a factory that consists of m distinct machines, each of which is designed to perform a single distinct assignment, such that all machines are different from each other. Moreover, there are two so-called jobs Job_1 and Job_2 that each consist of m different tasks. Each of these tasks needs exactly one machine and the $(i+1)$ th task may not start to be processed before task i is finished. As long as Job_1 and Job_2 request different machines at the same time, the work can be parallelized. However, if they request the same machine, one of the two has to be delayed. Our aim is to construct an online algorithm that minimizes these delays.

If we label the machines from 1 to m , an example input is

$$Job_1 = (1, 2, 3, 4, 5, 6, 7, 8),$$

$$Job_2 = (3, 1, 2, 4, 6, 5, 8, 7),$$

which means that Job_2 first requests to perform a task on machine 3, after it is finished, it wants to perform a task on machine 1, and so on. To simplify the problem, we suppose that each task exactly needs one time unit (thus talking about unit length tasks). It is important to state that each job needs each machine *exactly* once, and therefore, the above sequences are permutations of the numbers from 1 to m .

A feasible solution S to this instance is

$$\text{Schedule } (Job_1) = (1, 2, 3, 4, 5, 6, /, 7, 8, /, /),$$

$$\text{Schedule } (Job_2) = (3, 1, 2, /, 4, /, 6, 5, /, 8, 7),$$

which means that the first three tasks of both jobs are served greedily (in parallel), but after that, both jobs request machine 4. Job_2 is delayed while Job_1 is served. In the next time step, Job_1 performs its fifth task on machine 5 and Job_2 performs its fourth task on machine 4 which is now free. This schedule induces a delay of 3.

In an online scenario, as we consider it, for each job, the $(j+1)$ th request is only known after the j th task is finished, i. e., while we have not processed task j , we have no idea which machine task $j+1$ wants to use.

2.4. A Graphical Representation

This section covers an important lesson. Before we are able to describe results about the problem in a coherent way, we want to introduce a more comprehensible presentation of input instances to get a grip of the problem as shown in Fig. 4.

This description was introduced by Brucker (1988) and it is frequently used in the literature (Hromkovič *et al.*, 2009; Hromkovič, 2006; Hromkovič, 2009). Consider a 2-dimensional grid of size $m \times m$. We label the x -axis with the numbers from 1 to m in the order given by Job_1 . We do the same thing with the y -axis and Job_2 . Every cell that has the same number on both axis is marked by a grey square and we refer to these cells as *obstacles*. A feasible solution is a path through this grid from the upper left corner to the bottom right corner. Whenever two tasks can be performed in parallel, a diagonal step may be made (of course, it is also allowed to make a horizontal or a vertical step). However, if the path arrives at the upper left corner of a grey square, one job has to be delayed, and we say the path *hits an obstacle*. In this case, only a horizontal step (Job_2 is delayed) or a vertical step (Job_1 is delayed) is possible (we have to bypass the obstacle). Figure 4b shows the path that represents the solution S . Let us denote the number of horizontal [vertical, diagonal] steps of any strategy by $h[v, d]$. We can make the following observations.

- O1 There always is *exactly* one obstacle in every row and every column.
- O2 For any feasible solution, we have $v = h$.
- O3 Also, for such a path, $d + v = d + h = m$.
- O4 Obviously, at least m steps have to be made in total.
- O5 Finally, the costs of every feasible solution are $d + h + v = m + h = m + v$.

It is immediate that we aim at designing an online algorithm A that calculates a path of minimal length, i. e., makes as few non-diagonal steps as possible.

2.5. An Adversary that Creates Hard Input Instances

Using the above graphical representation, we now want to show how an adversary can create a bad input instance for any online algorithm.

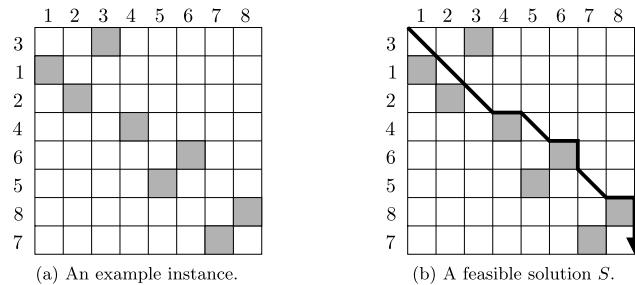


Fig. 4. A graphical representation.

There exists an adversary ADV that, by putting obstacles onto the grid in a clever way, can make sure that every second step of a path any online algorithm calculates, is not a diagonal one.

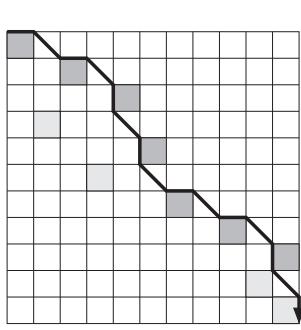
The idea of the proof is as simple as it gets. We do not prove it formally, but only show the students ADV's strategy on an example which is straightforward to generalize. It is important to emphasize that, for a formal proof, we are not allowed to fix the algorithm in any way, but we would have to consider *all* algorithms. The following strategy might be presented on the blackboard (Fig. 5a). At first, ADV puts an obstacle to the upper left cell by making both jobs request machine 1 at the start.

Note that the only restriction ADV has to obey in the following is given by Observation O1, that is, ADV is allowed to merely place *one* obstacle in every row and every column of the grid (otherwise, one job would request the same machine twice, which is forbidden by the problem definition). ADV's strategy is as follows: Whenever *A* performs a diagonal step, the corresponding path enters a cell for which no task of both *Job*₁ and *Job*₂ has been assigned yet. ADV may then just place an obstacle by saying that both jobs now ask for the same machine with the smallest index that has not been used until this point. The next step of *A* must be either horizontal or vertical. By Observation O1, *A* may then perform a diagonal step thus enabling ADV to place another obstacle by the above strategy. If the path arrives at the right or bottom border of the grid, ADV may place the remaining obstacles in an arbitrary fashion.

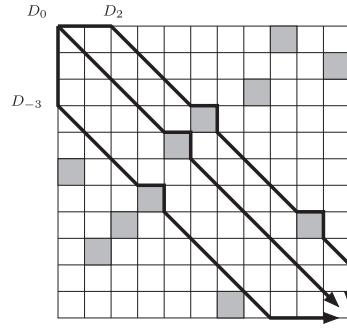
Since there are at least m steps in total (Observation O4) and every second of them is not diagonal, it follows that there at least $m/2$ non-diagonal steps in the sum. Consequently, using Observation O2, the solution makes at least $m/4$ horizontal and $m/4$ vertical steps. The costs of this solution are therefore, by Observation O5, at least

$$\text{cost } (A(x)) \geq m + \frac{m}{4}.$$

Hromkovič *et al.* even proved a stronger claim (Hromkovič *et al.*, 2009). However, for our needs the above bound is fine and we omit the proof for the sake of not getting too formal.



(a) An instance created by ADV.



(b) The strategies D_0 , D_2 , and D_{-3} .

Fig. 5. An example of how ADV can make sure that every second step of any path computed by *A* is not diagonal, and three strategies from \mathcal{D} .

So far, we have shown a lower bound on the costs of any online algorithm. But this is not sufficient to argue that the competitive ratio of any such algorithm is bad. It remains to show that there also always exists an optimal (offline) solution that has lower costs. To this end, we introduce the following set of strategies. For every $i \in \{1, \dots, \sqrt{m}\}$, the strategy D_i makes i horizontal steps at the beginning. Afterwards, it makes a diagonal step whenever possible. If it hits an obstacle, it bypasses it by making a horizontal step immediately followed by a vertical step. Eventually, D_i hits the right border. It then makes i vertical steps to reach the lower right corner. Analogously, for $i \in \{-\sqrt{m}, \dots, -1\}$, the strategy D_i makes i vertical steps at the beginning and i horizontal steps at the end, acting as above in between. Moreover, the strategy D_0 does not make any non-diagonal steps at the beginning or at the end unless it hits an obstacle. We now set

$$\mathcal{D} = \{D_i \mid i \in \{-\sqrt{m}, \dots, 0, \dots, \sqrt{m}\}\}.$$

Obviously, we are dealing with $2\sqrt{m} + 1$ strategies in total, sample strategies are shown in Fig. 5b. In the following, for the ease of presentation, we assume that \sqrt{m} is a natural number³. We are now interested in answering the following question.

What can we claim about the smallest costs from the set of these solutions?

Recall that we call the number of horizontal steps (h) the delay of a solution. Our idea is to first calculate the average delay. Let Obs_i denote the number of obstacles strategy D_i hits on its way through the grid. We conclude that

$$\text{Delay of strategy } D_i = h = i + \text{Obs}_i,$$

because D_i makes exactly one horizontal move for every obstacle it hits. We sum the costs for all strategies and get

$$\text{Total delay} = \sum_{i=-\sqrt{m}}^{\sqrt{m}} (i + \text{Obs}_i) = \sum_{i=-\sqrt{m}}^{\sqrt{m}} i + \sum_{i=-\sqrt{m}}^{\sqrt{m}} \text{Obs}_i.$$

Observe that we know that there are, in total, exactly m obstacles in the grid for any instance (Observation O1). It follows that the sum of all obstacles that are hit cannot be larger than m .

$$\begin{aligned} \text{Total delay} &\leq \sum_{i=-\sqrt{m}}^{\sqrt{m}} i + m = 2 \cdot \sum_{i=1}^{\sqrt{m}} i + m \\ &= 2 \frac{\sqrt{m}(\sqrt{m} + 1)}{2} + m = \sqrt{m}(\sqrt{m} + 1) + m \\ &= \sqrt{m}(\sqrt{m} + 1) + \sqrt{m} \cdot \sqrt{m} = \sqrt{m}(2\sqrt{m} + 1). \end{aligned}$$

³The following proof works without this restriction, but we think that it is easier to follow this way.

If we now divide this number by the number of all strategies, we get

$$\text{Average delay} \leq \frac{\sqrt{m}(2\sqrt{m} + 1)}{2\sqrt{m} + 1} = \sqrt{m}.$$

By Observation O5, it follows that the average costs are at most $m + \sqrt{m}$. Therefore, there has to exist a single strategy that has costs of at most $m + \sqrt{m}$.

Let us sum up what we have just learned. We showed the existence of an adversary that can make sure that any solution computed by any online algorithm has costs of at least $m + m/4$. On the other hand, we also know that there always exists a solution that has costs of at most $m + \sqrt{m}$. What does this mean for the competitive ratio of any online algorithm A for the problem job shop scheduling with 2 unit length tasks?

To answer this question, we simply plug these two values into the formula for competitiveness. There exists an instance \bar{x} such that, for any online algorithm A , we have

$$\frac{\text{cost}(A(\bar{x}))}{\text{Opt}(\bar{x})} \geq \frac{m + m/4}{m + \sqrt{m}} = \frac{5}{4} \left(\frac{m}{m + \sqrt{m}} \right) = \frac{5}{4} \left(\frac{1}{1 + \frac{1}{\sqrt{m}}} \right).$$

As computer scientists, we are interested in how this function behaves with respect to an increasing input m . This means that we ask

What does this expression converge to as m grows?

Obviously, it converges to $5/4$ and we may therefore state that there exist inputs on which the solution computed by A is almost 25% worse than the optimal solution.

2.6. Using Randomization Increases Performance

We now search for a way that overcomes the above drawback and introduce the concept of randomized algorithms which are allowed to base some of their computations on random decisions. To be precise and formally correct, in the following, we would have to talk about the expected competitive ratio of the investigated randomized online algorithm. As a matter of fact, it is known that it tends to 1 with m tending to infinity (Hromkovič *et al.*, 2009), but we do not want to introduce the formal definitions of random variables and expected values which are needed for the proof. Therefore, we use a different approach as suggested by Hromkovič (2009).

Let us first get back to the model of ADV .

- ADV knows A ,
- but he does not know A 's random decisions in advance
- therefore, ADV has to cope with a lot of “different” algorithms at once.

Indeed, if ADV knows the source code of A , it does not really help him to know that there exists a code block which shows some instructions that merely say “Choose one out of d possible strategies, each with the same probability”. Which one will actually be chosen will only be known at runtime. Therefore, this is consistent with our intuition about the adversary.

If an algorithm A is not allowed to use randomness and we show the existence of an adversary that is able to construct a bad input instance as we have shown in the previous section, A will *always* perform bad on this input. But actually, the property that makes this input “hard to deal with” might be very natural and thus appear frequently in practical applications. For randomized algorithms, we want to make statements of the form “the algorithm only performs bad on x with a very low probability”, but in most of the cases its costs are low. Only very rarely, its costs are high for some fixed x and therefore, x is not “bad” in the above sense.

We now consider the randomized algorithm R that has the following strategy.

R chooses one out of the strategies from \mathcal{D} and follows it. Here, any possible strategy gets chosen with the same probability.

Since there are exactly $2\sqrt{m} + 1$ strategies in total to choose from, we have

$$\text{Probability to choose strategy } D_i = \frac{1}{2\sqrt{m} + 1},$$

for every $i \in \{-\sqrt{m}, \dots, 0, \dots, \sqrt{m}\}$. For our next step, we need the following observation.

Let d be the average value of n natural numbers. Then at least half of these numbers have a value less than $2d$.

Informally, we can just draw the number line from 0 to $2d$ on the blackboard. Even if $n/2 - 1$ numbers out of the n numbers have a value of 0 (are as small as possible), not all remaining numbers can have a value of $2d$ or d is not the average (Fig. 6). Alternatively: to formally prove the claim, the students must understand the concept of an *indirect proof*. In this case, towards contradiction, suppose the claim is not true and at least $n/2 + 1$ numbers have a value of at least $2d$. It follows that

$$\begin{aligned} d &\geqslant \frac{\left(\frac{n}{2} + 1\right) \cdot 2d + \left(\frac{n}{2} - 1\right) \cdot 0}{n} = \frac{\frac{2dn}{2} + 2d + 0}{n} \\ &= \frac{dn}{n} + \frac{2d}{n} = d + \frac{2d}{n} > d, \end{aligned}$$

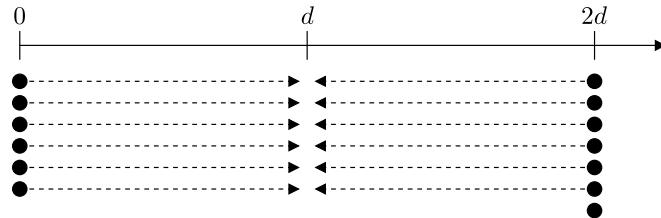


Fig. 6. An example to prove the claim. We can pair every element of value 0 with an element of value $2d$. Obviously, their average is d , but at the end, there will be at least one element left with a value of $2d$. Therefore, the average of all numbers cannot be d . Clearly, if we increase any of the values, the claim gets only more obvious.

which is a contradiction in itself. It is important to see that this statement can be generalized in a straightforward way.

Again, let d be the average value of n natural numbers. Then at least a $(1 - \frac{1}{c})$ th of these numbers have a value of less than $c \cdot d$ for any natural number c .

The corresponding proof follows from the fact that

$$\begin{aligned} d &\geq \frac{\left(\frac{n}{c} + 1\right) \cdot cd + \left(1 - \frac{n}{c} - 1\right) \cdot 0}{n} = \frac{\frac{cdn}{c} + cd + 0}{n} \\ &= \frac{dn}{n} + \frac{cd}{n} = d + \frac{cd}{n} > d. \end{aligned}$$

In the previous section, we have already seen that the average delay of the strategies in \mathcal{D} is at most \sqrt{m} . Applying the observation above, we get that, e. g.,

at most half of the solutions have a delay greater than $2\sqrt{m}$ and at most a tenth has a delay greater than $10\sqrt{m}$.

Furthermore, we know that any solution has costs of at least m (Observation O4). Therefore, at least a 9/10th of the strategies have a competitive ratio of at most

$$\frac{m + 10\sqrt{m}}{m} = 1 + \frac{10}{\sqrt{m}},$$

which tends to 1 with an increasing m . This means that the competitive ratio of R tends to 1 with probability at least 9/10.

The competitive ratio of R is compared to the one of A in Fig. 7. Actually, we can

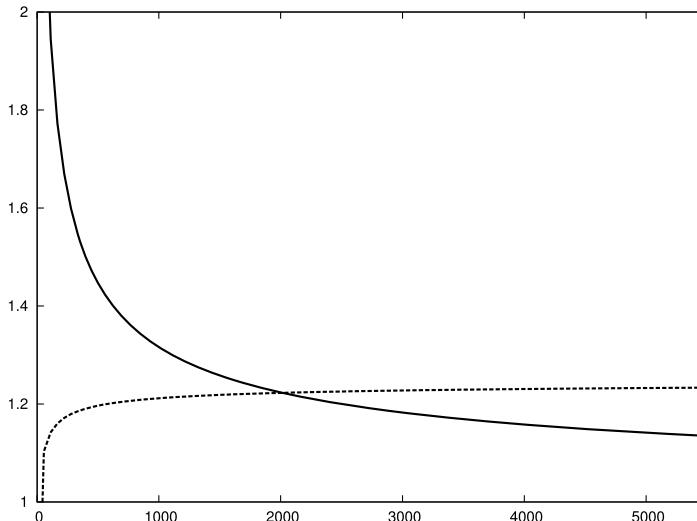


Fig. 7. Comparison of both competitive ratios, where the solid line shows the competitive ratio of R and the dashed line shows the ratio of A .

make the same statement for any probability that is constant with respect to m . Thus, we can also claim that the competitive ratio of R tends to 1 with probability 999/1000. The only difference is that in this case the constant of the upper bound is 1000 instead of 10.

3. Conclusion

In this paper, we proposed a concrete way how to teach secondary school students the basic concepts of online computation, worst-case analysis via an imaginary adversary, and the power of randomization. To do so in a comprehensible way, we omitted formal definitions and proofs whenever possible. We started with an easy example and slowly increased the technical difficulties by trying to involve the audience as much as possible.

At the end, we showed a rather deep result which is new to the majority of secondary school students, although it seems straightforward to us as computer scientists: Allowing an algorithm to make random decisions helps it to deal with an unknown future.

References

- Borodin, A., El-Yaniv, R. (1998). *Online Computation and Competitive Analysis*. Cambridge University Press, New York.
- Brucker, P. (1988). An efficient algorithm for the job-shop problem with two jobs. *Computing*, 40(4), 353–359.
- Hromkovič, J. (2006). *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms*. Springer-Verlag.
- Hromkovič, J. (2009). *Algorithmic Adventures*. Springer-Verlag.
- Hromkovič, J., Mömke, T., Steinhöfel, K., Widmayer, P. (2007). Job shop scheduling with unit length tasks: bounds and algorithms. *Algorithmic Operations Research*, 2(1), 1–14.
- Irani, S., Karlin, A.R. (1997). On Online Computation. In: *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, 521–564.
- Manasse, M.S., McGeoch, L.A., Sleator, D.D. (1990). Competitive algorithms for server problems. *Journal of Algorithms*, 11(2), 208–230.
- Sleator, D.D., Tarjan, R.E. (1985). Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2), 202–208.



D. Komm (1982), PhD student at the chair of Information Technology and Education, Department of Computer Science at ETH Zurich, studied computer science at RWTH Aachen University. His research interests focus on algorithmics and the advice complexity of online problems.

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

Tomasz KULCZYŃSKI, Jakub ŁACKI, 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), Dagienné (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 n th 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 $[a, b]$, $b \leq 10^{18}$.

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.

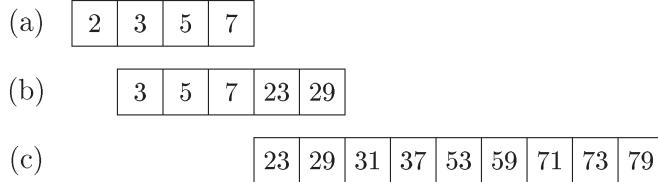


Fig. 1. The queue with right-truncatable primes: (a) the initial queue; (b) the queue after 2 is processed; (c) the queue after all single-digit right-truncatable primes are processed.

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 , $c \leq 2\,000\,000\,000$.

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

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

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 $n \geq 3$ 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 n th element of this sequence, for a given $n \leq 1\,000\,000$.

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 $n = 3, 4, \dots$ 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 $n \leq N$. We will show how to utilize the observations to improve the effectiveness of computing a_n from a_{n-1} . This

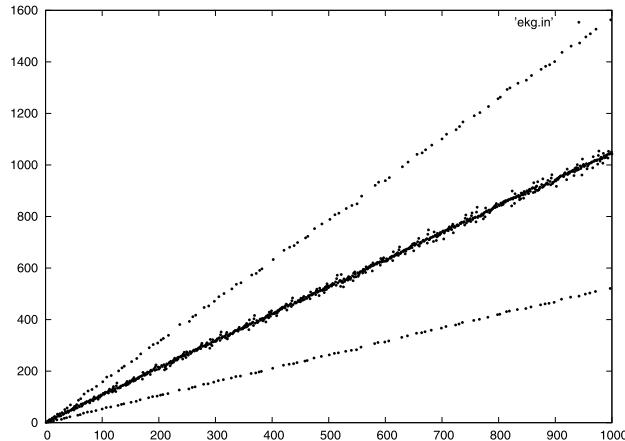


Fig. 2. A graph of the first 1 000 elements of the EKG sequence.

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 $1 \dots 2N$ 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 $N \leq 1\,000\,000$.

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 $-2 \leq x_i, y_i \leq 2$), we are to find the number of nuts that will lie on the ground after t units of time ($n \leq 7, t \leq 10^9$).

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 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.

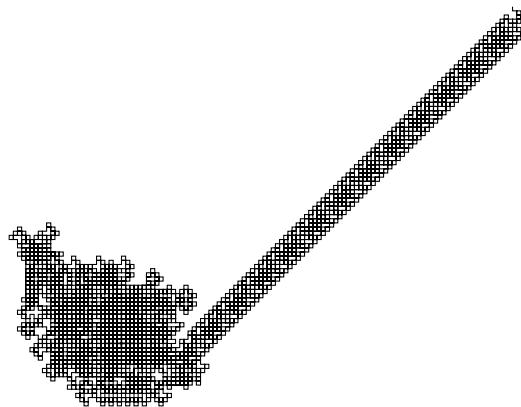


Fig. 3. The first 14 000 moves with initially empty plane.

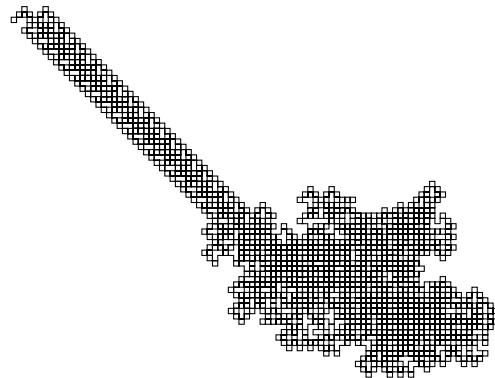


Fig. 4. The first 11 000 moves with some other initial configuration.

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

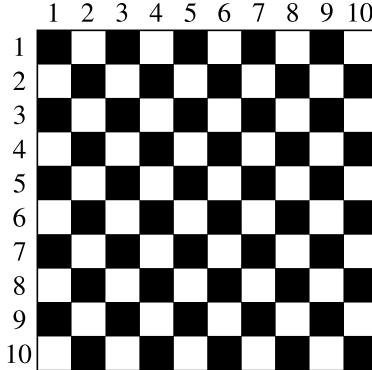


Fig. 5. Number game: a matrix describing winning positions. A cell $a[i, j]$ is white iff a position (i, j) is winning.

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 $x \times y$ ($1 \leq x, y \leq 2 \cdot 10^9$) 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 $n, n+1, \dots, 2n-1$ 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 $n, n+1, \dots, 2n-1$ are distinct integers in the range $[0, n-1]$. Given a pile with $2n$ stones, the reachable positions are exactly $n, n+1, \dots, 2n-1$. 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 $n+1, n+2, \dots, 2n$ 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 pre-computation 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.

Reconstruction of Trees Using Metric Properties *

Krassimir MANEV¹, Nikolai NIKOLOV², Minko MARKOV¹

¹*Department of Mathematics and Informatics, Sofia University*

J. Bourchier blvd. 5, 1164 Sofia, Bulgaria

²*Institute of Mathematics and Informatics, Bulgarian Academy of Sciences*

G. Bontchev str. 8, 1113 Sofia, Bulgaria

e-mail: manev@fmi.uni-sofia.bg, nik@math.bas.bg, minkom@fmi.uni-sofia.bg

Abstract. To reconstruct a graph from some of its elements or characteristics is a hard problem. Reconstructions require very good mathematical background and programming skills. That is why some not so difficult graph reconstruction problems may be appropriate for competitions in programming both for university and school students. In this paper two such problems are considered – reconstruction of a tree from the distances between its outer vertices and from the distances between all its vertices. It is proved that any tree can be reconstructed in either case. The corresponding algorithms are presented and their time complexities are estimated.

Key words: graphs, trees, shortest paths, reconstruction of tree.

1. Introduction

In the preface of a book dedicated to the 60th anniversary of Tutte the great Paul Erdős writes: “There are three diseases in graph theorists. The first is four-color-disease, the second is reconstruction-disease, and the third – Hamiltonian-disease.” Regarding *reconstruction* Paul Erdős without any doubt had in mind a hypothesis conjectured by Kelly (1942) and recited by Ulam (1960). The hypothesis became popular as the *graph reconstruction conjecture* from the paper of Harary (1964), where it was reformulated in a weaker form: every graph $G = (V, E)$, $V = \{v_1, v_2, \dots, v_n\}$, $n \geq 3$, can be reconstructed from the set of its vertex-deleted subgraphs $\{G_1, G_2, \dots, G_n\}$, where G_i is obtained from G by deleting v_i and its incident edges.

That conjecture is still neither proved nor disproved for arbitrary graphs and is considered one of the hardest graph problems. An easier problem of graph reconstruction, to reconstruct a graph from the set of subgraphs, each one obtained by a single edge removal from the original graph, was formulated and solved in Harary and Palmer (1966).

Graph reconstruction problems can be formulated in different ways and using different elements or characteristics of the graph. These problems are usually not trivial and good mathematical and algorithmic background is necessary for solving them. It is possible, however, to formulate reconstruction problems for restricted classes of graphs that are not too hard and thus to propose interesting tasks for competitions in programming.

*This work was partly supported by Scientific Research Fund of Sofia University through Contract 242/2010.

In this paper we consider two problems of reconstruction of trees from given metric characteristics, *viz.* the distances between some pairs of vertices. We prove such reconstructions are possible and the reasoning is not very involved. That is why we included these problems in two programming contests, one for university students and one for high school students, and they were successfully solved by some students.

In Section 2 we give some basic definitions and prove some properties that are used further on. In Section 3 we formulate the first problem and propose an algorithm that solves it. In Section 4 we formulate the second problem and present a version of the algorithm from Section 3 that solves it. In Section 5 we estimate the time complexity of both algorithms. We outline and discuss an important subproblem that can arise in that estimation. Section 6 contains some conclusions and ideas for future research.

2. Basic Notions and Properties

Some necessary notions are defined below. The other definitions we use can be found in each textbook on graph algorithms. We consider *finite undirected graphs* without loops or multiple edges. The graph G with vertex set V and edge set E is denoted by $G = (V, E)$. By $V(G)$ we denote the vertex set of G in case that vertex set is not specified. Without loss of generality we assume $V = \{1, 2, \dots, n\}$. For each edge e with endpoints, say, u and v we denote e by either (u, v) or (v, u) . For any vertex v , the number of edges incident with v is denoted by $\deg(v)$ and called the *degree of v* . To each edge (u, v) , a positive real number $c(u, v)$ is assigned called the *length of (u, v)* . If the edge lengths are not specified explicitly each one of them is assumed to be 1 and such graphs are called *non-weighted*.

The alternating sequence $\pi = v_0, e_0, v_1, e_1, \dots, v_{k-1}, e_{k-1}, v_k$ of distinct vertices and edges where $v_i \in V$ for $i = 0, 1, \dots, k$ and $e_i = (v_i, v_{i+1}) \in E$ for $i = 0, 1, \dots, k - 1$, is called a *path between v_0 and v_k* . The *length of π* is equal to $\sum_{i=0}^{k-1} c(e_i)$ and is denoted by $c(\pi)$. Every $v \in V$ is a *trivial path* between v and v of length 0. The edge names are omitted when we describe a path. The path $\pi_0 = v_0, v_1, \dots, v_k$ is called a *shortest path* between v_0 and v_k if $c(\pi_0) \leq c(\pi)$ for each other path π from v_0 to v_k . A graph G is *connected* if there is a path between every two of its vertices.

The *distance between u and v* is the length of any shortest path between them and is denoted by $d(u, v)$. We emphasize that the function $d(u, v)$ defined in any connected graph has the three properties of any other function in mathematics called distance:

1. $d(u, v) \geq 0 \forall u, v \in V$, and $d(u, v) = 0$ iff $u = v$;
2. $d(u, v) = d(v, u) \forall u, v \in V$;
3. $d(u, v) + d(v, w) \geq d(u, w) \forall u, v, w \in V$ (triangle inequality).

