Olympiads in Informatics

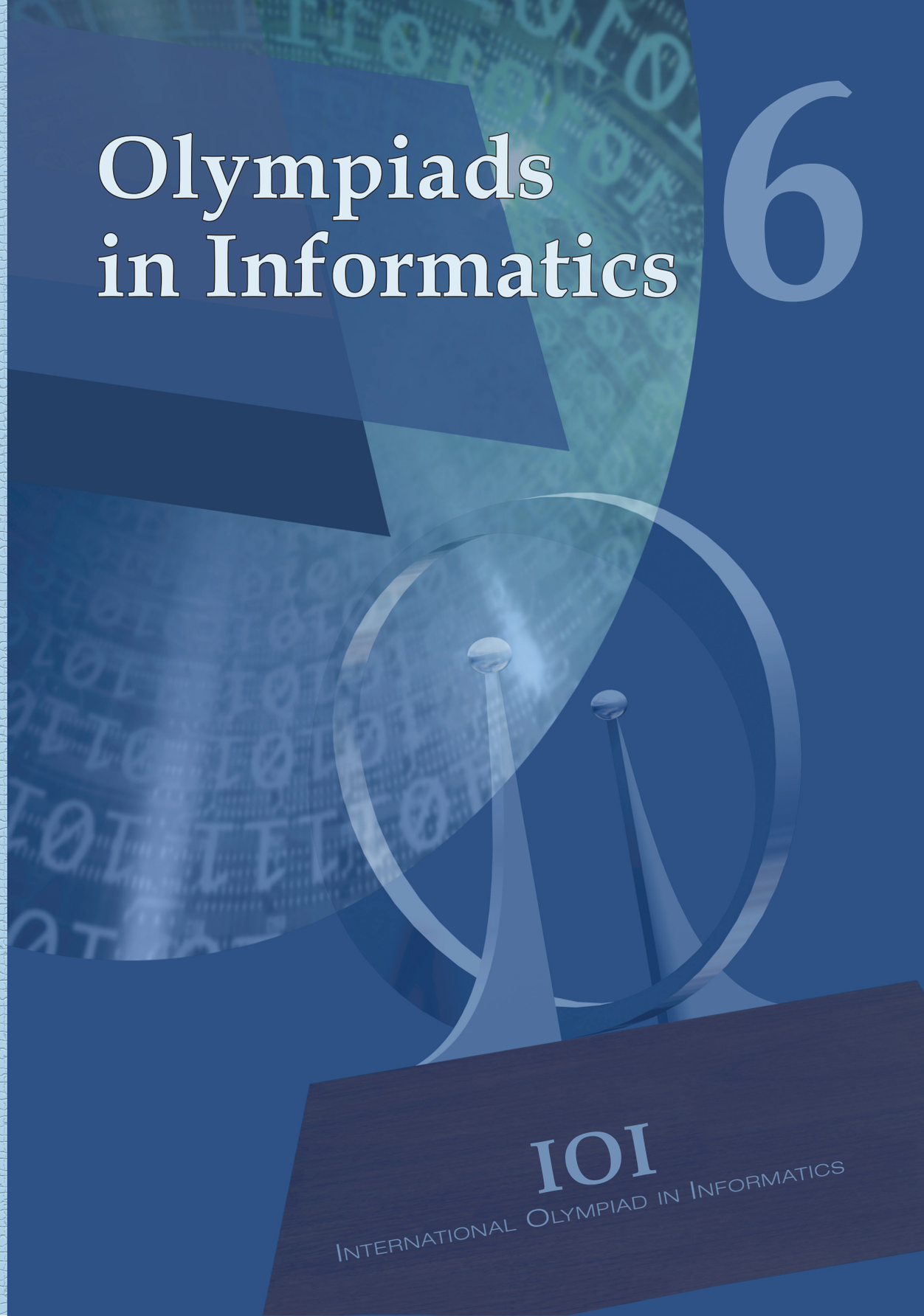
Volume 6, 2012

ISSN 1822-7732

9 771822 773007

Olympiads in Informatics Volume 6, 2012

Olympiads in Informatics 6



ISSN 1822-7732

INTERNATIONAL OLYMPIAD IN INFORMATICS

VILNIUS UNIVERSITY
INSTITUTE OF MATHEMATICS AND INFORMATICS

OLYMPIADS IN INFORMATICS

Volume 6 2012

Selected papers of
the International Conference joint with
the XXIV International Olympiad in Informatics
Sirmione – Montichiari, Italy, September 23–30, 2012

OLYMPIADS IN INFORMATICS

ISSN 1822-7732

http://www.mii.lt/olympiads_in_informatics

Publisher: Vilnius University Institute of Mathematics and Informatics
Akademijos 4, LT-08663 Vilnius, Lithuania

Olympiads in Informatics, 2012, Vol. 6, 1–2

Foreword

Writing the introductions to the successive volumes of *Olympiads in Informatics* gives a moment to reflect. Typically on the changing face of informatics olympiads and education but also informatics and wider society in general. In the time between the previous (5th) conference and this one, back in October 2011, one of the pioneer computer scientists passed away. This was someone whose work had a noticeable influence on most informatics olympiads, indeed on much of computing as we know it today, and someone whose death whilst making the obituary columns in many major papers rarely made it any further. Quickly forgotten, or passing under most peoples' radar, a man without whom the current IT landscape would look somewhat different.

We refer, of course, to *Dennis Ritchie* whose contributions included such things as the UNIX operating system and the C language. It is perhaps telling that almost all informatics olympiads – excluding perhaps those aimed at a relatively junior level – include C or its successors as programming options. On receiving the US National Medal of Technology he, along with his co-recipient, said "*We take this award to us as symbolic of the seminal contributions of our immediate colleagues . . . many of the most interesting ideas were not ours, but generated in collaboration*".

It would be easy to focus on the "*collaboration*" statement. We are, after all, a conference so our very existence relies on collaboration within our community. Individuals to write the papers, review them, edit them and of course read and use them, without which they are but "*sound and fury, signifying nothing*". Rather, let us reflect on the "*seminal contributions*". Our field is a relatively young one, one which leaving aside its mathematic roots (claiming the 300 BCE Euclidean algorithm as evidence for computer science existing for over 2 millennium would be stretching a point) and some programable machines in the 1800s, really kicked off in the 1930s. Some of the early pioneers are still amongst us and many of us have worked with and known others.

Our up-and-coming computer scientists have that opportunity to be pioneers in the field. Our informatics olympiads are a means for capturing and enthusing potential students; to quote the IOI regulations "*To bring the discipline of informatics to the attention of young people*". We all have different requirements when putting together our olympiads. We want to attract a wide range of students, introduce real computing to students who otherwise only see the end results; the games they play and the word processing they are 'taught' in schools. We are also there though for our best students – our participation at global olympiads proves that point. How do we encourage those students? Our best students are often the best students in a multitude of subjects – why should they choose informatics? How do we grab our new pioneers?

Informatics in general is young and the teaching of informatics, plus teaching with informatics, younger still. We have opportunities, in many of our countries, to have an impact – positive or negative – on the way students are taught. Get it right and paradigms for future teaching will be set; a pioneering footstep. Get it wrong and we will just a part of a passing trend; an inevitable footnote.

This volume is quite extensive: we are publishing 22 papers and reports. The format for the journal follows three tracks: the primary section of the journal focuses on research (the acceptance rate is around 40%); the second report section is devoted to sharing our national experiences – potentially of less interest to those outside the community; the last section presents some book reviews which we hope will be of interest to our readers. We are pleased that this volume succeeded in getting national reports from various countries. Again, we would like to encourage writing 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, Jonas Blonskis, Ben Burton, Giorgio Casadei, Sćbastien Combéfis, Hugo Duenas, Gerald Futschek, Gintautas Grigas, Sari Haj Hussein, Ville Leppänen, Krassimir Manev, Timo Poranen, Rhein Prank, Jūratė Skūpienė, Peter Taylor, Ahto Truu, Troy Vasiga, Willem van der Vegt, and Peter Walker.

Last, but by no means least, particular thanks are due to the organisational committee for IOI'2012 in Italy without whose assistance we would be unable to hold the conference. Their assistance, during what is an already busy period, is gratefully received.

Editors

Olympiads in Informatics, 2012, Vol. 6, 3–20

The Role of Contests in Changing Informatics Education: A Local View

Giorgio AUDRITO[1], G. Barbara DEMO[2], Elio GIOVANNETTI[2]
[1]*Dipartimento di Matematica, Università di Torino*
via Carlo Alberto, 10-10123 Torino, Italy
[2]*Dipartimento di Informatica, Università di Torino*
corso Svizzera, 185-10149, Torino, Italy
e-mail: giorgio.audrito@gmail.com, {barbara, elio}@di.unito.it

Abstract. Changing the teaching of informatics in secondary (and primary) education is in many countries difficult to achieve, owing to several reasons: the natural tendency of institutions and people to stick to well-established practices, a distorted view of informatics prevailing in society and among those who take the decisions, the specific fact that in several school systems the teaching of informatics is largely committed to teachers of other scientific disciplines. We describe the Italian situation and our experience in Turin where we have organised training stages for the Olympiad in Informatics and supported the organisation of other informatics contests. On its basis we argue how the different kinds of competition can help to change informatics education in Italy, by making problem solving and programming play a central role.

Key words: competitions, problem solving, programming in education, data structures, algorithms, programming techniques.

1. Introduction

In the last few years the need for a change in curricula, learning objectives and teaching techniques of computer-related disciplines in (primary and) secondary education has emerged in the public debate in a number of countries. The recent speech of the British Secretary of State for Education Mr. Michael Gove at BETT 2012 (BETT is the annual British Educational Training and Technology show) is a good example of governmental awareness on the subject. As is now widely recognized, time has come to replace the teaching of the so-called ICT with a real computer science curriculum, with emphasis on the word "science"; if we adopt the term traditionally used for computer science in continental Europe, but now also in UK, we can say that we must replace ICT with Informatics, or at least to promote a change in that direction.

In Italy too there has been a great discussion involving the academic and industrial computer scientists' communities, particularly during the months when the recent reform of secondary education was worked out, and eventually came into effect in September 2010. The reform, though introducing some positive novelties, is still far from satisfactory from the above point of view; nevertheless, the hope is that the general framework provided by the law may still be filled with some contents of the desired type.

In May 2010 the national associations of academic computer scientists of Science Faculties and Engineering had published a manifesto for informatics in the reform of secondary school (CINI, GRIN, GII, "Manifesto sull'Informatica nella riforma della scuola superiore", 2010), which is still the basis for the present efforts to change the way informatics is (not) taught in most Italian schools. It distinguishes three different, though related, meanings of the word "informatics":

1. *operational*: the set of all software and hardware objects;
2. *technological*: the technology that allows to make such objects;
3. *cultural*: the scientific discipline that is the foundation of such technology and thus makes it possible.

Informatics in the first meaning is what is sold in shops and department stores, and is for most people the main meaning of the word. The technological aspect is usually recognized as important, though often associated with computer geeks. As for the scientific aspect, few are aware even of its existence; also some scientists from the more traditional disciplines doubt that informatics is a separate science in its own right and not, at best, a (trivial) part of mathematics.

As has been repeatedly stated by many influential computer scientists, non-technical secondary education ("liceo" in Italy) should privilege this third aspect, giving informatics as a science the same dignity and importance as mathematics or physics, because of the essential contribution it can carry to a person's intellectual and practical formation. For example, in a world increasingly ruled by algorithms of the most diverse kinds, the very notion of algorithm and the basic principles of algorithmics should be familiar to everybody with a sufficient level of culture, exactly as any pupil is supposed to become familiar with the basic laws of physics.

On the contrary, in this kind of schools, like in society, the view of informatics as restricted to the first or at best the second aspect is still largely prevailing, also owing to the scarcity of teachers with a computer science degree (graduates in informatics tend to find jobs in industry or services and are not traditionally oriented to school-teaching careers).

To change school curricula has always been hard, as already observed by Papert many years ago (Papert, 1997); unfortunately, this is still true nowadays, particularly in informatics, for several reasons. On the one hand, as we have observed above, among those who according to the rules of the reform are going to teach informatics many are teachers of other disciplines, without specific competences in the subject. On the other hand, the computer science community can hardly compete with much older scientific communities, such as those of mathematics and physics, which have a long tradition in didactic research and a well established and organized presence and curricula in secondary education.

In this situation, the idea of using competitions as a lever to introduce in our schools "sideways" what is hard to be obtained directly, has gained in the latest years a growing consensus. Starting from an increased interest by many institutions in the traditional Olympiad in Informatics, several new competitions have been introduced over the years in many countries, and recently also in Italy: the Olympiad in Problem Solving, Kan-

gourou, Bebras, the ScratchDay, ZeroRobotics, RoboCUP, etc. Like the Olympiad in Informatics, they are organized at different territorial levels, and are therefore able to reach a great number of teachers and students. How to make them most effective for the required change of informatics education is a subject of discussion, to which we would like to give, in the following, the small contribution of our experience.

In the next section we briefly describe the situation of informatics teaching in Italian secondary schools, and in the short Section 3 we comment on a natural role of competitions. In Section 4 we focus on the Olympiad in Informatics, while Section 5 describes other competitions, less aimed than Olympiad at discovering and cultivating excellence. In Section 6 we comment on the central role of programming induced by competitions. In Section 7 we draw some general conclusions.

2. Informatics in Italian Secondary Schools: Short History and Perspectives

The Italian system of secondary education consists of two levels, the lower secondary school or "Scuola Media", 11–13, and the upper level, 14–19, which is divided into three different channels: the general channel or "liceo", analogous to the French "lycée" or German "Gymnasium", the technical school type, and the vocational school type. The general channel, in turn, is divided into the scientific and the humanistic sub-channels, while the technical and vocational channels are obviously divided into the various technological specialisations.

Differently from most other countries, the subjects and their weights in numbers of hours are rigidly fixed within each kind of channel or sub-channel, with no possibility of free choice of optional subjects on an individual basis. However, in the pre-reform system a great number of variants had developed over the years within each established kind of school, and it had eventually grown into an almost inextricable but extremely wide spectrum of education programmes, thus putting a remedy to the formal rigidity of each of them. Many of these programmes were largely appreciated by pupils and families. The reform system has pruned the number of alternatives, in some cases by taking as a new standard for a channel what previously was a sub-sub-channel that had proved to be particularly successful; in other cases it has cut possibilities in a way that was strongly criticised by teachers.

Informatics as a distinct and autonomous subject in Italy has been traditionally present only in the technical schools for informatics, and – with a much lower importance – in other technical types of school. On the contrary, it has been almost completely absent in the general channel or "liceo".

As a matter of fact, the so-called National Programme for Informatics ("Piano Nazionale Informatica" or PNI) was launched in non-technical secondary education almost thirty years ago, but it was a programme mainly conceived by mathematicians as instrumental to a strengthened teaching of mathematics, and committed to mathematics teachers, whose competence in informatics was usually poor. Nevertheless, in the pre-internet era, when few or no software aids to mathematics teaching were available, PNI

meant introduction to programming, usually with Pascal, which was certainly, for the first time in that kind of schools, a pupils' (and teachers') exposure to informatics proper. Over the years, however, when tools like "Derive" or "Cabri" became of universal use, the need of even a limited capacity of programming in a general-purpose language disappeared, and informatics reduced to the use of specialized tools. Also the widespread offering, in schools, of the ECDL (European Computer Driving Licence) certification – though initially valuable – strongly contributed to the vision of computer science as the mere usage of software tools.

With the reform not much has changed with respect to informatics. The new "liceo scientifico" has two separate curricula: the first one, called "traditional", is similar to those previous special programmes that exhibited a strengthening of mathematics and scientific subjects, and like those includes not well-defined elements of informatics as part of the mathematics syllabus; the second curriculum, called "Applied Sciences" ("Liceo delle Scienze Applicate'), is also similar to a former programme ("Liceo Scientifico Tecnologico"), and features no Latin, a lighter weight of philosophy and history, a still increased weight of all the traditional scientific subjects, in particular physics and biology, and informatics as a separate subject, whose teaching is allowed to mathematicians and physicists besides computer scientists.

