Olympiads in Informatics, 2011, Vol. 5, 131–139

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

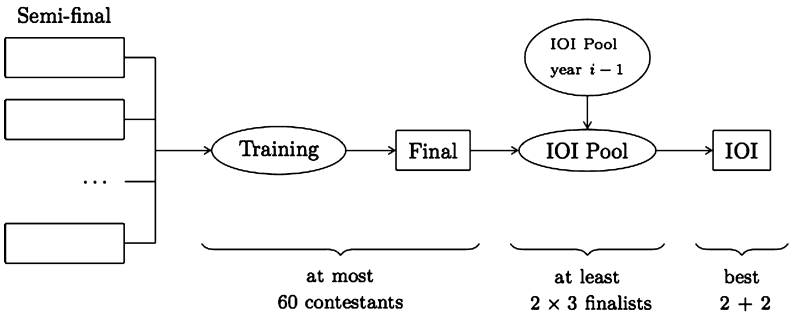


Fig. 1. Structure of the be-OI contest.

and algorithmic problems. At most 60 of these contestants are selected for the final. That exam is an exclusively pen-an-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 later 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 task 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 to 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 f 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).

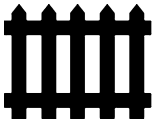
ming language to be better than other ones. The grading steps are estimated so as to differentiate the main classes of solutions, e.g., $O(n)$ versus $O(n^2)$ versus $O(2^n)$. An improvement of the solution is anyway preferred to better performances.

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

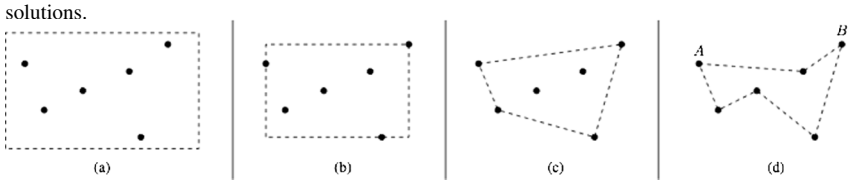
Fence problem



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.

- $3 <= N <= 100.000$.
- $0 <= X,\ Y <= 20.000$, 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.

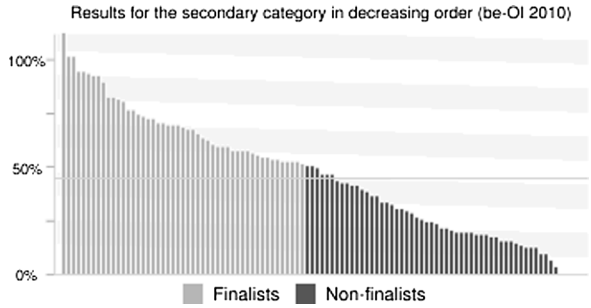


Fig. 5. Histogram with the scores of the semifinalists (be-OI 2011).

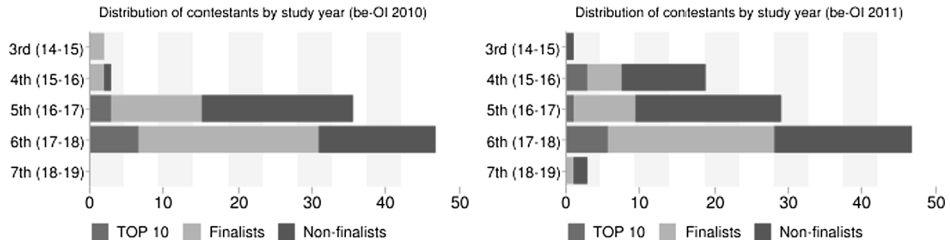


Fig. 6. Scores of the semifinalists grouped by study year. Between parenthesis are the corresponding ages.

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.