Therefore, when solving distance problems on graphs it is helpful to use analogies with the more familiar Euclidean space. Of course, these analogies have to be used carefully. For example, in Euclidean space the triangle inequality will turn to equality iff the point v lies on the linear segment between u and w , while in the metric space of the graph the triangle inequality will turn to equality iff vertex v lies on any shortest path between u and w .

The commonplace definition of a *tree* is a connected graph with no cycles, and of a *rooted tree*, a tree in which one vertex is chosen to be the *root* and the *leaves* are defined via the maximum length paths having the root as one endpoint. We find the following inductive definition of *rooted tree* more useful:

- (i) The graph $T = (\{r\}, \emptyset)$ is a rooted tree, r is both the root and the sole leaf of T .
- (ii) Let $T = (V, E)$ be a tree with root r and leaves $L = \{v_1, v_2, \dots, v_k\}$, let $u \in V$, and $w \notin V$. Then $T' = (V \cup \{w\}, E \cup \{(u, w)\})$ is also a rooted tree. Its root is r and its leaves are $(L - \{u\}) \cup \{w\}$.

Obviously, every rooted tree is a tree and every tree can be turned into a rooted one. Many properties are valid both for trees and rooted trees. For example, for each (rooted) tree $T = (V, E)$ the equality $|V| = |E| + 1$ holds, there is a unique path between each two vertices u and v of a (rooted) tree, etc.

Suppose T is a non rooted tree. For every $u \in V(T)$, the subtrees of T induced by u are the connected components obtained after the deletion of u and its incident edges. The vertices of degree one are the *outer vertices* and the other vertices are the *inner vertices*. A tree with one vertex only is called *trivial*. If all inner vertices have degree ≥ 3 , T is called *homeomorphically irreducible*, shortly *irreducible*. We find the following equivalent (except for the trivial tree) inductive definition of irreducible trees useful:

- (i) $T = (\{u, v\}, \{(u, v)\})$ is an irreducible tree with outer vertices u and v .
- (ii) Let $T = (V, E)$ be an irreducible tree, $|V| \geq 2$, and the outer vertices be $W = \{v_1, v_2, \dots, v_t\}$. Let $w \in W$, and let x_1, \dots, x_k for some $k \geq 2$ be vertices not in V . Then $T' = (V \cup \{x_1, \dots, x_k\}, E \cup \{(w, x_1), \dots, (w, x_k)\})$ is also an irreducible tree. Its outer vertices are $(W - \{w\}) \cup \{x_1, \dots, x_k\}$.

3. Reconstruction of a Tree from the Distances Between Its Outer Vertices

Consider the following reconstruction problem.

Problem 1. Let $T = (V, E)$ be non-weighted tree with outer vertices $L = \{1, 2, \dots, k\}$. Let the function $d': L \times L \rightarrow N$ be the restriction of the distance function $d: V \times V \rightarrow N$ of T on $L \times L$, i.e., $d'(i, j) = d(i, j)$, $1 \leq i \leq k$, $1 \leq j \leq k$. Given L and d' , reconstruct T . In the discussions below we will use only $d(i, j)$, having in mind that $d(i, j) = d(i, j)^c$ when i and j are outer vertices. Otherwise $d(i, j)$ is calculated by our algorithm.

The main question regarding every graph reconstruction is, is it possible to reconstruct the graph from the given elements or characteristics. That is why we proceed immediately with the following theorem.

Theorem 1. *Each non-weighted tree $T = (V, E)$ is uniquely determined by the distances between all couples of its outer vertices.*

Lemma 1. *Let $T = (V, E)$ be non-weighted tree and u, v , and w be any three pairwise distinct vertices from V . Let p, q_1 , and q_2 be the (unique) paths between u and v , between*

u and w , and between v and w , respectively. Let q' be the maximum common subpath of q_1 and q_2 that has w as one endpoint. Let x be the other endpoint of q' , if $c(q') \geq 1$, or $x = w$, else. Call x , the connecting point of w and p . Then x is uniquely defined by $d(u, v)$, $d(u, w)$, and $d(v, w)$.

Proof of Lemma 1. It is obvious that x is a vertex in p , therefore:

$$d(u, x) + d(x, v) = d(u, v).$$

Furthermore,

$$d(u, x) + d(x, w) = d(u, w), \quad \text{and}$$

$$d(v, x) + d(x, w) = d(v, w).$$

The system of these three equations in three unknowns has a unique solution:

$$d(u, x) = (d(u, v) + d(u, w) - d(v, w))/2,$$

$$d(v, x) = d(x, v) = (d(u, v) + d(v, w) - d(u, w))/2,$$

$$d(w, x) = d(x, w) = (d(u, w) + d(v, w) - d(u, v))/2.$$

Because of the triangle inequalities for u , v and w , these three values are always nonnegative integers. So, x is a uniquely defined vertex in T . See Fig. 1 for illustration.

Proof of the Theorem. Let L be the set of the outer vertices of T . Choose arbitrarily two distinct $u, v \in L$. Let $d = d(u, v)$ and p be the path between u and v (see Fig. 1). Call p , the *backbone* of T . Since we know the number $d - 1$ of the inner vertices of p it is trivial to reconstruct p . As suggested by Lemma 1, for each outer vertex $w \notin \{u, v\}$ we compute x , the connecting point of w and p , and the distance $d(w, x)$. Let T_0, \dots, T_d be the rooted subtrees of T relative to the backbone (see Fig 1). Let $L_i = L \cap V(T_i)$, for $0 \leq i \leq d$. Each nontrivial T_i can be reconstructed using L_i as follows. Choose an arbitrary $w \in L_i$. Let x be the root of T_i . Apply the procedure described so far for the subtree T_i with $L_i \cup \{x\}$ as set of its outer vertices and w and x as the endpoints of its backbone. Note that x must be treated as an outer vertex although $\deg(x)$ in T_i can be ≥ 1 , the

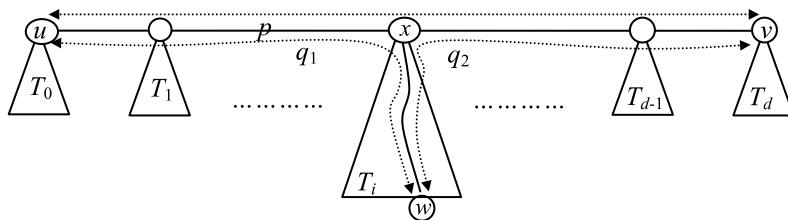


Fig. 1. Reconstruction by outer vertices. x is the connecting point of w and p .

reason being that we know (by Lemma 1) and need the distances between x and the other vertices from L_i .

Proceed recursively within the rooted subtrees of T_i relative to its backbone in order to reconstruct T_i . Applying the same procedure to each nontrivial subtree relative to the backbone of T , we reconstruct the whole tree T .

Suppose that the distances between outer vertices of T are kept in the two dimensional matrix D , $D[i][j] = d'(i, j)$ and `vert` and `dist` are defined as `int`. The calculation of Lemma 1 is performed by the function

```
vert find(vert u, vert v, vert x, dist *d),
```

which returns a not used vertex w , such that the vertex x is the connection point of w and the path from u to v . Further, `find` stores in d the distance between x and w . If there is no such w (i.e., the subtree rooted in x is trivial) the function return 0.

The proof of Theorem 1 allows us to verify the following algorithm which by a set L of outer vertices of a tree T , $|L| = K$, and the distances between each couple of vertices of L , reconstructs T (as a list of edges):

Algorithm 1

```
vert new; dist D[MAXK] [MAXK];
int used[MAXK];

reconstruct(vert from, vert to, dist d) {
    vert w, old; dist di;
    used[to]=1; old=from;
    if(w=find(from,to,from,&di))
        reconstruct(from,w,di);
    for(int i=1;i<=d-1;i++){
        output(old, new);
        if(w=find(from,to,new,&di)){
            reconstruct(new,w,di);
            old=new++;
        }
        output(old,to);
        if(w=find(from,to,to,&di)){
            reconstruct(to,w,di);
        }
    }
    int main() {
        input(K,D[][]);
        used[1]=1; new=K+1;
        reconstruct(1,2,D[1][2]);
    }
}
```

Note that if the tree is weighted the reconstruction by the distances between outer vertices is impossible in general. Indeed, if x is a vertex of degree 2 and y and z are its neighbors there are infinitely many possibilities for $d(x, y)$ and $d(x, z)$ such that $d(x, y) + d(x, z) = d(y, z)$. However, a small modification of Algorithm 1 solves the problem for irreducible weighed trees. In this case every vertex of the backbone is a root of some nontrivial tree relative to that backbone. We can identify the inner vertices, call them x_1, x_2, \dots, x_q , of the backbone using Lemma 1. To reconstruct the backbone it suffices to sort the distances $d(u, x_1), d(u, x_2), \dots, d(u, x_q)$, where u is one of the endpoints of the backbone. So we proved the following theorem.

Theorem 2. *Suppose T is a weighted tree. T can be reconstructed from the distances between its outer vertices iff T is irreducible.*

4. Reconstruction of a Tree from the Distances Between All Vertices

Problem 2. Let $T = (V, E)$ be weighted tree and $d: V \times V \rightarrow N$ is the distance function of T , i.e., $d(v_i, v_j)$ is the weighted distance between v_i and v_j , $1 \leq v_i \leq n$, $1 \leq v_j \leq n$. Given V and d , reconstruct T .

If all edges have the same weight C then the problem is trivial: the smallest $N - 1$ of the distances will be equal to C and they will identify the edges uniquely. So we consider trees with arbitrary lengths of the edges and we prove the following theorem.

Theorem 3. *Each weighted tree $T = (V, E)$ can be reconstructed from the distances between all its vertices.*

Proof. Proceeding as in Theorem 1 we choose arbitrarily two distinct vertices u and v and call the path p between them the *backbone*. Any vertex x belongs to p iff $d(u, x) + d(x, v) = d(u, v)$. Let the inner vertices in p be x_1, x_2, \dots, x_q . As mentioned above, p can easily be reconstructed by sorting the distances $d(u, x_1), d(u, x_2), \dots, d(u, x_q)$. Using Lemma 1, for each vertex w not in p we can compute the corresponding connecting point x_i by searching for the value $(d(u, v) + d(u, w) - d(v, w))/2$ in the sorted list of distances.

Consider, as in Theorem 1, a non trivial subtree T_i relative to the backbone, T_i being rooted at x_i . Let w be any other vertex in T_i . Let the path between x_i and w be the backbone of T_i . Reconstruct that backbone as above and proceed recursively with the rooted subtrees of T_i relative to its backbone. Thus we reconstruct T_i . Applying that procedure to each non trivial subtree T_j we reconstruct the whole tree T .

Some elementary modifications of Algorithm 1 are necessary in order to obtain Algorithm 2 that solves Problem 2. First, `dist` could be defined now as `double`. The function `int build_path(vert u, vert v, int **p int)` reconstructs the path p from u to v by all vertices x such that $d(u, x) + d(x, v) = d(u, v)$, sorted

by $d(u, x)$ and mark all these vertices as used. The function int find(vert x) returns an unused vertex w such that its connecting point to previously found path p is x or 0 if no such w exists.

Algorithm 2

```

dist D[MAXN] [MAXN]; int used[MAXN]

reconstruct(vert from, vert to) {
    vertex w, path[MAXN]; dist di; int psize;
    psize=build_path(from, to, path);
    for(int i=0;i<psize;i++) {
        output(path[i], path[i+1]);
        if(w=find(path[i])) reconstruct(path[i],w);
    }
    if(w=find(to)) reconstruct(to,w);
}

int main() {
    input(N,D[][]); reconstruct(1,2);
}

```

5. Time Complexity of the Algorithms

Let us first estimate the time complexity of Algorithm 2. Because the size of the input is at worst $\Theta(n^2)$ where $n = |V|$ we estimate that complexity as a function, say $t_2(m)$, of $m = n(n - 1)/2$. The reading of the input data necessitates $\Theta(m)$ steps. The algorithm breaks the vertex set of the tree into subsets, each one of which induces a path, those paths being pairwise edge-disjoint (but not necessarily vertex-disjoint), and each recursive call works on one of those paths. The number r of those paths cannot be greater than the number $n - 1$ of edges so r is $O(n)$. Let the paths have n_1, n_2, \dots, n_r vertices respectively. Then $n_1 + n_2 + \dots + n_r < n + r < 2n$ because each path but the first one contains exactly one vertex that we have already considered.

During the i th recursive call we need $O(n)$ steps to find the n_i vertices that belong to the corresponding path, $O(n_i \lg n_i)$ steps to sort them and to reconstruct the path and $O(n)$ steps to choose one vertex from each of the subtrees rooted at a vertex from that path. So all recursive calls need $2r.O(n) + \sum_i O(n_i \lg n_i)$. But $n_1 + n_2 + \dots + n_r < 2n$, therefore $\sum_i O(n_i \lg n_i) = O(n \lg n)$. So the complexity of Algorithm 2 is linear:

$$\begin{aligned}
t_2(m) &= O(m) + 2r.O(n) + \sum_i O(n_i \lg n_i) \\
&= O(m) + O(n).O(n) + O(n \lg n) = O(m).
\end{aligned}$$

The estimation of the time complexity of Algorithm 1 presents us with the following challenge. Assume the output is T represented by, say, adjacency lists. Then the number

of steps performed by Algorithm 1 is at least as big as the number of the inner vertices. That number can vary dramatically for different trees with the same number of outer vertices. As one extreme example let T be a path. The input in this case is a single number $n - 1$: the length of the path. If we make the standard simplifying assumption that any integer necessitates only a constant amount of space to be represented and can be operated upon in constant time, it will turn out that the worst case time complexity is *infinite* since n can be arbitrarily large, therefore there is no *a priori* upper bound on the number of vertices of T . Even if we take into consideration the length of n and express the time complexity of Algorithm 1 as a function of that length, in the worst case the time complexity will be exponential for obvious reasons.

However, if instead of T we output its *equivalent irreducible tree* (defined below) it turns out the time complexity is linear in the length of the input, under the standard assumption that integers require constant space and can be operated upon in constant time. Notice that the problem with the traditional representation is the possibility of long paths in the tree of vertices of degree two. It is because of such paths that the inner vertices can be exponentially more than the outer ones. If there are no vertices of degree two then the number of inner vertices cannot even exceed the number of outer ones, as shown by the following:

Lemma 2. *Let T be an irreducible tree, n_{out} be the number of its outer vertices, and n_{in} , the number of its inner vertices. Then $n_{in} \leq n_{out} - 2$.*

Proof. By induction on the number of steps that the inductive definition has been carried out. The basis is a single edge: obviously, $0 \leq 2 - 2$. Assume the claim holds for some irreducible tree. Consider any irreducible tree obtained from it by adding k new outer vertices to some outer vertex w where $k \geq 2$. Since w is not an outer vertex any more, the number of outer vertices goes up by $k - 1 \geq 1$, and the number of inner vertices goes up by one. The inequality is preserved.

The equivalent irreducible tree of any nontrivial, non-weighted tree T is the weighted irreducible tree T' that is obtained from T after substituting every maximal path $p = u, x_1, x_2, \dots, x_{q-1}, v$, such that x_1, x_2, \dots, x_{q-1} are vertices of degree two by the single edge $e = (u, v)$ with weight $c(e)$ in T' equal to length of p in T , i.e., q . Clearly, the degree two vertices disappear so T' is indeed irreducible. Notice that the other vertices are not affected by this operation and it is in that sense that T' is equivalent to T . It is trivial to restore T from T' .

We estimate the time complexity of Algorithm 1 as a function, call it $t_1(m)$, of $m = n_{out}(n_{out} - 1)/2$. Let n_{in} is the number of inner vertices of degree at least 3 and $n = n_{in} + n_{out}$. From Lemma 2, $n_{in} = O(n_{out})$ and so $n = O(n_{out})$ too. The reading of input data will need $O(n_{out}^2)$ steps and printing of the output in irreducible form – $O(n) = O(n_{out})$ steps. Let us denote again with t the number of paths without common edges passed to the recursive calls and with n_1, n_2, \dots, n_t – the number of inner vertices of these paths, respectively. As it was mentioned above t is $O(n_{out})$ and

$n_1 + n_2 + \dots + n_t < n_{in} + t = O(n_{out})$, because each path use at most one inner vertex of degree at least 3, used by some other path also.

Now the estimation of t_1 is almost the same as of t_2 . During i th recursive call of Algorithm 1 we first find for each not used outer vertex w the vertex x from the backbone where the subtree of w is rooted – $O(n_{out})$ steps – and identify in such way n_i inner vertices. For reconstruction of the backbone in irreducible form it will be enough to sort the found inner vertices – $O(n_i \lg n_i)$ steps. So all recursive calls will need $t.O(n_{out}) + \sum_i O(n_i \lg n_i) = O(n_{out}^2) + O(n_{in} \lg n_{in}) = O(n_{out}^2) + O(n_{out} \lg n_{out})$ and the complexity of Algorithm 1 is $t_1(m) = O(n_{out}^2) + O(n_{out}) + O(n_{out}^2) + O(n_{out} \lg n_{out}) = O(n_{out}^2) = O(m)$.

We suppose that it is not a problem for the reader to make the necessary changes in Algorithm 1 and to obtain a new one with linear time complexity.

6. Comments and Conclusions

The first problem was proposed to the South-East European Regional Round of ACM ICPC in October 2010 (ACM ICPC, 2010 → Problems → Problem G) where 52 teams of university students from 7 countries took part, among them 33 teams from traditionally very strong Ukraine, Romania and Bulgaria. Only 11 teams succeeded in solving the problem (one problem was not solved at all and one was solved by 5 teams). This gave us some reason to classify the Problem 1 as harder than average.

The second problem, which we consider easier, was proposed to the traditional Bulgarian Autumn Tournament in Informatics for school students (INFOS, 2010). The problem was included in the problem set for the second age group (till 16 years) with 53 participants, including students from Croatia, Greece, FYR of Macedonia, Serbia and Romania. We supposed that it would be the hardest problem in the set. Unexpectedly 43 students submitted solutions for this problem (more than for any of the other two problems), 15 of the solutions got at least 70% of the grading marks and 10 solutions got 100%.

In our opinion the second version was more attractive for the contestants than the first because of the existence in the first problem of the mentioned above dependence of the time complexity from a hidden (searched) parameter of the graph. Each such parameter could involve additional difficulties. It seems that some efforts have to be spent during the training of contestants in order to be able to manage tasks with hidden parameters like graph reconstruction problems. Anyway, graph reconstruction problems seem to be a good way to enlarge the scope of the tasks given at programming contests.

With respect to the reconstruction of trees by some metric properties, it will be interesting to try to reconstruct the tree from the distances between all vertices or between the outer vertices when it is not given which distance is between which two vertices. For solutions of the discussed two problems this knowledge is crucial.

References

- ACM ICPC (2010). *ACM ICPC South-East European Regional Contest*. <http://acm.ro>.
- Harary, F. (1964). On the reconstruction of a graph from a collection of subgraphs. In: Fiedler, M. (Ed), *Proc. of the Symposium Theory of Graphs and its Applications*, Czechoslovak Academy of Sciences, Prague, 47–52.
- Harari, F., Palme, E.M. (1966). The reconstruction of a tree from its maximal proper subtrees, *Can. J. Math.*, 18, 803–810.
- Kelly, P.J. (1942). *On Isometric Transformations*, Doctoral Thesis, University of Wisconsin.
- INFOS (2010), *Autumn Tournament in Informatics “John Atanasov”*. <http://math.bas.bg/infos/>.
- Ulam, S.M. (1960). *A Collection of Mathematical Problems*, Wiley (Interscience), New York.



K. Manev is an associated professor of discrete mathematics and algorithms in Sofia University, Bulgaria, PhD in computer science. He was a member of Bulgarian National Committee for Olympiads in Informatics since 1982 and president of NC from 1998 to 2002. He was member of the organizing team of IOI'1989, IOI'1990, vice president of IOI'2009 and leader of Bulgarian team for IOI in 1989, 1998, 1999, 2000 and 2005. From 2000 to 2003 he was elected member of IC of IOI, since 2005 to 2010 represented in IC the host country of IOI'2009, and during IOI'2010 was elected as a member of IC of IOI again for the period 2010–2013.



N. Nikolov is an associated professor (2002) in section "Complex Analysis", Institute of Mathematic and Informatics, Bulgarian Academy of Sciences; Dr.Sc. (2010), Ph.D. (2000). His main scientific interests are in analysis of functions of several complex variables. Leader from 2009 and deputy leader (2004–2008) of the Bulgarian team for the International Mathematical Olympiad as well as leader (2004–2008) and deputy leader (1999–2003) of the team for the Balkan Mathematical Olympiad.



M. Markov is a lecturer and lab instructor of discrete mathematics and algorithms at the Faculty of Mathematics and Informatics, Sofia University, Bulgaria. He is a PhD in computer science from the Institute of Mathematics and Informatics at the Bulgarian Academy of Sciences.

Fairness of Time Constraints

Martin MAREŠ

*Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
e-mail: mares@kam.mff.cuni.cz*

Abstract. Many computer science contests use automatic evaluation of contestants' submissions by running them on test data within given time and space limits. It turns out that the design of contemporary hardware and operating systems, which usually focuses on maximizing throughput, makes consistent implementation of such resource constraints very cumbersome. We discuss possible methods and their properties with emphasis on precision and repeatability of results.

Key words: automatic grading, time limits, precision.

1. Introduction

Many programming contests in the world employ automatic grading of programs submitted by contestants. This is usually accomplished by running the submissions on batches of input data and testing correctness of the output. To distinguish between correct solutions of different efficiency, the tested program must finish within given resource limits. In most cases, the only limited resources are execution time and memory, but there are other possibilities, e.g., communication complexity in case of interactive programs.

For the competition to be fair, all submissions must be treated the same by the grader and the grading process must be repeatable. Contest rules often forbid non-deterministic programs for this reason, or at least they disclaim repeatability in such cases. For deterministic programs, memory consumption and communication complexity are well-defined quantities which can be measured exactly. However, several concerns were raised recently about the precision of time measurements.

In this paper, we investigate this issue. We have performed multiple experiments with timing real contestants' submissions from the IOI 2009 on different machines. We will present a summary of our results and we will try to explain them. As far as we know, this is the first study of this kind based on such broad data.

2. Possible Sources of Inaccuracy

Programming contests are typically run on commodity PC-class hardware with an OS based on the Linux kernel. However, unlike their ancestors from the 1980's, contemporary PC's are fairly complex and the operating systems used on them doubly so. A variety

of optimization techniques are used in both software and hardware with the common goal of improving performance of frequent code patterns at the expense of decreased predictability of instruction timing.

Possible sources of inaccuracy include:

- *Context switches.* Although Linux is usually considered to be a monolithic kernel, it defers many activities (like some kinds of I/O) to system processes. Therefore even if the contestant's program is the only user process running, the scheduler occasionally switches context to a system process and back. The kernel tries to keep track of execution time of each process separately, but the precision of is mostly limited to a single timer tick (on the order of milliseconds). Therefore short-lived context switches are seldom accounted for properly.
- *Cache hierarchy.* Since processors are much faster than the main memory, the processor contains cache memory which remembers recently accessed data in a hope to speed up access to frequently used values. There is typically a hierarchy of several caches of different sizes and speeds. Unfortunately, the caches are not fully associative, so their efficiency can vary with exact placement of data in the main memory. The placement depends on too many external factors. Furthermore, a context switch to a different process leads to flushing data from the cache.
- *Multiple cores.* Parallel computation is no longer limited to expensive servers. Desktop processors contain several processing cores, together forming a small multi-processor system on a chip. Some caches are local to a single core, some are shared by more (but not necessarily all) cores. The process scheduler in the OS assigns running processes to cores, trying to exploit parallelism as much as possible. On the other hand, when a process is moved to another core, it loses a part of the cached data, so it slows down for a short while. The scheduler therefore tries to balance inter-core movements, but the success varies.
- *Non-uniform memory architecture (NUMA).* On some machines, the main memory is not uniform. There are multiple memory controllers, each of them handling a different part of memory. Usually, the machine consists of several NUMA nodes, each of them containing a set of cores and a set of memory controllers connected to each other. Memory accesses within a node are then faster than inter-node ones. Again, the placement of processes and their data on the nodes is rather unpredictable.
- *Power management.* When the system load is small, the OS tries to save electric energy by putting functional blocks of the hardware to various power saving modes. E.g., an idle CPU core can have its operating frequency lowered (so it runs slower), or it can be even turned off completely. Transitions between these states are not immediate and they can incur flushing data from caches. Fortunately, most of the power management functions can be turned off and we recommend to do so on grading machines.
- *System management mode.* Machine firmware (the BIOS) can act behind the scenes. Most importantly, the hardware can ask the processor to interrupt execution of the program (and even of the OS kernel), to switch to a so-called system management mode and run a firmware routine. Significant amounts of time can be

spent in the SMM and since the OS is not notified, all this time is accounted to the currently running process. This is a common problem on laptops, but effects caused by the SMM have been observed on desktop and server computers as well.

3. Experiments

3.1. Test Setup

For our experiments, we have chosen three different tasks from the IOI 2009 in Plovdiv, Bulgaria:

- *Raisins* – this is a usual batch task (the program reads an input file and produces and output file) with small inputs (the largest input file has 10 kB), but requiring a lot of time and memory. The model solution runs in time $\mathcal{O}(n^5)$ and memory $\mathcal{O}(n^4)$.
- *Mecho* – another batch task, but with larger inputs (the largest test case reaches ca. 600 kB) and smaller resource requirements. The model solution runs in time $\mathcal{O}(n \log n)$.
- *Regions* – this is an interactive task requiring implementation of a data structure. The program interacts with a judge program, it receives queries from the judge and it has to answer them on-line. However, the judge's queries are not adaptive, so we can also run the task as a batch task for comparison. The test data contain about 105 queries per test, the model solution runs in time $\mathcal{O}(n^{0.5})$ per query and in linear space.

For the full description of the tasks, see Tzanov *et al.* (2009).

For all three tasks, we have used real contestants' submissions from the IOI, written in Pascal, C and C++, and also real test data. In some experiments, we had to reduce the submissions and the tests to a (hopefully representative) subset in order to run them within reasonable time. Also, we have dropped several C++ programs which failed to compile with a recent compiler.

We have run the tests on the following machines:

- *Camellia* – AMD Athlon64 X2 6000+ with 2 cores running at 1 GHz, equipped with 2 GB of RAM.
- *Turing* – Intel Core2 Quad with 4 cores running at 2 GHz, 4 GB of RAM.
- *Corbu* – Intel Core i7 with 4 cores at 2.66 GHz, 6 GB of RAM.
- *Arcikam* – AMD Opteron 2218, 2 processors with 2 cores each, running at 2.5 GHz. Arcikam's 8 GB of RAM are split to 2 NUMA nodes, each containing one processor.

All machines run Debian Linux 5.0 (Lenny) with kernel 2.6.37. The programs were compiled by GCC 4.3.2 and FreePascal 2.4.2. Effects of different versions of the kernel and of the compilers were also investigated, but they were negligible, so we do not discuss them here in detail. Also, where the behavior of several machines or of several tasks is sufficiently similar, we show only a single example. For grading, we have used the Moe

contest environment developed by us; see Mares (2009) for details. The only part of Moe's grader, which can affect timing in a significant way, is the sandbox. It monitors all system calls for security reasons, passing them to a supervisor process for inspection. For some of the tests, we turned the sandbox off to check how much slowdown it causes.

3.2. Error Distribution

First of all, we try to establish statistical behavior of time measurement errors. We pick a single submission and run it repeatedly on a single input file. Histograms of two such experiments on batch tasks can be seen on Figs. 1 and 2, other experiments always resembled one of these two types. Our interactive task behaves in a more complex way, which will be discussed separately in Section 3.5.

In the first case (Arcikam), we got a distribution close to Gaussian, the second case (Turing) is more peculiar: most values are concentrated near the mean, but there is a couple of exceptional values much farther. Table 1 shows the fractions of samples within k -times the standard deviation from the mean.

Another interesting question is how much the standard deviation depends on the mean (that is, whether the error is primarily additive or multiplicative). To answer this, we have run all submissions of a single task 5 times on a single input and plotted standard deviation against mean. A typical plot looks like Fig. 3.