3. Algorithms and Creativity: A Role for Competitions

Teachers of scientific disciplines (both in secondary and in higher education) have witnessed in the last decades a growing request – from students and pupils – of precise and detailed "recipes", i.e., algorithms, for solving the problems and doing the exercises that are submitted to them. Many students seem to implicitly ask for a precise catalogue of all the possible problem schemes, with a solving algorithm for each of them: all they wish to learn is to execute such algorithms after putting in the initial data, as human computers. This parallels, not surprisingly, the general evolution of society where, as observed in Section 2, algorithms are going to permeate every aspect of life, from the famous financial algorithms blamed of being a cause of the present crisis, to medical procedures and protocols, to bibliometric methods for the evaluation of research, etc.

At the same time, and somewhat paradoxically, there is among pupils and students a widespread search for creative activities outside school, as in the adult society there is a search for creative jobs in contrast with routine work (the word "routine" used to indicate, in the early days of computer science, a program procedure). In particular, pupils of non-technical schools who approach informatics for themselves are mostly driven by a specific goal: to build their own computer game. Also our average undergraduate student of informatics, when it comes to the obligatory final stage with industry seems to show, according to the external tutors' reports, autonomy, originality and creativity, i.e., exactly those qualities they were lacking during the previous study years. It is presumably a problem of motivation, and of personal adequacy to the level of difficulty.

So, it is also not surprising that a number of pupils in any kind of school are attracted and strongly motivated by competitions, where they can test themselves on more cre-

ative tasks than the ones usually assigned in normal schoolwork. The challenge is how to exploit competitions for the needed change in education systems.

4. The Olympiad in Informatics

4.1. *General Characteristics and Present Trends*

The Italian Olympiad in Informatics (Olimpiadi Italiane di Informatica, OII) has, like their analogous in other countries, the ultimate aim of selecting the national team for the International Olympiad in Informatics (IOI). The whole selection process is organized on three different levels: a school level, which selects a fair number of pupils from every participant school, a regional level, which in each region selects a small number of pupils out of the previous phase, and finally the national level which selects the national team. For a detailed analysis, with statistical data, of ten years of Olympiad in Italy, see Italiani (2010) and Scarabottolo (2011).

It is interesting to single out the different characterizations of the three levels, and to recognize through the years some evolution trends, distinct for the different levels. First of all, we must remark that school-level tests also contain logical and mathematical problems and exercises besides programming problems, so as to be able to select potentially good candidates for the following phases even in that majority of schools where informatics is not present in the curriculum. The regional and the national tests, on the other hand, consist exclusively of programming problems, to be solved in the official languages of IOI (C or C++, besides Pascal).

At the regional level the difficulty, after a considerable increase in the years from 2000 to 2005, has settled on a kind of programming problems that often require good competences on the use of recursion and/or on graph algorithms. However, since at this level there are no execution time limits for the submitted solutions, often problems may be solved by brute-force methods, thus discouraging algorithmic creativity.

At the national level, on the other hand, after an analogous, though less pronounced, increase of difficulty, in the last years the extent of algorithmic competences has been narrowed, with the intent to reduce the advantage of informatics technical schools with respect to the other schools. So in the last editions dynamic programming problems, backtracking, minimum path or spanning tree or flow graph problems have no longer turned up. The effect of this has been a shift of the difficulty towards the mathematical aspect, with the introduction of types of problems whose solutions have little algorithmic content but are based on the results of preliminary non-trivial mathematical reasoning, often of combinatorial nature.

As an example of the latter trend we may cite the problem "A superheroes' school" of the national 2010 contest, where one requirement is the organization of a tournament with $\mathbf{2}^n$ participants in $2^n - 1$ rounds, so that each superhero meets every other: the solution lies in the fact that at the **k**th round the superhero **j** met by the superhero **i** is given by **j = i XOR k** (bitwise exclusive-or).

As one can see, these trends, which in some way go against the characterization of computer science as an autonomous discipline, stem from the absence of computer programming and algorithmics in Italian general education, and are therefore a symptom of the very disease they should help to treat.

4.2. *The Role of the Olympiad in Italian Secondary Education*

Informatics Olympiad, like all homologous initiatives, e.g., the Mathematical Olympiad, seeks individual excellence, which is extremely important, but one may wonder what their impact is on the general level of education, in the same way as one may ponder on the difference between searching for prospective sport champions and ensuring the possibility of a healthy sport practicing for the greatest number.

Also, the Olympiad is naturally centred on algorithmics, which as remarked above is certainly at the core of computer science, but does not exhaust it: other areas are at least as important, for example, abstraction and modelling of reality, concurrent and interactive programming, multidisciplinary informatics, just to name a few.

Algorithmic problems, however, are very good examples of tasks requiring a disciplined creativity, and may therefore be an answer to pupils' demands, at least in principle. As a matter of fact, we explain in our courses that there cannot exist an algorithm which, given a specification, i.e., the description of a computational problem, is able to derive a solving algorithm for it. The big issue is how to present such problems in an attractive and motivating way, not just as another kind of abstract exercises completely disconnected from reality.

The imaginative language in which Olympiad problems are traditionally formulated is a possible help, though to a university professor it may look a bit childish, or a useless and misleading disguise of a simple problem. Actually, at a more thorough reflection, this very disguise has a precise educational value for prospective computer scientists, who in their professional activities are often faced with the task of extracting from the customer's requirements, expressed in a concrete and often confusing language, the underlying abstract computational problem (which may sometimes be a well known problem with a known solving algorithm).

The organisation at three levels (school, region, country), common to most kinds of competitions, is useful since each participant may thus find a level roughly adequate to her/his abilities, instead of having to achieve a compulsory – and equal for all – level of competence, as in ordinary school learning. The drawback is that pupils that have poor performances in school contests or even in regional contests may feel frustrated and may develop a feeling of rejection for the whole discipline, even though, in some cases, their school proficiency is good. However, this is not necessarily a bad thing, since it may help pupils to assess their capacities and talents more precisely.

On the other hand the Olympiad experience, either in mathematics or in informatics, certainly has, for successful participants, a strong impact on the choice of the university course: they tend to enrol at Mathematics or Informatics respectively, while otherwise they would have probably chosen Engineering, especially in a city like Turin where the

renowned Polytechnic University also attracts students that would be better suited to other courses. This clearly emerges from the answers to the questionnaires submitted to pupils at the end of Olympic training stages and from the enrolment data in the following years.

4.3. *The Training Stages at the University of Turin*

The Department of Informatics of the University of Turin, among its various activities of popularization and promotion, in the last four years has organized a training stage for the regional contest, addressed to the pupils got through the school competitions.

Since in many school contests programming may play a minor role, we found ourselves facing a great disparity of competences: from people who knew a great deal of algorithms and data structures, to boys[1] who hardly know how to write a program, or were lacking basic concepts like recursion. So our challenge was to take also the latter, in a few days, to a sufficient competence level for the regional tests.

Each day was organized in two parts: the morning devoted to a small number of lectures on programming or algorithmic techniques, for example recursion, loop design possibly through invariants, C++ STL, exhaustive search, greedy algorithms, graph visits, etc.; the afternoon in the computer lab, devoted to problems where the techniques explained in the morning must be applied. In the lab, each student tries individually to solve at his computer the problem proposed, with the teacher's active assistance.

In this activity the experience of one of the authors (G.A.), former medalled participant at IOI and now officially among the trainers of the national team, was of great value for devising exercises, while his direct knowledge, both passive and active, of competition mechanisms allowed him to help pupils with simple advice and tips.

The stage was supplemented by a virtual environment on a Moodle platform of the Department of Informatics, containing the presentations, the proposed problems, discussion forums, assignments, etc., as a means to continue the training and to keep a link with the department. The stage obviously involved a small sample of all the secondary school pupils in Piedmont; nevertheless, we think the method could be extended to learning informatics even independently from competitions.

4.4. *Two Problem Examples*

As a partial illustration of the techniques used in the stage, we report here a couple of problems among the ones we have proposed.

The first problem, in Fig. 1, is an example of how the greedy technique can be applied for finding a quick solution that, although not correct, may give the right result in many simple cases, and may therefore – according to the Olympiad's rules – ensure a non-null score with a little effort. The problem also allows to show how a correct solution can often be reached by means of the slightly more sophisticated technique of recursion with *memoisation*, i.e., (the recursive form of) dynamic programming.

The text is simple and the specification of the required result is easy to understand.

[1] Girls were unfortunately completely absent, see also Scarabottolo (2011).

In the new amusement park they want to build a copy of the tower of Ouchnoi, but due to budget restrictions they only have at their disposal a stock of rectangular stone slabs, and all the slab sides are different from one another! In building the tower, each slab (except the base, of course) must rest on a bigger slab, i.e., on one whose both sides are greater, possibly through a rotation by 90 degrees. Help the workers to build the highest possible tower.

Fig. 1. The tower of Ouchnoi.[2]

At each step, if the new slab is larger (in both dimensions) than the present base, add it as new base under the present one; if it is not, check whether it is still larger than the second slab of the tower: in such case remove the base and replace it with the new slab, because in this way you get a tower of the same height but with a smaller base, which is of course an advantage; otherwise simply discard it. Because of the last clause, this strategy is unable to find the maximal height for any input.

```
...
Slab base, second;
base = slabs[0];
int maxHeight = 1;
second.x = second.y = 0; // dummy null slab put on top
for (int i=1; i<N; i++){
  if (slabs[i].y > base.y) { // if the slab is larger than the base
    second = base; // the old base becomes the second slab . . .
    fout « base.x « ",," « base.y « endl;// . . . definitively
    base = slabs[i]; // the new base
    maxHeight ++;
  } // otherwise check if you can replace the base
  else if (slabs[i].y > second.y)
    base = lastre[i];
}
```

Fig. 2. The tower of Ouchnoi: incorrect greedy algorithm.

Each slab is characterized by two integers, the x-side and the y-side, and in reading the slab list from input the slabs may all be saved with, say, the x-side as the greater. Then the first action that usually comes to mind consists in sorting the slab sequence w.r.t. to one side, for example x.

At that point we have to find the maximal non-consecutive subsequence that is also sorted w.r.t. the other side. To the more expert trainee this task may remind problems like the longest common subsequence, which may be solved by dynamic programming techniques. Less prepared pupils, like the ones in our class, tried a much simpler greedy method: build the tower from the top to the base by successively taking the slabs previously sorted in ascending order with respect to x, as in Fig. 2.

The reason why the naïve greedy strategy is not correct is that there are cases where consecutively removing more than one slab in order to accommodate a smaller base is necessary to allow building later a higher tower. For example, if the sorted sequence of slabs is [(6,4), (7,5) (8,1), (9,2), (10,3)], the greedy strategy builds a tower of height 2 with the first two slabs and then discards all the following slabs, since they have a smaller

[2] In Italian "Ahinoi", which sounds similar to "Hanoi", but literally means "ouch us", "poor us".

The highest tower having the kth slab as top slab is 1 plus the height of the highest tower having a larger slab as top slab; the candidate larger slabs are those that follow – or, in reverse order, precede – the kth slide in the sorted input sequence.

```
...
int maxHeightFromTop(int k) {
  int maxHeight = 1;
  int top_y = slabs[k].y;
  for(int i=0; i < k; i++) {//assuming slabs sorted in reverse x-order
    if(slabs[i].y > top_y) {
      int max = maxHeightFromTop(i) + 1;
      if (max > maxHeight) maxHeight = max;
    }
  }
  return maxHeight;
}
...
```

Fig. 3. The tower of Ouchnoi: recursive solution.

y-side than the base; on the other hand, by removing from the tower the first two slabs and starting anew from the third, one is able to build a tower of height 3. It is therefore natural to look for a recursive solution, i.e., a recursive function which returns the height of the highest tower having the kth slab as top slab, as described in Fig. 3.

Pupils rapidly discovered that, while for small inputs the recursive algorithm returns the right solution, with a greater number of slabs the execution time rapidly grows beyond any reasonable limit.

They were then helped to reflect on why this happens, and thus came to the obvious conclusion that the cause is the exceedingly great number of recursive calls that re-compute solutions of the same sub-problems, like in the paradigmatic example of the naïve recursive version of the function that computes the nth Fibonacci number. So the *memoisation* was added (using a vector) and an efficient solution, partially described in Fig. 4, was finally found.

Problems directly or indirectly involving graphs are rather common in the Olympic competitions, and are also a kind of problem that is usually a fun for pupils. An example from the ones proposed in our training stage was, in its abstract form, the following: given a weighted directed acyclic graph, and given a source vertex *A* and a sink vertex *B*, find a path from *A* to *B* such that the weight of the heaviest of its edges is minimal. Its formulation in Fig. 5 translated the acyclicity condition into the metaphor of downhill-only paths, and interpreted weights as danger or risk values.

A similarity with the least path problem immediately suggested an *à la Dijkstra* approach: keep for each vertex a *distance*, representing the global risk of the best path to that vertex, and visit the graph possibly updating the distances. The distance is, as in Dijkstra's or Prim's algorithms, initialized to zero for the start vertex and to infinity for the others. However, differently from those algorithms, breadth-first visit does not work, since the temporary best path to a vertex becomes definitive only when all paths to that vertex have been considered.

An array is added to keep the computed results of the recursive calls.

```
...
int maxHeightMemo[MAX_N];
...
int maxHeightFromTop(int k) {
  if (maxHeightMemo[k] > 0) // if the value has been computed before,
    return maxHeightMemo[k]; // return it without re-computing it
  int maxHeight = 1;
  int top_y = slabs[k].y;
  for(int i=0; i < k; i++) {//assuming slabs sorted in reverse x-order
    if(slabs[i].y > top_y) {
      int max = maxHeightFromTop(i) + 1;
      if (max > maxHeight) maxHeight = max;
    }
  }
  // memo(r)izes the result and returns it
  return maxHeightMemo[k] = maxHeight;
}
...
```

Fig. 4. The tower of Ouchnoi: recursive solution with *memoisation*.

Tom runs in the Boxcar Race of the Alps where each participant, in a self-made motorless vehicle, has to reach the base of a mountain from an upper starting point without destroying or damaging her/his rudimentary vehicle. They can choose any downhill path in a network of roads and trails. Tom has a map of all the downhill pathways with their intersections and has annotated with a risk value every stretch between two intersections. The global risk of a path is the maximum risk of the stretches composing the path. Help Tom to find the least risky path."

Fig. 5. Boxcar race.

In the morning lectures on graphs we had presented a number of basic graph algorithms and techniques: some pupils were then able to see through the "downhill" metaphor an altitude order, and recognize it as a topological order. As a matter of fact, by visiting the vertices in a topological order we are ensured that when we visit the ith vertex we have already examined all the edges to that vertex, and its distance is therefore definitive; we only need to check and, if necessary, update the distances of the adjacent vertices. The topological sorting was performed in advance by the algorithm presented in the lectures.

We found that the fancy formulation of the problem, owing to its concrete character, was for our pupils easier to understand than the abstract formulation which, on the contrary, is usually easier for teachers. At the same time, knowing how to obtain the topological order without having to reinventing it from scratch was of course important, and contributed to convince the trainees that study is useful.

First perform (through depth-first visit) the topological sorting on the graph, and let *sortedVertices* be the result array; then examine the vertices in that order, and for each vertex u consider all its adjacent vertices v: for each v, the global risk of the path from *start* to v via u is the maximum between the global risk from *start* to u and the risk of the edge from u to v; if the global risk of the new path is lesser than the previously computed global risk from *start* to v, update the path to v and the global risk to v. In the code fragment below we only show the updating of the global risk.

```
...
// the vertex 0 is the start vertex
topologicalSorting(0);
globalRisk[0] = 0; // is the distance: the start vertex has distance 0
                   // the other vertices have distances initialized to infinity
for (int i = 1; i < N; i++) globalRisk[i] = INFINITY;
for (int i= N-1; i > 0; i-) {
  int u = sortedVertices[i];
  for (int j=0; j < degree[u]; j++) {
    int v = adjacent[u][j]; // for each vertex v adjacent to u
    // evaluate the global risk in the path to v via u
    int vRisk = max(globalRisk[u], risk[u][j]);
    // if lesser, update the global risk to v
    if(vRisk < globalRisk[v]) globalRisk[v] = vRisk;
  }
}
...
```

Fig. 6. Boxcar race: a solution.

5. Other Informatics Contests in Italy

Olympiad in Informatics, with its search for excellence, cannot alone serve the purpose of stimulating the interest in computer science. In the last years some other informatics contests have been introduced in Italy at various levels. Among them, Olympiad in Problem solving and Kangourou Informatics already involve large numbers of participants from throughout the country; the Scratch Day, celebrated every year in schools and institutions throughout the world, and joined last year by a Piedmont's technical school with a local initiative, has become this year a national event, the Italian Scratch Festival. In addition, there is a number of educational robotics yearly meetings and competitions, where being able to program robot behaviours is often a key element for success in the competition.