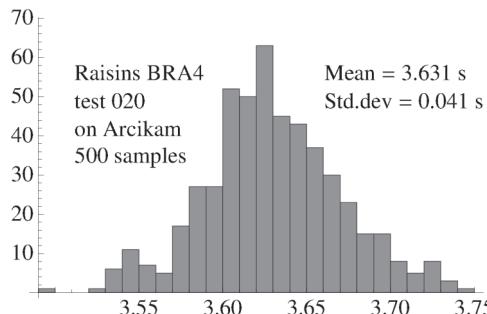


Fig. 1. Histogram of time [s] spent on a single test, type 1

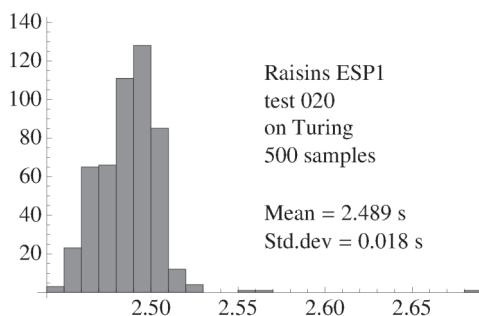


Fig. 2. Histogram of time [s] spent on a single test, type 2

Table 1
Probability of values within k -times standard deviation

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
Arcikam	0.694	0.938	0.998	1.000	1.000	1.000
Turing	0.740	0.982	0.994	0.996	0.998	0.998
Gaussian	0.683	0.955	0.997	0.999	0.999	1.000

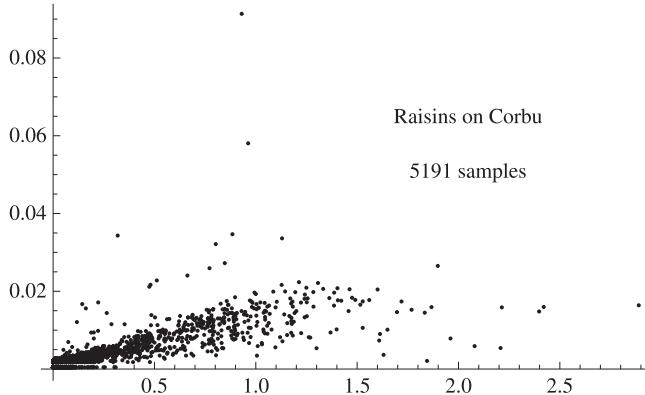


Fig. 3. Dependence between mean and standard deviation [both in s]

The data look roughly linear and indeed, linear regression gives a good fit with small variance. Furthermore, the linear approximation has an almost zero constant term, so we can conjecture that the error consists of many small parts well distributed over the whole run-time of the program. Indeed, Gaussian distribution is typical for such processes.

We also note that when we run the test on Camellia with power management turned on, there is a significant constant term, which corresponds to the time needed for an idle core to gain speed. On newer machines, the transition delay is much smaller, so power management causes only the standard deviation to increase.

3.3. Tuning the Scheduler

We have already explained several effects, which contribute to measurement errors. Most of them are related to the way how the OS schedules processes. We however need not blindly follow the default settings of the scheduler – many parameters affecting the scheduling decisions can be indeed configured. For example, a set of processes can be pinned to a specific core or NUMA node. In this section, we will consider several tricks of this type and quantify their effects.

We have observed that different submissions (or even different tests of a single submissions) are affected in different ways. It even happens that a single change speeds some submissions up, while slowing down the others. In order to average out these effects, we want to test several different submissions on several different inputs.

For each task, we have manually selected 5 submissions such that they finish within a reasonable time (larger than the time limit originally set for the task) without any runtime errors or security violations. We have also picked 3 input files such that the execution times of the selected submissions are not too small, so that they lie in the approximately linear region of Fig. 3. Each pair of a submission and an input (we will call it an *instance*) was run 50 times, alternating different submissions, so that the caches are flushed like in a real contest environment.

To make it possible to combine the results, we normalize them. For each instance, we calculate a reference time by dropping the smallest 2 and the largest 2 values and taking a mean of the remaining values. We then divide each measurement by the reference time for that particular instance. (Under our hypothesis on the source of the errors, this normalization is correct.) When we run the tests again under different conditions, we re-use the reference times from the base experiment.

The graphs in Figs. 4 and 5 show results for 5 experiments with the task Raisins run on Corbu. Results for other tasks and machines are similar. The left graph shows order statistics: the gray box shows the range between quantiles 0.25 and 0.75, the horizontal line inside the box is the median and the “whiskers” show the full range of the values. The boxes in the right graph are centered on the mean and their height is twice the standard deviation.

The five experiments are as follows:

- **K** – the base experiment.
- **C** – the scheduler was told to handle the submission as a real-time task (FIFO class) with priority higher than all other user processes. This was intended to decrease effects of contexts switches. Surprisingly, while it has decreased the mean, it has increased variance.
- **R** – again with real-time scheduling, but this time we have pinned the process to a specific core, hoping to eliminate inter-core movement and cache pollution. The variance is better than **C**, but nowhere near **K**.

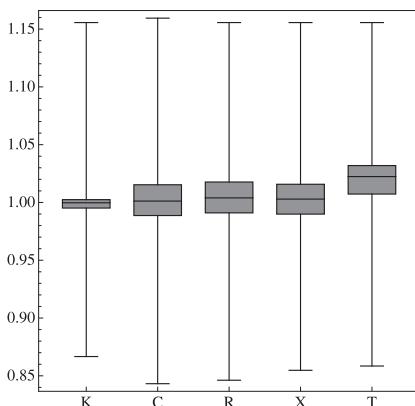


Fig. 4. Raisins on Corbu: quantiles

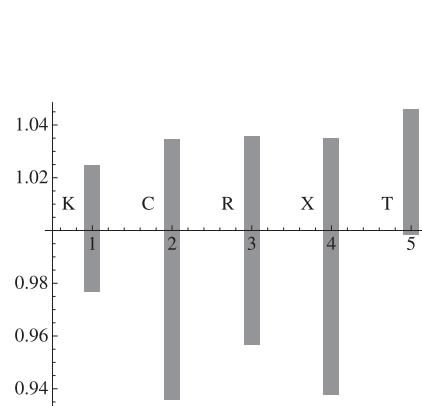


Fig. 5. Raisins on Corbu: mean and deviation

- **X** – in addition to **C**, we have turned off syscall interception in the sandbox. It is clear that the overhead of the sandbox is negligible. (In Mecho, it is slightly visible, because we need much more system calls for I/O, but it is still insignificant.)
- **T** – the base experiment performed with an older kernel (2.6.32, approximately 13 months old). It is well visible that the scheduler was significantly improved since that version.

We can conclude that the process scheduler in the kernel is doing its job very well and that our attempts to reduce the variance by tweaking scheduler parameters are in fact making the results worse.

3.4. Parallel Execution

With a multiple-core machine, it is tempting to evaluate multiple submissions in parallel on different cores. Given the possibilities of interference between the submissions (e.g., competing for access to a single common memory controller or for space in caches), one should be very careful about that. We will use the method of normalized measurements from the previous section to estimate the effect of parallelization.

The results can be seen in Figs. 6 and 7: **R1**, **R2** and **R4** is the task Raisins run on Corbu (a 4-core machine) with 1, 2 or 4 processes running in parallel (**R1** is used as a base case for our normalization). **M1**, **M2** and **M4** are the same for Mecho.

For both tasks, parallel evaluation gives both larger mean (by almost 20% in **R4**) and much higher variance. Mecho is harmed much less, we conjecture that because it has much smaller memory footprint, most of the time it runs from the cache, which shields it from the other processes.

We also tried to make use of the NUMA at Arcikam: we have run two instances of the grader, each pinned to a single NUMA node. Figures 8 and 9 show the results: **R1** is the base case with a single process, **R2** uses 2 processes with default scheduling, **R2'** uses node pinning. **M1**, **M2** and **M2'** are analogous tests for Mecho. We can see that even

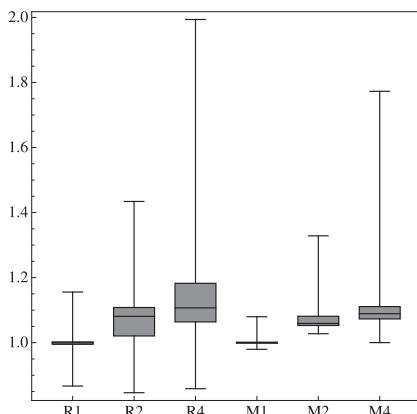


Fig. 6. Parallal grading: quantiles

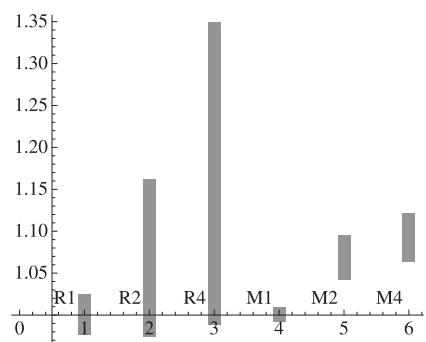


Fig. 7. Parallal grading: mean and deviation

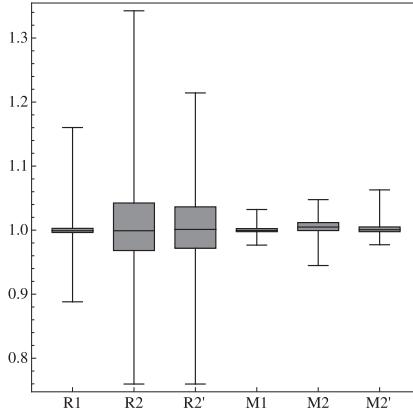


Fig. 8. Parallel grading on NUMA: quantiles

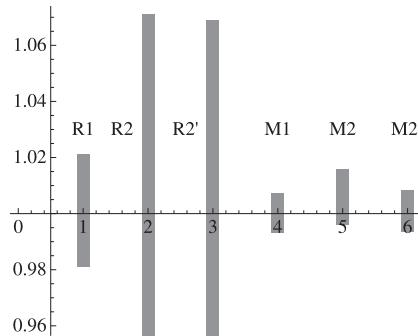


Fig. 9. Parallel grading on NUMA: mean and deviation

with default scheduling, the NUMA exhibits a much smaller variance and explicit node pinning helps even more. Again, Mecho is affected significantly less than Raisins.

3.5. Interactive Tasks

Let us turn our attention to Regions, our interactive task. We can expect serious problems here, because of huge amount of context switching between the submission, the judge (that is, the program generating and checking the queries) and the system call monitor of the sandbox. This was already noted by an earlier paper by Merry (2010), who studied the same task and compared performance in different types of sandboxes. We will try to gain more insight into the related issues.

Figures 10 and 11 summarize results of several experiments with Regions (we again use our normalization technique, but with only 5 submissions and 2 test inputs to make it

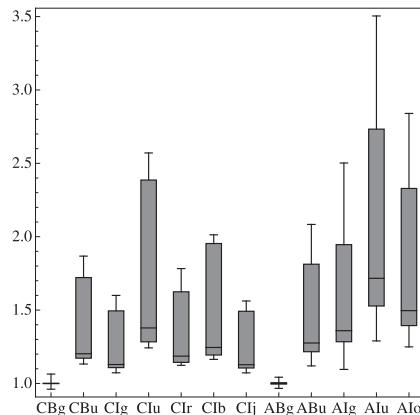


Fig. 10. Regions: quantiles

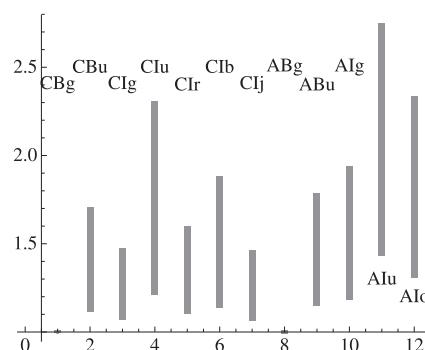


Fig. 11. Regions: mean and deviation

finish within reasonable time). The experiments are assigned three-letter codes: the first letter encodes the machine (**C** for Camorra, **A** for Arcikam), then comes **B** for the batch version of the task or **I** for the interactive version with the judge, and finally:

- **g** when running with no system call monitoring,
- **u** for a normal run (with system call monitoring),
- **b** for **u** with pinning to a specific core,
- **r** for **b** with real-time scheduling,
- **j** for **g** with pinning the submission to one core and the judge to another,
- **o** for **u** with pinning to a specific NUMA node.

As the base case for normalization, we have chosen the most straightforward version: batch mode with no system call monitoring.

It is clearly visible that system call monitoring has very significant overhead, even in batch mode (this has a simple reason: in order to work with the interactive judge, all submissions flush their output after every line, so they need at least one system call per query in batch mode). Not only that – the difference between interactive and batch runs is also immense regardless of the sandbox. Explicit pinning of processes to cores or nodes helps, but the slowdown is still very high.

A histogram of normalized execution times (e.g., of the **CIu** test) in Fig. 12 reveals a surprise: there are two clusters where the values are concentrated, located far from each other. Careful inspection of the raw data confirms that it is indeed the case – for some instances, the slowdown is much higher than for others. We do not have a good theoretical explanation of this phenomenon yet.

We get a completely different picture when we calibrate each test run against itself (that is, we do not compare with the non-sandboxed batch-mode base case and we only check consistency of each test run per se). Normalized standard deviations drop to values around 0.02, which is similar to what we got for batch tasks. Pinning still gives a small improvement. A histogram of values, again for the **CIu** test, can be seen on Fig. 13.

Grading interactive tasks using online communication with a judge process therefore does not provide realistic timing, when compared to the behavior outside the grader. This

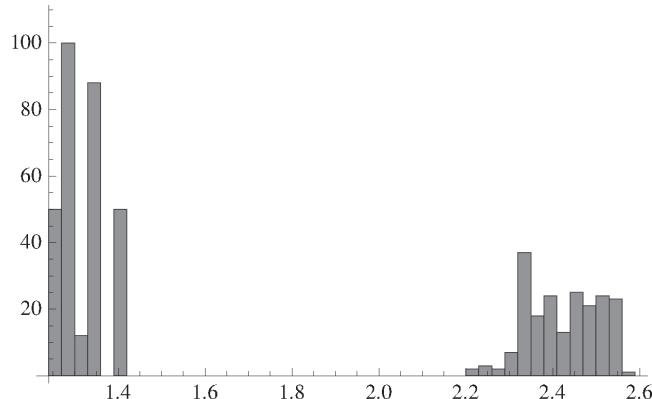


Fig. 12. Surprising behavior of the test **CIu**

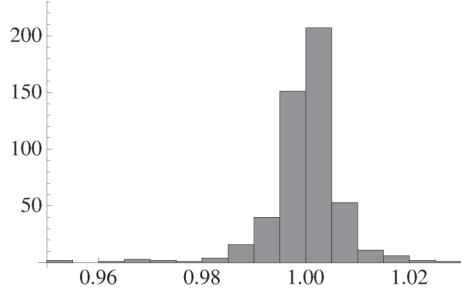


Fig. 13. Test **CIu** normalized with respect to itself

holds regardless of the sandbox. Despite of this, the timing can be still self-consistent and fair.

Compared to the Merry's paper, we have observed much more reliable results. We suspect that the difference is caused by improvements in the Linux kernel since version 2.6.30 used by Merry.

4. Conclusions

We have studied the problem of measuring program execution time in programming contests. We have performed multiple experiments with data from a real contest on different machines. We have also given theoretical explanation of some of the observed phenomena.

For batch tasks, we have found that the variance of measured values is reasonably low and that the process scheduler in recent Linux kernels utilizes the hardware fairly well. All attempts to override the scheduling decisions manually by pinning the processes to specific CPU cores or NUMA nodes made the variance worse. Also, the overhead of system call checking by our sandbox turned out to be very small.

For interactive tasks, the situation is much more complicated. We considered the typical setup with an interactive judge process communicating with the graded program over pipes. In this case, context switches between these two programs have drastic effects on timing, regardless of the sandbox used. When compared with execution outside the grader, the timings are not representative; when compared with each other, they can still be considered fair in some cases.

We recommend grading system authors take into account that unlike memory use or communication complexity, execution time behaves as a random variable encumbered by measurement errors. There are several ways how to cope with this fact. One of them is to repeat the measurements and use either minimum or mean of the results. This however makes grading much slower. Another possibility is to repeat the measurement only if the value is near the time limit (according to our data, the distribution of errors is concentrated around the mean). Or we can replace the binary time limit (passed/failed) by a continuous function transforming time to points – this is already used in the Polish olympiad in informatics for several years.

We also recommend avoiding the type of interactive tasks with a separate judge process in all cases where the number of queries is large. Uses with only a few queries (e.g., when playing combinatorial games) should be safe. When the number of queries has to be large, integrating the judge with the submission to a single process will have much better performance, but it is hard to do in a secure way. One possibility applicable to judges with non-adaptive queries could be reading the queries from a file, decrypting each query using the answers to the previous queries as a key (this enforces on-line execution). More possibilities of grading interactive tasks should be examined, for example communication through shared memory guarded by spin-locks, which might avoid scheduling altogether.

Finally, we have examined the possibility of grading multiple submissions simultaneously when a multiple-core machine is available. The variance turned out to be high, but in cases where the number of cores exceeds the number of submissions, it seems to be acceptable. Assigning processes to cores or nodes manually seems to improve the situation somewhat. Nevertheless, parallel grading should be done with a great care and checked carefully.

References

- Mareš, M. (2009). Moe – design of a modular grading system. *Olympiads in Informatics*, 3, 60–66.
Merry, B. (2010). Performance analysis of sandboxes for reactive tasks. *Olympiads in Informatics*, 4, 87–94.
Tzanov, V. et al. (2009). Tasks and Solutions. In: *21st International Olympiad in Informatics*, Plovdiv, Bulgaria.



M. Mareš is an assistant professor at the Department of Applied Mathematics of Faculty of Mathematics and Physics of the Charles University in Prague, a researcher at the Institute for Theoretical Computer Science of the same faculty, organizer of several Czech programming contests, member of the IOI Scientific Committee and a Linux hacker.

Measuring the Startup Time of a Java Virtual Machine

Bruce MERRY¹, Carl HULTQUIST²

¹*ARM Ltd.*

110 Fulbourn Road, Cambridge, CB1 9NJ, England

²*D.E. Shaw & Co.(U.K) Ltd.*

55 Baker Street, London, W1U 8EW, England

e-mail: bmerry@gmail.com, chultquist@gmail.com

Abstract. For many olympiad problems, an execution time limit is used to constrain the classes of algorithms that will be accepted. While Just In Time (JIT) technology allows Java bytecode to be executed at a speed comparable to native code, the Sun Java Virtual Machine has a reputation for a large startup time that can significantly affect measured execution time. We present a technique for measuring the startup time of the virtual machine for each execution run, so that it may be subtracted from the total execution time. We found that the startup time was lower than expected (about 70ms), with little variation between programs or runs.

Key words: Java, startup time, time limit.

1. Introduction

Numerous programming contests allow Java to be used as a programming language (ICPC, 2010; Kolstad, 2009; Merry *et al.*, 2008; Tani and Moriya, 2008; Trefz, 2007). Additionally, it is common practise in programming contests to impose an execution time limit on solutions, both to protect the system against long-running solutions and to enforce some degree of algorithm efficiency on solutions. When multiple programming languages are made available, the question that naturally arises is whether the choice of language affects execution time and if so, whether this is fair to contestants.

Java is different from C-like languages in that it is typically compiled to bytecode and run in an interpreter, rather than compiled directly to machine code. The de-facto standard interpreter is the Sun Java Virtual Machine (JVM). While the execution speed of such interpreted code remains a concern (even with technologies such as Just-in-Time (JIT) compilation), this paper specifically addresses startup overhead. In evaluating Java for the International Olympiad in Informatics, it has previously been noted (Pohl, 2006) that the JVM can take anywhere up to a second to start, and that the startup time is highly variable.

To address this, the South African Computer Olympiad (SACO) grading system measures the startup time each time a Java solution is run, and subtracts this from the total execution time. Section 2 describes how this is achieved.

In order to evaluate the effectiveness of our wrapper, we are interested in the following questions:

- What is the average startup time of a C++ program?
- What is the average startup time of a Java program?
- Does the Java wrapper have any impact on execution time?
- Is the startup time of a Java program greater than that of a C++ program?
- Is the solution time of an empty Java program greater than that of a C++ program?
- Does the choice of program (e.g., a simple versus a complex one) impact the startup time?
- Does the solution time computed using our wrapper have less variation than execution time? In other words, does our wrapper lead to more consistent timing?

Section 3 explains how we tested the effectiveness of our technique, and we report the results of our tests in Section 4. We finish with conclusions in Section 5.

2. Measuring Startup Time

A simple approach to measuring startup time would simply be to run a trivial program multiple times to determine an average. However, the startup time could vary from program to program or even run to run, and so the average is only a crude approximation. The measured average would also only be valid for the combination of hardware and software and so would need to be recalibrated each time a system update was done. It could also vary over time even if the system is not changed, due to factors such as file system fragmentation.

Instead, we measure the startup time of each run as it is happening. To accomplish this, we do not launch the user's Java class directly. Instead, we launch a helper class that does the following:

1. Uses reflection to look up the `main` method in the user's class.
2. Calls a native function we have written that measures the time elapsed so far, and passes it to the process that manages time limit enforcement.
3. Invokes the `main` method in the user's class.

We discuss these steps in detail in the following subsections.

2.1. Class Lookup

Unlike languages such as C, Java provides reflection mechanisms that allow classes and methods to be looked up by name at run time. The name of the user's class is passed to the wrapper on the command-line, and its `main` method is located with the following code:

```
Class<?> childClass;
Method childMain;
Object childArgs;
```

```
childClass = Class.forName(args[0], false,
    JavaWrapper.class.getClassLoader());
childMain = childClass.getMethod("main", String[].class);
childArgs = Arrays.copyOfRange(args, 1, args.length);
```

It is not strictly necessary to do this class lookup dynamically; one could instead generate a wrapper on the fly for each possible user class name. We have chosen to use reflection largely for convenience.

2.2. Reporting Startup Time

Standard Java libraries do not provide a means of determining the CPU time consumed so far. Instead, we use Java Native Interface (JNI), a mechanism that allows Java methods to be implemented in a native language such as C. This C code is compiled into a shared object which the JVM loads at run time.

Since our evaluation system is based on GNU/Linux, our native code calls `getrusage` to determine the CPU time that has been consumed by startup overheads. It then needs to report this to the parent process (which is doing timing and enforcing resource limits), in a secure way. We chose to have the parent process open a pipe between parent and child when the child is created, with a fixed file descriptor number (3). The native code running in the child writes the startup time to this pipe, and then closes it to ensure that the user's code cannot access it.

2.3. Launching the User's Class

Having measured all startup overheads to this point, we are ready to launch the user's code. This is done by calling

```
childMain.invoke(null, childArgs).
```

This passes any remaining command-line arguments to the child. Most olympiad problems do not process command-line arguments, but doing so makes the wrapper general-purpose.

2.4. Exception Handling

The sample code listed above does not include any exception handling. There is relatively little exception handling required, because an exception will usually indicate a fault in the user's submission (whether because their code threw an exception, or their class didn't have a `main` method with the appropriate signature or permissions, or some other reflection issue), and the submission will score zero with the exact reason being irrelevant. There is one special case: because the parent process is waiting for the child to inform it of the startup time, this must be reported even if the class lookup throws an exception.

Our use of reflection causes exception messages to be different from those that would be seen had the code been run outside the wrapper. This is because exceptions thrown by the invoked method are wrapped in another exception, such as `InvocationTargetException`, to allow exception handlers to distinguish them from exceptions in the invocation process itself. To make our wrapper more transparent, we catch these exceptions and re-throw the underlying exception, causing the error log to match what would have appeared in the absence of the wrapper.

2.5. Security Considerations

No matter how well this scheme works, it would be of dubious practical value if contestant code could either perform computation in the nominal “startup” time (thus bypassing the resource limit), or was able to report a false startup time to the parent process. In this section we consider some possible ways that user code might attack the system.

First we consider whether user code can execute during “startup” time. Because we load the user’s class before recording the startup time (so that the time taken for this reflection process does not count against the user), we must not allow any user initialisation code, such as a static class initialiser, to be run during this loading. This is done by passing `false` to `Class.forName`, which indicates that the class should not be initialised at this point. The class is instead initialised when the `main` method is invoked.

Since no user code is able to run before the startup time is reported to the parent, it cannot interfere with the reporting process. It cannot even send false data along the same pipe, because the native code that does the reporting closes the pipe before it returns.

One other risk is the presence of the native code: the user’s code can cause it to execute again (for example, by calling the `main` method in the wrapper class), and so it must be made robust to repeated execution.

3. Test Setup

In order to test the usage of our wrapper, we used the solutions to several olympiad problems to generate statistics about run time under various scenarios. Specifically, we took the following measurements:

- *Execution time* – the total time taken by the process, including any startup and wrapper overheads.
- *Startup time* – the startup time measured by our wrapper, prior to the user’s code being launched.
- *Solution time* – the time taken by the user’s code, computed as the difference between execution time and startup time.

Where our wrapper has not been used, only execution time can be measured directly.

Our test machine has a 2.16G Hz Core 2 Duo T7400, with 1GB RAM, and runs a Linux 2.6.36 64-bit preemptable kernel and 64-bit Gentoo OS. C++ solutions were compiled with GCC 4.4.5 with options `-O2 -s -static` while Java solutions were compiled and executed with Sun Java SE Runtime Environment 1.6.0_24-b07.

The C++ and Java solutions for each problem are functionally identical, so as to minimise the effects of different algorithms impacting our measurements. We also include a special `empty` program which, as its name implies, does no processing and returns immediately. The C++ and Java `empty` programs are as follows:

```
// C++
int main()
{
    return 0;
}

// Java
public class empty
{
    public static void main(String[] args)
    {
    }
}
```

Since we are aiming to measure startup time rather than solution speed, all our tests were done with very small input files for which processing is expected to take almost zero time.

For statistical significance, each combination of solution, language, and the presence or absence of the wrapper was run 200 times.

One limitation in our approach for collecting test data is the quantisation of the results returned by the `getrusage` system call. The kernel on our test system has a tick frequency of 1000 Hz, which quantises our results to 1 ms buckets. We have not attempted to take these quantisation effects into account when applying statistical tests, and so p values quoted may be inaccurate.

4. Results

The figures below show box plots. Each box represents the inter-quartile range, the bar inside the box represents the median, and the small circles represent outliers (the 1 ms quantisation causes more data points to be flagged as outliers than might otherwise be the case).

We now address the questions listed in Section 1. Firstly we consider C++: every run on all our test programs reported an execution time of 1 ms. It is likely that less than 1 ms was required, but that the 1 ms quantisation caused everything to be rounded up to 1 ms.

We can estimate Java startup time without the use of our wrapper. Since our `empty` program performs no computation, its execution time is essentially the startup time. Figures 1 and 2 show that, on average:

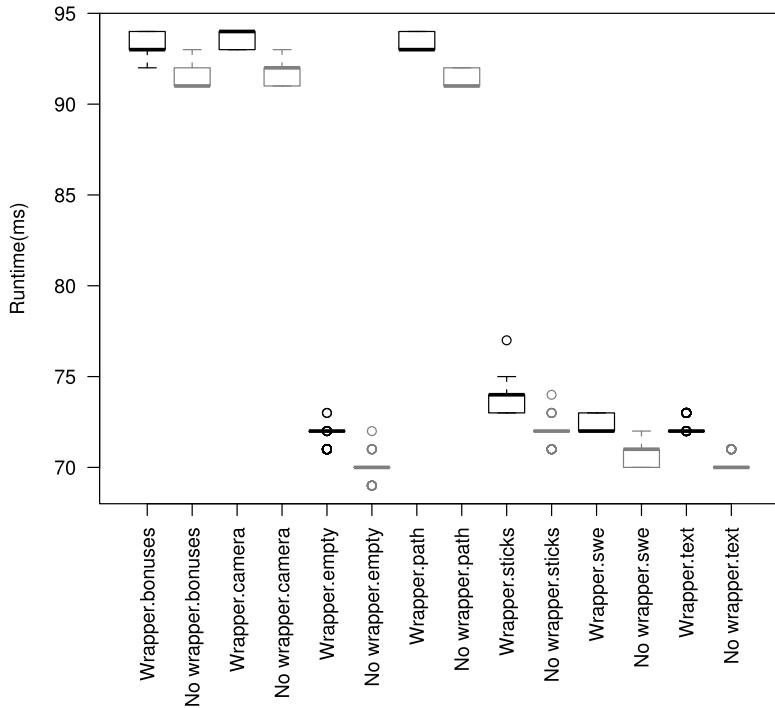


Fig. 1. Execution time for various Java programs, without the use of our wrapper.