Some of the initiatives emphasize problem solving as a preliminary ability for programming, and are based on the observation that technicalities of actual coding may often obfuscate the clarity of the algorithmic core (as anybody who has written a non-trivial program in a real language knows). At the same time, programming remains the end to which, in the intention of the proponents, these initiatives should ultimately lead: programming is present, though with small exercises, in the upper competition levels.

In the following, after a short account of the Olympiad of Problem Solving and Kangourou, amply described elsewhere, we extend a bit more on the newly introduced Scratch Festival, which – being completely based on a "programming for everybody" paradigm – may act as a driving factor for the introduction of programming-based curricula at all school levels.

5.1. *Olympiad of Problem Solving*

The Italian Olympiad of Problem Solving (IOPS), created – after some years of local experiences – by professor Giorgio Casadei, was organized by the Italian Ministry of Education for the first time as a national contest in the school year 2008–09. It was reserved to pupils of the last year of both primary education and lower secondary education ("scuola media"), but because of the big interest it aroused among the teachers, the following editions were opened also to other year classes: in 2011–12 it globally involved pupils from the last two years of primary school, all the three years of lower and the first two years of upper secondary education, so in the ages from 9 to 16, of course with different features and difficulty levels for each age range. The attendance at such editions reached about 30.000 pupils.

IOPS is based on a methodology of the kind "Computer Science Unplugged" (CSU), i.e., computer science without a computer, but it also aims at making young pupils able to understand and use simple semi-formal languages. As a matter of fact, typical contest problems are in a CSU form, but some exercises require the usage of a pseudo programming language for describing the solution clearly and unambiguously (Casadei and Teolis, 2011).

The specific contribution of IOPS to the change of informatics education is its addressing the lower age ranges with tasks that develop typical informatics abilities in a pure way, without the technicalities of a real programming language. Another essential feature is the mode of interaction with schools, which is not limited to one or two annual events, but continues along the whole school year, with frequent training sessions, intermediate assessments, monthly school competitions, etc., all supported by a dedicated web environment.

5.2. *Kangourou of Informatics*

It is the extension to informatics of a mathematics contest which was originally started in France where it has a well-established and successful tradition. Conceived and organized by a group of professors and researchers of the University of Milan, its features are thoroughly described in Lissoni *et al.* (2008) and Lonati *et al.* (2011). It is addressed to lower secondary schools and to the first two years of upper secondary schools, i.e., to students in the age from 11 to 16.

Characterizing aspects are the game-contest modality with its emphasis on fun, the absence of prerequisites in programming or other specific knowledge, and – not least – the fact that competitions are not among individuals, but among four-person teams, thus giving value to cooperation capacity, which is essential in today's working environments, particularly in computer science, and more generally in today's society.

All these features, and especially the last, should profitably find a place in the reorganisation of informatics education.

5.3. *Italian Scratch Festival*

Scratch, the visual drag-and-drop programming environment for children and young pupils created by the MIT Resnick's team (Resnick *et al.*, 2007), is being adopted, after the reform, by a number of Italian schools in the first year of technical education as an alternative to the ICT syllabus, a sort of ECDL-like general introduction to computers, which remains the first-year standard in most technical schools. In this way introduction to programming is no longer reserved to the informatics specialization, but for the first time it is present in the curricula of every type of technical school from the very beginning.

With Scratch the emphasis is therefore back to programming like in the earliest times of computers in education, but in a completely new non-textual approach which is a lot of fun (with the words of the above cited British education secretary: "Instead of children bored out of their minds being taught how to use Word and Excel by bored teachers, we could have 11-year-olds able to write simple 2D computer animations using an MIT tool called Scratch"). In the higher classes of technical education in informatics, of course, Scratch is replaced by C, C++, Java, etc.

In the year 2010–11, the first one in which the school reform was effective and new curricula had to be established, a group of teachers of the "Vallauri" Technical School in Fossano (Piedmont), who had been (and continued to be) involved in IOPS competitions, adopted Scratch, and published a Scratch-based textbook in Italian (Barbero and Demo, 2011; Barbero *et al.*, 2011).

On the occasion of the international Scratch Day 2011, the "Vallauri" School organized an internal Scratch Day with the participation of all its first-year classes; the teachers' association Dschola, the Piedmont regional consortium CSP for applied ICT research, and the Department of Informatics (Dipartimento di Informatica) of the University of Turin all gave their support to the event and ensured cooperation for it. The Day was a great success, and also attracted teachers from other schools; so this year for the Scratch Day 2012 Dschola decided to organize an Italian Scratch Festival (`http://www.associazionedschola.it/isf`).

The importance of Scratch lies exactly in its being an easy-to-use programming environment that also attracts the less talented or less interested pupils. In an era dominated by multimedia web contents, smartphone apps, sophisticated electronic games, etc., an initial approach to informatics based on a poor programming environment, on a textual programming language and on algorithmic problems of abstract nature, such as finding the maximum of a number sequence, performing a binary search or sorting, is often perceived as an obsolete technology and leads to rejection.

Scratch is a real, though non-textual, programming language, "that makes it easy to create interactive stories, animations, games, music, art, and to share them on the web" (scratch.mit.edu). For that reason it is highly motivating for pupils of different ages, and therefore also easily accepted by teachers who can thus avoid being faced with uninterested and bored audiences.

Its expressive limitations, such as the absence of recursion or of classical data structures do not matter in the context of an initiation to programming, and are counterbal-

anced by its being based on programming concepts like concurrency and message-passing that are of paramount importance in today's computer science. Besides, such limitations have been mostly removed in the BYOB Scratch extension developed at Berkeley University, and in fact some teachers, after Scratch, are going to experiment with BYOB in the next classes (Harvey and Mönig, 2010). Also, a Scratch extension with procedures is announced by MIT for the end of May 2012.

6. Programming as a Core of Informatics Education

Olympiad in Informatics and – with a completely different character – Scratch contests are both exclusively based on programming; the other kinds of competitions are also aimed at developing programming abilities. We think this is highly positive, since we believe, like many authors, that programming should be at the centre of informatics education in schools. The basic fascination of informatics is its creative dimension: inventing an object, the program, which seems to find a life of its own, whatever the level of difficulty and the field of application.

Obviously the goal is not to make of every student a professional programmer, but the not much easier one of turning into reality the idea expressed by Sysło (2011) that "everybody can learn programming", and therefore "computer programming (in any sense) . . . should be the competence of everyone".

Actually, while initiation to elementary programming may be made almost deceptively easy, teaching how to write elegant and correct programs for even basic algorithmic problems is difficult, unless one only considers the few gifted students. Giving pupils the solution of a problem, even at the end of their unfruitful efforts, and explaining why it works without telling how to get to it, is not enough. An introspective effort is necessary by the teacher in order to communicate how the solution is reached.

Learning to program requires a delicate balance between creativity and application of rules, between talent and study, as in any non trivial human undertaking. If on the one hand we must contrast the widespread idea that the ability to solve problems may be gained by memorizing a catalogue of suitable recipes to be applied mechanically (i.e., algorithms), on the other hand we must counter the opposite belief that talent is the only thing that matters, and that it is not necessary to study programming patterns and algorithmic techniques.

For example, it may be necessary to show how to use inductive reasoning not only for recursion, but also for iteration, where often a non-trivial loop can be conceived by an invariant-driven design method (it is important, however, that loop invariants are not presented in a formal way but applied as an informal technique: imagine the situation at the generic intermediate step, describe it on paper, possibly with a drawing, and then start coding the iteration step on the basis of it).

In 1971 Wirth drew attention to the role of the chosen programming language in the intellectual process of solving a computational problem: "Program construction consists of a sequence of refinement steps [...] The direction in which the notation devel-

ops during the process of refinement is determined by the language in which the program must ultimately be specified [...]" (Wirth, 1971). Some years later Iverson considered programming languages as "tools of thought", by remarking that their executable character opened the possibility of experimenting with thought (Iverson, 1980), in the same way as physicists always experimented with matter. Although solving problems (or writing procedures) with pencil and paper is a valuable means for countering the thoughtless approach to computing and showing the need of thinking before typing, running and debugging the programme one has designed and coded should remain the final aim.

Modern programming languages are, as Simone Martini writes in Martini (2011), "elegant and sophisticated linguistic tools that informatics has developed to describe effective procedures". Being confronted with an expressive programming language of the present generation should therefore be inescapable for anyone in the course of secondary education. The usage of languages like Python, where indenting is part of the syntax, is then a possible choice, which has been successfully made by A.R. Meo's group in lower secondary school (Martina *et al.*, 2010). Moreover, Python can be combined with a multimedia approach (Guzdial and Ericson, 2010) which makes it more attractive.

Also, since "in informatics, whose primary essence lies in the immateriality of the *linguistic expression* of computation and interaction, really *form is substance*" (italic in Martini's Italian original), even at an early stage the importance of program elegance and "beauty" should not be neglected, starting from a correct indenting, a meaningful choice of identifiers, up to the clarity of the program structure, the avoidance of redundancies, etc. Unfortunately, these aspects cannot be checked by an automatic corrector, and therefore cannot be evaluated in programming contests. In the Olympiad in Informatics, for example, the important thing is to get to write whatever program, possibly quick and dirty, gives the right outputs for as many inputs as possible, if not for all. However, we found that even for the Olympiad these aspects are not completely immaterial, since it is often the case that an "ugly" program is also wrong or inefficient, and bad style is usually error prone.

The problem is that pupils entering secondary schools have very different levels of familiarity with computers and informatics, thus partially contradicting the idea of youth as "digital natives" (Prensky, 2001). The idea has nevertheless some truth, if one considers the fruitfulness usually achieved by an approach based on games and playful environments. In this sense, programming competitions are certainly not the only means to stimulate interest in computer science and to promote an improvement of informatics education. For example, also books like the one by Tomatis (2010), a graduate in CS from our university, on "Mathematics and informatics for crimes" are worth attention. The author shows in it how to apply simple algorithmic or programming techniques to police investigations, criminal cases, etc.; an attractive speaker, he gives lectures and presentations in schools and at the university. Again, the topics play an essential role, for the same reason why detective stories and thrillers have hugely multiplied in the last decades both in literature and in the cinema, becoming a key factor of success.

7. Conclusions

In an educational system like ours in Italy, where computer science is still largely perceived as a merely practical skill and is being taught by teachers with poor informatics competence, competitions may play a key role.

As a matter of fact, creating new teachers through special university degree courses or qualifying the existing teachers through formal educational programmes are processes that may take years if performed in isolation with respect to present school life. On the other hand, the introduction of competitions, starting from the lower secondary classes, and not immediately aiming at excellence, is a means for promoting the desired change, provided some conditions are satisfied.

Firstly, competitions must not be limited to small groups of gifted persons, but have to involve the greatest possible number of pupils and teachers. As a matter of fact, the involvement of present teachers is essential for reshaping syllabi and the way their contents are learnt. Competitions represent a unique opportunity for teachers and pupils to learn together and from one another, working on concrete tasks of common interest.

To this purpose, activities must be initially proposed which are algorithmically elementary but attractive for their character (games, multimedia applications, etc.) and with friendly programming environments. Languages like Scratch, EasyLogo (Salanci, 2010), Python, and contests based on that kind of programming can be useful in such first phase. In the following years more algorithmic aspects should be introduced: a core, roughly coinciding with what is needed for the regional level of the Olympiad in Informatics, should be common to all kinds of schools; other aspects should possibly depend on the different school channels, with or without dedicated competitions. See for example the educational robotics activities described by Barbero *et al.* (2011).

Secondly, and equally important, training for competitions should be integrated in everyday school work, and supported by virtual environments where pupils may interact with schoolmates and teachers and thus find help, discuss problems and solutions, etc., on the model of what is being done with the IOPS. School networks and cooperation with the university are necessary to factorize the efforts and reach a critical mass for the management of such platforms.

This second condition is not easily put into practice in a generalized way: even the best-promoted experiences have only reached well-definite groups of schools and pupils. Extending them to the whole school population would require a qualitative leap that seems hard to perform. A greater commitment is needed by the university, whose support should be more continuous than usually is. Competitions may in any case work as a master key to open the doors of the most insensitive schools and institutions to other actions, which finally lead to a correct teaching of this still largely misrepresented science.

Acknowledgments. We thank Paolo Pasteris, of the technical staff of the Dept. of Informatics of the University of Turin for the constant support in the organization and management of the training courses and laboratories for IOI.

We also thank Giorgio Pidello from Liceo Scientifico “Marie Curie” of Grugliasco, Alberto Barbero from the Istituto Tecnico per Informatici “Vallauri” of Fossano (both

schools are in Piedmont), and all the members of the Dschola working group "New informatics curricula in secondary education" for the many useful discussions and suggestions on the problems of secondary education.

References

Barbero, A., Demo, G. B. (2011). The art of programming in a Technical Institute after the Italian secondary school reform. In: *Proceedings ISSEP 2011*, Bratislava.

Barbero, A., Demo, G. B., Vaschetto, F. (2011). A contribution to the discussion on informatics and robotics in secondary schools. In: *Proceedings RiE*, 2nd International Robotics in Education Conference, Wien.

Casadei, G., Teolis, A. Comprendere e comunicare in modo effettivo: computational thinking. *Annali della Pubblica Istruzione*, 4–5, 15–36.

CINI, GII, GRIN, Manifesto sull'Informatica nella riforma della scuola superiore. `http://www.grin-informatica.it`.

ECDL – European Computer Driving License. `www.ecdl.org/`.

Gove, M. 'Harmful' ICT curriculum set to be dropped this September to make way for rigorous Computer Science. `http://www.education.gov.uk/inthenews/inthenews/a00201864/harmful-ict-curriculum-set-to-be-dropped-this-september-to-make-way-for-rigorous-computer-science`.

Guzdial, M., Ericson, B. (2010). *Introduction to Computing and Programming in Python*, A Multimedia Approach, 2/E, Pearson.

Harvey, H., Mönig, J. Bringing "No ceiling" to scratch: can one language serve kids and computer scientists? In: *Proceedings Constructionism*, Paris.

Italiani, M. (2011). Italian Olympiad in informatics: 10 years of the selection and education. *Olympiads in Informatics*, 5, 140–146.

Iverson, K. (1980). Notation as a tool for thought. *Communcations of ACM*, 33(8), 444–465.

Lissoni, A., Lonati, V., Monga, M., Morpurgo, Torelli, M. (2008). Working for a leap in the general perception of computing. In: *Proceedings of Informatics Education Europe III*, Venice, Italy.

Lonati, V., Monga, M., Morpurgo, A., Torelli, M. (2011). What's the gun in informatics? Working to capture children and teachers into the pleasure of computing. In: *Proceedings of ISSEP 2011 – Informatics in Schools: Contributing to 21st Century Education*, Bratislava.

Martini, S. (2011). Lingua Universalis. *Annali della Pubblica Istruzione*, 4–5, 65–69.

Martina, A., Meo, A., R., Moro, C., Scovazzi, M. (2010). Passo dopo passo impariamo a programmare con Python. `http://linuxdidattica.org/polito/manuale-python-V2.pdf`.

Papert, S. (1997). Why school reform is impossible. *The Journal of the Learning Sciences*, 6(4), 417–427. `www.papert.org/articles/school_reform.html`.

Prensky, M. (2001). Digital natives, digital immigrants. *On the Horizon*, 9(5).

Resnick, M. *et al.* (2009). Scratch: programming for all. *CACM*, 52(11), 60–67.

Salanci, L. (2010). EasyLogo – discovering basic programming concepts in a constructive manner. In: *Proceedings Constructionism 2010*, Paris.

Scarabottolo, N. (2011). Olimpiadi di informatica bilancio di un decennio. *Mondo Digitale*, 38/39.

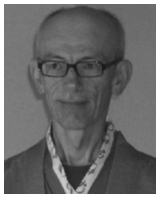
Sysło, M.M. (2011). Outreach to prospective informatics students. In: *Informatics in Schools: Contributing to the 21s Education*, *LNCS*, 7013, *Proceedings ISSEP 2011*, Bratislava.

Tomatis, M. (2010). Mathematics and informatics for crimes. In: *Seminar at the Department Computer Science, University of Torino*, October 2010, `www.marianotomatis.it/index.php?lang=en`.

Wirth, N. (1971). Program development by stepwise refinement. *Communications of the ACM*, 14(4), 221–227.

G. Audrito is a former IOI participant, bronze medal at the 2004 and 2005 editions. He has been involved as a staff member in the Italian IOI project since 2006, and in the regional IOI training courses held in Turin since 2010. He is currently a PhD student in mathematics at the University of Turin.

G.B. Demo is an associate professor of informatics at the University of Turin. She coordinates the working groups on cooperation with schools for informatics teaching both of her department and of GRIN, the National Group of Researchers and Teachers in Informatics of the Italian Universities.



E. Giovannetti is an associate professor of informatics at the University of Turin. He is responsible for the training stages of preparation to the Regional OII in West-Piedmont. He has been teacher in Italian secondary schools, and researcher at the former Italian Telecom Research Centre (CSELT).

Olympiads in Informatics, 2012, Vol. 6, 21–30

FCL-STL, a Generics-Based Template Library for FreePascal

Vladimír BOŽA, Michal FORIŠEK
Comenius University, Bratislava, Slovakia
e-mail: {usama,misof}@ksp.sk

Abstract. We present the design and usage of a new set of FreePascal units we created. These units aim to be the counterpart of the Standard Template Library in C++. The library is already included in the development branch of FreePascal (v. 2.7.1), and it should be a part of the stable branch (v. 2.6.?) at some point in future. Available data structures include vectors, sets, maps (both ordered and unordered) and more.

For many tasks used in past programming competitions there were C++ solutions that were significantly easier to implement than any Pascal solution. Problem setters usually needed to ensure that a reasonable Pascal solution exists. This library, once available, will mitigate the difference between the powers of these two languages.

Key words: FreePascal, templates, generic programming, algorithm library, STL.

1. Overview

In this section we give an overview of topics related to this paper. First, we discuss generic programming, in particular its use when creating algorithm and data structure libraries. Next, we explore the connection between such libraries and programming contests. At the end of this section we give an outline of the rest of the paper.

Generic Programming

In recent years the global trend in programming is towards library reuse. Modern programming languages such as Python come equipped with an extensive set of libraries that cover all the common needs of programmers.

One particular area contains the basic data structures and algorithms. From our point of view, this area is special and really stands out among the others. More precisely, the one thing that makes the data structure/algorithm libraries special is the need for a generic approach. For most other libraries the input domain is clearly defined. E.g., the regex library processes strings, the datetime library works with timestamps, the OS library provides a platform-independent API to access the directory structure. The situation is different with algorithms and data structures: everybody needs to store, access and sort records, but the content of those records and their correct comparison differ between programs. The libraries that provide data structures and algorithms have to be prepared to deal with this issue.

Modern programming languages deal with this issue by introducing *generic programming* (Musser and Stepanov, 1988). The basic syntactic elements of generic programming are usually called *templates*. A template essentially allows the programmer to use metavariables that represent unknown *types*. This language construct allows the library authors to write generic functions that can later be *instantiated* by a library user to work with any suitable data type(s). For instance, the following is a simple templated version of a swap function in C++:

```
template<typename T>
void swap(T &a, T &b) { T c; c=a; a=b; b=c; }
```

Note how the template metavariable `T` plays the role of an unknown type. Once you replace `T` by a particular type (e.g., `int`), you will get a valid C++ function that swaps the elements of that type.

The benefit of generic programming both for library authors and for library users is obvious. First of all, it prevents unnecessary duplication of code. E.g., there is just one sorting metafunction, instead of thousands ("a sort for lists of ints", "a sort for arrays of strings", etc.). This directly reduces bloat – the resulting library is physically smaller – and makes testing much simpler.

Finally, this approach is more versatile than traditional libraries in that it can be used even for data types not explicitly known to the library author. The users can define their own data types and process them using the same functions provided by the library.

Algorithm Libraries in Programming Languages; Their Use in Contests

Most of the modern programming languages come with extensive algorithm and data structure libraries. To name a few:

C++: The Standard Template Library (STL) has all the basic data structures and algorithms. (Many additional ones, e.g. including graph algorithms, are included in the Boost libraries (Boost C++ Libraries). Note that Boost is not allowed in most programming contests.)

Java: All basic data structures are in the Collections package.

Python: Some data structures (array-lists, dictionaries) are built-in, others are provided in the "Data Types" and "Numeric and Mathematical Modules" parts of the Python Standard Library.

On the contrary, no such library was available for FreePascal until now. Some data structures, but with no consistent interface and without generic programming, were available in the component library (`AVL_Tree`: an AVL tree containing pointers; `contnrs`: lists, stacks and queues of pointers or objects).

At programming contests this provided a clear disadvantage to contestants who use Pascal. For instance, this is the case at the International Olympiad in Informatics (IOI) where for many years the allowed programming languages are only Pascal and C/C++. The lack of a standard library for FreePascal has directly influenced the choice of past

competition tasks. For instance, tasks solvable using balanced binary trees were used in the competition only if there was an alternate solution in Pascal without balanced trees, but with the same asymptotic time complexity. (E.g., solutions that use various interval trees fall into this category.) In addition to leveling the playing field, this library should broaden the spectrum of suitable competition tasks for the IOI.

In the IOI Syllabus (Verhoeff *et al.*, 2006; Verhoeff *et al.*, 2012) most algorithms and data structures from standard libraries have the status "not for task description", meaning that such concepts should not be discussed in the task statements but using them may be necessary to solve some of the competition tasks. Here we see a direction in which the Syllabus ought to be improved in the future: At the moment, there is no clean distinction between *understanding* such data structures, *using* their library implementation and *implementing* them from scratch. The general consensus is that the first two are surely necessary; further discussion about requiring their implementation would be helpful.

About FCL-STL

The core of the FCL-STL FreePascal library was implemented in 2010 and 2011 by Vladimír Boža as a part of his Bachelor Thesis at Comenius University, Slovakia (Boža, 2011). The supervisor of this Thesis was Michal Forišek. At the moment, the library is available in the FreePascal development snapshot (Free Pascal team, 2012) in `fpc/packages/fcl-stl/`.

Outline of the Paper

Sections 2 and 3 list the algorithms and data structures currently implemented in FCL-STL. In Section 4 we provide a comprehensive table that compares the main features to C++ STL. In Section 5 we present data from practical benchmarks of FCL-STL.

2. Library Contents – Algorithms

In this section we list and briefly describe algorithms implemented as a part of the FCL-STL FreePascal library. If unclear about the semantics of a particular algorithm, please refer to (Cormen *et al.*, 2009) or an equivalent algorithm textbook. The same applies to the next section.

Sort

`Sort` is a generic array sort function. Internally the implementation uses IntroSort (Musser, 1997) – which is basically a QuickSort with a fallback to HeapSort in order to achieve a guaranteed $O(n \log n)$ worst case time complexity.

Random Shuffle

`RandomShuffle` randomly shuffles the elements in an array. The time complexity is worst case $O(n)$, and the random distribution is uniform assuming a uniform internal random function.

Next Permutation

`NextPermutation` rearranges the elements in an array to obtain the next permutation in lexicographic order and returns `true`. The only exception: if the input is an array with elements in descending order, the array is reversed (to obtain the lexicographically smallest permutation) and `NextPermutation` returns `false`.

The time complexity of a single call is $O(n)$. More precisely, the time complexity of a single call is linear in the number of necessary changes. Hence a full cycle iterating over all permutations of a given n-element array runs in $O(n!)$.

3. Library Contents – Data Structures

Vector

`TVector` is an array that can be resized at the end efficiently (doubling storage size when necessary). Most important operations:

- indexing using `[]` in $O(1)$;
- `PushBack` in amortized $O(1)$.

Stack

`TStack` is a traditional stack data structure. Most important operations:[1]

- `Push` in amortized $O(1)$;
- `Top` and `Pop` in $O(1)$.

Queue

`TQueue` is a traditional queue data structure. Most important operations:

- `Push` in amortized $O(1)$;
- `Front` and `Pop` in $O(1)$.

Deque

`TDeque` is a deque (usually pronounced [deck], also called a double-ended queue) – a self-resizing array that supports indexing and fast element addition/removal at both ends. Most important operations:

- `PushFront` and `PushBack` in amortized $O(1)$;
- indexing using `[]` in $O(1)$;
- `PopFront` and `PopBack` in $O(1)$.

[1] The implementations of `TStack`, `TQueue` and `TDeque` all use `TVectors` for internal data storage, hence the amortized complexity bounds on insertions. Although there are possible implementations of stacks and queues with guaranteed constant time complexity, this is the industry standard in other languages. For most uses in practice the amortized time complexity does not matter, and in the remaining cases the stacks/queues can easily be implemented as linked lists.

Priority Queue

`TPriorityQueue` is a priority queue that allows insertion of ordered elements and fast extraction of the maximal element. Internally the priority queue is implemented as a binary heap. Most important operations:

- `Push` in amortized $O(\log n)$;
- `Top` in $O(1)$;
- `Pop` in $O(\log n)$.

Ordered Sets and Maps

`TSet` is an ordered container of unique elements. `TMap` is an ordered associative array. Internally, sets are implemented using left-leaning red-black trees (Sedgewick, 2008), and maps are implemented as sets of pairs (key,value).

Most important operations:

- `Insert`, `Find`, `Delete` in worst-case $O(\log n)$;
- `Min`, `Max` in worst-case $O(\log n)$;
- `FindLess[Equal]`, `FindGreater[Equal]` in worst-case $O(\log n)$;
- iteration over all elements in worst-case $O(n)$;
- maps: indexing using `[]` in worst-case $O(\log n)$.

Unordered Sets and Maps

`THashSet` is an unordered container of unique elements. `THashMap` is an unordered associative array. Internally, unordered sets and maps are implemented as self-resizing hash tables, with collisions resolved by chaining.

Most important operations (assuming a good hash function is used):

- `Insert`, `Contains`, `Delete` in expected $O(1)$;
- iteration over all elements in worst-case $O(n)$;
- maps: indexing using `[]` in expected $O(1)$.

(At the moment there are no pre-written hash functions, the programmer has to provide one when using a data structure with a hash table. Default hash functions for basic types are a possible addition in the future.)

4. Comparison of Contents to C++ STL

Table 1 summarizes the correspondences between FPC-STL and the Standard Template Library for C++. We also highlighted some algorithms and data structures that are available in C++, but are not a part of FCL-STL yet.

Table 1
Correspondence between FCL-STL and its C++ counterpart

FreePascal FCL-STL	**C++ STL equivalent**
`Sort`	`sort`
`RandomShuffle`	`random_shuffle`
`NextPermutation`	`next_permutation`
`--`	`stable_sort, nth_element`
`TVector`	`vector`
`TStack, TQueue`	`stack, queue`
`TDeque`	`deque`
`TPriorityQueue`	`priority_queue`
`TSet, Tmap`	`set, map`
`THashSet, THashMap`	`unordered_set, unordered_map`
`--`	`list`
`--`	`bitset`
`--`	`(unordered) multiset`

5. Performance Tests

We made several benchmarks to test the efficiency of our implementation and to compare it to the implementation of STL in C++. The benchmarks were of two different types, focusing on two different topics:

- benchmarks measuring the efficiency of individual library components;
- benchmarks focusing on solving entire tasks from programming contests.

Together, the chosen benchmarks cover all relevant parts of FCL-STL. Some details on the benchmarking environment:

- AMD Athlon(tm) II X4 640 Processor (single core used), 4 GB RAM;
- Linux version 3.2.0-24-generic (Ubuntu/Linaro 4.6.3-1ubuntu5);
- gcc/g++ version 4.6.3, switches -std=gnu++0x, -O2;
- fpc version 2.4.4-3.1 with FCL-STL, switch -O2;
- All measured times are processor times (i.e., time actually spent running the application and executing its system calls);
- Each test was executed 10 times. The main plotted values are averages. All plots also have error bars showing the minimum and maximum, but as the measurements are pretty accurate, the error bars are usually invisible.

Benchmark #1: Push Back, Shuffle and Sort

In this benchmark the program creates an empty vector, inserts values 0 through $n - 1$ (using the push back method), randomly shuffles the vector and then sorts it. Obviously,

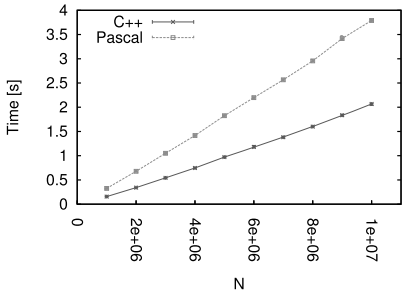


Fig. 1. Vectors and sorting.

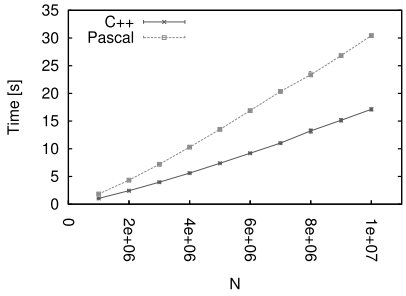


Fig. 2. Sets and iterators.

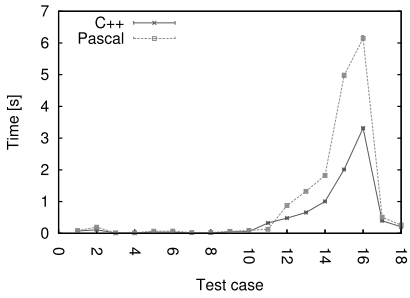


Fig. 3. The task “poet”.

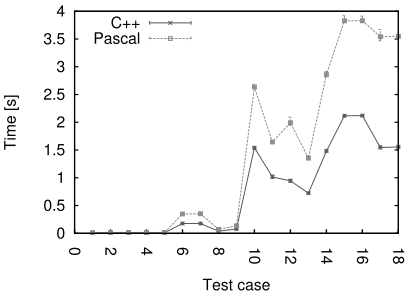


Fig. 4. The task “asphalt”.

most running time is spent in sorting the random permutation. The Pascal implementation is shown in the Appendix. Results of this benchmark for various n are plotted in Fig. 1.

Benchmark #2: Sets

In this benchmark the program creates an empty vector and pushes back n random values, each between 0 and $n - 1$, inclusive. Then all of these values are inserted into a set (effectively sorting them and discarding duplicates). Finally, the set is traversed using an iterator and the sorted values are copied into a new vector. This benchmark tests the efficiency of sets. The Pascal implementation is shown in the Appendix. Results of this benchmark for various n are plotted in Fig. 2.

Benchmark #3: Task “Poet”

For this benchmark we used the task “poet” (Slovak OI, national round 2011). We want make a poem of n couplets (two-line stanzas). For each of the n couplets we are given multiple options. We need to pick the couplets in such a way that for each i, line 2 of couplet i rhymes with line 1 of couplet $i + 1$. Also, the last line of the poem must rhyme with the first line, making the rhymes cyclic.

The solution is to search the state space: for each possible ending of the first line of the poem, inductively construct sets of all possible endings of last lines for couplets 1

through n. To store these sets, our implementations use unordered associative arrays (i.e., hash sets). Results are plotted in Fig. 3.

Benchmark #4: Task "Asphalt"

For this benchmark we used the task "asphalt" (Slovak OI, national round 2011). The goal is to find a shortest path in a 2D landscape by building roads, bridges and tunnels. The solution is a modified version of Dijkstra's shortest paths algorithm, using a priority queue to achieve a better time complexity. The data structures used are: a priority queue, vectors (and vectors of vectors), and stacks. Results are plotted in Fig. 4.

6. Conclusions

The FCL-STL library aims to become a standard algorithm and data structure library for FreePascal. It is worth noting that nowadays this is as far as a Pascal library can go. There is no standard for modern Pascal, and different compilers (e.g., Delphi) choose their own syntax for all language extensions such as generics.

The benchmarks quite consistently show that the Pascal implementations tend to be slower than their C++ counterparts approximately by a factor of 2. In our opinion, this makes the library usable enough for many practical uses, including programming contests.

The main plan for the near future is to have this library included into a stable FreePascal release.

To conclude this paper, the authors would like to thank two anonymous referees for their helpful comments and remarks.

References

Boost C++ Libraries. Available online at `http://www.boost.org/`.

Boža, V. (2011). Knižnica štandardných algoritmov pre kompilátor FreePascal. Comenius University Bratislava, Bachelor thesis, Comenius University Bratislava, Slovak. Available online at `http://oldwww.dcs.fmph.uniba.sk/bakalarky/obhajene/getfile.php/boza11306397896881.pdf?id=159&fid=315&type=application%2Fpdf`.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2009). *Introduction to Algorithms*, 3rd edn., The MIT Press.