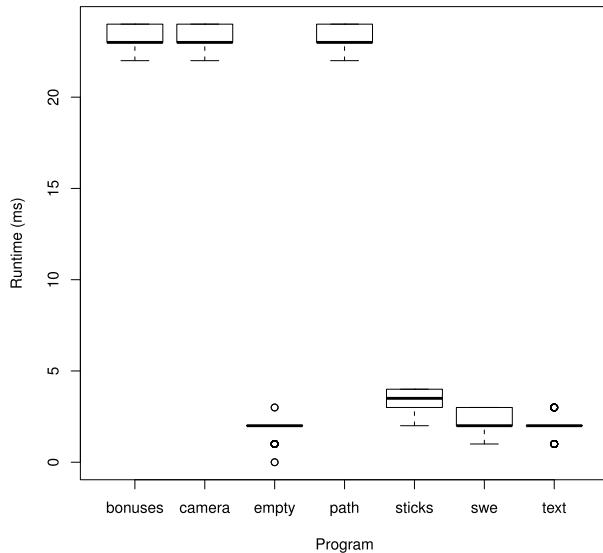


Fig. 2. Solution time for various Java programs with their sample test-case as input, after subtracting the measured startup time of the JVM.

- Java with no wrapper has an execution time of 70.0 ms.
- Java with our wrapper has an execution time of 71.8 ms.

In each of these data-sets, the standard deviation of the measurements is less than 0.5 ms and is thus unreliable in telling us how the individual times vary, due to the quantisation of our measurements to 1 ms buckets. Figure 3 gives an alternative view of the effect of our wrapper, showing the average startup time and solution time for each program. It shows that of the 71.8 ms average time for the empty program, only 1.8 ms is solution time and the remainder is startup time.

These data also address our question of whether the use of our wrapper impacts execution time. The mean execution time for the `empty` Java program is clearly greater than that of the same program without our wrapper, and a t-test confirms that this finding is statistically significant with $p < 0.001$.

Furthermore, it is clear both from the data and conventional wisdom that the startup time of a Java program is greater than that of a C++ program – and our data shows that this is true even when taking the startup time into account. Applying a t-test to our data, we find that the solution time of the wrapped `empty` Java program (1.8 ms) is statistically significantly more than the 1 ms execution time of the `empty` C++ program, with $p < 0.001$. Similarly, the execution time of the unwrapped `empty` Java program (70 ms) is also statistically significantly more than that of C++, again with $p < 0.001$.

The figures also show a clear variation in execution time between the programs; from Fig. 3 and 4 it is clear that this variation is part of the solution time, rather than the startup time. Investigation showed that the three slow programs (`bonuses`, `camera` and `path`) all used the `Scanner` class for parsing input, while the remaining programs did not. This class is more convenient than other methods of parsing input (such as

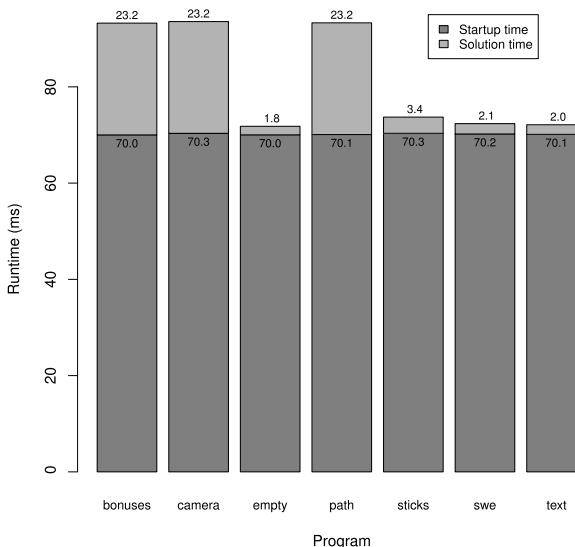


Fig. 3. The average startup time and solution time for various Java programs.

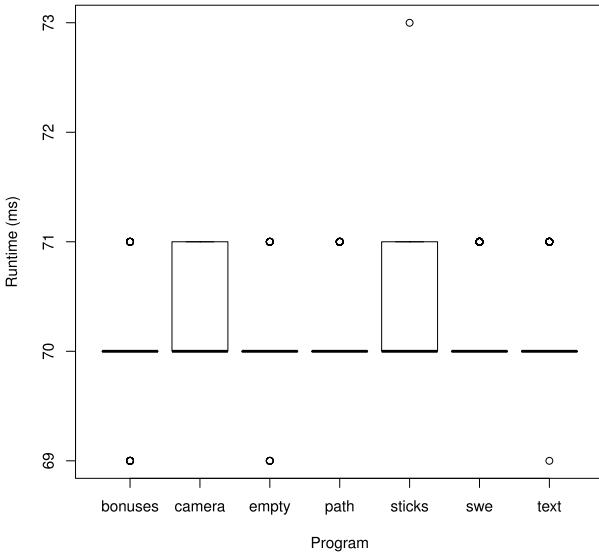


Fig. 4. Startup time measured by our Java wrapper for various Java programs.

`StringTokenizer`), but apparently this convenience comes at a price in initialisation time.

Finally, we address the question of whether subtracting the startup time for a Java program reduces the variation in reported run time. From Figs. 1 and 2, it appears that the standard deviation in run time for each program has not improved after subtracting the startup time. This is further supported by the box plots in Fig. 4, which show that the measured startup time is almost constant at 70 ms. However, given that all the standard deviations in these data sets are less than 1 ms, we cannot draw any definite conclusions due to the 1 ms quantisation of our measurements.

5. Conclusions

Prior experience, both our own and that reported by Pohl (2006), led us to expect that the startup time of Java programs would be significant and highly variable, and that our technique would be valuable for programming contests using Java with time limits of up to a few seconds.

We were thus surprised to find that the startup time we measured was only around 70 ms, and highly consistent across different programs and different runs. Indeed, the variation observed between programs was not due to the startup time of the JVM, but to run time consumed by the `Scanner` class. It is not currently known whether this small and consistent startup time is due to improvements in the Virtual Machine implementation since 2006, or to changes in the underlying environment (faster CPUs and larger caches).

Given the consistency of the measured startup time, it seems that a simple solution of applying a fixed correction to all Java run times is likely to be sufficient. Nevertheless,

our method is particularly convenient since it does not require separate calibration. It also adds only a few milliseconds to the total execution time, and thus will have minimal impact on the rate at which solutions can be judged.

After correcting for the JVM startup time, a trivial Java program runs almost as fast as a trivial C++ program – the difference is less than the quantisation error caused by operating system time-slicing. We thus feel confident that while execution speed may still be a concern (and is beyond the scope of this paper), startup time is no longer an issue when supporting Java as a programming language in a contest.

5.1. Future Work

Although we have addressed the issue of startup time in Java programs, there are several additional avenues for research that would assist in supporting the more widespread adoption of Java in programming contests:

- **Class-specific wrapper.** In Section 2.1 we described how we use reflection to look up the user’s class, and noted that an alternative approach would be to generate a wrapper on the fly and avoid the reflective lookup. This approach may reduce the additional run time overhead incurred by the wrapper.
- **Shutdown time.** Although our technique focuses on reducing startup time, a similar approach might be used to measure *shutdown time* (that is, the time required for the JVM to terminate after the user’s program has completed).
- **Preloading of key classes.** Our analysis identified that the use of certain classes (such as `Scanner`) can have a measurable impact on the startup time of a program. One possible means of ameliorating the effect of such classes would be to preload them in the wrapper, and thus separate their initialisation time from the time attributable to the user’s program. Further investigation would be required to determine whether this overhead is for class initialisation or is per-instance.
- **JVM priming.** The JVM is well known for performing just-in-time compilation and adaptive optimisation of executing bytecode, which can result in fully optimised machine code only being executed at some indeterminate point in the program’s execution. Java programs may therefore be at a disadvantage to those written in other languages, such as C++ or Pascal, where program code is precompiled straight to optimised machine code. One possible means of reducing this disparity would be to “prime” the JVM by executing the Java program many times within the same JVM instance, thus giving it an opportunity to fully optimise and compile all the code. Care would, however, need to be taken in such an approach to ensure that the user’s program did not store any information for use in successive runs.
- **Other JVM implementations.** This paper has focused on the Sun JVM; it would be interesting to perform similar analysis on other JVM implementations.

References

- ICPC. ICPCWiki: World finals rules. (2010).
<http://cm.baylor.edu/ICPCWiki/Wiki.jsp?page=World%20Finals%20Rules>.
<http://cm.baylor.edu/ICPCWiki/Wiki.jsp?page=World%20Finals%20Rules>.
- Kolstad, R. (2009). Infrastructure for contest task development. *Olympiads in Informatics*, 3, 38–59.
- Merry, B., Gallotta, M., Hultquist, C. (2008). Challenges in running a computer olympiad in South Africa. *Olympiads in Informatics*, 2, 105–114.
- Pohl, W. (2006). *IOI Newsletter*, 1.
<http://ioinformatics.org/newsletters/html/ioinews6.htm>.
<http://ioinformatics.org/newsletters/html/ioinews6.htm>.
- Tani, S., Moriya, E. (2008). Japanese olympiad in informatics. *Olympiads in Informatics*, 2, 163–170.
- Trefz, N. (2007). The coding window – TopCoder wiki.
<http://www.topcoder.com/wiki/display/tc/The+Coding+Window>.
<http://www.topcoder.com/wiki/display/tc/The+Coding+Window>.



B. Merry took part in the IOI from 1996 to 2001, winning two gold medals. Since then he has been involved in numerous programming contests, as well as South Africa's IOI training programme. He obtained his PhD in computer science from the University of Cape Town and is now a software engineer at ARM.



C. Hultquist took part in the IOI from 1999 to 2000, winning a silver medal and a bronze medal. He has since taken part in and organised several other programming contests, including active involvement in South Africa's IOI training programme. Carl obtained his PhD in computer science from the University of Cape Town and is currently working as a software engineer for D.E. Shaw.

Tasks of “Mission Impossible” and “Mission Impeded” Types

Pavel S. PANKOV¹, Kirill A. BARYSHNIKOV²

¹*International University of Kyrgyzstan*

A. Sydykov str. 252, apt. 10, 720001 Bishkek, Kyrgyzstan

²*OJSC “Finance Credit Bank”, Bishkek, Kyrgyzstan*

e-mail: pps50@rambler.ru, kiryakg@gmail.com

Abstract. We propose a way to derive new tasks by reversing the goals of existing tasks as follows. Some operation is possible now and we can perform it in some number of steps. We would like to make this operation impossible (or to increase the number of steps to complete it). What is the minimum number of steps we should do to achieve our goal? Some tasks of the proposed types and ways to create such tasks in various branches of informatics are presented in the paper.

Key words: olympiads in informatics, tasks, impossibility, impediment.

1. Survey of Types of Tasks and the Aim of Paper

There are number of publications on the topic of creating tasks for informatics olympiads, for example Kemkes *et al.* (2007), Diks *et al.* (2008), Pankov (2008), Pankov and Baryshnikov (2009), Pankov *et al.* (2010).

We cite Burton *et al.* (2008) “The question might involve finding a specific object (or set of objects) that has some property, or that maximizes or minimizes some property. It might involve aggregating some attributes over a set of objects, or setting the attributes of each object to reach a particular condition.”

Thus, most of tasks given earlier may be classified as follows:

- TA) to detect whether the object exists (or the operation is possible; the aim is attainable) (call them **alternative**);
- TC) to find the number of cases (**combinatory**);
- TO) to find the extreme value, the minimum number of steps (**optimization**);
- TB) to build the object.

All these goals may be called positive. We propose to build tasks with negative goals using environments of known types of tasks. We shall call such positive tasks basic (for reversing).

We propose the following pendants to the types of TA), TC), TO):

- TA-A) to detect whether the possible operation can be made impossible (is it possible to make the existing solution impossible in exactly N steps? If yes, then provide these steps); and the optimization tasks:

- TA-L) to find the minimum number of steps to make the possible operation impossible;
- TC-L) to find the minimum number of steps to make the number of cases less than a given number;
- TC-O) to make the number of cases the least possible by means of the given number of steps;
- TO-L) to find the minimum number of steps to make the extreme value worse than a given number (boundary);
- TO-O) to make the extreme value the worst possible by means of the given number of steps.

We are going to demonstrate that many well-known tasks can be transformed in such a way. We also consider general tasks (*G*-Tasks), i.e., such schemes that can generate concrete tasks. If the above-listed "negative" aim is already attained then the output is zero.

We classify tasks by the traditional subject fields: text processing, graphs, rectangular geometry and miscellaneous tasks. A general proposal on subject fields was presented in Verhoeff *et al.* (2006), surveys of actual subject fields were given in (2008), Verhoeff (2009).

We simplify the texts of tasks to the phrases beginning with: "Can ... be made ...", "Output ...", "How many ..." instead of the formal "Write a program finding/detecting ...". Restrictions on data also are not specified. They are to be put when the simplest (brute force) algorithm and effective ones for each task are built. Comments are given in square brackets. All numbers given in tasks are supposed to be integer and greater than zero.

2. Tasks on Text Processing

Task 1 (TA-L). (The Kyrgyzstan quarterfinal of the International Collegiate Programming Contest administered by the ACM, October 2010). Given a word W of 4..100 capital letters. How many, at least, letters must me erased from W in such a way that the word 'SU' cannot be obtained from the rest of W by means of further erasing letters?

We shall call such relation as "scattered embedding" or "S-embedding" of 'SU' into W .

Example 1. Input: UKRS. Output: 0.

Example 2. Input: BSSSSKKRRSSUU. Output: 2.

Solution

```
If (not(S in W)) or (not(U in W)) then Output 0 else
  {M := min{number of S in W; number of U in W};
   for all clearances C in W {M1 := (number of S in W left to C)
                             +(number of U in W right to C);
   M := min{M, M1}}; Output M}.
```

Remark. Let $N := \text{length}(W)$. If $M1$ is counted directly then the complexity is $O(N^2)$; if preceding values of numbers are used then the complexity is $O(N)$. This complexity can put restrictions of the length of the given word W .

A more general version can be defined as follows.

Task 2. (TA-L). Given words W (of length N) and $W1$ (of length M). How many, at least, letters must be erased from W in such a way that (*) $W1$ cannot be S -embedded into the rest of W ?

Under the additional assumption, that all letters in $W1$ are distinct (Task 2A), U. Degenbaev proposed the following

Solution (dynamical programming)

Denote the solution for words $W[1..i]$ and $W1[1..j]$ correspondingly as $D[i, j]$.

```

for all  $i$   $D[i, 0] := i$ ;
for all  $j > 0$   $D[0, j] = 0$ ;
for all  $i > 0$ ,  $j > 0$  { if not (letters  $W[i] = W1[j]$ ) then  $D[i, j] := D[i - 1, j]$ 
else  $D[i, j] := \min\{D[i - 1, j] + 1, D[i - 1, j - 1]\}$  };
Output  $D[N, M]$ .

```

Other types of tasks in the environment of Task 2 – Given words W (of length N) and $W1$ (of length M).

Task 3 (TA-A). Can the goal (*) be attained by erasing the given number K of letters from W ?

G-Task 4 (TC-L). How many, at least, letters must be erased from W in such a way that the number of S -embeddings of $W1$ becomes less than the given number K ?

G-Task 5 (TC-O). Find the minimum possible number of S -embeddings of $W1$ after a given number Q letters have been erased from W .

Certainly, these *G*-Tasks are too complicated for vast initial data. To derive interesting tasks from them, additional conditions are to be put.

Task 6 (classical, as basic). Given the set S of words and the word W . Can W be composed of a subset of S under the condition A) without overlapping or B) with possible overlapping?

Other types of tasks in the environment of Task 6 – Given the set S of words and the word W :

Task 7 (TA-A). Can K words be removed from S to make such composing impossible?

Task 8 (TA-L). How many, at least, words must be removed from S to make such composing impossible?

If all words are made of a same (one) letter then we obtain:

Task 9 (classical, as basic). Given a set S of numbers and a number N . Can N be presented as the sum of a subset of S ?

Other types of tasks in the environment of Task 9 – Given the set S of numbers and the number N :

Task 10 (TA-A). Can K numbers be removed from S to make such presentation impossible?

Task 11 (TA-L). How many, at least, numbers be removed from S to make such presentation impossible?

3. Tasks on Graphs

We recall two classical tasks and give adjacent tasks due to the schemes of Section 1.

Task 12 (TA-L) (classical, negative in our terminology). Given a connected graph and two of its vertices (which are not subject for removal). How many, at least, A) arcs or B) vertices are to be removed to disconnect these two vertices?

Other types of tasks in the environment of Task 12:

Task 13 (TO-L) ... to make the distance between these vertices greater than a given number?

Task 14 (TO-O) ... to make the distance between these vertices as large as possible?

Task 15 (TC-L) ... to make the number of paths implementing the distance between these vertices less than a given number?

Task 16 (TA-L) (classical). How many, at least, A) arcs or B) vertices are to be removed to make a given connected graph non-connected?

Other types of tasks in the environment of Task 16:

Task 17 (TA-A). Can we remove the given number of A) arcs or B) vertices to make these vertices disconnected?

Task 18 (TO-L) ... to split a given graph into a given number of independent graphs?

G-Task 19 (TO-L) ... to make a) the diameter of the graph or b) the radius of the graph greater than a given number whilst preserving connectedness?

G-Task 20 (TO-O) ... to make a) the diameter of the graph or b) the radius of the graph as large as possible whilst preserving connectedness?

G-Task 21 (TC-L) ... to make the number of paths implementing a) the diameter of the graph or b) the radius of the graph less than the given number whilst preserving connectedness?

Demonstrate pendants to a certain task:

Task 22 (as basic), Pankov, 2008. Given a graph (its arcs are of length 1) and two of its neighbor vertices. The train has the length 1, its head is at the 1st vertex, and its tail is at the 2nd vertex. The train can move ahead only. Output the length of the shortest way to be passed by the train such that its head and tail would swap.

Types of pendant tasks in the environment of Task 22:

Task 23 (TA-L). How many, at least, A) arcs or B) vertices are to be removed to make such swap impossible?

Task 24 (TO-L) . . . to make the length of such a way greater than a given number?

Task 25 (TO-O) . . . to make the length of such a way as large as possible?

4. Tasks on Rectangular Geometry

We shall consider squared paper and call points with integer coordinates "integer points".

Demonstrate pendants to two tasks.

Task 26 (as basic). Given a number $N > 1$ and a list of integer points (obstacles) which are not within the square $Q: 0 \leq X \leq N, 0 \leq Y \leq N$. Can Q go away (far from all obstacles) by shifts parallel to coordinate axes?

Example. Input: $N = 5$, four integer points: $(0, 4); (5, 1); (5, 3); (6, 9)$. Output: It can.

Types of pendant tasks in the environment of Task 26:

Task 27 (TA-L). . . . How many, at least, integer points (additional obstacles) are to be added to make the going away of Q impossible?

The same Example. Output: 2 points.

Task 28 (TA-L) . . . How many, at least, integer points are to be added to make the number of shifts for the going away of Q greater than a given natural number K ?

The same Example; Also input: $K = 2$. Output: 3 points.

Task 29 (as basic), Pankov, 2008. The train stands on the segment $(0, N)(Head) - (0, 0)(Tail)$. The train can move (forward only) along edges of the rectangular grid not self-touching and cannot pass given integer points (obstacles) $(X_1, Y_1), (X_2, Y_2), \dots, (X_K, Y_K)$. Find the length of the shortest ways to be passed by the train in order to reach the state $(X_H, Y_H)(Head) - (X_T, Y_T)(Tail)$. Main restriction for these given points: $|X_H - X_T| + |Y_H - Y_T| \leq N$.

Types of pendant tasks in the environment of Task 29.

Task 30 (TA-L). How many, at least, obstacles (not on the initial segment) are to be added to make such motion impossible?

Task 31 (TO-L) . . . to make the length of such a way greater than a given number?

5. Miscellaneous Tasks

G-Task 32 (TA-L). Given a set U (a rectangle, a segment etc.) and a covering C of it. How many, at least, elements, must be removed from C to make it not a covering of U ?

Task 33 (as basic, Pankov, 2010). There are M known chemicals; the first N of them are present.

Some of the chemicals can be obtained from other ones (we will only consider reactions that cause two chemicals to become one). All reactions are given as four numbers $B1$, $B2$, A (all different natural numbers in $1..M$) and T (integer number denoting releasing heat, if $T > 0$, or required heat, if $T < 0$) indicating the A th chemical is obtained of $B1$ th and $B2$ th ones. Find the most profitable way to obtain the given $A0$ th chemical of the list $M - N + 1..M$ (if it is possible), i.e., such sequence of reactions $B1$, $B2$, A , T that:

- all A s are in $M - N + 1..M$ and different; the last A is $A0$;
- each $B1$, $B2$ are either in $1..N$ or of preceding A s;
- all A s except $A0$ are used in following reactions;
- the sum of all T s has the greatest possible value.

A pendant task in the environment of Task 33.

Task 34 (TA-A). How many, at least, A) present chemicals of $1..N$ or B) permitted reactions must be excluded to make obtaining the $A0$ th chemical impossible?

Task 35 (TA-L). Given a set of segments with natural lengths. How many, at least, segments must be excluded to make construction of a (non-degenerate) triangle impossible?

An example of “negative” task given earlier.

Idea of Task 36 “Training” (IOI’2007). Mirko and Slavko are training for the tandem cycling marathon in Croatia. They need to choose a route to train on. . . . There are N cities and M roads in their country. Exactly $N - 1$ of those roads are paved, while the rest of the roads are unpaved trails . . . Riding in the back seat is easier. Because of this, Mirko and Slavko change seats in every city. To ensure that they get the same amount of training, they must choose a route with an even number of roads.

Mirko and Slavko’s competitors decided to block some of the unpaved roads, making it impossible for them to find a training route satisfying the above requirements. For each unpaved road there is a cost (a positive integer) associated with blocking the road. It is impossible to block paved roads.

Find the smallest total cost needed to block the roads so that no training route exists . . .

Probably, among the vast scope of tasks given at numerous competitions in informatics there were ones which could be considered as “negative” in our terminology. We propose to develop such tasks systematically.

6. Conclusion

We hope that successful application of methods proposed above would yield new tasks with “short and elegant formulation” (Dagienė and Skūpienė, 2007), and being interesting to solve. This would enlarge the scope of tasks involved into national and international informatics olympiads.

References

- Burton, B.A., Hiron, M. (2008). Creating informatics olympiad tasks: exploring the black art. *Olympiads in Informatics: Tasks and Training*, 2, 16–36.
- Dagienė, V., Skūpienė, J. (2007). Contests in programming: quarter century of Lithuanian experience. *Olympiads in Informatics: Country Experiences and Developments*, 1, 37–49.
- Diks, K., Kubica, M., Radoszewski, J., Stencel, K. (2008). A proposal for a task preparation process. *Olympiads in Informatics: Tasks and Training*, 2, 64–74.
- Kelevedjiev E., Dzhenkova Z. (2008). Tasks and training the youngest beginners for informatics competitions. *Olympiads in Informatics: Tasks and Training*, 2, 75–89.
- Kemkes, G., Cormack, G., Munro, I., Vasiga, T. (2007). New task types at the Canadian computing competition. *Olympiads in Informatics: Country Experiences and Developments*, 1, 79–89.
- Pankov, P.S. (2008). Naturalness in tasks for olympiads in informatics. *Olympiads in Informatics: Tasks and Training*, 2, 115–121.
- Pankov, P.S., Baryshnikov, K.A. (2009). Representational means for tasks in informatics. *Olympiads in Informatics*, Selected Papers of the International Conference Joint with the XXI Olympiad in Informatics, Plovdiv, 3, 101–111.
- Pankov P.S. (2010). Real processes as sources for tasks in informatics. *Olympiads in Informatics*, Selected Papers of the International Conference Joint with the XXII Olympiad in Informatics, Waterloo, 4, 95–103.
- Verhoeff, T. (2009). 20 years of IOI competition tasks. *Olympiads in Informatics*, 3, 149–166.
- Verhoeff, T., Horvath, G., Diks, K., Cormack, G. (2006). A proposal for an IOI Syllabus. *Teaching Mathematics and Computer Science*, 4(1), 193–216.



P.S. Pankov (1950), doctor of physical-math. sciences, prof., corr. member of Kyrgyzstani National Academy of Sciences (KR NAS), is the chairman of jury of Bishkek City Olympiads since 1985, of Republican Olympiads since 1987, the leader of Kyrgyzstani teams at IOIs since 2002. Graduated from the Kyrgyz State University in 1969, is a main research worker of Institute of Theoretical and Applied Mathematics of KR NAS, a manager of chair of the International University of Kyrgyzstan.



K.A. Baryshnikov (1985), OJSC "Finance Credit Bank", Bishkek, Kyrgyzstan. Participated in IOI'2002, in training the Kyrgyzstani teams for IOI'2003 and IOI'2004. Graduated from the Kyrgyz-Russian Slavic University in 2007.

Beyond the Competitive Aspect of the IOI: It Is All about Caring for Talent

Tom VERHOEFF

*Department of Mathematics and Computer Science, Eindhoven University of Technology
Den Dolech 2, 5612 AZ Eindhoven, The Netherlands
e-mail: t.verhoeff@tue.nl*

Abstract. The IOI is not just an informatics competition, but a means to care for talent in informatics. Caring for talent involves a broad range of issues, including identification of talent and education adjusted to that talent. There is (almost?) no generally accessible literature focusing on informatics talent. To show what such literature could offer, we review several books that address talent in mathematics. These books also contain much that is directly applicable to talent in informatics.

Key words: international olympiad in informatics, international mathematical olympiad, developing talent, training, books.

1. Introduction

The statutes of the International Olympiad in Informatics (IOI website, 2011; Regulations) state in article S1.7:

“The main objectives to be accomplished by the IOI are:

- to discover, encourage, bring together, challenge, and give recognition to young people who are exceptionally talented in the field of informatics;
- [further objectives omitted ...]”

Similar phrases appear in the regulations of other olympiads. All olympiads have chosen the competition form as a means to achieve these ulterior goals, but they clearly do not have the competition as their first goal. Note that not all talented pupils are motivated by a competition (Verhoeff, 1997).

In this article, we address various aspects related to the goals of discovering, encouraging, challenging, and, in particular, developing talent. We do so by reviewing six books about discovering, developing, training mathematical talent. Our hope is that similar books will, one day, be published in the area of informatics, so that these can be reviewed. But even the math-oriented books offer good advice in general, and they can serve as examples worthy of imitation. Keep in mind that informatics resembles mathematics, because it is also a so-called *science of the artificial*, as opposed to an empirical science, like physics.

2. How to Care for Talent

It is one thing to say that you wish to discover, encourage, and challenge talent. How to do so is quite another thing. The latter is a recurrent concern of teachers, parents, school administrators, and policy makers. There are many prejudices, misunderstandings, and myths on how to care for talented pupils, even among those with (some) experience. And, as it turns out, that included the author.

There is not much literature on developing talent when compared to the amount of ‘regular’ teaching material. In this section, we will review three books that do address this topic (see Fig. 1).

On the side, we teach enrichment classes at a primary school, touching on topics in mathematics and informatics. While looking for some new material, we accidentally came across Assouline and Lupkowski-Shoplik (2005). This was before the second edition (Assouline and Lupkowski-Shoplik, 2011) came out, so the review here is for the older edition. However, the second edition has the same structure, with some rewrites to put it in a broader perspective and some interspersed updates.

Assouline and Lupkowski-Shoplik (2005) was bought with the hope of finding concrete material for enrichment classes. At first, it was disappointing not to find anything technical that could be used in classes. But after reading the book, its real purpose became clear, and we got hooked.

The book is written by two authors who both were introduced into the world of pupils with a talent for mathematics by Julian Stanley. Stanley acquired fame in this area in the 1970s, by doing research, publishing about it, establishing an institute devoted to math talent, and training a new generation of researchers. The authors now direct their own research institutes for math talent.

Soon, it was understood that we always had been somewhat naïve, because there has been considerably more research on talent than many parents and teachers realize. The first chapter treats a series of myths (in the second edition, renamed to “excuses”)

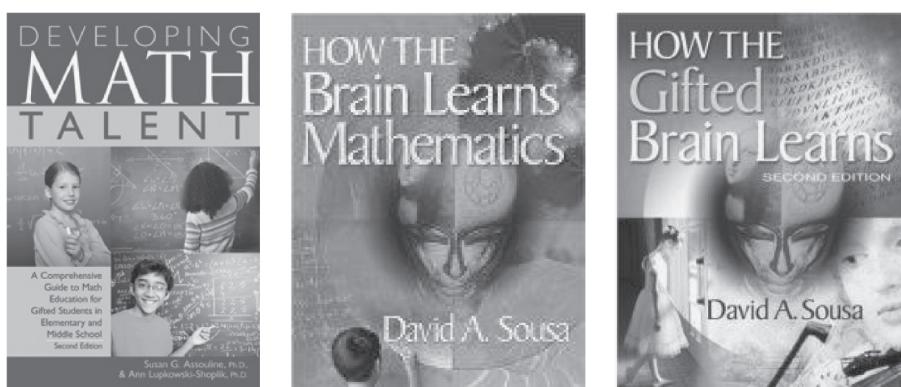


Fig. 1. Three books concerned with (mathematical) talent (Assouline and Lupkowski-Shoplik, 2011, Sousa, 2008, 2009).

and provides brief counterarguments, which are elaborated in subsequent chapters. Some myths are well known to mathematicians, such as “mathematically talented students are computation whizzes”, or “if mathematically able students study mathematics at an accelerated pace, they will run out of math curriculum before they reach high school” (the vastness of mathematics is not easy to explain to the layperson). But other myths are more subtle: “results from standardized, grade-level testing are sufficient for identifying mathematically talented students”, or “the best option for mathematically talented elementary school students is enrichment”, or “mathematically talented students cannot be identified until high school”.