Free Pascal team. (2012). Free Pascal 2.7.x Daily Source Snapshot of Development Tree. Available online at `ftp://ftp.freepascal.org/pub/fpc/snapshot/trunk/source/fpc.zip`.

Musser, D.R. (1997). Introspective sorting and selection algorithms. *Software Practice and Experience*, 27, 983–993.

Musser, D.R., Stepanov, A.A. (1988). Generic programming. In: *Symbolic and Algebraic Computation: International Symposium ISSAC*, 13–25.

Sedgewick, R. (2008). Left-leaning red-black trees. *Workshop on Analysis of Algorithms*, Maresias, Brazil. Available online at `http://www.cs.princeton.edu/ rs/talks/LLRB/RedBlack.pdf`.

Verhoeff, T. Horváth, G., Diks, K., Cormack, G. (2006). A proposal for an IOI syllabus. *Teaching Mathematics and Computer Science*, 4(1), 193–216.

Verhoeff, T., Horváth, G., Diks, K., Cormack, G., Forišek, M. (2012). IOI Syllabus. Available online at `http://ksp.sk/ misof/ioi-syllabus/`.

Appendix

Usage Examples

Below we show a few Pascal source codes that use the FCL-STL library. The first source code is the code used for benchmark #1: random shuffling and then sorting a vector.

```
uses gvector, garrayutils, gutil;

type iLess     = specialize TLess<longint>;
     iVector   = specialize TVector<longint>;
     iOrdUtils = specialize TOrderingArrayUtils<iVector, longint, iLess>;
     iUtils    = specialize TArrayUtils<iVector, longint>;

var  V : iVector;
     n, i : longint;

begin
    read(n);
    V := iVector.Create;
    for i := 0 to n-1 do V.PushBack(i);
    iUtils.RandomShuffle(V,n);
    iOrdUtils.Sort(V,n);
end.
```

The second example is benchmark #2: sets and set iterators.

```
uses gvector, gset, gutil;

type iLess   = specialize TLess<longint>;
     iVector = specialize TVector<longint>;
     iSet    = specialize TSet<longint,iLess>;

var V : iVector;
    S : iSet;
    N, i : longint;
    it : iSet.TIterator;

begin
    read(N);
    V := iVector.Create();
    for i:=1 to N do V.PushBack(random(N));
    S := iSet.Create();
    for i:=0 to N-1 do S.Insert(V[i]);
    V.Clear();
    it := S.Min();
    repeat V.PushBack( it.GetData() ); until not it.Next();
end.
```

V. Boža is a master's degree student at the Comenius University in Slovakia. He has multiple medals from IOIs and IPhOs and now he is an active organizer of various national and international programming contests. In the last two years he also worked as an intern in Google Zurich. His master's thesis is in the area of cryptography.

M. Forišek is an assistant professor at the Comenius University in Slovakia. Since 2006 he serves as an elected member of the International Scientific Committee (ISC) of the IOI. He is also the head organizer of the Internet Problem Solving Contest (IPSC). His research interests include theoretical computer science (hard problems, computability, complexity) and computer science education.

Olympiads in Informatics, 2012, Vol. 6, 31–43

Teaching Programming and Algorithm Design with Pythia, a Web-Based Learning Platform

Sébastien COMBÉFIS, Vianney le CLÉMENT de SAINT-MARCQ
Department of Computer Science Engineering, Université catholique de Louvain
Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium
e-mail: {sebastien.combefis, vianney.leclement}@uclouvain.be

Abstract. In Belgium, there are no or very few programming and algorithm design courses at secondary schools (12–18 years old). Students who can program are self-learners. The selection process for the Belgian delegation for the IOI, that has been set up two years ago, takes this situation into account. More initiatives should be taken to introduce computer science in secondary schools. This paper presents Pythia, a solution grader for programming and algorithm design problems based on a web application. Following the *learning by doing* motto, the proposed framework provides an accessible, usable, effective way to learn programming. To compensate for the lack of teachers with programming or algorithm design skills, Pythia delivers direct feedback to the students. The paper describes the design of the courses and the architecture of the tool. As future work, the proposed teaching technique has yet to be tested at large and evaluated.

Key words: solution grader, teaching, programming, algorithm design, learning platform, online course.

1. Introduction

In Belgium, there are no or very few programming and algorithm design courses taught at secondary schools. Nevertheless, some 12–18 years old Belgian pupils are good at programming and designing algorithms, but they do not know it (Combéfis and Leroy, 2011). As far as we know, there are very few opportunities to encourage such people to learn programming. The contestants participating to the Belgian Olympiad in Informatics (be-OI) are mainly pupils with self-taught programming experience.

When taught in secondary schools, computer science is taken in charge by teachers from other disciplines such as mathematics or physics. The same situation happens in many other countries. Teachers do not always have the suited experience to be able to teach computer science. Sometimes they are even asked to teach computer science but they just do not know what to do.

Discussions with the contestants of the be-OI raised several ideas. One of them was to gather on a website existing online programming contests and to allow the contestants to publish their scores. As such, they can compare with each other and develop a competition spirit. Another idea was to propose online programming courses targeted at people with zero background in programming and algorithm design.

The main problem addressed by this paper is to find and develop a solution providing an easy way for people to learn programming. The solution should not require any difficult installation and configuration of the personal computer. Moreover, the solution should be flexible enough so that pupils can learn autonomously at home and so that secondary school teachers could use it to teach in classes.

As an attempt to implement the ideas given above, this paper presents Pythia, a web-based solution grader application. Pythia provides both online programming and algorithm design courses, and a collection of problems users can solve and compare with each other. Code submitted by students is executed safely inside a sandbox. The Pythia tool is actually more general than what it is used for in the context of this work. The tool can be used for many other purposes, such as providing automatic grading for students' homework or supporting online contests for example.

Section 2 draws up a related work presenting some existing solutions to address the problem of providing means to learn programming. Besides the description of courses and problems in Section 3, this paper shows the technical aspects of Pythia in Section 4. Section 5 details the plan for evaluating the tool. Finally the last section concludes the work and draws up some perspectives.

2. Related Work

Several initiatives exist to support people in learning programming. For example programming contests are a good motivation for people to learn and improve their skills. However, the contestants must learn by themselves if no trainings are organised. Contests like olympiads are also targeting excellence, excluding less-skilled people. A positive aspect of contests is the discussions it triggers between contestants, especially for non-online contests.

A computer science festival for young people is another initiative. In Belgium, such festivals sensitise people to computer science without using computers. Useful resources include CSUnplugged (Bell *et al.*, 2009) and Scratch (Maloney *et al.*, 2004). School teachers come to such events with their pupils. They follow some activities and learn at the same occasion. Teachers often continue the activity with the pupils back in their classroom. While festivals spark great interest, they require a lot of specially-trained human resources. Due to their large audience, festivals also do not dive deep into the details. As such, they are adequate for advertising purposes, but not for learning purposes.

A third kind of initiative is online learning. Web-based applications avoid the need for the learner to install anything on his/her computer. There are plenty of online applications like *"Rubymonk"*, *"Try Ruby"* or *"Try Python"*. The main goals of such applications is usually to learn a specific programming language rather than general programming or algorithm design skills. The learner can execute small snippets of code to get a grasp of the language. Some sites, like *"Codecademy"*, try to learn programming as a general skill. However, *"Codecademy"* does not teach algorithm design and lacks support to provide quality feedbacks to learners.

The solution proposed in this paper shares the technology of learning sites. In contrast to other sites, if focuses on learning programming and algorithm design skills. Special care has been taken to provide a language-agnostic framework for the safe execution of students' code.

Systems for executing students' code in a safe way have been proposed in the literature. The hackzor system is used for online competition contest judging. The Moe Contest Environment (Mareš, 2009) is targeted at competitions in the spirit of the International Olympiads in Informatics. Moe sandboxes forbids dangerous system calls. Pythia implements a less restrictive approach by leveraging virtual machines. Such approach is shared by the uevalrun system. However, uevalrun lacks the surrounding framework provided by Pythia.

3. The Problems

The proposed learning platform basically consists of two activities. The learner can follow a course where he/she is guided through several lessons that will teach him/her programming and algorithm design concepts. The goal of the lessons is to bring theory to the learner and to allow him/her to practice directly. The learner can also try to solve isolated problems. The goal of those problems is to train algorithm design by allowing the learner to solve problems. The two kinds of activities are uniformed around the concept of *task*.

This section reviews the two kinds of activities and illustrates them with examples. Some intuition about how to design such problems and about what is possible with the proposed learning platform is also provided.

A *task description* is an XML file which contains the following information:

- Basic information: author, title and difficulty level.
- Constraints: programming language, maximal execution time and memory. Optionally, a limit on the total number of submissions and/or on the number of submissions per day can be specified. A deadline can also be given.
- Task contents: context and questions.

Besides the above information, a task description contains two programs. The *template program* contains placeholders to be filled with the student's answers. The completed program will be run in a sandboxed environment. The result of the execution will be analysed by the second program, i.e., the *analyser script*, whose goal is to generate a feedback for the user.

We now explain how isolated problems (Section 3.1) and courses (Section 3.2) are designed using task descriptions. Section 3.3 expands on the feedback, an important point in the learning process.

3.1. *Isolated Problems*

Isolated problems can be solved by the students independently from any other problems.

The key elements of isolated problems are:

1. The problem should be put in context and explained with a concrete situation.
2. Simple instances should be provided to be solved by hand in order to get the intuition of the problem.
3. Larger instances should be provided to be solved with an algorithm.

The first item is required to increase the motivation of the user to solve the problems. The context links the abstract problem to concrete situations the student encounters in everyday life. The user thus gains more value to the activity of solving the problem. It is essential for the student to perceive an interest and a pedagogical utility with respect to his goals for the task he is doing (Viau, 1998). According to Viau, the student's perception of his controllability of the activity is also important in the motivational dynamics. Such concern is addressed by allowing the student to choose the problems by himself according to his level and tastes.

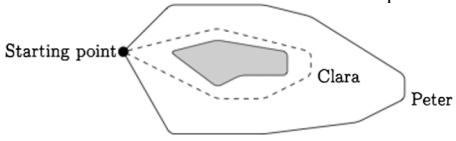
The second and third items, i.e., simple and larger instances, promote computer science. By solving a simple instance by hand, the student gets an insight on why a computer program is needed for solving the larger instances. While solving the small instance manually, the student also gets the logical intuition of the algorithm he is applying by hand. The answer to the simple instance, typically a single value, is used in the feedback to check whether the student has understood the purpose of the problem. In the very last question, the student has to provide an algorithm to solve the general problem.

Figure 1 shows an example of an isolated problem, as shown to the student. The algorithmic problem hidden behind the concrete situation is the least common multiple between two strictly positive integers.

Let's go for a tour around the lake

Context

Peter and Clara decided that they are going to go running around the lake. There are several possible paths around the lake. Peter and Clara both have their favourite paths. The two paths have the same starting point and Peter and Clara both arrive at the same point after having run.



Question

Let's suppose that Peter's path is **five** kilometres long and that Clara's one is only **three** kilometres long. If they start at the same time and if they are running at exactly the same speed, after **how many rounds** will Clara cross Peter **for the first time**?

Write a function that **takes two parameters** A and B which are **non-zero natural numbers** corresponding to the lengths of the paths of Peter and Clara. The function **returns a pair of natural numbers** containing the minimal number of rounds after which Peter and Clara (in that order) will cross each other at the starting point.

```
def toursNumber (A, B):
```

Fig. 1. The concrete situations introducing isolated problems motivate the students to complete the task. The algorithmic problem hidden behind the questions is finding the least common multiple.

Isolated problems are mainly targeted at people with some background in programming, but little knowledge of algorithm design. Students participating in mathematical olympiads are very interested in similar problems proposed by Project Euler. Such problems require the students to understand the problem, to be able to solve small instances by hand and then to build an algorithm out of their reasoning.

Small logic games used to increase algorithmic thinking (Burton, 2010) are well suited for isolated problems. Figure 2 shows an example of such tasks.

3.2. *Courses*

The Pythia framework aims to be a learning platform for people with zero programming knowledge or people wanting to improve their programming skills. We leverage the same platform used for isolated problems described in the previous section to provide full courses. The student can directly code what he is learning and gain feedback. A course in Pythia is a list of tasks. Unlike isolated problems, the student must complete the different

The mystery sequence

Context

In this problem, we are going to build a sequence of numbers. The first number of the sequence is the digit 1. To build the next numbers of the sequence, you have to execute the following procedure: you start from the previous number (in the sequence) and you read it from left to right, while counting the number of identical digits; you then indicate the obtained sum followed by the digit that was repeated and you do so until you have read the whole number.
Concretely, when you read a number from the sequence, digit by digit, it is in fact the description of the previous number in the sequence. To better understand, here are the first numbers of the sequence:

1. 1
2. 11
3. 21
4. 1211
5. 111221
6. 312211

Let's take the last number (312211). To build it, we start from the previous number (111221) and we read it from left to right. There is first **three** times the digit 1 (31), followed by **two** times the digit 2 (22) and finally **one** time the digit 1 (11).

Question

What is the 7th number of the sequence?

Write a function that **takes one parameter** N which is a **non-zero natural number**. The function **returns the** N**th number** of the mystery sequence.

```
def mysterySeq (N):
```

Fig. 2. Small logic games are good candidates for isolated problems and to learn programming and algorithm design.

tasks of a course in the given order, as they build upon each other. Courses are developed around the *learning by doing* motto (Dewey, 1938) which inspired active pedagogy learning methodologies.

Figure 3 shows a task which belongs to the *Introduction to programming* course. The task introduces the concepts of variable, initialisation and value modification. Compared to isolated problems, context is replaced by theory and questions by practice. Theory is explained with illustrations and examples. Students can practise the new concepts on small exercises. For example, in Fig. 3 the student must declare and initialise one variable named "x" and then change its value.

3.3. *Feedback*

In order to keep a student motivated, and to help him/her learn something, he should receive feedback. With the proposed framework, the feedback can be directly provided to the student, as soon as his code has been executed. The framework is flexible enough to allow different kinds of feedback.

Using the capabilities of the Python language, the template program for the first isolated problem example presented in Fig. 1 is shown in Fig. 4. The tags `@@id@@` are placeholders for the student's answers. The tag `@ @q2@@` means that the student's answer tagged with the q2 id will be inserted with every line prefixed with four spaces.

The code for the tester is gathered in the `TestSuite` class. The class has the following methods.

Using variables

Theory

Now that you can print text on the screen, we are going to learn a key concept in programming: variables. A *variable* is a box which can store a value in the memory of the computer. The box is given a *name* which is used to interact with the variable, i.e., to read or change its value.
For Python, the name of a variable can only contain letters among a-z, A-Z and digits among 0-9. For example, "var", "i" and "rowNb42" are valid names, but "$4" and "$4 + 3$" are not valid.
A variable is declared and initialised in one instruction. Here is the instruction which declares a variable name *size* and whose initial value is set to 42:

```
size = 42
```

The following figure illustrates the variable which is created in memory, with its value.

The first time a variable is encountered, we say that it is *initialised*. Afterwards, you can use exactly the same instruction to change its value.

Practice

Declare a variable named "x" and whose initial value is 10:

Change the value of the "x" variable to 12:

Fig. 3. During a course, the students can immediately practise their newly learned skills on small exercises.

```
import random
def toursNumber(A, B):
@  @q2@@
class TestSuite:
  def correct(self, A, B):
    (x, y) = (A, B)
    if x == 0:
      return y
    while y != 0:
      if x > y:
        x -= y
      else:
        y -= x
    gcd = A * B / x
    return (gcd / A, gcd / B)
  def check(self, data, exp):
    try:
      ans = toursNumber(*data)
    except Exception as e:
      return 'exception:%s' % (str(e))
    if type(ans) is not tuple:
      return 'bad_type:%s:%s:%s' % (str(ans), str(data), str(exp))
    if len(answer) != 2:
      return 'bad_size:%s:%s:%s' % (str(ans), str(data), str(exp))
    try:
      a, b = (int(x) for x in answer)
      if a <= 0 or b <= 0:
        raise ValueError
    except ValueError:
      return 'bad_domain:%s:%s:%s' % (str(ans), str(data), str(exp))
    if ans != exp:
      return 'bad_answer:%s:%s:%s' % (str(ans), str(data), str(exp))
    else:
      return 'correct'
  def run(self):
    # Predefined tests
    inputs = [(1, 1), (3, 5), (1, 2), (2, 1)]
    outputs = [(1, 1), (5, 3), (2, 1), (1, 2)]
    correct = 0
    for i in range(len(inputs)):
      res = self.check(inputs[i], outputs[i])
      if res.split(':')[0] == 'correct':
        correct += 1
      else:
        print(res)
        break
    # Random tests
    if correct == len(inputs):
      nbtests = 100
      correct = 0
      for i in range(nbtests):
        A = random.randint(1, 100000)
        B = random.randint(1, 100000)
        ans = self.correct(A, B)
        res = self.check((A, B), ans)
        if res.split(':')[0] == 'correct':
          correct += 1
        else:
          print(res)
          break
      if correct == nbtests:
        print('correct')
TestSuite().run()
```

Fig. 4. Placeholders in the template program will be replaced by the student's answers. The completed program is run in a sandboxed environment.