There are a number of interesting things that you can learn from this book, for instance, that it is important to distinguish *good*, *talented*, and *exceptionally talented* pupils. This is especially important when fine-tuning their (math) education. Standardized grade-level tests are not appropriate, because all three groups will end up in the top 5% where they cannot really be distinguished. This is the saturation region or *ceiling* of the test. Research involving thousands of pupils, has provided evidence that so-called *above-level tests*, i.e., standardized tests designed for older pupils, can be used to pull these three groups apart.

To develop math talent it does not suffice to offer some more assignments of material already mastered (the book calls that *busywork*), nor extra topics in a general or cultural enrichment class, like philosophy or debating. It can be the case that pupils with talent for mathematics obtain only average scores in other subjects, and therefore they will not be identified as gifted and do not even qualify for a general enrichment class. Talent, be it in sports or music or math, needs to be developed in an educational program tuned to that talent. Furthermore, math talent is not really developed by offering some ad hoc enrichment material from the field of mathematics. It needs a systematic approach that *accelerates* a solid training in math.

Gifted pupils with a talent for math are ahead of the class in mathematics from the very start, and the gap only increases when their development is properly facilitated. Simply skipping a grade (even if only for math) is not a good solution, because also there the tempo is too low, and the dangers of boredom lurk. The authors also provide a more important reason to accelerate: “If a student is not permitted to accelerate, he or she may be ... denied the opportunity to study a more advanced subject later. ... [W]aiting to study a subject eliminates the chance to take new, more challenging courses.” Society needs talent to make progress.

The book by Assouline and Lupkowski-Shoplik (2005) has a systematic organization, larded with tables, lists, and examples. There is a chapter aimed at parents, who are the prime advocates for their children to get schools to offer an adjusted curriculum (also see Bosse and Rotigel (2006)). A full chapter is devoted to talent search, and one to organizing assessment and individual mentoring. For teachers there is a chapter on various options, either inside the regular classroom or outside (including mathematics competitions and clubs). The final chapter treats a couple of concrete cases of exceptional pupils and summarizes all insights. The book has a focus on the situation in the United States of America, with ample references to American literature and sources. However, most of

the advice is generally applicable. People involved in olympiads, in whatever role, often encounter talented students and are asked for advice. This book provides a solid basis.

Cognitive Psychology and Brain Research

Two books by Sousa (2008, 2009) caught our attention. They present recent research in neuroscience and the implications this is having for education. The first book is specifically about cognitive development and how this affects the learning of mathematics, from a very young age on. It turns out that the human brain is much more capable of dealing with abstract mathematical concepts than many parents and teachers believe. Devlin (2000) even provides evolutionary arguments for this.

In music and sports it seems to be accepted practice to offer special support for developing talent, but this is much less the case for academic talent, for no good reason. As in music and sports, it is worthwhile to start young with the development of talent in mathematics and informatics. We have the impression that a lot of talent is already lost and wasted before secondary education starts. The book provides many concrete aids to improve math education, from preschool up to adolescence.

The second book (Sousa, 2009) starts by exploring the phenomenon of gifted persons in a broad sense, comparing various definitions, emphasizing that giftedness is not equivalent to high IQ (intelligence test score), but also involves creativity, problem-solving ability, intense interest, and motivation. It cites experimental evidence that giftedness is not brought about by either nature (inherited genes) or nurture (environment and upbringing) alone, but by a suitable combination of nature *and* nurture. Other chapters address topics like how to challenge, how to deal with underachievement, and the twice-exceptional brain (gifted in one area, deficient in another). There are separate chapters on gifts for language, mathematics (though Assouline and Lupkowski-Shoplik (2011) goes deeper), and arts. One of the main theses of the book is debunking the myth that “[gifted] students can take care of themselves”. Or put differently: “Teachers at all grade levels have the responsibility to recognize and plan for the needs of the gifted. . . . The loss of such potential is a serious blow to society as well as to the student and teacher.”

3. How to Train Talent

The three books discussed in the preceding section did not offer specific advice or material on the technical side. In this section, we will review three books that arose from training initiatives for mathematical olympiads and other math contests. The book by Faires (2006) was conceived in the United States of America, that by Zawaira and Hitchcock (2009) in Zimbabwe, and Holton (2010) in New Zealand. To compare the books on their contents, it is necessary to know more about the mathematical olympiad.

The International Mathematical Olympiad (IMO) was established in 1959 with as primary aim “to discover, encourage and challenge mathematically gifted young people in all countries” (IMO 2011 website, 2011). (Note how the IOI objectives cited in the introduction resemble this wording.) To achieve this aim, the IMO offers a challenging

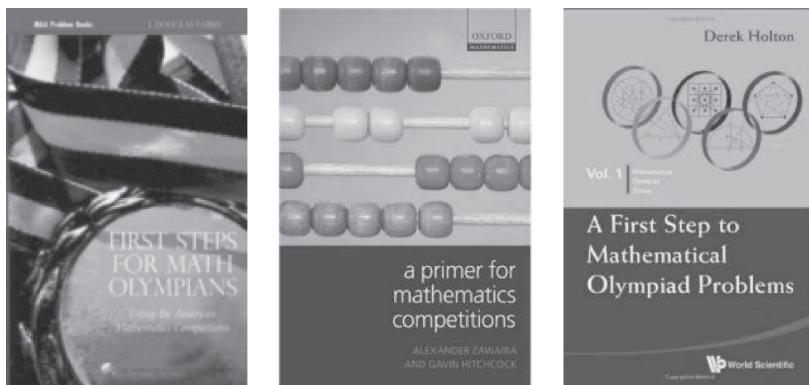


Fig. 2. Three books concerned with the training of mathematical talent (Faires, 2006; Zawaira and Hitchcock, 2009; Holton, 2010).

competition with math problems that require an investigative mentality. To ensure that the problems are accessible to participants from all over the world, they cover a limited range of topics, and the problems and solutions must be ‘elementary’ but ‘non-routine’. Traditionally these topics are geometry, number theory, algebra, and combinatorics, specifically without calculus and probability theory.

Over the years, the level of the competition has risen. Currently, the harder problems at the IMO are even challenging for professionals. In part, this increase happened because more effort is put into discovering talent, resulting in more competitors with raw talent. But also because more and more countries have a systematic training program to develop discovered talent.

Faires (2006) intends to cater for the top 10% to 15% of an average senior high school class, and not just for the elite. The material is divided into eighteen short chapters, which each treat a theme, such as polynomials and their zeroes, or the geometry of the circle. Some of the material covered is not so relevant for the IMO, but can be useful for other math contests; this includes probability theory, solid geometry, and complex numbers. Each chapter closes with examples and multiple-choice exercises. Complete solutions are provided in the back, together with references to original sources.

To give you an impression of the level that Faires (2006) strives for, it can be noted that geometry proceeds up to Ceva’s Theorem, the power of a point, and cyclic quadrangles; number theory up to prime factorization and modular arithmetic; combinatorics up to inclusion-exclusion and the pigeonhole principle (although the latter is presented in a chapter on number theory); and algebra has an underdog position. There is extensive attention for (special) functions, whereas function equations and inequalities are not treated. Still, it would be wonderful if the top 10% of a regular class would really perform at this level.

Zawaira and Hitchcock (2009) are more ambitious and have organized their material with the math olympiad in mind. They start with an overview of that olympiad and even pay attention to such questions as “what benefits does ... training bring” and “what careers are appropriate for people with mathematical aptitude”. There are eight hefty

chapters built around grand themes. Each chapter starts with a few problems as appetizers; the solutions appear only at the end of the chapter. Motivation, theorems (mostly with complete proofs) and examples are intertwined. Separate sections with problems and solutions close a chapter.

The chapter themes of Zawaira and Hitchcock (2009) are: geometry (including theorems of Menelaus en Ptolomeus), algebraic inequalities and induction (Cauchy–Schwarz, AM–GM), Diophantine equations, number theory (including little Fermat, Wilson, and Chinese remainder), trigonometry, sequences and series, Newton’s binomial theorem, and combinatorics (including partitions and derangements). The ninth chapter consists of miscellaneous problems, where you have no clue in what direction to start looking for a solution. As with Faires (2006), also here, many of the problems are multiple choice, which is somewhat counter to IMO tradition, but the intention is that you prove your answers.

Holton is an experienced author on mathematical problem solving. Holton (2010) is Volume 1 in the new *Mathematical Olympiad Series*, which features attractive titles of various authors, mostly on specific subareas. Although nowhere mentioned, Holton composed this book from booklets Number 1 through 8 of his *Problem Solving Series*, without major changes. This older series was published by the *The Mathematical Association* (of New Zealand) in 1988–90. It can be expected that booklets Number 9 through 15 will also be combined and republished in the future. At the time (over twenty years ago), those booklets were aimed at training and selection of the national team for the IMO. Nowadays, the IMO level is much higher and the book is not longer suitable for that purpose. Holton makes it clear that he considers problem solving the essence of all mathematics, and he emphasizes the need for mathematical creativity.

In the first chapter, Holton illustrates several general problem-solving techniques using some accessible problems. The subsequent chapters treat combinatorics, graph theory, number theory (little Fermat, but no Chinese remainder), geometry (two chapters, but no Ceva nor cyclic quadrilaterals), and proof techniques. The exposition is aimed at beginners and not at advanced students. Except for graph theory, Holton (2010) presents less theory than Faires (2006). The final chapter offers several IMO problems and accompanying background information.

A Comparison

The three training books have considerable overlap in content and purpose, but the approaches are rather different. Faires (2006) is clearly aimed at breadth and a lower level, and it comes across as dry. Also Holton (2010) keeps it simpler, especially on theory, but the presentation style is much more appealing, especially for self-study. All three books contain an index, but only Zawaira and Hitchcock (2009) refer to other books, journals, and websites. The typography of Faires (2006) and Holton (2010) is beyond reproach, whereas that of Zawaira and Hitchcock (2009) looks amateurish.

It is remarkable that only Holton (2010) pays attention to the *meta-cognitive level*, like Pólya in his famous *How to Solve It*. None of these books explicitly refers to Pólya,

not even Holton, who presents his own problem-solving guidelines. We would think that talented students would especially be interested in developing an understanding of the problem-solving process.

Good training establishes a solid foundation of basic knowledge, develops experience with many and diverse problems, and compares different techniques to solve each problem. These books supply useful ingredients for such training. With Faires (2006) and Holton (2010), at most one or two of the six IMO problems come in range, whereas with Zawaira and Hitchcock (2009), maybe a third or even fourth IMO problem might become accessible. Of course, contestants still need to practice, practice, practice.

4. Conclusion

In this article, we have drawn your attention to the broad set of issues surrounding talent and how to care for it. Our aim has not been to treat this topic in any depth, but to point you to informative literature.

Since discovery, encouragement, and challenging of talent are among the main objectives of the IOI, it would be good if more literature on talent in informatics would emerge. For lack of such literature, we have reviewed six books concerning talent in mathematics, a mature and neighboring discipline. Because informatics is so close to mathematics in many respects, the lessons and insights set forth in these books are also of direct use to persons, such as in the IOI community, who encounter and work with pupils talented in informatics.

References

- Assouline, S., Lupkowski-Shoplik, A. (2005). *Developing Math Talent: A Guide for Educating Gifted and Advanced Learners in Math*. Prufrock Press.
 Quotes on www.win.tue.nl/~wstomv/quotes/developing-math-talent.html.
- Assouline, S., Lupkowski-Shoplik, A. (2011). *Developing Math Talent: A Comprehensive Guide to Math Education for Gifted Students in Elementary and Middle School*, 2nd edn. Prufrock Press.
- Bosse, M.J., Rotigel, J.V. (2006). *Encouraging Your Child's Math Talent: The Involved Parents' Guide*. Prufrock Press.
- Devlin, K. (2000). *The Math Gene: How Mathematical Thinking Evolved and Why Numbers Are Like Gossip*. Basic Books.
- Faires, J.D. (2006). *First Steps For Math Olympians*. MAA.
- Holton, D. (2010). *A First Step to Mathematical Olympiad Problems*. World Scientific.
International Mathematical Olympiad 2011 Website.
<http://www.imo2011.nl/> (accessed March 2011).
- International Olympiad in Informatics Website*.
<http://www.IOInformatics.org/> (accessed March 2011).
- Sousa, D.A. (2008). *How the Brain Learns Mathematics*. Corwin Press.
- Sousa, D.A. (2009). *How the Gifted Brain Learns*, 2nd edn. Corwin Press.
- Verhoeff, T. (1997). The role of competitions in education. In: *Future World: Educating for the 21st Century*. A conference and exhibition at IOI'97.
olympiads.win.tue.nl/ioi/ioi97/ffutwrld/competit.pdf (accessed March 2011).
- Zawaira, A., Hitchcock, G. (2009). *A Primer for Mathematics Competitions*. Oxford University Press.



Tom Verhoeff is assistant professor in computer science at Eindhoven University of Technology, where he works in the Group Software Engineering & Technology. His research interests are support tools for verified software development and model driven engineering. He received the IOI Distinguished Service Award at IOI 2007 in Zagreb, Croatia, in particular for his role in setting up and maintaining a web archive of IOI-related material and facilities for communication in the IOI community, and in establishing, developing, chairing, and contributing to the IOI Scientific Committee from 1999 until 2007.

REPORTS

Mexican Olympiad in Informatics

Arturo CEPEDA¹, Margarita GARCIA²

¹*Comite Mexicano de Informatica AC*

*Hacienda de Coaxamalucan 145, Col. Hda. de Echegaray
Naucalpan, Estado de Mexico, Mexico CP 53300*

²*Comite Mexicano de Informatica AC*

Circuito Medicos 30-1, Cd. Satelite Naucalpan, Estado de Mexico, Mexico CP 53100

e-mail: acepeda@auronix.com, mgarcia@auronix.com, www.olimpiadadeinformatica.org.mx

The Mexican Olympiad in Informatics (OMI) is a National Competition that every year the “Comite Mexicano de Informatica A.C. (COMI)” does in order to select the best young programmers in Mexico below 19 years old.

Mexico (whose official name is “United States of Mexico”) is a country composed of 32 states, with 112 Million Inhabitants and close to 2 Million square Kilometers in territory. We have been doing the OMI every year since 1996, the method that we use is as follow:

1. National Invitation to participate.
2. Exams via Internet.
3. 32 State Olympiads, one in each State.
4. Each State sends the best 4 to the National Olympiad OMI.
5. From the OMI we select the best 32.
6. The 32 Pre-Selected are in training over one year, via Internet and 4 training camps of 10 days each; from them we get the 4 Contestants that represent Mexico at the IOI.

This process is repeated every year and we have to quote that:

During 2010 we registered 16,521 students from all over the country and during the first exams via Internet we reduce the amount to 2081 students, then we sent the best 60 of each State to the delegate in that State. They bring all the students to some place in that State, train them and do the State or Regional Olympiad in order to get their four best Students in order to represent the State in the OMI.

We ask the competitors to be 19 years old or below and that they have at least one more year to finish their High School, this is in order to comply with the IOI requests after the one year they will be in training after they get in the Pre-Selected team.

Table 1
General statistics OMI 2010 Mérida, Yucatán

Concept	Quantity	Percentage
States participating	28	87%
States not participating	4	13%
Students that could compete	8,500,000	
Registered competitors	16,521	0.19%
Contestants at the OMI-2010	106	100%
Women contestants	20	19%
Men contestants	86	81%
row 12		
Gold medals	9	17%
Silver medals	18	33%
Bronce medals	27	50%
All medals	54	100%
Women medalists	5	9%
Men medalists	49	91%
Government school medalists	38	70%
Private school medalists	16	30%
Government school contestants	84	79%
Private school contestants	22	21%
High school contestant medals	53	98%
Secondary school medals	1	2%
Primary school contestant medals	0	0%
Average contestant per state	3.78	
Perfect score per contestant	800	100%
First place score	676	84.5%
Average points per medalist	468	58%
Average points per contestant	260	33%
Zero points contestants	4	4%
40 perfect score per state 800×4	3200	100%
Best state score	2303	72%
Minimum score per state	10	1.25%
Average age of contestants	16.6	years

The 32 students Pre-Selected in 2010 are still in training and we will have our National Selection Team in order to compete in Thailand IOI 2011, at the end of May this year.

In this report we present the results obtained during the OMI 2010, celebrated in September of that year in the City of Merida, State of Yucatan; the same place in which we had the IOI 2006. We also include some statistical data, which will help to understand the process we carry on in the OMI each year.



A. Cepeda has a communications and electronics engineer degree (1967) from the Escuela Superior de Ingenieria Mecanica y Electrica ESIME, National Politechnic Institute in México City. He also has a bachelor degree in physics and mathematics (1969) from the same institution and a MSc in solid state physics (1971). He is the president of the Mexican Olympic Committee in Charge of Organizing the Mexican Olympiad in Informatics every year. He is also the president of the International Olympiad in Informatics IOI (2008–2011). He is the planning director of Auronix SA de CV, group of companies developing software for automatic processes and telecommunications in Mexico.



M. Garcia has a communications and electronics engineer degree (1971) from the Escuela Superior de Ingenieria Mecanica y Electrica ESIME, National Politechnic Institute in México City. She is in charge of the general secretary at the Mexican Olympic Committee. She is also the administrative director of Auronix SA de CV, group of companies developing software for automatic processes and telecommunications in Mexico.

Belgian Olympiads in Informatics: The Story of Launching a National Contest

Sébastien COMBÉFIS, Damien LEROY

Department of Computer Science Engineering, Université catholique de Louvain

Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium

e-mail: {sebastien.combefis, damien.leroy}@uclouvain.be

Abstract. This paper describes the story of a new national contest through the experience of Belgium where the first olympiads in informatics were launched in 2010. Belgian Olympiads in Informatics is a multi-stage algorithmic, programming and logic contest composed of both pen-and-paper and on-computer tasks. The great focus on pen-and-paper tasks is a peculiarity of this contest. The context in Belgium is not the most favourable: programming courses at schools are quite rare, there are several official languages but none spoken by the entire population and the government does not give much means for helping organizing such a contest. This paper states how the contest was launched, explains the motivations behind the structure of the contest, the kind of tasks, the national delegation selection process and the training of the contestants.

Key words: olympiad, programming contest, be-OI.

1. Introduction

In 2009, there was almost twenty years without any participation of Belgium at the IOI. It was time to have a Belgian delegation again. So, in 2010, a small team lead by the two authors launched a national contest: the Belgian Olympiads in Informatics (be-OI). From the beginning, we focused on a double opportunity: having Belgium participating in the IOI and promoting informatics in secondary schools. In order to get more people involved, and especially higher education people, and to make the event bigger, we chose to organize two contests in parallel: the first one for secondary schools and the second one for first year higher education schools. That also gave an opportunity for secondary school contestants and teachers to meet students which chose to study informatics. In this paper, we will only focus on the contest for secondary schools.

Belgium is probably not the easiest place to organize such a contest. First, there are no, or very few, algorithmic and programming courses at secondary schools. So, we cannot only focus on good programmers but we should also detect talented people, which is done through logic and simple tricky algorithm problem sets. The idea was so to have a selection based on pen-an-paper tasks, in opposition to what is done in many other countries. Secondly, official and financial supports are really missing. Although most people from education institutions and from authorities agree to support the initiative, many of them do not bring more than moral support. Finally, the Belgian political context

is not easy to deal with. The country has three official languages, each of those being associated with a community having its own government with education as a competence. Unlike the Mathematical Olympiad, we preferred to organize a single contest for the whole country. That requires more complicated synchronization and translation tasks. It also makes searching for public funding more difficult as the educational authorities are different for each community.

This paper explains how we dealt with those difficulties to organize a national contest in 2010 and 2011 to pick out the best candidates for the IOI. The remainder of the paper is organized as follows: Section 2 describes the structure of the contest and the tasks that are given to the contestants. Section 3 presents some statistics about the two editions of the be-OI. Finally, Section 4 states perspectives about the future of the contest.

2. The Contest

The Belgian Olympiads in Informatics (be-OI) was initiated in 2009 by the two authors of this paper. The contest was actually first organized as a new activity of the UCLouvain ACM Student Chapter. The initiative has quickly been supported by some universities and higher education institutions which helped in advertising the event and organizing the semifinal in the French part of the country. The Dutch part of the country joined us this year through the Antwerp ACM Student Chapter, and several universities offer some human resources to write and review questions for the national contest. In the few next months, a specific targeted association will be created to organize the be-OI.

The remaining of the section describes the steps within the yearly contest, the kind of tasks that are given and how they are graded, and finally the training that is given.

2.1. Stages of the Contest

The Belgian Olympiads in Informatics is a multi-stages competition (Fig. 1). The first step consists of semifinals, organized the same day in several regional centres throughout Belgium. This year, it took place in mid-February in eleven centres. That first stage consists in a 3-hour pen-and-paper exam during which the contestants have to solve logic

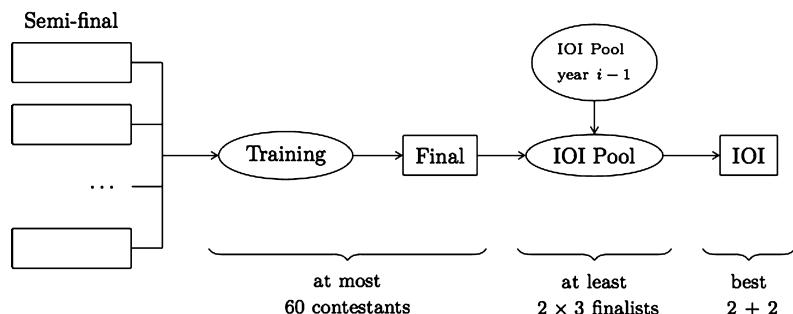


Fig. 1. Structure of the be-OI contest.

and algorithmic problems. At most 60 of these contestants are selected for the final. That exam is an exclusively pen-and-paper test as we think that it is the best way to precisely identify talented contestants.

A few weeks before the final, a training is organized in Dutch and in French. The training is mainly organized for the finalists, but it is in fact open to any contestant, even those who have not been selected for the final. That training aims at introducing some algorithmic and programming concepts.

The final lasts an afternoon and is composed of two parts: a pen-and-paper and a programming (i.e., on computer) one. The former is similar to the semifinal but is shorter. The latter consists in typical programming tasks as the ones given at IOI. The first task is a very simple one and the second one is much more difficult with a smooth grading between zero and the maximum score. A few days later, a public ceremony is organized to award prizes.

The contestants with the higher potential, i.e., with the highest scores or younger promising ones, are selected to enter into the *IOI pool*. In fact, they join one of the two pools according to their mother tongue. Members of these pools are invited to train on IOI-style questions. After one last week of training, the best four (two from each pool) are selected based on their competence and motivation to attend the IOI.

2.2. Tasks

As explained earlier, there are two kinds of tasks in our competition. The first ones are pen-and-paper tasks and the second ones are tasks to be solved by directly programming on a computer. The tasks are found via a call among the members of the national committee or from existing problems. The writing and reviewing of these tasks is aimed to be spread among the committee. A particular attention is paid to the balance between the different categories of questions.

There are three main categories of pen-and-paper tasks: multiple-choice questions, algorithms to fill in, and short algorithms to write down. The contestants may answer using pseudo-code, diagrams, natural language or any of the authorized languages (Java, C, C++, Pascal, Python and PHP).

Two pen-and-paper tasks are shown on Figs. 2 and 3. In the task *Numeric anagrams* (Fig. 2), the contestants were asked to find an algorithm to check whether two positive integers, given as arrays of digits, are anagrams or not. To help them, they were provided with a sorting algorithm (selection sort) which they may use. The naive solution is quite trivial but rather inefficient ($O(n^2)$). To obtain an extra score, we asked them to give an efficient algorithm in $O(n)$. In this task, they had to write the whole algorithm. So, it is a rather long task in comparison with other ones.

The task *Subarray of maximal sum* (Fig. 3) consists of, given an array *tab*, computing the sum of the elements of a subarray, the latter being chosen so that the sum is maximal. Contestants were provided with a naive $O(n^2)$ algorithm and were asked to fill the partial algorithm given in order to get an $O(n)$ algorithm. This is a typical task, where contestants have to first understand the partial solution given before adding the few instructions requested.

Numeric anagrams

In this problem, we consider that two numbers are *anagrams* if they are composed exactly of the same digits, in the same order or in a different one. For example, 121, 211, 112 and 121 are anagrams but 411, 144 and 511 are not.

You have to write an algorithm which establishes whether two positive integers, a and b , represented as arrays of digits, are anagrams. You can define new functions as long as you only write out inside the dedicated area. You can also declare new integer arrays, whose elements are initialized to 0, using the notation $newtab \leftarrow new_array(size)$.

Input: - a, b , two arrays with the same length n , such that each cell contains one digit

Output: - **true** if a and b are anagrams and **false** otherwise
- arrays a and b may have been modified

Bonus

You can get a bonus if your algorithm is **efficient**. For that, you have to propose an algorithm for which the number of times it accesses to the elements of the array (i.e., the number of $tab[index]$) is smaller than $10n$ for n strictly greater to 10. If you use the given sort function, array accesses of this function have to be taken into account.

Fig. 2. Pen-and-paper task from the be-OI semifinal 2011 (translated from French).

Even though those two examples include complexity through some efficiency requirements, few tasks require the contestants to know something about complexity. No more than 25% of the tasks involve complexity notions and no knowledge of complexity theory is required to find the best answer.

In addition to these questions, we have added in 2011 small logical games which aim at encouraging algorithmic thinking and opening the contest to a wider public just as proposed in the Australian Informatics Competition (Burton, 2010).

There are two tasks to be solved on a machine for the final. For these, the contestants have the choice between six programming languages: Java, C, C++, Pascal, Python and PHP. The reason why we accept more languages than at IOI is that we want to make the contest accessible to the larger number of people. We also think that a good Python or Java programmer can quickly switch to C++ if he is selected.

The description for programming tasks follows the classical structure: context, task description, constraints, inputs, outputs and scoring information. The *Fence problem* (Fig. 4), the task given at be-OI 2010, is actually an instance of the *convex-hull problem*. In addition to what is given on Fig. 4, the contestant received an example of input and output files together with information about the grading. The score depended on the length of the produced fence, with 80% of the score given for datasets with less than 1.000 trees and 20% for datasets with more than 1.000 trees. The maximum execution time was 10 seconds.

The constraints that are to be taken into account for the computer tasks is that there must always be a trivial not so bad solution that every contestant should be able to solve. In the “Fence problem”, the trivial solution is a rectangular fence around the trees. The contestants should then be able to improve their solution gradually. There are two ways of improving one’s solution: providing a better result, i.e., lowering the length of the fence in our example, or improving performances, i.e., compute the fence faster. However, small steps in the performance should not influence the score as we do not want one program-

Subarray of maximal sum

The following algorithm takes as input a non-empty integers array *tab* and computes the maximal sum that it is possible to get while taking a non-empty subarray of *tab* and summing its elements. For example, let *tab* = [1, -2, 4]. There are six possible subarrays which are [1], [-2], [4], [1, -2], [-2, 4] and [1, -2, 4]. The one with the maximal sum is the third one ([4]) whose sum is 4.

Input: - *tab*, a non-empty integer array whose length is *n*
Output: - the sum of the elements of the non-empty subarray with maximal sum

```
max ← tab[i]
for (i ← 0 to n - 1 step + 1)
{
    sum ← 0
    for (j ← i to n - 1 step +1 )
    {
        sum ← sum + tab[j]
        if (sum > max)
        {
            max ← sum
        }
    }
}
return max
```

The proposed algorithm is not efficient. For an array of length *n*, the execution time is indeed proportional to n^2 . The *tab* array is traversed too many times. It is possible to write an algorithm much more efficient so that the execution time is proportional to *n* rather than to n^2 . You are asked to fill up the following algorithm which read the array only once.

```
i ← 1
s ← tab[0]
max ← tab[0]
while (i != n)
{
    [...]
}
return max
```

Fig. 3. Pen-and-paper task from the be-OI final 2011 (translated from French).

ming language to be better than other ones. The grading steps are estimated so as to differentiate the main classes of solutions, e.g., $O(n)$ versus $O(n^2)$ versus $O(2^n)$. An improvement of the solution is anyway preferred to better performances.

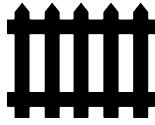
2.3. Evaluation

It is quite a difficult task to select which contestants should be part of the Belgian IOI delegation. The goal of the semifinal is to identify candidates who have the capacity to reason on algorithms and to solve problems. They do not need to know a programming language or to be able to program. The target is thus typically people that are good at mathematics and sciences at school.