- The `correct` method contains the reference code with correct implementation.
- The `check` method executes the student's code on one data test. It returns a string code indicating the kind of error and that will be used in the analyser script.
- The `run` method runs the whole test suite. A test suite is composed of a set of predefined tests (to force testing some limit cases) and of a set of random tests (to disallow the student to hardcode the answers in his/her solution).

Note that, as the program is executed inside a sandbox with no contact to the outside world, it cannot provide direct feedback to the student. Instead, the program prints out the results to be analysed by the analyser script.

The template program just shown in Fig. 4 is a standard test suite program. The flexibility of Pythia makes it possible to have richer kind of feedbacks. In the task shown in Fig. 5, the student is required to implement a search algorithm on an array of natural numbers. The expected solution is a dichotomic search. The parameter stack that the student receives is an array-like object whose [] method has been overridden so as to be able to count the number of times the array is accessed. The provided feedback then includes a plot of the number of accesses compared to the size of the problem (i.e., the size of the array). The plot graphically shows to the student the time complexity of his algorithm, compared to the expected complexity. The plot is generated by executing the student's code and the reference code on arrays with increasing size. Outputs generated by the template program contains the number of array accesses and are used by the analyser script to generate the plot with Google Chart Tools.

The distinction between the template program and the analyser script opens a lot of opportunities.

The teacher can also use feedback to grade students, e.g., in the context of using Pythia for homework. Grading is achieved by customising the analyser script to generate private information for the teacher alongside the public information dedicated to the student. The two parts can be linked to a private and a public test set respectively, defined in the template program.

Moreover, by separating the analyser script, it is possible for a teacher to support multilingual courses and provide feedbacks in multiple languages (which is useful in Belgium which has three national languages). The multi-language support is managed in the analyser script.

For now, those customisation are not easy to realise since it should be done in the analyser script by the teacher. Future work includes developing a framework to ease the task of writing template programmes and analyser scripts for teachers.

4. The Pythia Framework

This section describes the global architecture of the Pythia framework. The framework allows students to solve tasks and get useful feedback about their submissions. Such feedback involves running user-submitted code. Special care has been taken in Pythia to handle the risks of executing buggy or malicious programs. Please note that the framework is

Where are my shoes?

Context

Whenever you need new shoes, it is always the same routine. You go to the shop, find the kind of shoes you like and then you have to search through the stack of boxes in order to find the pair with the most likely suited shoe size.

Hopefully, the boxes are not ordered randomly. They are ordered by increasing shoe size, the box on the top being the one with the smallest shoe size. Such ordering helps a lot in the process of finding if the shoe size you are searching for is available or not.

Question

Suppose that there are 13 boxes for the shoes you are interested in. How many boxes will you have? to check **in the worst case** to find whether there is or not a pair with size 8

13

Your answer is right! In the worst case, you will have to check each box once in order to find whether one box has a pair of shoes with size 8.

Write a function that takes two parameters: stack represents **the stack of shoes boxes** and size is the shoe size you are searching for. The function you have to write must return True if a pair of shoes with the specified size exists in the stack and False otherwise. Your function should be **as efficient as possible**, that is the number of times it looks at the size of a box should be as small as possible.

```
def hasShoes (stack, size):
```

```
for x in stack:
     if x == size:
       return True
   return False
```

All the test sets passed successfully on your code. But your code may be more efficient as testified by the following plot showing your code's execution time compared to the execution times of the reference solution.

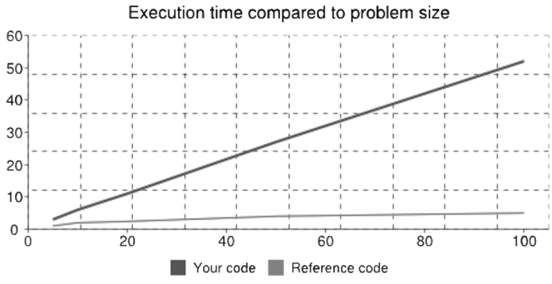


Fig. 5. The feedback mechanism is very versatile. For this problem, the student is presented a feedback with a plot representing the time complexity of his algorithm in a graphical way. The expected solution, a dichotomic search, has logarithmic time complexity.

language-agnostic. Problems can be written in different programming languages as long as an interpreter or compiler is available. Figure 6 illustrates the different components of the framework and the relations between them.

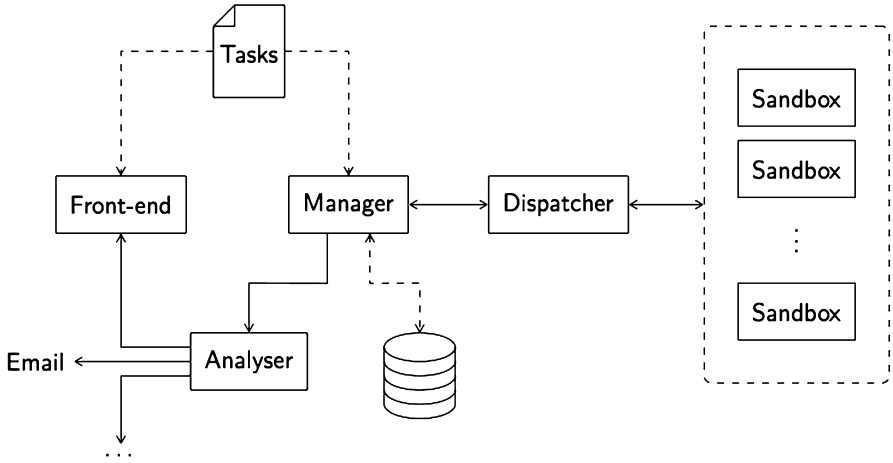


Fig. 6. The Pythia framework is subdivided into separate components communicating with each other with simple HTTP requests. Plain arrows represent communication between components and dashed arrows represent information access.

The programming courses and the problems are defined as *task descriptions* as shown in the previous section. Explanation and questions are presented to the student through the *front-end*. Using the front-end web application, the student can submit answers to the questions. Answers are used by the *manager* to fill in the placeholders of the template program. The resulting complete program is executed in a safe environment and its outcome is analysed by the analyser script. The remainder of this section dives more into the inner working of the system.

The manager, dispatcher and sandboxes are all small web servers communicating with each other with simple HTTP requests. Currently, we assume the existence of a shared filesystem between all components. Such assumption holds when all components are run on the same machine or if a networked filesystem is used.

The *manager* serves two roles. When it receives a submission from the frond-end, the manager merges the template program with the student's answers. The complete program, called a *job*, is then sent to the dispatcher. Later, when the job's execution has completed, the manager launches the *analyser* script. The analyser is responsible for providing feedback, based on the results of the computation. Feedback to the student can be given through the front-end web interface, by email, or by other means. The analyser can also provide the student's grade to the teacher. In order to provide statistical information to the users and to follow them, the manager stores information in a database.

Jobs are executed in *sandboxes* due to security concerns posed by buggy or malicious code. A sandbox is a disposable virtual machine that is created specifically for a job. Full virtual machines that emulate hardware peripherals, like KVM or VirtualBox, are not suitable for such task as they are too slow to start up. Instead, Pythia uses User-mode Linux (Dike, 2000), a port of the Linux kernel implementing drivers by forwarding the requests to the host Linux kernel. The sandbox runs a heavily trimmed down version of ArchLinux and is able to boot in under one second. The root filesystem being read-only,

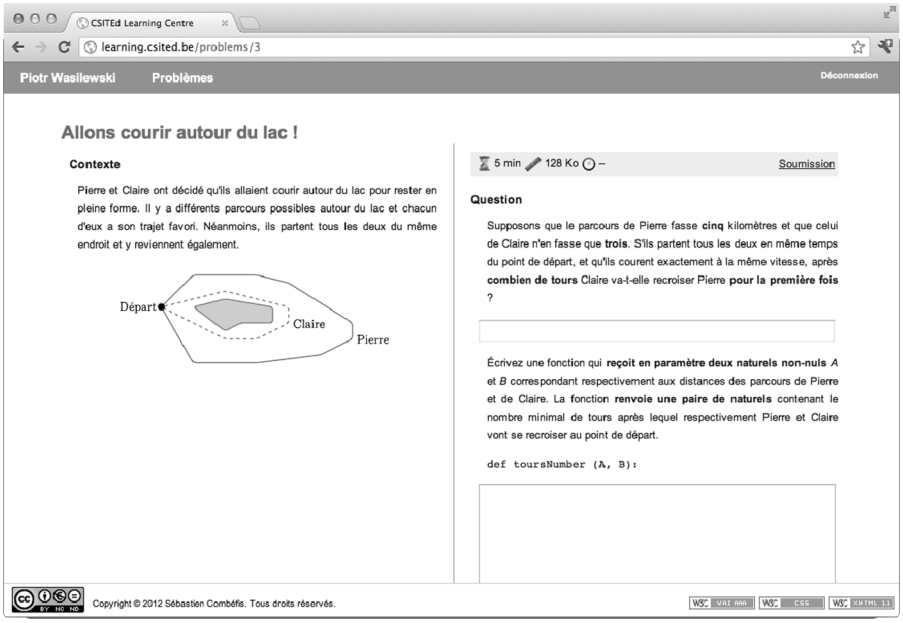


Fig. 7. The students complete tasks using the web-based front-end.

the job's output is written to a RAM-backed temporary filesystem. As such, the total amount of main and storage memory a job can use is capped by the amount of memory that is made available to the sandbox kernel. At the end of the job execution, the output files are copied back to the host in the shared filesystem.

Finally, the *dispatcher* makes the link between the manager and the sandboxes. The dispatcher maintains a queue of jobs to be run and dispatches them to idle sandboxes, which may or may not be on the same machine. Once a job is finished, the dispatcher notifies the manager.

5. Evaluation

An implementation of the Pythia framework is currently under development. Figure 7 shows the front-end of the *"Let's go for a tour around the lake"* problem. Once finalised, the working prototype will be used to evaluate our approach in two steps.

During the first step, a small group will be acting as beta-testers for both the framework itself and the courses and problems available on the platform. The group will consist of about ten persons, mostly bachelor students in computer science and in engineering but also older people with very few programming knowledge.

The second step will be to send invitations to secondary school pupils (12–18 years old) who are the main target audience of Pythia.

6. Conclusion

To promote computer science and to answer the needs highlighted by the contestants of the Belgian Olympiad in Informatics, the Pythia framework, a web-based learning platform, is being developed.

The framework proposes a set of programming courses and isolated problems. Students' programs submitted as answers are safely executed and the students receive fully automated feedback. The proposed problems are designed with the students' motivation in mind. Following the philosophy of learning by doing, the tasks aim at keeping the students interested in continuing and learning.

A prototype has been developed to evaluate our approach.

Future work includes strengthening the framework, creating more tasks and courses, supporting more programming languages (we currently target Python), thinking about the internationalisation of the front-end (useful in a multi-languages country such as Belgium). Future work also includes extending the framework to ease the task of writing a problem for the teacher, that is the description, the template programme and the analyser script. In the long run, the framework should also be tested in other contexts such as automated homework grading or online contests. In such applications, extensions like cheating detection could be considered.

References

Bell, T., Alexander, J., Freeman, I., Grimley, M. (2009). Computer science unplugged: school students doing real computing without computers. *Journal of Applied Computing and Information*, 13(1), 20–29.

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

Codecademy. `http://www.codecademy.com`.

Combéfis, S., Leroy, D. (2011). Belgian olympiads in informatics: the story of launching a national contest. *Olympiads in Informatics*, 5, 131–139.

Dewey, J. (1938). *Experience and Education*. New York, The Macmillan Publishing Company.

Dike, J. (2000). A user-mode port of the Linux kernel. In: *Proceedings of the 4th Annual Linux Showcase & Conference*, Atlanta, GA, Usenix.

hackzor. `http://code.google.com/p/hackzor/`.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., Resnick, M. (2004). Scratch: a sneak preview. In: *Proceedings of the 2nd International Conference on Creating, Connecting, and Collaborating through Computing*, Kyoto, Japan, 104–109.

Mareš, M. (2009). Moe – design of a modular grading system. *Olympiads in Informatics*, 3, 60–66.

Project Euler. `http://projecteuler.net`.

Rubymonk. `http://rubymonk.com`.

Try Python. `http://www.trypython.org`.

Try Ruby. `http://tryruby.org`.

uevalrun. `http://code.google.com/p/pts-mini-gpl/wiki/uevalrun`.

Viau, R. (1998). *La Motivation en Contexte Scolaire*. Édition de Boeck (2ème édition), Bruxelles.

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.

V. le Clément de Saint-Marcq is a PhD student at the Université catholique de Louvain in Belgium, funded as a research assistant by the FNRS, the national fund for scientific research. He works for the Computing Science Engineering Division of the ICTEAM Institute. Together with Sébastien Combéfis, he is involved in training the Belgian delegation to the IOI.

Olympiads in Informatics, 2012, Vol. 6, 44–52

Insight Tasks for Examining Student Illuminations

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

Abstract. Students who attend the national Olympiad process join the activity with diverse programming backgrounds and experience. Thus, in the process of selecting the better students, we examine their competence, given their heterogeneous backgrounds. In various parts of this process, we may want to "mask" programming advantage. This may be done by posing insight tasks that require limited programming knowledge. In this paper, we display such tasks, of various levels of difficulty, and elaborate on their relevance. Particular emphasis is put on the importance of assertional observations, creative operational ideas, and suitable problem representations.

Key words: insight tasks, problem solving, problem representation.

1. Introduction

Consider the following *Find Duplicate* task (posed to me by Eric Roberts from Stanford University). Given an array $\boldsymbol{A}$, of N integers, such that each integer is in the range $[1..N-1]$, devise an $O(N)$ time, $O(1)$ space algorithm for finding an integer that appears more than once in $\boldsymbol{A}$. (Note 1. The array $\boldsymbol{A}$ serves as a ROM, and the $O(1)$-space requirement refers to the RAM needed for the computation. Note 2. There may be one or more integers that appear more than once; output just one of them.)

The above task is a non-trivial algorithmic task. Yet, it requires little knowledge in programming. The desired outcome is an algorithm composed of basic programming features. No special knowledge of advanced computational schemes, data structure or design patterns (e.g., Astrachan *et al.*, 1998) is needed. It may also be posed without the terms $O(N)$ and $O(1)$. (One may require, instead, "no more than $5N$ iterations in the computation, and no more than a few variables".) Yet, two important elements are required – insightful illuminations and creativity.

Olympiad competitions require insight and creativity, embedded in (often rather subtle) programming. When students join the Olympiad activity, their programming backgrounds divert. Some are well acquainted with programming schemes and data-structures, whereas others are not. During the process of recognizing and selecting the top students, we should take into account their heterogeneous backgrounds.

In particular, at various points of the national selection process, we may want to "mask" programming advantages. Gaps in programming competence are much easier to "close" than gaps in insight and creativity. Thus, it may be beneficial to pose to students

tasks that require challenging insight, but basic programming, in order to learn about their: insight, illuminating perspectives, and creativity.

In this paper, we offer three algorithmic tasks, which require different levels of insight and creativity. We use these tasks, and additional ones, in our national practices and competitions (in conjunction with more involved programming tasks.) All three tasks involve sequences, which are common entities in programming. Their solutions require rather simple algorithmic schemes, and no knowledge of data structures. However, they require illuminating observations, which are reached by applying suitable problem solving heuristics (Polya, 1954; Schoenfeld, 1985) and discrete mathematics elements. The heuristics particularly involve various perspectives of problem representation. The discrete mathematics elements involve the notion of invariance, element decomposition, induction, basic mathematical principles (e.g., pigeon-hole), basic arithmetic, and features of mathematical entities.

Our objective is to elaborate on the tasks' required illuminations, and demonstrate the relevance of such tasks to Olympiad activities. We indicate our experience with posing these tasks to students. In the next section, we display the tasks, in increasing level of challenge. We then reflect, in the Discussion section, on the relevance of posing (such) insight tasks, with little programming, to Olympiad students.

2. Insight Tasks with Basic Programming

In what follows we display three tasks that encapsulate different insight characteristics. The tasks are suitable for different levels of problem solving experience. The first task is suitable after rather limited problem solving practice, whereas the latter two tasks are suitable after more advanced practice. The reader may better appreciate the illuminations needed in the task's solution, by trying to solve the tasks herself before reading their displayed solutions. Each task is displayed in a separate sub-section, which is titled according to its relevant problem solving perspective and discrete mathematics elements.

2.1. *Graphical Representation and Shape Invariance*

The following task involves on-the-fly sequence computation. We devised this task, in order to examine whether students conceive a sequence of values not just as a collection of separate elements; but rather as a chain of related elements, with some invariant characteristic.

> **Longest Balanced Sub-sequence**. We call a sequence of numbers *balanced*, if the amount of positive values in the sequence equals the amount of negative values. Given a list of N positive and negative integers, output the length of the longest balanced sub-sequence (of consecutive elements).
> For example, for the sequence $-1\ \ -5\ \ 1\ \ -7\ \ 8\ \ -6\ \ -9\ \ -2$ the output will be 4 (due to the sub-sequence $-5\ \ 1\ \ -7\ \ 8$ (or the sub-sequence $1\ \ -7\ \ 8\ \ -6$)).

A naïve solution to the task involves the examination of all the $(O(N^2))$ sub-sequences. One way to do that is by examining the longest balanced sub-sequence that

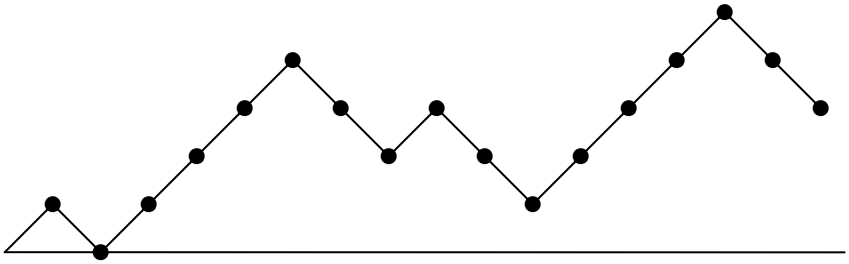


Fig. 1. Graphical illustration of the accumulated value of the difference between the amount-of-positives and the amount-of-negatives of the sequence: $5\ \ -2\ \ 1\ \ 3\ \ 7\ \ 9\ \ -9\ \ -1\ \ 6\ \ -7\ \ -1\ \ 2\ \ 8\ \ 3\ \ 1\ \ -2\ \ -1$ (the horizontal line marks the X axis, i.e., value 0).

starts from each element in the sequence. In our experience, students with limited abstraction competence demonstrate difficulties in devising a better solution.

Others seek illuminations from the basic task feature of the accumulated difference between "positives" and "negatives" throughout the sequence. The difference between the amount-of-positives and the amount-of-negatives accumulated so-far, changes exactly by 1 with each additional element. The intuitive tendency of problem solvers (in our experience) is to look at the absolute value of this difference. This absolute value depends on all the elements read so far. Yet, since the local change, of $+1$ or -1, in this difference depends at each point only on the new, additional element, we may also choose to look at the relative "behaviour" of this difference. One way to look at such behaviour is by drawing its graphical representation, as illustrated in Fig. 1.

In examining the graph of the accumulated difference of the 17-point sequence above, we may notice that the same values appear more than once. For example, the value 0 appears initially, and after processing the 2nd element; the value 1 appears three times (after processing the 1st, the 3rd, and the 11th elements); and the value 2 appears four times.

In the naïve computation, the longest sub-sequence for each starting point is found upon reaching the furthest location in which the initial value 0, of the accumulated difference, returns. However, upon looking at the above graphical illustration, we may notice that it is unnecessary to separately repeat the computation for each starting point. If we remove any prefix of the original sequence, the *shape* of the graphical behaviour of the remaining accumulated difference does not change. The only feature that changes is the "height" of this shape; i.e., its "distance" from the X axis.

This implies that if we start the computation, for example, only after the 4th point, then upon looking at the above shape we may regard the "height" 2 (which would be reached after processing the first four points of the original sequence) as 0; and we should seek the furthest location in which this "height" returns. More generally, we may formulate the following two observations:

> *The **shape** of the accumulated-difference behaviour, of any postfix of the original input, **remains invariant** after removing its corresponding prefix.*

A sub-sequence between two points, for which the value of the accumulated-difference is the same, is a balanced sub-sequence.

The above assertions imply that the longest balanced sub-sequence is the longest distance between two locations for which the value of the accumulated difference is the same. Thus, in the example of figure 1, the output will be 12, which is the largest distance between the furthest two points for which the value of the accumulated difference is 3 (points 5 and 17).

The latter observations yield a simple algorithm, of on-the-fly, $O(N)$ time computation, using $O(N)$ space. The algorithm will use an array $\boldsymbol{A}$, in which $\boldsymbol{A}[i]$ will keep the location of the first time that the value of the accumulated difference is i. (We leave open the question whether the computation may be conducted in $O(1)$ space.)

All in all, the algorithm is very simple. It is obtained after reaching the illuminations specified in the above observations. These illuminations derived from a graphical problem solving perspective, which illustrated the shape invariance of the behaviour of the accumulated-difference values.

2.2. *Problem Decomposition and Majority Characteristics*

The task presented in this section was originally posed and solved by Boyer and Moore (1991) (see note there), and later by Misra and Gries (1982). It involves the computation of majority in a sequence of elements. The primary asset of this task, with respect to our focus on illuminating observations, is that it may be solved in several ways, each based on different illuminations. The algorithmic schemes are simple and require no data structures (apart from a few variables). However, the required illuminations are not easily reached.

Majority Number of Appearances. Given a very long list of N integers, output a message saying whether one of the sequence elements appears a majority number of times (i.e., more than $N/2$ times); and if so – output this element.
For example, in the sequence $1\ \ 2\ \ 2\ \ 1\ \ 2$, the output will be "2 is majority".
Since the list is very long, it is impossible to keep it all in memory. However, the computation may *read the input twice*.

We posed this task to students in our advanced practice stage. They offered a few different solutions, based on different perspectives of decomposition, as thoroughly described in Ginat (2002). The very elegant solution that appears in the literature was offered by students only after some hint. Yet, a few students devised without a hint a creative solution of slightly higher time complexity (than the literature's elegant solution). We first elaborate on the student illuminations, and then display the elegant, literature solution.

One perspective that one may attempt upon approaching the task is to turn to the binary representation of the input elements, as each bit may only be of one of two values – 0 or 1. If we look at a single bit across the input, either the appearances of the value 0 will be in majority, or the appearances of 1 will be in majority; or, 0 and 1 will have exactly the same number of appearances. Thus, it may be beneficial not to look at each

integer in the input list as a separate unit, but rather follow an orthogonal point of view, and look separately at each position of the integers' binary representation, across the input. This yields an illuminating observation:

If there is a majority, then the value of each of its bits should appear a majority number of times across the input.

We may illustrate the above observation with the example 1 2 2 1 2. The binary representation of these integers is: 01 10 10 01 10. Since 2 is a majority, the value 1 appears a majority number of times in the left (most significant) bit, and the value 0 appears a majority number of times in the right bit.

However, we may obtain a majority value for each bit, without having a majority in the given input. This may occur for example with 1 2 2 3 0, whose binary representation is 01 10 10 11 00. (Although there is no majority, the value 1 appears a majority number of times in the left bit, and the value 0 appears a majority number of times in the right bit.) The case of no majority may also occur if one of the bits' values exactly $N/2$ times appears across the input.

In order to cope with the latter observations, we may seek a way to capitalize on the task characteristic that allows us to read the input twice. The following creative idea is very helpful:

We may divide the computation into two stages. In the first stage, we may compute the majority value for each bit across the input, and construct a ***candidate*** *for majority. If there is a majority, it can only be that candidate. In the second stage, we may go over the input again, and check whether the constructed candidate is indeed majority.*

This nice solution requires $O(N \log M)$ time and $O(\log M)$ space, when the binary representation of each integer requires M bits. (A separate counter should be kept for each bit of the binary representation.) The computation is simple, and requires little memory. Its novel illuminations stem from an orthogonal view of the input's bit representation.

The literature's solution is also based on the notion of candidate, but involves an inductive perspective rather than an orthogonal one. The majority in a list encloses a very simple, but powerful arithmetic characteristic:

If a value v is majority in a list, and we remove from the list two elements – v and an element u not equal to v, then v is still majority in the remaining list.

In fact, simple arithmetic shows that the removal of two non-equal elements even increases the "weight" of the majority in the list (examine, for example, a list of five elements that is reduced to a list of three elements).

The above observation yields a very elegant algorithm, which is based on "removing" pairs of different elements, in an on-the-fly processing manner, using only two variables – a variable that keeps an on-the-fly candidate and a variable that keeps the number of accumulated appearances of the candidate that were not yet matched with different values. The candidate may change again and again throughout the first stage of reading the input. The candidate that remains at the end of this stage is verified as majority in a second

stage, as in the previous solution. The detailed, elegant algorithm (see in the indicated literature) requires $O(N)$ time and $O(1)$ space.

All in all, both algorithms are based on simple characteristics of majority, and employ the notion of *candidate*. They both employ the notion of decomposition, but in very different forms – orthogonal decomposition across the input in the first solution, and inductive decomposition in the second one. Although both forms yield simple computations, they are not easy to reach, and require both insightful observations and creativity.

2.3. *Values as Addresses and Cycle Detection*

The task discussed in this section is the **Find Duplicate** task displayed in the Introduction. Although the final solution involves a simple computation scheme, the problem solving process here requires a series of illuminations, which in our experience are not trivial for problem solvers.

In our experience, students that were less acquainted with programming schemes, sometimes performed better than the more experienced ones. In particular, the more experienced students turned at first to familiar searching and sorting schemes that did not help them much. The less experienced students were more "open", and less fixated on familiar schemes, which they only knew for a limited extent.

The initial consideration that should be examined is how to capitalize on the primary task characteristic that all the values in $\boldsymbol{A}$ are in the range $1..N-1$. One simple observation that derives from the pigeon-hole principle is:

1. *Regardless of how we peek N times into cells of $\boldsymbol{A}$, we will surely see a value twice.*

We can devise more creative observations from the primary task characteristic (of range $1..N-1$), if we follow the perspective that cell values may be used during computation not only for arithmetic calculations, but also as pointers to addresses.

2. *Since each **value** in $\boldsymbol{A}$'s cells is an integer larger than 0 and smaller than N, we may look at each value **as a pointer** to a cell in $\boldsymbol{A}$.*
3. *No cell in $\boldsymbol{A}$ points to the cell $\boldsymbol{A}[N]$.*

The above observations yield a creative operational idea – to try to "walk" through $\boldsymbol{A}$'s cells:

4. *If we start a "walk" from $\boldsymbol{A}[N]$, our first step will surely lead to another cell in $\boldsymbol{A}$. (Note that this may not necessarily be the case if we start from $A[i]$, for $i <> N$.)*
5. *In walking N steps, we will surely visit some cell twice.*
6. *Once we visit a cell again, we also visit its successor again, and its successor's successor again, and so on.*
7. *Therefore, when we visit a cell for the second time, we actually just completed a cycle; and all our future steps will be in this cycle.*
8. *This cycle has an "entry cell", which is the first cell visited in the cycle. Two cells point at this "entry cell" – the cell visited just before entering the cycle, and the last cell of the cycle, just before "starting" it again. **These two cells, which point to the cycle's entry cell, keep the same value.***

Thus, our goal is to devise an algorithm that reaches one (or both) of these two cells, and extracts its value. How can we tell that we have reached one of these cells upon "walking" through $\boldsymbol{A}$'s cells? The following creative scheme answers this question:

9. ***After advancing*** N ***steps*** *through* $\boldsymbol{A}$*, starting at* $\boldsymbol{A}[N]$*, and referring to* $\boldsymbol{A}$*'s values as pointers,* ***we are guaranteed to be in the cycle****.*
10. *Once we are in cell* c*, in the cycle, we may start counting the number of steps that it takes to return to* c*.* ***This will tell us the length,*** l***, of the cycle****.*
11. *Now we are almost done. We will start a "walk" from* $\boldsymbol{A}[N]$ *again, and advance* l *steps. At this point, we will start an additional "walk" from* $\boldsymbol{A}[N]$*, with an additional pointer. We will continue with both "walks" concurrently. Since the distance between the two pointers that lead the "walks" is exactly* l *(the cycle length),* ***it will take us less than*** N ***concurrent steps until both pointers will reach, at the same time, the two different cells that point to the cycle's entry cell.***

We may observe the algorithm's execution in the following example. Let the array $\boldsymbol{A}$, composed of 12 cells, keep the following values: 3 10 2 1 4 3 10 2 7 5 1 9. We regard the index of the first cell as 1 (and not 0). We will "walk" 12 steps, starting at $\boldsymbol{A}[12]$: $12 \rightarrow 9 \rightarrow 7 \rightarrow 10 \rightarrow 5 \rightarrow 4 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 10 \rightarrow 5 \rightarrow 4 \rightarrow 1$. Now we are in a cycle. We will count the number of steps until we reach the value 1 again. This will take 6 steps. We will now start a new "walk", $w1$, from $\boldsymbol{A}[12]$, and an additional "walk", $w2$, after 6 steps of $w1$. We will proceed concurrently with the two "walks", and after two concurrent steps with the "walks", both "walks" will reach the value 10 – $w1$ will reach it in $\boldsymbol{A}[2]$ and $w2$ will reach it in $\boldsymbol{A}[7]$. The algorithm will output the value 10.

All in all, the time complexity of the computation is $O(N)$, and the space complexity – $O(1)$. We believe that the primary reason for the task's challenging nature is the need to progress gradually through a series of illuminating observations, as displayed above. These observations combine both illuminating insight and algorithmic creativity.

3. Discussion

The three presented tasks involve both assertional and operational observations (Dijkstra *et al.*, 1989). In order to solve each task, one had to first recognize some assertional characteristics, such as the two assertions in the first task, and the initial observations in the other two tasks. But, this by itself was insufficient. After recognizing these characteristics, one had to capitalize on them, and devise a suitable algorithmic solution. In the latter two tasks, this capitalization required creative operational ideas, of how to elegantly conduct the computation.

The assertional characteristics correspond to Ryle's notion of "knowing what", and the operational ideas correspond Ryle's notion of "knowing how", in devising operational schemes (Ryle, 1949). The assertional characteristics encapsulated insightful illuminations that one had to extract from the problems' features. They did not require knowledge in programming and computational schemes, but rather a focused reasoning perspective. A competent problem solver could reach them without advanced knowledge in algorithmics.

The operational ideas in this study did require some knowledge and experience in algorithmics, but only to a limited extent. In the first task (Longest Balanced Subsequence), they involved simple array utilization. In the second task (Majority), they involved the notion of a candidate. Novices see a trivial example of the notion of candidate already in the computation of Max in a list. Here, they had to "take it one step further" and combine it in a two-stage computation. This required creativity. In the third task (Find Duplicate), the required operational ideas were subtler, as they involved cycle detection. Yet, experience with common algorithmic schemes, such as searching and sorting, as well as experience with Data Structures, could not help much. One needed a creative operational idea for conducting the computation.

One primary aspect that was required in all three tasks was that of suitable task representation. In the first task, the illumination derived from graphical representation. In the second task – it derived from an orthogonal view of the data, across the input (the students' solution), and an inductive perspective of majority characteristics in a list. In the third task, the illumination started by viewing the array values as array addresses.

Problem representation, and problem restructuring, play a key role in problem solving, particularly in the case of insight problems (Knoblich *et al.*, 1999; Ash and Wiley, 2006). Both notions relate to Representational Change Theory (Knoblich *et al.*, 1999), and the invocation of a suitable heuristic, of "restating the problem" (Polya, 1954).