The final is used to identify the best of these candidates. The contestants must have the ability to understand and write algorithms as well as to program them on computer. The

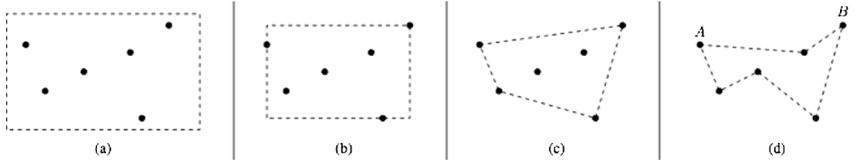
Fence problem

You have recently inherited a large orchard with a certain number of trees. To prevent malicious people from entering into your property, you want to put a fence around the orchard. Nevertheless, this kind of equipment is quite expensive and you should absolutely minimize the length of fence you will buy.



The fence consists of posts along which the fence will be tightened. Your friend can give you for free as many posts as you like. You just have to choose how to place your fence so as to minimize its length and to ensure that all the trees are inside. Another constraint is that, to ease the moving inside the orchard, the fence must be convex. That means that you must also be able to travel between any two points inside the orchard without leaving and having to climb the fence.

As illustrated by the following figure showing an orchard of six trees, there are several possible solutions.



Solutions (a), (b) and (c) are all three acceptable and are from the least to the most optimal. The third one being better than the first one, it will give you more points. The last solution (d) is not correct because the formed polygon is not convex. For example, when you want to travel from A to B in a straight line, you will have to quit the orchard.

Task

Write a program that, given Cartesian coordinates (positive integers), of the N trees, computes the coordinates of the posts to be placed to get a fence which contains all the trees and forms a convex polygon. To get the maximum score, you should minimize the length of the fence.

Limits and constraints

Your program only needs to manage problems within the following constraints. All the tests will remain in these limits.

- $3 \leq N \leq 100.000$.
- $0 \leq X, Y \leq 20.000$, the coordinates of the trees and posts.

In the datasets used by the judges:

- Trees will not be colinear, i.e., they are not all on the same line.
- There are no two trees on the same coordinate.

In the result produced by your program:

- Posts can be placed at the same coordinate as a tree, but you cannot place several posts at the same place.
- The produced fence must be convex.
- Cartesian coordinates of posts must be given in the order we have to link them so that the fence so formed is to be covered in counterclockwise order.

Input

The input file is built as follow:

- The first line contains a single positive integer N : the number of trees.
- The N next lines contains the cartesian coordinates of the trees, given as two positive integers separated with a single space.

The file ends with a new line.

Output

The output file to produce defines the coordinates of the posts to be placed. Each coordinate is defined on a single line in the file, as two positive integers separated with a single space. Coordinates must be given in counterclockwise way. The file must end with a new line.

Fig. 4. Computer task from the be-OI final 2010 (translated from French).

two-day training is an opportunity for them to learn some important algorithm concept and a programming language if they have never programmed before.

The most critical part is the choice of the four contestants for the IOI delegation. For the first edition, in 2010, the best four contestants at the final were chosen to be part of the delegation. After observing other countries at IOI 2010, we understood that we have to create a pool of candidates among which we select the best and more motivated ones. So, from 2011 on, we created a specific pool for IOI. After one week of intensive training, an IOI-like test is organized and the delegation team is created.

2.4. Training

As previously mentioned, several trainings are organized throughout the course of the contest. The first training mainly aims at making the contestants discover algorithmic notions and a programming language. In practice, over two days, we introduce them to time complexity, decomposition into sub-problems and recursion and make small pen-and-paper and on-computer exercises. Some of the finalists just do not know any programming languages. That is just a consequence of the way we choose to organize the semifinals. During that first training, we thus teach them a programming language and the basis of programming. We choose Python as it is easy for them to learn it quickly and to be able to use it directly (Georgatos, 2002). The main goal of this training is thus educational and aims at promoting computer science.

The second training is organized for the IOI pool. The goal of this one-week training is to train the finalists who are in the pool, teaching them classic algorithms (Skiena, 2008). A side goal of the training is also to help us identify the contestants that will be chosen to form the Belgian IOI delegation.

3. Some Statistics

This section gives some statistics about the first edition and about the first stages of the second edition as the 2011 final is not yet completed while this paper is being written. There is a total of about 500,000 young between 14 and 18 years old in Belgium. Among that huge amount, only about one hundred took part to the contest. Attracting people is a very difficult task. Table 1 summarizes the number of semifinalists and finalists to the be-OI.

Table 1
Contestants registered at the be-OI

Year	Semifinalists	Finalists
2010	83	43
2011	105	49

The exact number of finalist was chosen by the committee according to the score they had. Figure 5 shows the scores of each semifinalist for be-OI 2011, ordered by decreasing values. The vertical line separates the selected from the non-selected contestants and the horizontal line is the mean value.

It is interesting to observe that the distribution of scores is uniform when the distribution for such tests is usually a Gaussian. That observation was also observed for the be-OI 2010 edition. Another interesting observation is that among the contestants, the older ones are not necessarily better than the younger ones since there are few programming and algorithmic courses at secondary school. For the be-OI 2010, we only targeted 5th and 6th years students (16-18 years) but we opened the contest to any secondary school student for the be-OI 2011 while keeping a single task set. As shown on Fig. 6 (right), we can see that the percentage of selected contestants is good even for fourth year contestants (15-16 years old). That observation reinforced our intuition that having an IOI pool with younger students may be a good idea.

A final interesting observation we can do is about the programming language that the contestants used for the machine-task during the be-OI 2010 final. There were 11 contestants using Python, 10 using PHP, 5 using C++, 3 using Java, 3 using C, 2 using Pascal and 1 using Ruby (which was allowed in 2010 but not anymore in 2011). We notice clearly that the majority used Python which was the language they learned during the training and the second most used language was PHP which is quite popular among young people creating websites.

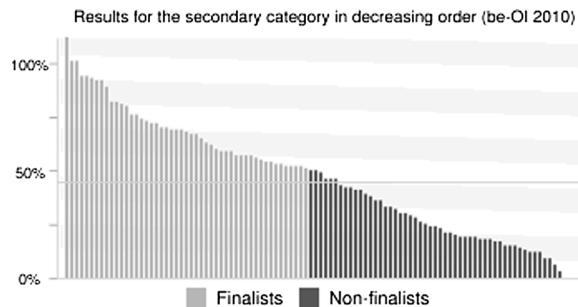


Fig. 5. Histogram with the scores of the semifinalists (be-OI 2011).

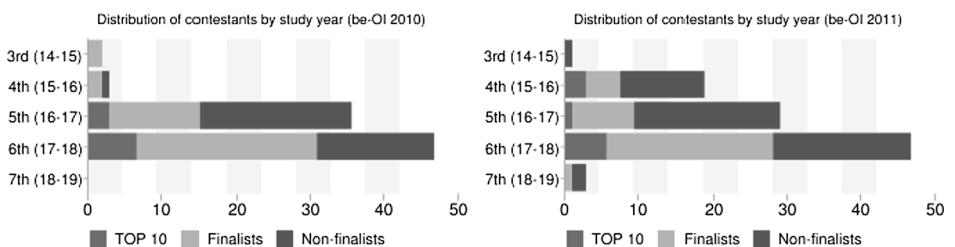


Fig. 6. Scores of the semifinalists grouped by study year. Between parenthesis are the corresponding ages.

4. Development of the Contest

The be-OI has only existed for two years and so needs to be improved on many points. The first change that is currently developed and will be deployed for the be-OI 2011 final is an online submission and testing system. That system will allow the contestants to directly submit their code for the on-computer task and to get their score immediately. The system is based on uevalrun (<http://code.google.com/p/pts-mini-gpl/wiki/uevalrun>). The advantage of such a system is that it could be used by the coaches to provide the Belgian IOI delegation with exercises that they can do to train themselves for the IOI.

Some additional work should also be done for advertising the contest and to make secondary students and teachers more informed and implied. An idea to do better advertising is to identify more small games and distribute flyers with these games to secondary school students. Those games should be appealing and give rise to some interest for the contest.

Finally, the last important thing that we still need to work hard to get more financial and more human resource for the be-OI project. The new association that will be created soon should help us in completing this goal.

References

- Burton, B. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.
Georgatos, F. (2002). How applicable is Python as first computer language for teaching programming in a pre-university educational environment, from a teacher's point of view? *Master Thesis*, AMSTEL Institute, Faculty of Science, Universiteit of Amsterdam.
Skiena, S. (2008). *The Algorithm Design Manual*, 2nd edn, Springer.



S. Combéfis is a PhD student at the Université catholique de Louvain in Belgium and works as a teaching assistant for the computer science Engineering Department. He is also following an advanced master in pedagogy in higher education. In 2010, he founded, with Damien Leroy, the Belgian Olympiads in Informatics (be-OI). He is now part of the coordinating committee that is in charge of managing everything which is related to the national contest. He is also trainer for the Belgian delegation to the IOI.



D. Leroy is currently a postdoctoral research assistant at Université catholique de Louvain (B). He obtained his PhD in computer science engineering in 2011 for his researches on computer and network security. He is one of the main initiators of the Belgian Olympiads of Informatics and was the Belgian delegation leader in 2010. In 2011, he is still coordinating both the national and international competition for Belgium.

Italian Olympiad in Informatics: 10 Years of the Selection and Education Process

Mario ITALIANI

*Dipartimento di Scienze dell'Informazione Università di Milano
Via Comelico 31, 20131 Milano, Italy
e-mail: mitaliani@alice.it*

Abstract. This article describes the selection and training process of the Italian Committee for the International Olympiad in Informatics to identify and prepare the Italian team members participating in the international olympiad. Special attention is dedicated to the first step in the selection process, which involves schools from all over the country. Examples of problems submitted to the contestants in this first selection phase are also provided.

Key words: olympiads, programming contest, selection, training.

1. Introduction

In the year 2000 Italy started to participate in the International Olympiad in Informatics, following an agreement between the Italian Ministry of Education, University and Research (MIUR) and AICA, the most important association of Italian computer scientists, which represents teachers and researchers of Italian Secondary Schools and Universities, as well as professionals working at institutions or companies using or providing information and communication technology.

The agreement led to the establishment of a “National Olympic Committee”, made up of members appointed by MIUR and AICA on a joint basis. The costs for the preparation and participation of Italian contestants in the olympiad are borne by MIUR and AICA on an equal basis.

The Committee relies on a group of researchers and teachers, the so-called “national trainers”, who take care of a complex training and selection process which lasts two years, due to the fact that Italian secondary schools do not give the necessary importance to computer science. It is therefore necessary that students showing special talent in the preliminary selection phases are provided with a solid preparation in programming, particularly in the algorithmic area.

2. The Selection and Training Process

2.1. *The First Step: The School-Based Selection*

The first step in the selection and training process for the annual international competition is the “school-based selection”, which is carried two years before the international olympiad, in the months of November/December, at the schools which have applied. The school selelection usually involves 500 schools for a total number of approximately ten thousand students.

The school selection takes place on the same day at all participating schools, which receive the text of the competition by telematic means shortly before the begin of the test. School personnel subsequently collects and marks the answers given by the students in ninety minutes, and then transmits (again by telematic means) the ranking to the Olympic Committee.

During the test, the students are required to answer 20 questions (mainly math and logic questions or algorithmic ones) and are allowed to use only paper and pencil.

Math and logic questions aim to assess the students’ capacity to rapidly make logical connections and use simple mathematical instruments, while the objective of algorithmic questions is to evaluate the student’s capacity to perform a simple succession of operations to solve elementary problems.

These two sets of tasks are supplemented by further questions to determine whether the student possesses programming fundamentals. These last questions are not particularly challenging and take into account the inadequate preparation of Italian schools in programming, as talented students should not be prevented from participating to subsequent phases in the selection and training process, which could give them the opportunity to fill their gaps in programming.

Some examples of the questions submitted in the latest school selection are provided in the Appendix.

2.2. *The Second Step: The Territorial Selection*

Based on the ranking provided by each school at the end of the first selection phase and on some rules aiming to guarantee, where possible, the presence of at least one student from each school, the Olympic Committee drafts a national ranking and selects approximately one thousand students that are admitted to the second step in the preparation and selection process, which is the so-called territorial selection, which takes place at various schools all over the country (at least one for each Region), chosen according to the availability of appropriate IT equipment and of competent personnel.

The territorial selection is carried out in the month of April of the year preceding the international competition.

The test requires participants to design algorithms and realize the related programs for the solution of problems whose complexity is not high and anyway substantially lower than that of problems submitted in the international olympiad.

The programs realized by participating students by using Pascal or C/C++ language in Linux or Windows environment are submitted to a centralized server that automatically compiles them and submits them to test data. Territorial rankings are therefore drafted, as well as a national one to identify about eighty students who are admitted to the following selection phase.

In the period between the school-based and the territorial selection, participating schools are invited to organize training activities for students who have succeeded in the first selection phase, by involving in-house and external teachers, and relying on the teaching material available on a site managed by the Olympic Committee.

2.3. The Third Step: The National Selection

The national selection aims to identify a group of 15–20 students who are considered as the “probable olympiad participants”, among which the members of the olympiad team are finally chosen. The olympiad team is made up of 4 regular members and 2 substitutes, who will participate in the international olympiad.

The national selection takes place on one single day in the month of November of the year before the international olympiad, in a venue chosen by the Olympic Committee, every year in a different region. The problems submitted to the participating students are similar to those of the international contest, although the level of complexity is lower, because in the period between the national selection and the international olympiad the preparation of probable olympiad participants (or a part of them) is expected to be improved thanks to the specific training activities planned.

The regional MIUR offices organize training activities for students participating in the national selection, often by relying on University professors and researchers of IT Departments in the various regions.

An IT tool has recently been developed to make individual training activities more effective. This tool is interesting for both students preparing for the national selection and those training for the territorial selection. This tool operates as an on-line teacher that provides students with a set of problems of different levels of complexity and origin and allows them to choose a specific problem and to submit the solution in a file containing the text of the source programme. The “on-line teacher” compiles the program, identifies any errors, and, if possible, executes it on a certain number of predefined test cases, and finally informs the user of the outcome.

2.4. The Italian Olympiad

For some years now the national selection has been considered as an Italian Olympiad, in order to make the participation to the selection more appealing to students and, especially, their teachers.

The best students are awarded with gold, silver and bronze medals, following criteria in line with those adopted for the international olympiad.

The prize-giving ceremony takes place on the day after the contest and consists in a party with the participation of contestants, their teachers and the Olympic Committee,

as well as school and public officials/authorities of the Italian region hosting the competition. A small ceremony is also organized in which the school hosting the competition passes its role to the school which will host the competition in the following year.

The initiative has been appreciated, as it has become an opportunity for teachers to meet, while contestants are different every year.

2.5. *The Fourth Step: Training the Olympiad Team*

In the months before the international competition, the probable members of the olympiad team selected through the national competition, who make up a team of 15 to 20 students (5 gold medals, 10 silver medals and some particularly promising bronze medals) participate in a training provided both in form of residential seminars managed by national trainers (lasting 3–4 days each) and of Internet-based training. The latter is provided in order to reduce the interference of olympiad training on regular school activities.

It can happen that some of the selected students leave the training after some time because they realize they do not measure up to their colleagues, while some others are advised to withdraw from the training before its completion. At the end of the last residential seminar, the olympiad team, made up of four regular members (as well as two substitutes), is chosen and will be followed by the trainers through remote systems until the departure to the city hosting the international olympiad.

The selection of the four regular team members is mainly based on the results obtained by the contestants in a series of tests simulating the conditions of international olympiad competitions.

3. Conclusion

In the first years of its participation to the international olympiad, the Italian team obtained only some bronze medals, but throughout the years results have improved and become more satisfying partly due to the effective selection process and training activities, but especially thanks to the contestants' commitment. The Italian team has achieved a gold medal (unfortunately only one so far), eleven silver medals and seventeen bronze ones.

These results, which hopefully will constantly improve in the following years even if the international competition is very tough, show that young Italian students have a good level of talent and skills, despite the insufficient attention given by school institutions to teaching of Information Science's foundations.

A negative aspect noted in the Italian participation in the competition, which the Olympic Committee has not succeeded in remedying, is the low percentage of women in the selection phases and the mediocre results obtained by the few girls who have passed the school-based selection.

The Committee has long been analyzing this problem and intends to define a strategy to improve the female students' commitment to participate in olympiads and drive them to the achievement of the success results that can surely be obtained.

4. Appendix

Some of the questions submitted to the students in the latest school-based selection are reported below. Examples of the problems presented in territorial and national selections are not reported, as they are similar, although less complex, to those submitted in the international olympiad.

a) Math and logic question (weight = 1 point)

If you throw two dice, what is the probability of obtaining 6 (obviously by adding the values of the two dice)?

Possible answers:

- a) 18/36
- b) 12/36
- c) 5/36
- d) 7/36

b) Math and logic question (weight = 3 points)

Consider the following multiplication:

$$\begin{array}{r}
 A * * * \times \\
 B * \\
 \hline
 C * * * *
 \end{array}$$

where each of the figures of the three numbers A, B and C (indicated by the symbol *) can be only 3, or 5, or 7. What are the three numbers A, B and C?

c) Algorithmic question (weight = 1 point)

In order to respect the timetable of reservations for the delivery of pizzas at the odd-numbered houses of the same street, pizzas should be delivered by following the instructions based on a code specifying how to go forwards (e.g., F2 to move forward of two buildings) and backwards (e.g., B5, to move backwards of five buildings) along the street starting from a specific point.

An example for the delivery of 4 pizzas: if starting from house no. 1, the instructions were [F2, F1, B2] the delivery would be in this order [1,5,7,3] which indicate the number of the odd-numbered houses where pizzas should be delivered.

Seven pizzas should be delivered. The first one should be delivered at house no. 1, while the remaining ones should be delivered by following these instructions: [F3,F4,B5,F6,B3,B4]. Find the list L containing the right order of the house numbers where pizzas will be delivered.

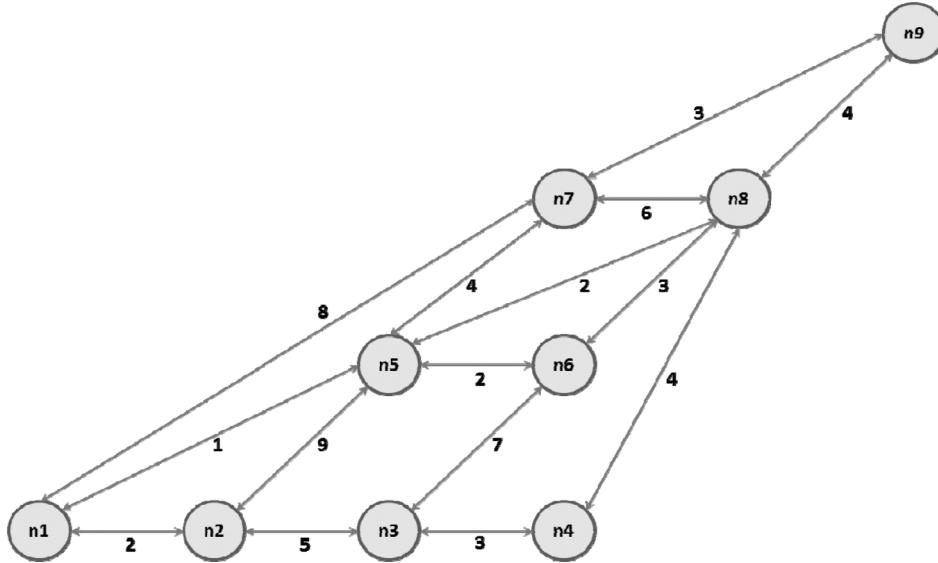


Fig. 1. A street map.

d) Algorithmic question (weight = 3 points)

The term $a(<\text{node1}>, <\text{node2}>, <\text{distance}>)$ describes the street connection between node1 and node2, and provides the relating distance in kilometres.

Consider the street map composed by the following connections:

$a(n1,n2,2)$	$a(n2,n3,5)$	$a(n3,n4,3)$	$a(n4,n8,4)$	$a(n5,n6,2)$	$a(n6,n8,3)$
$a(n1,n7,8)$	$a(n8,n7,6)$	$a(n5,n1,1)$	$a(n2,n5,9)$	$a(n3,n6,7)$	$a(n5,n7,4)$
$a(n9,n7,3)$	$a(n8,n9,4)$	$a(n5,n8,2)$			

as shown in the Fig. 1.

The distance between two nodes is described by the list of nodes, which are ordered from the departure node to the arrival node. The total distance in kilometres of each connection can obviously be calculated.

For example, the connection: $L = [n1, n7, n8, n6]$ has a distance K of 17 Km.

Find the number N of different connections departing from the node $n2$, ending in the node $n9$ and crossing once all the nodes in the picture. Among these connections, provide the list $L1$ of the shortest distance and the list $L2$ of the longest one.

e) Programming question (weight = 1 point)

Consider the following fragment of a program:

```
int r, c, s;
printf("Input a full number between -10 000 and 10 000: ");
scanf("%d", &r);
c = 1;
```

```

c = r * c;
s = 1;
if (c <= r) {
    s = s + c;
    c = c * 2;
}
printf("the variable s is %d\n", s);

```

Which of the following statements is true?

- a) *The sum of all numbers from 1 to r + 1 is shown.*
- b) *The value r + 1 is shown.*
- c) *The value r + 1 is shown only if r >= 1.*
- d) *The value 2r + 1 is shown.*

f) Programming question (Weight =3 points)

Consider the following fragment of a program:

```

int succ(int i) {
    if (i <= 2)
        return(1);
    else
        return(3*succ(i - 1)+2*succ(i - 2)-succ(i - 3));
}
main() {
    printf("num=%d\n", succ(7));
}

```

What appears on the screen if main() is executed?

References

- Fadini, B., Grossi, R. (2006). *Olimpiadi dell'informatica, giovani talenti cercansi*, Mondo Digitale.
- Combéfis, S., Leroy, D. (2011). Belgian Olympiads in Informatics: Story of a National Context Launching. *Olympiads in Informatics*.
- Scarabottolo, N. (2011). *Olimpiadi dell'Informatica: Bilancio di un decennio di attività*. Mondo Digitale.
- Official Italian Site for Informatics Olympiads*. <http://www.olimpiadi-informatica.it>.



M. Italiani started its activity in 1959 in the Research & Development Team of Olivetti's Electronic Division in Milan. He has been full professor of computer science subjects at the Universities of Turin, Pavia and Milan since 1976. He has been the president of AICA, the association that, in co-operation with the Italian Ministry of Education, University and Research, promotes the Italian participation in the International Olympiad in Informatics. He is currently member of the Italian Olympic Committee.

The National Computer Olympiads and the IOI Participation in Finland

Jari KOIVISTO

*School Principal, LUMO Upper Secondary School
Urpiaisen tie 14, FIN-01450 Vantaa, Finland e-mail: jari.koivisto@vantaa.fi*

Abstract. This paper describes the concise history of Finnish National Computer Olympiad and the participation in the IOI. It is entirely based on the author's own experience because he has been involved in running or participating as a team leader both of the olympiads since 1988.

Key words: Finish olympiads, teaching computer science, IOI.

1. The National Olympiads

The first national computer olympiad in Finland was organized in 1988 during the time when the first initiatives to accept IT and later ICT as a subject appeared in education. At the same time organized work to create regional curriculum documents in ICT started. The obvious reason for the increased interest in ICT in society was the added value accumulating in business and public administration when using the ICT as a productivity tool. The first really useful software in educational administration appeared at the same time.

The first trials to use computers in education took place in the 70s when the schools started to buy the Swedish ABC-80 computers. In the 80s there were a lot of computers in the Finnish schools mainly from Nokia but there was no coherent way to teach computer science as a subject and only a very few trials to create educational software. Some teacher training to teach programming was organized in the beginning of the 90s but this initiative did not survive the 1993 curriculum reform.

The basic idea for the first national olympiads in computer science was to test general knowledge about computers and their use in the society. In addition there were some tasks to write simple pieces of software. The programming language mainly used in schools was Basic and fortunately some good environments to write software in Pascal were also available. Turbo Pascal afterwards became a popular programming language in Finnish schools.

The idea to test general knowledge in the national computer olympiad has survived to present time and it is still producing 40% of the final count of the points whereas solving the programming tasks produces 60%. The reason is that in Finland the whole process from the national olympiads to the IOI is financed as a part of the general education in Finland. General knowledge about ICT as one of the most important productivity tools

is highly appreciated in Finland. By emphasizing general knowledge from ICT, the vast spectrum of possibilities to use the ICT in the society is acknowledged.

Previously the national computer olympiad took place in three steps:

1. The first step was the primary round in all the Finnish lower and upper secondary schools with separate tasks for both in September and teachers doing the evaluation. Participation was about 4000 students.
2. A qualifying round in November for the best competitors in each school, and the national organizers doing the evaluation. Participation was about 200 students.
3. The final round in January the following year in Helsinki for 20 students from both series a big national bank as a sponsor.

2. Participation in the IOI

In 1993 Finland started its participation in the IOI. Gradually the programming tasks in the national competition became more demanding in order to better prepare the students for the IOI. At the same time the national competition lost the bank as a sponsor, and the possibilities to organize a three-step competition vanished. The global recession took its toll.

Since 1997 the competition has taken place in two steps. First the on-line competition takes place in November with two weeks answering time and then the final in Helsinki in January next year. The number of participating students has reduced below 100 but on the other hand all the participants are expected to be able to solve difficult algorithmic tasks. It is a pity that because of lack of resources it is not possible to organize the first round in schools, but it has to be accepted as a reality.

The programming tasks in the two-step competition are normally kept at the same difficulty level as in the IOI. To encourage the students at least to try to write something and also to increase the number of participants some icebreaker tasks are introduced. The winners have generally done programming for many years before the competition but their success is still fully dependent on their own initiative because te programming studies at Finnish schools have in practice completely disappeared in the wake of several national curriculum reforms.

Training for the Baltic IOI and the main IOI begins immediately after the national olympiad. The training is organized on-line on the Internet and only a few days face-to-face training is provided. The purpose of the training is to strengthen the algorithmic skills of the participants and enable them to analyze the tasks in order to find a good starting point.

The students having participated in the IOI have normally continued their academic career at the Finnish universities mostly studying technology or computer science generally ending their studies to a doctoral degree. Before a doctoral degree only a very few students have decided to continue their studies at a foreign university but naturally a part of their post doc studies are almost always done in foreign universities.

The IOI is considered in Finland as one of the international science olympiads which in practice means that most of the funding is in the government educational budget.

The reasonable success in those competitions guarantees that the procedure to organize the Finnish participation in those competitions remains quite steady. This of course is very dependent on the estimated added value of the competitions in the Finnish education. The organizers of the national science competitions are very aware of that fact.



J. Koivisto has been a member of the Finnish National Computer Olympiad organizing committee and occasionally chairing the committee since 1988. His first IOI was in Haninge, Sweden in 1994 as a deputy team leader. In 1995–1997 he participated as the Finnish team leader. In 1998–2004 he was a member of the IC. In 2001–2002 he was the president of the IOI organizing the IOI2001 in Tampere, Finland being at the same time the chair of the IC. In 2006–2008 he was the ED of the IOI.

Preparing Students for IOI: Thailand Country Report

Kanchit MALAIVONGS

*Fellow, Royal Institute of Thailand
e-mail: kanchit.ma@gmail.com*

Abstract. This paper describes the training process which is used to prepare Thai students for International Olympiad in Informatics. Thailand sent two observers to Minsk for the second IOI to find out about the contest and types of tasks used in the contest. Subsequently a selection process has been carried out every year to screen students for the training camps which are organized to provide students with adequate programming skills to solve the IOI tasks. At the beginning the training is under the responsibility of the Institute for the Promotion of Teaching Science and Technology (IPST). At present a new foundation, the Promotion of Academic Olympiad and Development of Science Education Foundation under the Patronage of Her Royal Highness Princess Galyani Vadhana Krom Luang Narathiwat Rajanagarindra helps share the training responsibility of all Academic Olympiads including IOI. Both organizations now work together to make sure that students in all parts of the country have equal opportunity to receive training and equal chance to be selected as the student representatives of Thailand to IOI.

Key words: olympiad, programming contest, programming camp.

1. Introduction

Thailand started its involvement with the IOI by sending two observers to participate in the second IOI at Minsk, Belarusian Republic, Soviet Union in 1990. The author was one of the observers at that time and the author was convinced that the contest was of high standard and, without proper trainings, Thai students would not be able to compete with other students at the IOI. This is because computer programming was not a part of the required curriculum for secondary education at that time. Even nowadays only some schools offer a programming course to students and the content is very basic indeed. If Thai students are to be successful in IOI, they must be trained on the topics used in IOI tasks.

The visit to Minsk was sponsored by the Institute for Promotion of Science and Technology Teaching (IPST). IPST is a semi-governmental organization reporting directly to the Minister of Education (www3.ipst.ac/eng/). IPST has responsibility in developing curriculum and textbooks on science, technology and mathematics for primary and secondary education. IPST had previously developed a computer curriculum for secondary schools. The first computer curriculum consisted of an Introduction to Computers and Basic Programming. Later, the curriculum was expanded to include an Introduction to Operating System, Introduction to Spreadsheet, and an Introduction to Databases.

The Ministry of Education officially recommended this curriculum for use in secondary schools throughout the country. However, the schools were not able to adopt the curriculum because they lacked qualified teachers and the curriculum itself was not compulsory. In addition, even in the schools that could adopt these courses, the students were not interested to take them. This is because the computer subject is not included in the entrance examination which is used for admitting students in various governmental universities.

Realizing these problems, the author and his co-observer recommended IPST to immediately start selecting students and prepare the training materials for them. The first problem was the lack of programming experts in IPST. Although IPST is involved with many teachers in high schools, we found that they did not have adequate background in programming and advanced problem solving. Fortunately, IOI activities are of interest to several faculty members in different universities and they were pleased to help in selecting and training students for IOI. The IPST director, through our advice, appointed a group of university instructors to be the members of the Subcommittee on IOI. The author was also appointed as the Chairman of this subcommittee.

2. Training

The academic year of Thai schools normally starts in the middle of May and ends in early March. The selection of students in the early years of Thai IOI trainings started by announcing a countrywide selection examination. The announcement was around July each year. The examination took only one day and consisted of only three main topics: mathematics, logic, and programming. The idea behind these topics is obvious. Mathematics is included to indicate to students that programming contest will definitely needs mathematical talent. Logic is to ensure that students are able to think logically when solving programming problem. The programming part in the selection examination is a simple algorithmic problem which can be programmed in Basic. In the early years only a few hundreds students applied to sit in the selection examination. At the present time, several thousands students are attracted to the examination.

Only the thirty topmost students were selected to attend the first training camp in October for four weeks. The camp site is the IPST office in Bangkok. IPST is well equipped with accommodation facilities as well as computers for practicing. After the three week training, an examination was given and 16 top students were selected for the second camp. This second camp was organized in two periods. The first period was in March and lasted for three weeks. At the end of this part, an examination was given to select ten students for the second period which lasted ten days. Four students were then selected for the IOI contest. Between April and July, these four students were required to practice tasks provided by their tutors during weekends. Students normally communicated with their tutors through the Internet and sometime they were required to attend a special class provided on weekends by the tutors. The overall training period normally lasted about ten months. Tasks that students are required to practice are those past IOI tasks. Test data were created for each task to test students' programs.

Table 1
Summary of medals achieved by Thai students

Year	Gold	Silver	Bronze
1991	–	1	2
1992	2	1	1
1993	–	2	1
1994	–	–	2
1995	–	–	2
1996	–	2	1
1997	–	1	3
1998	–	1	2
1999	1	–	2
2000	–	3	1
2001	–	1	3
2002	–	–	3
2003	–	3	1
2004	1	1	2
2005	2	2	–
2006	–	–	3
2007	1	3	–
2008	2	1	1
2009	1	2	1
2010	1	2	1
Total	11	26	32

The tutorial topics comprised of the following topics: discrete mathematics, data structure, linked lists, stack, queue, tree, sorting, merging, searching, graph, hashing, etc.

We are proud to say that the training programs enable our students to achieve very good results starting from the third IOI in Greece. Table 1 summarized the medals that Thai students received so far.

3. Selection Process

The success of Thai students in IOI very much depends on the patronage of Her Royal Highness Princess Galyani Vadhana, the Princess of Naradhiwas, who was very keen in promoting science education in Thailand. Learning that IPST could not obtain adequate budget for the Academic Olympiads, she helped provide financial supports to send students and team leaders to several olympiads. After discussion between IPST and Her Royal Highness, in 2000 the Promotion of Academic Olympiad and Development of Science Education Foundation under the Patronage of Her Royal Highness Princess Galyani Vadhana Krom Luang Narathiwat Rajanagarindra was founded with an understanding that working under the Foundation is more flexible (www3.ipst.ac/eng/). The ob-

jective of the Foundation is to provide trainings to secondary school students and to select suitable representatives to Academic Olympiads and other contests. The Foundation makes agreement with the IPST to take responsibility in the first stage trainings. This agreement has changed the procedure outlined above into the following.

Between July and August each year, the Foundation invites students to attend the Computer Programming Camp in October. The Foundation has previously established 14 training centers in a few schools in Bangkok and in different universities throughout the country. The centers receive financial supports from the Foundation and are responsible for inviting instructors to provide training. In 2010 there were 4,100 applicants and the Foundation selected only 600 students for this camp (Camp 1). The topics provided in the first camp are programming fundamentals and problem solving. Basic data structures are also taught to acquaint students with advanced topics. The main objective is to encourage students to write programs to solve various kinds of tasks. At the end of one month training course, 300–400 students are selected to attend the second camp in March (Camp 2). This camp lasts only 2 weeks and about 100 students are selected to the National Olympiad in Informatics contest in May.

The National Olympiad in Informatics is organized to select 25 students for the IPST training programs as outlined above. At present time the schedule of the IPST training is the same as that initiated several years ago. The difference are that the first selection process is carried out by the Foundation through the National Olympiad in Informatics and students are required to participate in four camps before being selected to IOI.

4. National Olympiads in Informatics

The National Olympiad in Informatics has a similar structure with that of the IOI. The students will compete in two contests each last about three hours. Three tasks chosen by the student trainers from each region are given in each contest. The tasks are similar to the IOI tasks but are less difficult. Submitted tasks are graded by a grader and results are automatically shown on the Internet.

The work of the Promotion of Academic Olympiad and Development of Science Education Foundation under the Patronage of Her Royal Highness Princess Galyani Vadhana Krom Luang Narathiwat Rajanagarindra is not limited to trainings of students for the IOI. The Foundation is responsible for trainings of all students to be selected for other Academic Olympiads as well. The Foundation also publishes textbooks and guidelines on different scientific subjects which stimulate interests among Thai youths.

To encourage students to undergo advanced study in science and technology, IPST provides scholarships for students who are selected as representatives in every Academic Olympiad to study in universities up to doctoral degree. However, students will receive the scholarship only when they select to study in the same field that they have been selected as a representative. Thai universities also agree to admit students who passes the second camps to those fields of choices without having to sit in entrance examinations.

5. Conclusion

In conclusion, all Academic Olympiads play a very important role these days. Newspapers and other media help promote the activities by making news on students who get medals. As a result, more and more students apply for the seats in the academic camps every year. We hope that this is a good means to strengthen the science education in the country. Most importantly, almost all Academic Olympiad's contestants have been attracted to academic and research professions which will help improve the quality of science and technology teaching at all levels of education in the country.

References

www3.ipst.ac/eng/. This website provides information about objectives, activities and achievements of IPST.
www.posn.or.th. This website currently displays only in Thai.



Dr. K. Malaivongs is the chairman of the IOI Selection Committee of the Institute for the Promotion of Teaching Science and Technology. His involvement with IOI started in 1990 when he went as an observer to the second IOI in Minsk, Belorussia, Soviet Union. He was the leader of Thai student team from 1991 to 2001. Besides his activity in IOI he is also the chairman of Sripatum University, one of the large private universities in Bangkok.

Kyrgyzstan National Report on Olympiads in Informatics

Pavel S. PANKOV¹, Timur R. ORUSKULOV²

¹*International University of Kyrgyzstan*

A. Sydykov str. 252, apt. 10, 720001 Bishkek, Kyrgyzstan

²*Ministry of Education and Science of Kyrgyzstan*

Microrayon Vostok 5, 14/2, apt. 3, 720065 Bishkek, Kyrgyzstan

e-mail: pps50@rambler.ru, toruskulov@mail.ru

1. Education and Initiation of Olympiads in Informatics in Kyrgyzstan

Informatics (under the traditional name "Foundations of Informatics and Computer Facilities") is taught in all secondary schools of Kyrgyzstan since the autumn of 1985. Firstly it was taught on a blackboard with calculators; pupils sometimes visited local computer centers. Later, more schools obtained computer rooms, now almost all schools have computer labs, some of them linked up with the internet.

By the State Educational Standard, now Informatics is obligatory subject in 7th (1 hour per week), 8th and 9th (2 hours per week) grades. The decision to teach or not to teach Informatics in older (10th and 11th) grades is granted to the schools themselves; many choose to set the subject as a part of the compulsory curriculum. Recently a new curriculum has been created for both primary and secondary school, from 1st till 11th grade.

The Olympiads in Bishkek city, the capital of Kyrgyzstan, are conducted since 1985, annually; National ones are conducted from 1987. From the beginning, they are led under the conduct of the corresponding Educational State Institutions. Besides Informatics the National Olympiad is conducted among other subjects (for instance Kyrgyz language and literature, Mathematics, Sociology) by the Ministry of Education and Science. Kyrgyzstan teams participate in IOIs since 2000.

2. Organization and Participants

B.1 Structure

The Olympiad is arranged in four levels: I (in each school; November); II (district or area; January); III (city or region; February), IV (National; March) and Selection competition for IOI (April). Then, for the two last levels the contestants are divided into two groups:

the first one includes 10th grade pupils (maximum age 16 years) and the second one does 11th grade pupils (17 years). Olympiads of the III and IV levels are conducted in two days; three tasks and 2.5 hours per each day. Selection competition as the final of Spring school (without division to groups) for IOI is conducted in one day: five tasks and 4 hours.

B.2. Participation and Funding

All pupils beginning from 8th grade have the right to participate in the I level. Winners of the II level of 8th and 9th grades who have achieved excellent results may be allowed to the first group of the III level together with winners of 10th grade.

Each of the seven regions of the country, the capital Bishkek and Osh city ("the southern capital") send 2 pupils in each group of the IV level. So, 19–20 pupils (including winners of the preceding year) participate in the first group and 17–18 pupils do in the second one. The teams are sent to the competition by the regional Departments of Education, the Ministry is responsible for the temporary residence and catering arrangements (at the National Computer Gymnasium in Bishkek).

Because of essential differences between the traditional content of IOI and ones of our Olympiads (see Section C below), Spring schools with the final competitions for the IOI are conducted for the winners (6–8 of the first group and 3–4 of the second one) of the IV level Olympiad. They are funded by the Bishkek Kyrgyz–Turkish Men's Lyceum.

B.3. Prizes and Other Preferences of Participants

Winners of City Olympiads (the III level) are given prizes by the city Mayor's office and by the sponsors; winners of National Olympiads (the IV level) are given prizes from the Ministry of Education and by the sponsors including the daily newspaper "Vecherniy (Evening) Bishkek" and company "Logic". The Kyrgyz–Russian Slavic University (KRSU) conducts annually the competition for winners of the second group of National Olympiads and accepts some of them free of charge; other universities in Kyrgyzstan also give discounts to winners.

Some universities including the International University of Kyrgyzstan grant computer classes for conducting the III level of the Olympiad.

B.4. Training

Pupils who wish to participate in selection for the forthcoming IOI, during the year practice solving tasks of the preceding IOIs. They also participate in Croatian Open Competitions in Informatics. Some of them go to the Zhauitykov Olympiad conducted annually in Kazakhstan.

3. Preparation and Evaluation of the Tasks

C.1. Preparation and Examples of Tasks

The different types of tasks and ways to develop them are described in Pankov *et al.* (2000, 2003, 2007, 2008, 2009, 2010). The CPU time is restricted to 5 seconds. A few examples of categories and tasks are listed below.

Tasks with implicit presentation of graphs, for example:

Task 1. Given two numbers P and Q in $(100 \dots 999)$. In one step any digit can be changed to a neighbor one or the number as whole can be changed to a neighbor one. How many steps are necessary to transform P into Q ?

Example: $929, 830 \rightarrow 2[929 - 829 - 830]$.

Common combinatory tasks, for example:

Task 2. Given natural number $2 \leq N \leq 2010$, how many essentially different pairs of A) rectangles composed of N equal squares or B) parallelepipeds composed of N equal cubes exist? (Permutations in pairs, rotation and reflection do not yield new pairs).

Example A): $N = 5 \rightarrow 3$ (pairs) $[4 \times 1 + 1 \times 1; 2 \times 2 + 1 \times 1; 3 \times 1 + 2 \times 1]$.

Acceleration of some algorithms, for example:

Task 3. Given the following algorithm [written in any semi-formal language]:

```
{ output ("Enter a natural number N, 1 <= N <= 2009"); input (N); W := 2009;
for I = N to 2009 { for J = 0 to 2009 { for K = 0 to 2009
{if I * (I + J) * (I + J + K) = 2009 then W := W - 1 } } };
output (W); }.
```

Write a program calculating same results in the CPU time 5 seconds.

Tasks with simple solutions but which demand labor expenditures, for example:

Task 4. Speed of a car is 60 kmph; an airplane takes one hour to fly from Bishkek to Osh. Karakol = $K - (220 \text{ km}) - \text{Balykchy} = \text{Ba} - (170 \text{ km}) - \text{Bishkek} = B - (60 \text{ km}) - \text{Karabalta} = \text{Kb} - (140 \text{ km}) - \text{Suusamyr-Cross} = \text{SC} - (380 \text{ km}) - \text{Jalal-Abad} = J - (50 \text{ km}) - \text{Osh} = O - (200 \text{ km}) - \text{Batken} = \text{Bt}; \text{SC} - (100 \text{ km}) - \text{Talas} = T; \text{Ba} - (200 \text{ km}) - \text{Naryn} = N$.

Output the minimum time necessary to deliver a copy of the newspaper "Vecherniy [Evening] Bishkek" [sponsor] to a given point on the roads (defined by the notation of each of the road segments and the distance to the first end of the segment).

Example: Kb, 8, SC \rightarrow 1 hour 8 minutes.

"Black box", for example:

A program is given as an exe-file (a contestant may run it as many times as s\he wishes). Write a program equivalent to the given one [which is sufficiently simple, does not contain large numbers and complex algebraic expressions]

Task 5. The program was the following: {*output* ("Enter a natural number $N \leq 10000$ "); *input* (N);

if $N \leq 2011$ *then output* $((N - 1) \text{div } 3 + 1)$ *else output* ("Too many").}

Graphical tasks, for example:

Task 6. Given [by jury] two arbitrary points between 40% and 60% of the height and 10% and 90% of the width of the display. Arrange the letters LOGIC [name of sponsor] containing the given points (marked with another color) symmetrically on the display.

Video-clips and controlled video-clips, for example:

Task 7. [Tyundyuk is the upper part of a yurt; its image is on the national flag of Kyrgyzstan]. The shadow of tyundyuk consists of the circle of radius 100 cm and two groups of three lines of parallel to each other (16 segments) within the circle; the distance between two nearby segments is 10 cm. A) Show this shadow on scale 1:10. B) Not to be seen by the Tiger [then was a year of Tiger], the Ant (a point, a pixel) must crawl within the shadow only. Given a point within the shadow, show a way or a slow motion simulation from this point till the center.

Some tasks of our Olympiads demand wide knowledge field in other subjects, such as physics, chemistry, geography, philology. Tasks like Tasks 3, 4, 5, 6, 7 are not used in training and selection to IOIs.

Tasks for the III and IV levels are created by the chairman and the deputy chairman of the jury of the Olympiad in two languages (the state one and the official one in Kyrgyzstan). Tasks for the III level with scopes of tests are brought in sealed envelopes by the representatives of the Ministry to all regions. Tasks for the Spring School and the Selection competition are created by the leader of it in English.

C.2. Evaluation of Tasks

During the period, the contestants verify their solutions by their own tests. After the time allowed, the contestants leave their computers and the jury begins check-up with inputting tests by hand in the contestant's presence. Variations in formats of input data (due to used algorithmic languages and to the contestant's will in graphical tasks) are permitted but none corrections are permitted.

Tasks such as Tasks 1, 2, 3, 4, 5 are graded by the standard method, i.e., by a range of 4..6 tests in increasing order of difficulty, with a condition: if the program responds "Impossible" (or alike) in all tests then it gets 0. Informal tasks such as Tasks 6, 7 are also graded by a range of 2..4 tests, and the marker may deduce points if some of the conditions are breached. There is sum total of 10 points for all of the tasks. The score for the III and IV level totals up to 60 points.

4. Informatics Curriculum

We have developed the following curriculum on "Information Communication and Technology" for secondary school (the project is funded by Asian Development Bank and Government of the Kyrgyz Republic).

Fifth year of education: information, its types and properties; the use of information processes in daily life, society and individual's activity; main features of a computer; computer technology in society; primary computer use in problem-solving; primary knowledge on software configuration and operating system and their use; models, their types and use in daily life.

Sixth year of education: information sources and their application; main and peripheral hardware and its use; treatment of constituents of standard software: graphical editors, music player, calculator; developing and investigating simple information models by computer means.

Seventh year of education: character systems as carriers of information, controlling systems using information processes; principles of computer program management; using utilities for customizing operating systems; files, directories, carriers of information treatment; application package Microsoft Office; electronic documents in text editors; objects and tables in text editors; main means of information-mathematical process modeling and real environmental objects; the algorithm concept; writing down a simple algorithm and implementing algorithm behavior algorithms in urgent situations.

Eighth year of education: types of controlling systems; cybernetics and cybernetic systems; digital systems of coding and their use; presentation of information and solving mathematical problems; operating systems: MS DOS, MS Windows-X, Linux, Macintosh; developing structures and designing multimedia projects; information saving, search and sorting; knowledge of types of algorithms, algorithmic presentation of functions; means and stages of developing computer programs; software life-cycle.

Ninth year of education: syntax, operators, procedures and functions of programming languages; developing programs to solve graphical and mathematical problems; concept about the structure of computer networks; use of the local and world-wide network resources; organizing collective work by using Internet.

Tenth year of education: developing and investigating simple information models by technological means; algorithm development and their execution; solving logical tasks; debugging and program testing in one of programming languages.

Eleventh year of education: solving logical, mathematical, scientific and economic problems; cellular communications, mobile telephones, smart phones; collective work with the use of GPS and WAP services; knowledge about syntax, tags, frames, forms and hyperlinks of Web-programming languages; developing information sites by the means of hypertext markup in documents; concepts of ethic and legal norms of human's information activity and information security.

5. Future Development

We hope that Informatics ("Information Communication and Technology") will be included into the State Educational Standard as an obligatory subject in 10th and 11th grades too. Programming skills would improve and number of schoolchildren participating in the I and II levels of the Olympiad would increase. We will have an opportunity to offer more difficult tasks at the III level and tasks of IOI types at the IV level.

Each year in the autumn the KRSU conducts the Kyrgyzstan quarterfinal of the International Collegiate Programming Contest administered by the Association for Computing Machinery by the means of Internet. The winners participate in semifinals. Also, the KRSU supports permanent competition site on solving tasks. Some universities conduct their own Olympiads and other kinds of competitions in Informatics (irregularly) for their students and for schoolchildren to attract undergraduates.

References

- Pankov, P.S. (2008). Naturalness in tasks for olympiads in informatics. *Olympiads in Informatics: Tasks and Training*, 2, 115–121.
- Pankov, P.S. (2010). Real processes as sources for tasks in informatics. *Olympiads in Informatics*, Selected papers of the International Conference joint with the XXII Olympiad in Informatics, Waterloo, 4, 95–103.
- Pankov, P.S., Oruskulov, T.R. (2007). Tasks at Kyrgyzstani olympiads in informatics: experience and proposals. *Olympiads in Informatics: Country Experiences and Developments*, 1, 131–140.
- Pankov, P.S., Baryshnikov, K.A. (2009). Representational means for tasks in informatics. *Olympiads in Informatics*, Selected papers of the International Conference joint with the XXI Olympiad in Informatics, Plovdiv, 3, 101–111.
- Pankov, P.S., Oruskulov, T.R., Miroshnichenko, G.G. (2000). School Olympiads in Informatics (1985–2000 years), Bishkek (in Kyrgyz and Russian).
- Pankov, P.S., Oruskulov, T.R., Miroshnichenko, G.G. (2003). Olympiad tasks in informatics, devoted to Kirghiz statehood, history of Kyrgyzstan and Great Silk road, Bishkek (in Kyrgyz and Russian).



P.S. Pankov (1950), doctor of physical-math. sciences, prof., corr. member of Kyrgyzstani National Academy of Sciences (KR NAS), is the chairman of Jury of Bishkek City OIs since 1985, of National OIs since 1987, the leader of Kyrgyzstani teams at IOIs since 2002. Graduated from the Kyrgyz State University in 1969, is a main research worker of Institute of Mathematics of KR NAS, a chairman of the International University of Kyrgyzstan.



T.R. Oruskulov (1954), candidate of pedagogical sciences, is the deputy chairman of Jury of Bishkek City OIs and of National OIs since 1988, the deputy leader of Kyrgyzstani teams at IOIs since 2005. Graduated from the Frunze (now Bishkek) Polytechnic Institute in 1977, worked in Institute of Automatics of KR NAS, Kyrgyz Research Institute of Pedagogy, studied at Post Graduate Office of the USSR Academy of Pedagogy in 1988–1991, was a professor of Kyrgyz Academy of Education, works in the II Project of Education of the Asian Development Bank at Kyrgyzstani Ministry of Education and Science.

Israel: The Regional and National Competitions

Ela ZUR¹, Tamar BENAYA¹, Oren BECKER², David GINAT³

¹*The Open University of Israel, Computer Science Department
108 Ravutzky st., 43107 Raanana, Israel*

²*The Hebrew University, Mathematics Department
Jerusalem, Israel*

³*Tel-Aviv University, Science Education Department
Ramat Aviv, 699978 Tel-Aviv, Israel*

e-mail: {ela, tamar}@openu.ac.il, oren.becker@gmail.com, ginat@post.tau.ac.il

Abstract. This year the Israeli Ministry of Education has decided to increase its support and involvement in the IOI project. The main goals of the increased involvement were to expose the project to a wider audience, and to expand the team selection and training process. For the first time, a regional competition was conducted before the national competition. In this paper we describe the aims, scope, and contents of the regional and national competitions, and provide some statistics about the students' backgrounds and views about these competitions.

Key words: regional competition, national competition.

1. Introduction

The IOI – the International Olympiad in Informatics – is the primary computer science (CS) competition for young (secondary school) students. The primary goal of the IOI is to stimulate challenge in CS among exceptionally talented young students from all over the world, and have them share scientific and cultural experiences. Each participating country conducts a preparation process, and brings to the IOI a team of (at most) four contestants. During the IOI, the contestants compete individually in solving and programming challenging algorithmic tasks. Different countries invest different amounts of effort and resources in preparing their IOI teams (e.g., Diks *et al.*, 2007; Casadei *et al.*, 2007; Philips, 2010; Tsvetkova, 2010; Wang *et al.*, 2010).

In Israel, until 2010, the IOI project was composed of four stages: a self-study towards the national competition; a national competition, an advanced training and team-selection stage, and the national team's preparation for the IOI. A detailed description appears in the Israeli IOI website and in our previous paper (Zur *et al.*, 2010). This year, an increased support by the Ministry of Education enabled us to conduct a preliminary stage of a regional competition. This competition was aimed for a large audience, including students with very limited programming experience. The better students in this competition were invited to the national competition. In what follows, we describe our experience with the new regional competition and its successive national competition, and provide some

statistics regarding the students' backgrounds, motivation, achievements and points of view.

2. The Regional and National Competitions

The regional competition was the first stage of this year's olympiad project in Israel. Our goal was three-fold: 1. to offer algorithmic challenge to an audience as wide as possible; 2. to engage CS secondary school teachers in posing the challenge; and 3. to identify competent students, who will advance in the project activities.

We developed a 5-question questionnaire, which was posted in the website of the CS inspector of the Ministry of Education. The questionnaire was posted at a given time in mid December, which was a-priori told to all the secondary schools in Israel. Secondary school teachers, in 110 schools, downloaded the questionnaire, and posed it to their selected students, as a 2-hour exam. Questions during the exam, about the exam tasks, were directed in real-time (phone) by the teachers to our team. All in all, 1442 students participated in the exam. The students wrote their answers on exam sheets, which were downloaded from the internet. All the sheets were sent to our team for grading. A couple of days after the exam, we posted the solution, with broader perspectives of notions that appeared in the exam questions.

The teachers' role in this activity was to encourage their better students, and have them take the exam. They supervised their students during the exam, and sent us the student notebooks. Our hope was that teachers will be enthusiastic about posing challenge to their competent students, be motivated themselves to solve the questionnaire questions, and become interested in the olympiad contents. This expectation was met only to a limited extent.

The amount of secondary schools that participated in the exam was about 20% of the secondary schools in Israel – much more than in previous years, but still limited. Most of the teachers who engaged their students in the activity cooperated well with us, and enjoyed the exam questions, but only some showed further interest in the olympiad contents. Many felt that the olympiad contents are too challenging for them, and their role is mostly to link their competent students to the project. In addition, some teachers did not expose their students to the project, as they felt incompetent with challenging questions, and did not want to reach an "embarrassing" situation in front of their students, in which they could not solve the exam questions themselves. We hope to change this attitude in future years.

As one of our goals was to expose the project to an audience as wide as possible, we posed algorithmic tasks for which the required answers were not an algorithm, but rather the outcome of an algorithmic computation. This is in line with the approach presented by Burton (2010) and Kubica and Radoszewski (2010). This approach offers the opportunity of reaching students who are less acquainted, or even unacquainted with programming. The exam questions focused on mathematical and algorithmic insight, on which one had to capitalize her/his computation. For example:

Question 2 of the Regional Exam.

Given the following set of integers {4, 97, 357, 29, 22, 7, 14, 377, 1, 80, 331, 2, 320, 401, 258}, calculate the following, and explain your calculation in a few sentences.

1. What is the lowest integer that cannot be generated by adding integers of that set? (For example, if the set was {1,2,6}, the answer would have been 4.)

Hint: the answer is an integer whose units-digit is one of the integers in the given set.

2. Is there an integer in the set, whose replacement with its doubled value (e.g., replacing 4 by 8) will yield a larger result in Section 1?

Section 1 of the above question appears in some mathematical-challenges texts, as well as in Kubica and Radoszewski (2010). The challenge is to identify two patterns: **1.** the relevance of ordering the given integers; and **2.** if we can generate all the integers up to the value S , with the first k integers in the ordered sequence; and the $k + 1$ integer is v , who is not larger than $S + 1$; then we can generate all the integers up to $S + v$; if, on the other hand, v is larger than $S + 1$, then $S + 1$ is the answer (output).

The question does not require the explicit phrasing of an algorithm, but rather an algorithmic computation with the given data. Thus, one may answer the question even when less acquainted with programming. Yet, there are two drawbacks:

- one may make a calculation mistake, and yield an erroneous integer as output;
- one may solve Section 1 without really obtaining full insight.

In order to address the first drawback, we offered a hint, with which one could check her/his calculation result. In addition, we asked for a short description of the result in free text. In examining the "hint statistics", we noticed that out of the 1442 students who took the exam, only 13 (less than 1%) yielded an erroneous calculation upon having the right insight (wrong output, correct text description). About 25 students ($\sim 2\%$) indicated that they guessed the output for Section 1, from the hint; but the vast majority of their guesses were erroneous.

Addressing the second drawback was more subtle. We felt that we had to ask for some further insight, as one could answer Section 1 by ordering the integers, and then advancing in a greedy manner, without sufficiently understanding the task characteristics. Thus, we added Section 2. And indeed, of the 1442 students, 433 answered Section 1 properly, but only 203 answered Section 2 properly. That is, less than 50% of those who answered the question, demonstrated sufficient insight.

The rest of the exam questions were characterized as follows. Question 1 required capitalization on a simple string pattern; Question 3 was similar to Question 2; Question 4 involved some basic recursive view, which was simple to calculate by applying dynamic programming; and Question 5 involved the specification of rules for a hat-colours challenge.

All in all, the total score that one could obtain was 125. We invited to the next stage all those who obtained a score of 80+, plus students who obtained a lower score but nicely answered one or more of the insightful sections in the questions. We expected students to learn from our posted solution, and from our previous national competitions, in preparing to the next stage – the national competition.

The national competition was held in the beginning of February, and was attended by 668 students. We posed four algorithmic tasks, to be answered by paper and pencil. Task 1 involved avoiding a misleading greedy computation and applying simple dynamic programming; Task 2 involved a mathematical game, for which one had to recognize an invariant; Task 3 involved optimization of counting the number of operator's operations; and Task 4 involved enumeration that was tied to inductive insight. Task 3 is displayed below.

Task 3 of the National Exam.

Given a $2 \times N$ matrix, randomly coloured black and white, and an operator that switches the colours of the cells in a given (as a parameter) rectangle of cells; output the minimal number of the operator's invocations in order to turn the matrix colours into a chessboard colouring.

E.g., for the input:

W	W	B
B	B	B

The output will be 2. (One way of operating the operator is by first applying it on the two right columns, and then – on the bottom-right cell.)

We devised the above task by simplifying (considerably) the XOR task of IOI 2002. Our intention was to examine whether students recognize two relevant patterns – the rather simpler pattern that the operator's relevant impact is primarily on the vertical sides of its rectangles; and the pattern that sole processing of single-row rectangles may not be sufficient for optimality, and one may have to "touch" both single-row rectangles and double-row rectangles.

3. Student Backgrounds and Views

We conducted a preliminary study among the students who attended the national competition, in an attempt to learn about their backgrounds, motivation, and viewpoints regarding the regional and national competitions. We briefly describe the study's methodology and then display and discuss the results.

We posed an 18-questions questionnaire that focused on the students' CS education, their demographic environment, their preparation for the competitions, and their opinions about the questions in both competition exams. The questionnaire was answered by 513 of the students who attended the national competition.

We found that 28% of the students have at least one parent who is involved in a CS-related career such as high-tech industry or CS education. 85% of the students were male students, in spite of our efforts throughout the years to increase female participation. The students of the national competition came from 93 different secondary schools in the country. Their age distribution is displayed in Table 1.

As we can see, over 75% of the students are in their last two secondary school years. This finding is correlated with CS education in Israel, which usually starts at 10th or 11th grade. The drawback in this finding is that we first meet the vast majority of the competent students in a rather late age, thus most of them may learn from us and develop for a very limited time.

Table 1
Students' age distribution in the national competition

8th grade (age 14)	9th grade (age 15)	10th grade (age 16)	11th grade (age 17)	12th grade (age 18–19)
0.6%	1.8%	21%	44.5%	32.1%

In Israeli secondary schools, every student has to study several required subjects and at least one optional subject in depth. Mathematics is one of the required subjects and it must be taken throughout the secondary school studies. A student may select the level of studies which suits her/his interests and ability. The mathematics curriculum may be studied in one of three levels: 3 points, 4 points, and 5 points. Each point represents 90 class hours. We found that 87% of the students who participated in the national competition have selected the highest level of 5 points, while the rest selected the second level (4 points).

Starting from the 10th or 11th grade, CS is one of the optional subjects that a student may select to study in depth. The CS curriculum includes several courses starting with a Foundations course followed by a Software Design course. A detailed description of the Israeli secondary school curriculum is described in Gal-Ezer *et al.* (1995) and Armoni *et al.* (2010). We found that 95% of the students selected CS as their optional subject. 12% of them have not yet started their CS studies, 88% are currently studying or have completed the Foundations course, and 61% are currently studying or have completed the Software Design course.

We also found that 85% of the students selected an additional scientific subject such as: Physics, Chemistry, Biology, or the like. The most popular additional subject is Physics which was selected by 75% of the students.

The students were asked about their plans for university studies and career direction: 53% of them said that they would like to study CS in the university, or plan to have a CS career. 43% of them said that they didn't decide yet.

Over 90% of the students indicated that they came to compete after being encouraged to do so by their teachers. We allowed participation in the national competition for those interested, even if they did not participate in the regional competition. 35% of the participants were such participants. When asked for the reason for not participating in the regional competition, most of these 35% claimed that the regional competition was (unfortunately) not offered in their schools.

When asked whether they studied for the competitions, 21% of the students replied that they prepared to the regional competition and 37% prepared to the national competition. In both cases, the students learnt from the examples in the Israeli IOI Website, and a few were helped by their teachers.

The students expressed different opinions about the levels of difficulty of the regional and national competitions. Their opinions are displayed in the Table 2.

Table 2
Students' viewpoints of the competitions' levels of difficulty

	Regional competition	National competition
Difficult	21%	74%
Partially difficult	56%	23%
Easy	23%	3%
<i>Interesting</i>	92%	90%
<i>First encounter with such questions</i>	59%	71%

Some student added comments to their evaluation, such as: "I lack the knowledge required for answering such a question", "I did not understand the question", "The question requires a lot of thinking" and "I have never encountered such a question".

We may notice that the vast majority of the students in both competitions found the questions interesting. Some additional comments were: "The questions were very interesting and challenging", "The questions were fun and stimulated creativity", "I like riddles and logic questions".

We took a further look, and examined the top 50 students which were selected to participate in the advanced training stage. Only 27 of them (54%) participated in the regional competition. The average grade of these 27 students in the regional competition was 101 (out of 125).

We examined the background of the top 50 students and compared it with the background of the national competition participants; the results are displayed in the Table 3.

We may notice that the top students are strong in mathematics, and have a strong inclination to Physics. The amount of females in the top group is too low, and we should seek ways to encourage the participation of the most competent girls. Finally, we are glad that the vast majority of the top students are not from 12th grade, yet we should seek ways to increase the amount of younger students in the top group.

Table 3
Comparison of students' background

	Top 50 students	Participants in the national competition
Selected the highest level of mathematical studies (5 points)	100%	87%
Chose to expand their CS studies in secondary school	95%	95%
Chose to expand their hysics studies in secondary school	92%	75%
Females	2%	15%
Students in the 10th or 11th grade	79%	75%

4. Conclusion

We displayed some contents and statistics of our regional and national competitions. For regional competitions, it may be beneficial to carefully examine, how exactly do tasks without programming reflect competence in algorithmic thinking. As for our statistics, one conclusion that evolves from the data is that we should develop additional ways to increase teachers' motivation and involvement in the IOI project. We should increase our efforts to attract more girls. In addition, we should familiarize students with CS challenges, of basic levels, prior to both competitions; and we should find ways to reach younger talented students prior to the time in which they start studying CS in secondary school. This will enable us teaching these students for a longer period of time, and better preparing them for the IOI.

References

- Armoni, M., Benaya, T., Ginat, D., Zur, E. (2010). Didactics of introduction to computer science in secondary school. In: *Teaching Fundamental Concepts of Informatics, Lecture Notes in Computer Science*, Vol. 5941, Springer-Verlag, Berlin, Heidelberg, pp. 36–48.
- Burton, B.A. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.
- Casadei, G., Fadini, B., Genovie, De Vita, M. (2007). Italian olympiads in informatics. *Olympiads in Informatics*, 1, 24–30.
- Diks, K., Kubica, M., Stencel, K. (2007). Polish olympiad in informatics – 14 years of experience. *Olympiads in Informatics*, 1, 50–56.
- Gal-Ezer, J., Beeri, C., Harel, D., Yehudai, A. (1995). A high school program in computer science, *Computer*, 28(10), 73–80.
- Kubica, M., Radoszewski, J. (2010). Algorithms without programming. *Olympiads in Informatics*, 4, 52–66. *Ministry of Education Website*. <http://www.csit.org.il/>.
- Phillips, M. (2010). The New Zealand experience of finding informatics talent. *Olympiads in Informatics*, 4, 104–112.
- The Israeli IOI Website*. <http://www.tau.ac.il/~cstasks>.
- Tsvetkova, M.S. (2010). The olympiads in informatics as a part of the state program of school informatization in Russia. *Olympiads in Informatics*, 4, 120–133.
- Wang, H., Yin, B., Liu, R., Tang, W., Hu, W. (2010). Selection mechanism and task creation of Chinese national olympiads in informatics. *Olympiads in Informatics*, 4, 142–150.
- Zur, E., Benaya, T., Ginat, D. (2010). IOI Israel – team selection, training, and statistics. *Olympiads in Informatics*, 4, 151–157.



E. Zur – is involved in the Israel IOI project since 1997, and repeatedly served as a deputy leader. She holds a PhD Degree in computer science education from Tel-Aviv University. She is a faculty member of the Computer Science Department at the Open University of Israel. She designed and developed several advanced undergraduate computer science courses and workshops, and currently serves as a course coordinator of several courses. Her research interests include distance education, collaborative learning, computer science education, computer science pedagogy, teacher preparation and certification and object oriented programming.



T. Benaya – holds a MSc in computer science from Tel-Aviv University. She is a faculty member of the Computer Science Department at The Open University of Israel. She designed and developed several advanced undergraduate computer science courses and workshops, and she serves as a course coordinator of several courses. She also supervises student projects. She is a lecturer of computer science courses at The Open University of Israel and Tel-Aviv University. Her research interests include distance education, collaborative learning, computer science education, computer science pedagogy and object oriented programming.

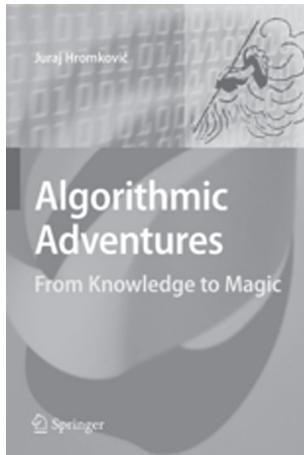


D. Ginat – heads the Israel IOI project since 1997. He is the head of the Computer Science Group in the Science Education Department at Tel-Aviv University. His PhD is in the computer science domains of distributed algorithms and amortized analysis. His current research is in computer science and mathematics education, focusing on cognitive aspects of algorithmic thinking.



O. Becker – is a former IOI contestant and has been involved as a staff member in the Israeli IOI project since 2005. He was Israel's team leader to IOI 2009, IOI 2010, and is currently Israel's team leader to IOI 2011. He is currently in his last year of BSc studies of mathematics in the Hebrew University of Jerusalem.

REVIEWS, COMMENTS



Algorithmic Adventures: From Knowledge to Magic

Author: Juraj HROMKOVIČ
Publishing house: Springer-Verlag
Year of edition: 2009
Language: English
Number of pages: 363
ISBN: 978-3-540-85985-7

What is an algorithm, what can we compute and what might we compute in the future? These are the questions which underline Hromkovič's book *Algorithmic Adventures: From Knowledge to Magic*. It covers a relatively broad range of topics, at an introductory overview level, which originated from a public lecture series at ETH Zürich (Swiss Federal Institute of Technology Zürich) in 2005.

Given the general scope of informatics olympiads, it should be pointed out from the beginning that this is in no form an introductory algorithm text, such as Cormen et al [1] or Sedgewick [2]. This is not the place to find a description (or mention) of standard approaches such as dynamic programming or greedy algorithms, standard algorithms or mathematical foundations. Rather it provides an essentially recreational overview of algorithms – a style that introduces some fundamental concepts and glosses-over some technical details.

The chapters within the book fit together well and tell an interesting story. Indeed, the structure of the book and the way in which the story flows between the chapters is one of the book's strengths. The book starts off by discussing scientific discipline, some basic logic and the notion of proof. We then lead in to the standard comparison of an algorithm and a recipe. This discussion is carried on in some depth, including discussions as to specificity of steps and non-termination.

The story then moves to a discussion of the different types of infinity, which is then used to explain why some problems are unsolvable. At this point (almost half way through the book) complexity is introduced, along with a discussion of what is tractable. Randomness is introduced next, along with witnesses and examples of how some hard problems can be solved with such algorithms.

The last third of the book contains two slightly esoteric topics – namely DNA computing and quantum computers, and two more standard topics – cryptography and online algorithms. These latter two topics make good use of the discussions that have preceded them in the book. The DNA and quantum chapters are interesting and different; perhaps a little too brief in both cases, especially when this is material that is not covered nearly as frequently as the other topics in the book. The DNA chapter, for example, lists various operations that can be performed on DNA (such as concatenation and length separation) and gives an example of a problem (on Hamiltonian paths) solved with these operation but makes no attempt to explain why these are sufficient to solve classical computational problems.

The book does not have nearly as many “milestones” and “spectacular facts” as the author may feel it has, but there are some gems within the text. It finishes on a strong note, with an explanation and proof of an online algorithm for a scheduling problem. That algorithm, and the vertex cover one discussed in the chapter on complexity, are perhaps the two instances in the book of problems which might serve as teasers for a starting informatics olympiad student.

Chapters follow a reasonably consistent format, with some background historical detail, discussion on a few parts of the chapter’s topic, accompanying exercises and a brief summary. Exercises vary in difficulty although most are easy and mechanical, and some of the exercises are answered at the end of the chapters. The choice of exercises to be answered appears erratic and the more interesting problems are typically unanswered.

The book could have done with better editing. There are numerous mistakes (distracting in general rather than misleading), repetitions, inconsistencies and undefined notations. The index is particularly poor; in almost every case where I went to look something up in the index there was no appropriate entry. The general reader might be excused for being frustrated on reading a sentence like “Remember the travelling salesman problem” but finding no reference to it in the index.

In the book’s introduction the author states that “The goal of this book is not typical for popular science writing, which often restricts itself to outlining the importance of a research area. Whenever possible we strive to bring full understanding of the concepts and results presented.” Much of the book fails to live up to this promise. Topics are rarely covered in enough depth to provide a full understanding, and a student who stops here is going to be missing much. Conversely, for the student who wants to get a basic understanding of some key concepts – and especially those who are not mathematicians or computer scientists – the book does provide a starting point.

It is difficult however to see the audience that would be best suited to the book¹. To the mathematician or computer science the book is far too shallow; perhaps suitable as reading for such a student, in their own time, prior to the commencement of a course. To the recreational science reader, quite simply there are far better books in this category out there. Infinity, complexity, randomness, cryptography, etc... are all frequently

¹As discussed in [3] this book has been used as the text in a course for second-year university students on the foundations of informatics as a science. That course, was for students “*from various disciplines, excluding computer science.*”

topics, handled well in the recreational mathematics literature. Perhaps the general, non-computer scientist might find parts of the book of interest, if they are able to persist. Unfortunately, the book's attempt to "lure the reader from her or his passive role" and "do some work by solving appropriate exercises" seem more likely to alienate than educate.

1. Cormen, T., Leiserson, C., Rivest, R., Stein, C. (??? year). *Introduction to Algorithmics*, MIT Press.
2. Sedgewick, R. (??? year). *Algorithms in C*, Addison Wesley.
3. Verhoeff, T. (2010). An enticing environment for programming. *Olympiads in Informatics*, 4, 134–141.

Richard Forster



Methods of Tasks Solving in Informatics. International Olympiads

Authors: Vladimir M. KIRYUKHIN
and Stanislav M. OKULOV

Publishing house: BINOM. Knowledge Lab (LBZ)

Country, city: Russian Federation, Moscow

Year of edition: 2007

Language: Russian

Number of pages: 600

ISBN: 978-5-94774-680-8

Website: <http://www.lbz.ru/katalog/products/literatura-dlya-shkol/informatika/olimpijskie-vysoty/metodika-reshenija-zadach-po-informatike-699>

An important place in preparing for the International Olympiad in Informatics is the methodical and informational support of all kinds and forms of work with gifted students. Since the olympiads themselves are just an evaluation of the preparatory process results, the systematic work with gifted students on development of their abilities is a key to the olympiad movement in informatics. And here there are a lot of questions such as – How to analyze complex and competition tasks in Informatics? How to explain their solution to students?

Traditional publications on this subject are usually based on the principle: do thus and thus and you'll get the result. What preceded this "doing so" is usually not disclosed by the authors of publications; this remains in the shadows. Moreover, in this case the students do not get the return of obtaining of correct solutions by themselves because all is explained about it to them. In this book, on examples of the tasks which were offered at the IOI between 1989–2006, the authors have attempted to show the problem-solving process from beginning to end. This proved to be much more complicated than simply

describing a solution in the form of algorithm description or program code (even with the proofs of mathematical facts, if they lie at the basis of the solution).

There is a viewpoint that the ordinary informatics teacher is not under force to prepare students for IOI, that this work is a lot of "elite". The book's authors disagree with this position, since they proceed from the fact that the main thing in working with gifted children is to establish a creative environment. Of course, a teacher or coach must be purposefully working on these problems, but to be "Guru" in tasks solving is not necessarily! Surprising but at the basis of almost any complexity is simplicity. To see and find this simplicity, to go from this simplicity to complete solution of the task in the process of joint activity with the students – this is the basis of work of teacher or coach. To learn how to do it, the authors have tried to write in this book, choosing the IOI competition tasks for this purpose.

In preparation for the IOI an informatics teacher still faces the same problem: What to teach? How to teach? What books and information resources to use? In spite of the increased capabilities of information technology, these problems have remained, as it remains an unchanged mission for the teacher – to develop the creative abilities of student in informatics. It has remained unchanged and a criterion for evaluating the work of the teacher as the successful performance of his students on the IOI.

Since the IOI is a very popular olympiad among students of many countries, in the book authors are talking about the solution of the IOI competition tasks, more precisely, the method of analysis of complex tasks in informatics, because on the status the IOI offers just such competition tasks. The tasks statements in this book are as close as possible to the original versions proposed at the IOI. Cuts and changes are minimal, which enables them to represent both the development contents and requirements for students.

The book provides analysis of all the IOI competition tasks from 1989 to 2006. Where there is enough material for a lesson, the authors discuss its analysis. In other words, using this material the teacher can "construct" a lesson. Otherwise, if the information is insufficient or the theory is too complicated for a student to understanding, the authors just talk about the ideas for solving it. The book ends with "a guide to tasks", which provides a classification of tasks both by complexity and topics.

It is clear that for full preparation for the IOI it is necessary to organize a creative environment and to use modern methods for the development of gifted students, but this problem is beyond the scope of this book. In this book the authors are talking about the contents aspect of teaching and methods of teaching. Importantly, this is considered not in separate chapters, not as common considerations about "what is good and what is bad", but presented with the concrete material during the discussion of each task. Instead of the traditional parsing of tasks or a general description of the algorithm to solve them the authors have tried to give a summary of studies with gifted students to teacher or coach and have tried to show the features of teaching during lessons, because all this laid down in the very structure of the discussion of each task.

Coming back to contents of teaching process, one can say that analysis of each task leads to the need to study these, or other topics in informatics, in more detail. But it is not possible to indefinitely extend the number of these topics in this part of working with

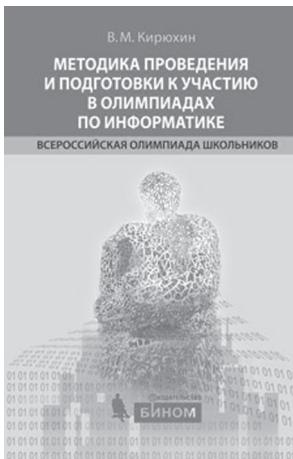
the students, because it is the problem of universities. In the book there are highlighted only those topics that are fundamental in informatics, and the material for learning them is sufficient in the scientific literature. In particular, such topics are:

- combinatorial calculus;
- sorting and search;
- algorithmic strategies;
- algorithms for graphs;
- string processing;
- dynamic programming;
- algorithms of computing geometry.

In the presented book "Guide" each task is classified to the one of these topics. In addition, a number of other topics are identified: "tasks on the idea", tasks on the programming techniques, "task without continuing". Due to this a teacher or coach has the opportunity to select a task, necessary for him to carry out lessons, and a student can select the task for his preparation, depending on the topic and the level of his development (the "Guide" provides an assessment of the complexity of each task). Thus, the "Guide" allows readers to organize the work with the book not on the level of easy familiarity and sequential read, but permanent use in preparation for the IOI.

It should be noted that in working with this book the process of considering tasks is not confined to the reading of their discussions. Readers are also required to analyze different approaches to solving tasks and use algorithms, the development of software code, testing of solution program and study the complexity of the solution. All this determines all the necessary conditions for fruitful preparation of students for the IOI and successful performance at it.

Sergey A. Beshenkov



Method of Carrying out and Preparation for Participation in the Olympiad in Informatics: All-Russian Olympiad

Author: Vladimir M. KIRYUKHIN

Publishing house: BINOM. Knowledge Lab (LBZ)

Country, city: Russian Federation, Moscow

Year of edition: 2011

Language: Russian

Number of pages: 271

ISBN: 978-5-9963-0464-6

Website: <http://lbz.ru/katalog/products/literatura-dlya-shkol/informatika/olimpijskie-vysoty/metodika-provedenija-i-podgotovki-k>

Russia has carried out many different olympiads and contests in informatics and information technology for students of secondary and high school. Among them the Russian Olympiad in Informatics (RusOI) has a special place, both in the number of participants, and by its influence on the development of their intellectual potential. For many years, this olympiad has implemented and improved innovative methods of search and supporting gifted students, regardless of where they live and study.

The RusOI has been held by the Ministry of Education and Science of Russia every year since 1988 in four stages: school, municipal, regional and final. The school stage involves all interested 5–11th grade students from all schools in the country (there is an eleven-year secondary education system in Russia). Only the winners and prize winners of previous stages participate in subsequent stages. Up to the final stage only about 200 students from the 9–11th grades are allowed, who demonstrated the best results among all the participants of the regional stage, which takes place simultaneously in all regions of Russia; using the same sets competition tasks for all participants of the regional stage.

This book presented the long experience of organizing and holding the RusOI, from school stage and ending with the final stage. The organization of each stage, their technical and technological equipment, formation of sets of competition tasks for participants of all ages are discussed in detail.

Particular attention in this book is paid to how to prepare for different kind of olympiads in informatics. Methods of preparation are considered that are not based on solving largest possible number competition tasks, but on developmental teaching, on forming individual trajectories of learning using modern techniques of developing the creative capacity of students. An important place is occupied here by the content of competition tasks as well as the use of modern information technology.

The book consists of five chapters. The first chapter describes the main features of the organization and holding for all stages the RusOI. There are considered in detail issues concerning the composition of participants, dates, forms and procedure of competition, as

well as the order of summarizing of each stage. Particular attention is paid to the school stage, because how this stage was organized and held depends in many respects on the success of further development of the olympiad movement in informatics.

The second chapter is dedicated to the organizational and technical support of all stages the RusOI. All the governing bodies of the RusOI and the basic functions they perform are considered. It also discusses in detail the requirements for the hardware and software of each stage.

The third chapter deals with the system of competition tasks. There are considered in detail methodological principles and characteristics of competition tasks, especially tasks for school, municipal, regional and final stages, formation of sets of tasks, as well as communication competition tasks with the Russian State School Educational Standard and a guide to competition tasks.

The fourth chapter contains a description of testing techniques and systems for evaluation of competition tasks solutions common to all stages of the RusOI. Particular attention is paid to methods of evaluation of participants solutions, including methods for determining violations of time and memory limits for testing programs. There are also considered features of evaluation of participants solutions in the final stage, since the process at this stage is the most established, and the organizers of other stages the RusOI should be familiar with the experience of its implementation and using this experience in their work.

The fifth chapter is devoted to consideration of most important issues related to preparations for olympiads in informatics, which are useful as participants of various stages of the RusOI, and their teachers and coaches. Particular attention is paid to developing teaching system as the basis for olympiad preparation and individual forms of preparation and self-preparation of students. The book also contains useful Internet resources for olympiad preparation that can be successfully used in various forms of preparation for olympiads in informatics.

The book will be very useful as organizers of competitions in informatics, and students who are interested in olympiads in informatics, as well as their parents, school teachers and coaches working with talented students. It will undoubtedly contribute to further development of the International Olympiad in Informatics and involvement for new students and countries.

Sergey A. Beshenkov

Instructions to Authors

OLYMPIADS IN INFORMATICS is a peer-reviewed scholarly journal that provides an international forum for presenting research and developments in the specific scope of teaching and learning informatics through olympiads and other competitions. The journal is focused on the research and practice of professionals who are working in the field of teaching informatics to talented student. OLYMPIADS IN INFORMATICS is published annually (in the summer).

The journal consists of two sections: the main part is devoted to research papers and only original high-quality scientific papers are accepted; the second section is for countries reports on national olympiads or contests, book reviews, comments on tasks solutions and other initiatives in connection with teaching informatics in schools.

The journal is closely connected to the scientific conference annually organized during the International Olympiad in Informatics (IOI). OLYMPIADS IN INFORMATICS is abstracted/indexed by

- Central and Eastern European Online Library (CEEOL),
- EBSCO.

Submission of Manuscripts

All research papers submitted for publication in this journal must contain original unpublished work and must not have been submitted for publication elsewhere. Any manuscript which does not conform to the requirements will be returned.

The journal language is English. No formal limit is placed on the length of a paper, but the editors may recommend the shortening of a long paper.

Each paper submitted for the journal should be prepared according to the following structure: 1) a concise and informative title; 2) the full names and affiliations of all authors, including e-mail addresses; 3) an informative abstract of 70–150 words; 4) a list of relevant keywords; 5) full text of the paper; 6) a list of references; 7) biographic information about the author(s) including photography.

All illustrations should be numbered consecutively and supplied with captions. They must fit on a 124 × 194 mm sheet of paper, including the title.

References cited in the text should be indicated in brackets, e.g., for one author – (Johnson, 1999), for two authors – (Johnson and Peterson, 2002), for three or more authors – (Johnson *et al.*, 2002). If necessary, the page number may be indicated as (Johnson, 2001, p. 25).

The list of references should be presented at the end of the paper in alphabetic order. Papers by the same author(s) in the same year should be distinguished by the letters a, b, etc. Only Latin characters should be used in references.

Please adhere closely to the following format in the list of references.

For books:

Hromkovič, J. (2009). *Algorithmic Adventures: From Knowledge to Magic*. Springer-Verlag, Berlin.

Schwartz, J.E., Beichner, R.J. (1999). *Essentials of Educational Technology*. Allyn and Bacon, Boston.

For contribution to collective works:

Batissta, M.T., Clements, D.H. (2000). Mathematics curriculum development as a scientific endeavor. In: Kelly, A.E., Lesh, R.A. (Eds.) *Handbook of Research Design in Mathematics and Science Education*. Lawrence Erlbaum Associates Pub., London, pp. 737–760.

Plomp, T., Reinen, I.J. (1996). Computer literacy. In: Plomp, T., Ely, A.D. (Eds.) *International Encyclopedia for Educational Technology*. Pergamon Press, London, pp. 626–630.

For journal papers:

McCormick, R. (1992). Curriculum development and new information technology. *Journal of Information Technology for Teacher Education*, 1(1), 23–49.

Burton, B.A. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.

For documents on Internet:

International Olympiads in Informatics (2011).

<http://www.IOInformatics.org/>.

Bebras – International Contest on Informatics and Computer Fluency (2011).

<http://bebras.org/en/welcome>.

Authors must submit electronic versions of manuscripts in PDF to the editors. The manuscripts should conform all the requirements above.

If a paper is accepted for publication, the authors will be asked for a computer-processed text of the final version of the paper, supplemented with illustrations and tables, prepared as a Microsoft Word or LaTeX document. The illustrations are to be presented in TIF, WMF, BMP, PCX or PNG formats (the resolution of point graphics pictures is 300 dots per inch).

Contacts for Communication

Valentina Dagiene

Vilnius University Institute of Mathematics and Informatics

Akademijos 4, LT-08663 Vilnius, Lithuania

Phone: +370 5 2109 732 Fax: +370 52 729 209

E-mail: valentina.dagiene@mii.vu.lt

Internet Address

Olympiads in Informatics provides an early access to the articles by placing PDF files of the papers on the Internet. All the information about the journal can be found at:

http://www.mii.lt/olympiads_informatics

Olympiads in Informatics

Volume 5, 2011

D. GINAT. Algorithmic Problem Solving and Novel Associations	3
L. HAKULINEN. Survey on Informatics Competitions: Developing Tasks	12
K. IMAJO. Contest Environment Using Wireless Networks: A Case Study from Japan	26
G. JAKACKI, M. KUBICA, T. WAŁĘŃ. Codility. Application of Olympiad-Style Code Assessment to Pre-Hire Screening of Programmers	32
V.M. KIRYUKHIN, M.S. TSVETKOVA. Preparing for the IOI through Developmental Teaching	44
D. KOMM. Teaching the Concept of Online Algorithms	58
T. KULCZYŃSKI, J. ŁĄCKI, J. RADOSZEWSKI. Stimulating Students' Creativity with Tasks Solved Using Precomputation and Visualization	71
K. MANEV, N. NIKOLOV, M. MARKOV. Reconstruction of Trees Using Metric Properties	82
M. MAREŠ. Fairness of Time Constraints	92
B. MERRY, C. HULTQUIST. Measuring the Startup Time of a Java Virtual Machine	103
P.S. PANKOV, K.A. BARYSHNIKOV. Tasks of "Mission Impossible" and "Mission Impeded" Types	113
T. VERHOEFF. Beyond the Competitive Aspect of the IOI: It Is All about Caring for Talent	120
REPORTS	128
A. CEPEDA, M. GARCIA. Mexican Olympiad in Informatics	128
S. COMBÉFIS, D. LEROY. Belgian Olympiads in Informatics: The Story of Launching a National Contest	131
M. ITALIANI. Italian Olympiad in Informatics: 10 Years of the Selection and Education Process	140
J. KOIVISTO. The National Computer Olympiads and the IOI Participation in Finland	147
K. MALAIVONGS. Preparing Students for IOI: Thailand Country Report	150
P.S. PANKOV, T.R. ORUSKULOV. Kyrgyzstan National Report on Olympiads in Informatics	155
E. ZUR, T. BENAYA, O. BECKER, D. GINAT. Israel: The Regional and National Competitions	161
REVIEWS, COMMENTS	169