We expect Olympiad students to turn to such representation perspectives, as a part of their demonstration of competence in problem solving. Those that meet difficulties with turning to the suitable representation may experience fixation or impasse that inhibits them from making progress. We would like our competent students to overcome impasses of this kind. Thus, part of our student evaluation is based on examining their behaviour in solving tasks like those presented here. Based on our experience, we recommend embedding such tasks during the process of recognizing the better students, at various stages of the national Olympiad activity.

References

Ash, I.K., Wiley, J. (2006). The nature of restructuring in insight: an individual-differences approach. *Psychonomic Bulletin & Review*, 13(1), 66–73.

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.

Boyer, R.S., Moore, J.S. (1991). A fast majority vote algorithm. *Automated Reasoning: Essays in Honor of Woody Bledsoe*. Automated Reasoning Series, Kluwer, 105–117. (The algorithm was invented in 1980.)

Dijkstra, E.W. *et al.* (1989). A debate on teaching computing science. *Communications of the ACM*, 32(12), 1397–1414.

Knoblich, G., Ohlsson, S., Haider, H., Rhenius, D. (1999). Constraint relaxation and chunk decomposition in insight problem solving. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 25(6), 1534–1555.

Misra, J., Gries, D. (1982). Finding repeated elements. *Science of Computer Programming*, 2, 143–152.

Ginat, D. (2002). On varying perspectives of problem decomposition. In: *Proc. of the 33rd SIGCSE Technical Symposium on CS Education*, ACM, 331–335.

Polya, G. (1954). *How to Solve It*. Princeton University Press.

Ryle, G. (1949). *The Concept of Mind*. Barnes and Noble.

Schoenfeld, A.H. (1985). *Mathematical Problem Solving*. Academic Press.

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.

Olympiads in Informatics, 2012, Vol. 6, 53–68

Learning Algorithms with Unified and Interactive Web-Based Visualization

Steven HALIM, Zi Chun KOH, Victor Bo Huai LOH, Felix HALIM
School of Computing, National University of Singapore
Computing 1, 13 Computing Drive, 117417, Singapore
e-mail: {dcssh, zichun, u0805026, felix.halim} @ nus.edu.sg

Abstract. We present a *unified* and *interactive* web-based visualization of various classical and non-classical algorithms at

`http://www.comp.nus.edu.sg/~stevenha/visualization`.

Our collection of algorithm visualizations has the following advantages over many other web-based algorithm visualizations in the Internet – (1) it has visualizations of various non-classical algorithms that currently cannot be found elsewhere in the Internet; (2) it is interactive; users (usually students) can enter *their own* input data to test the behavior of the algorithm; (3) it has a *consistent* user interface across different algorithm visualizations that are currently available; (4) it is built with HTML5 making it accessible on modern portable PCs including tablets and smartphones. User studies in two algorithm classes in NUS show that our visualizations is of immense help to *some* students who prefer to learn *visually*.

Key words: unified, interactive, algorithm, visualization.

1. Introduction

To teach *data structures and algorithms* (henceforth referred to as *algorithms*) in a typical Computer Science class, a professor/lecturer/instructor/teacher (henceforth referred to as *teacher*) has to illustrate at least one (and preferably several) working example(s) on how a particular algorithm works. The typical teaching methods employed are as follows:

1. The examples are pre-scripted in printed handouts, textbooks, or in PowerPoint (or equivalent presentation software) slides with a collection of **static** pictures/ diagrams. A slightly better approach is to use a hardcoded presentation with animations that clearly demonstrates the steps of the algorithm. One drawback of this approach is that it is possible for the teacher to advance the animation too quickly. Also, it is difficult for the teacher to show another example that was not hardcoded or to demonstrate spontaneous examples asked by the students; the teacher has to either draw it directly on the PowerPoint slide manually (using the mouse or a stylus pen) or to draw the example on the board by hand (method 2).
2. The teacher hand-draws the examples on the board. Unlike PowerPoint where the teacher can advance to the next slide rapidly, this hand-drawing method allows the students to assimilate the steps taken by the algorithm in proper "human (student)

pace" as drawing successive steps of the algorithm may take some time. However, the drawback is that such manual drawing can be very time consuming if not executed/prepared properly. It is also error-prone if the algorithm is complex or if the teacher is in a rush. Students have to not only understand the algorithm but also copy the teacher's examples into their notes in order to review the illustrations later. The conflicting task to internalize the algorithm and copy the examples manually is exacerbated if the lesson is not recorded.

3. The teacher simply points the students to certain existing websites (usually created by independent developers) with some algorithm animations. Some of these websites are effective tools for teaching algorithms, as will be elaborated in Section 2 below. However, this method has drawbacks: most of these websites only support *a few* classical algorithm visualizations. Thus, if the teacher refers to one website for one algorithm and *different* website for *another*, the students may be confused by the different conventions and styles employed. The diversity of the various user interfaces adopted by these websites adds *another layer* of learning curve for the student. Some students have certain inertia that reduces their will to visit many different links. Also, this method *cannot be used* if the teacher is teaching a rare, non-classical algorithm that nobody has visualized before.

In our opinion, a better way to teach algorithms is to give a combination of a well-chosen and scripted example with animation (to introduce the concept *for the first time*) combined with two-or-three more examples on student-supplied input data. There are two conditions for this approach to be effective – the visualization of the spontaneous examples must be animated in a manner that is consistent and bug-free, and it must be done quickly and not take up too much time.

We made the decision to deliver our visualization on a web-based platform through the browser so that algorithmic knowledge is accessible to a larger pool of audience worldwide. Almost every devices nowadays – desktops, laptops, tablets, and even smartphones – have *built-in* web browser(s), e.g., Mozilla Firefox, Google Chrome, Safari[1], etc., that support HTML5 natively. There is thus no need to install any program or worry about cross-platform issue.

The choice of having interactive over static visualizations is for the students to gain more in-depth understanding about the algorithm being visualized. We believe that the students will have better understanding on how the algorithm works if they can enter the input data themselves.

With the rapidly increasing proliferation of portable devices with Internet connectivity: laptops, tablets, and smartphones[2], many students can access the web-based visualization during the lesson and follow the teacher's examples on their own devices. After class, students can then play around with the visualization again at their own pace/time to further reinforce their understanding of the algorithm's behavior.

[1] Safari is the default browser built-in on iPhone and iPad. This is worth mentioning because one of the features of our visualization is its accessibility on modern devices.

[2] As an illustration, during the fourth quarter of 2011, Apple sold 17.07 million iPhones (smartphone) and 11.12 million iPads (tablet), `www.apple.com/pr/library/2011/10/18Apple-Reports-Fourth-Quarter-Results.html`, last accessed: April 2012.

On the pedagogy side, having such visualizations will reduce the number of questions in quizzes and exams that simply test students' knowledge of how the algorithm will behave given certain input data. Such questions are now redundant as the students are expected to know how the algorithm works by using the visualization. Thus, the teacher can focus on testing on *more advanced usages* of the algorithms.

The choice of unified interface will also benefit the students. It is reasonable to expect a typical Computer Science class having more than one algorithm covered in the syllabus. By having a unified interface, students can expect similar look and feel when using different algorithm visualizations. Also, the students can just bookmark one webpage and use it for the *entire semester*. It is better than having a collection of different websites to visit. The student need not have to navigate through the lecture slides or handouts to get to the URLs recommended by the teacher. Every extra inconvenience increases the student's inertia and can be a possible cause for the student to be lazier and not bother to understand the algorithms properly.

This paper is organized as follows: In Section 2, we will detail and dissect the current available web-based algorithm visualizations for the algorithms listed in "Competitive Programming 2" (Halim and Halim, 2011). This competitive programming handbook is written by the first and the last authors of this paper. A substantial number of algorithms are discussed in that book, of which many do not have publicly available visualizations yet. At the same time, we also discuss the weaknesses of the existing visualizations through the lens of the objectives that we want to achieve, namely: provide visualizations for the non-classical algorithms, use unified interface, provide the ability to interactively enter user (student)-supplied input data, and provide the ability to access the visualization using modern web browsers particularly on smartphones. In Section 3, we discuss the details of our visualization project. In Section 4, we report the initial feedbacks from 31 students in National University of Singapore (NUS) who used the first version of the visualizations during semester 2 of academic year 2011/2012. In Section 5, we list down our preliminary conclusions regarding the effect of this visualization project. We conclude this paper by listing down our plans to continually improve this visualization project in the near future.

2. Existing Web-Based Algorithm Visualization

In this section, we summarize the results of our analysis of various algorithm visualization websites on the Internet as of April 2012. This report does not take into account algorithm visualizations that are not publicly available in the Internet. The list of algorithms is taken from "Competitive Programming 2" handbook (Halim and Halim, 2011). From our analysis, we found that there is a huge imbalance on the number of visualizations of classical – and usually easier – algorithms like sorting algorithms, binary search on a sorted array etc., versus the non classical and rare (usually harder) algorithms like Binary Indexed (Fenwick) Tree (Fenwick, 1994), finding Strongly Connected Components of a directed graph (Tarjan, 1972), computing the Maximum Flow of a network (Edmonds and Karp, 1972), etc.

2.1. *Standalone Visualizations*

We split our results into two categories: Standalone and Unified visualizations. We categorize the existing web-based algorithm visualizations in Table 1 as standalone visualizations because these websites only have a single to no more than three algorithm animations inside, i.e., their developers do *not* create visualizations for other algorithms. This causes an unnecessary learning curve when students have to learn different algorithms and have to visit different visualization websites. In order to make Table 1 concise, we only pick several representative visualizations per topic for the classic algorithm visualizations.

Table 1
Standalone visualizations (all URLs are correct as of April 2012)

Topic	Representative visualization examples	N	H	O
Various linear data structures	apbrwww5.apsu.edu/myersb2/linked_list_animation.htm		H	O
	www.u-www.cosc.canterbury.ac.nz/mukundan/dsal/StackAppl.html		H	O
Various sorting algorithms	www.cs.auckland.ac.nz/~jmor159/PLDS210/Java/q_sort/tqs_new.html		H	O
	www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/mergeSort/mergeSort.html			O
	www.cs.oswego.edu/~mohammad/classes/csc241/samples/sort/Sort2-E.html		H	O
Bitmask or bit manipulation	(see Section 4.1)	N		
Basic BST (Table)	aleph0.clarku.edu/ ~ achou/cs102/examples/bst_animation/BST-Example.html			O
	www.cs.jhu.edu/~ goodrich/dsa/trees/btree.html			O
	www.csc.liv.ac.uk/~ ullrich/COMP102/applets/bstree/			O
Balanced BST (AVL)	webdiis.unizar.es/asignaturas/EDA/AVLTree/avltree.html			O
	www.site.uottawa.ca/~ stan/csi2514/applets/avl/BT.html			O
	www.cs.jhu.edu/~ goodrich/dsa/trees/avltree.html			O
Heap (priority queue)	nova.umuc.edu/~ jarc/idsv/lesson2.html			O
	www.cs.auckland.ac.nz/~ jmor159/PLDS210/heaps.html			O

To be continued

Continuation of Table 1

Topic	Representative visualization examples	N	H	O
Graph DS (Adj matrix or list)	(part of nova.umuc.edu/~jarc/idsv/) (see Section 4.4)	N		
Union find disjoint sets	research.cs.vt.edu/AVresearch/UF/			O
	www.cs.unm.edu/~rlpm/499/uf.html			O
Segment tree	(in our future works)	N		
Binary indexed (Fenwick) tree	(see Section 4.3)	N		
n-queens recursive backtracking	www.animatedrecursion.com/advanced/the_eight_queens_problem.html		H	O
	yuval.bar-or.org/index.php?item=9		H	O
Classic dynamic programming	(in our future works)	N		
Graph traversal: BFS/DFS+	www.cs.sunysb.edu/~skiena/combinatorica/animations/search.html		H	
	www.rci.rutgers.edu/~cfs/472_html/AI_SEARCH/SearchAnimations.html		H	
Finding cut vertices, bridges, SCCs	(see Section 4.5)	N		
MST (Prim's)	www.unf.edu/~wkloster/foundations/PrimApplet/PrimApplet.htm		H	O
	students.ceid.upatras.gr/~papagel/project/prim.htm		H	O
MST (Kruskal's)	www.unf.edu/~wkloster/foundations/KruskalApplet/KruskalApplet.htm		H	O
	students.ceid.upatras.gr/~papagel/project/kruskal.htm	H	O	
SSSP (Bellman Ford's)	(see Section 4.7)	N		
SSSP (Dijkstra's)	www.unf.edu/~wkloster/foundations/DijkstraApplet/DijkstraApplet.htm		H	O
	www.dgp.toronto.edu/~jstewart/270/9798s/Laffra/DijkstraApplet.html			O
APSP (Floyd Warshall's)	www.pms.ifi.lmu.de/lehre/compgeometry/Gosper/shortest_path/shortest_path.html		H	O
Network flow (Edmonds Karp's)	felix-halim.net/research/maxflow/index.php		H	O
	www.eecs.wsu.edu/~cook/aa/lectures/applets/ek/MaxFlowHome.html			
Algorithms on DAG	(in our future works, e.g., topological sort, shortest/longest path, counting paths)	N		
Algorithms on tree	(in our future works, e.g., shortest/longest path, all pairs shortest paths, Lowest Common Ancestor)	N		
Algorithms on bipartite graph	www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/main/index.shtml		H	O

To be continued

Continuation of Table 1

Topic	Representative visualization examples	N	H	O
String matching	cgjennings.ca/fjs/index.html			O
Suffix tree	illya-keeplearning.blogspot.com/2009/06/suffix-trees-java-applet.html			O
Suffix array	felix-halim.net/pg/suffix-array			
Algorithms on polygon	(see Section 4.9)	N		
Convex hull (Graham's scan, Andrew's, Jarvis March)	www.cs.princeton.edu/courses/archive/spr10/cos226/demo/ah/GrahamScan.html		H	O
	nms.lcs.mit.edu/∼aklmiu/6.838/convexhull/			O
	www.cs.unc.edu/∼snoeyink/demos/ch/Jarvis.html			O

In the same table, we also highlight *three* possible potential issues of these *standalone* visualizations with respect to our objectives listed in Section 1 using the following codes:

1. **N** (not available): This algorithm is the rare one where Internet searches using relevant keywords do not yield public web-based visualization for it yet.
2. **H** (hardcoded, not interactive): This algorithm visualization is hardcoded (especially the ones that are just animated GIF), i.e., users cannot use their own input data to see how the algorithm being visualized will behave on the *new* input data.
3. **O** (old technology): This (older) algorithm visualization is using older technology (e.g., Flash, Java Applet) that is not truly portable as it cannot be accessed via modern smartphones (e.g., iPhones) or tablets (e.g., iPads).

2.2. *Unified Visualizations*

Other than the standalone visualizations listed in Table 1, we also found the following unified visualizations that offer *more than three* visualizations in their website:

1. algoviz.org is basically a portal that contains links to various algorithm visualization websites. In the future, our visualization project will also be listed there. However, as of April 2012, some of our algorithm visualizations in Section 3 (e.g., Fenwick Tree, Tarjan's SCC, etc.) are not yet available there.
2. www.ansatt.hig.no/frodeh/algmet/animate.html is another portal that compiles URLs to other visualization websites (similar nature with algoviz.org).
3. nova.umuc.edu/∼jarc/idsv/ has visualizations for binary trees, several graph representation and algorithms, and sorting.
4. www.csse.monash.edu.au/∼dwa/Animations/index.html has several visualizations similar nova.umuc.edu/∼jarc/idsv/.

5. `research.cs.vt.edu/AVresearch/` has several algorithm visualizations developed between 2003 and 2009.
6. `www.cs.usfca.edu/~galles/visualization/Algorithms.html` is the closest unified visualization project that is the most similar with the ideas presented in this paper. It also built with HTML5. We highlight some of the differences between our visualization and this project in Section 3.

3. Our Unified and Interactive Web-Based Algorithm Visualization

These are the algorithm visualizations that we have built as of April 2012. Our focus at the early stages of this visualization project is to provide visualizations of the non-classical algorithms rather than the more classical ones. We also highlight the differences and the extra features that we have incorporated for the algorithms that have been visualized by some other programmers and can be found publicly on Internet.

3.1. *Bitmask/Bit Manipulation/Bit String/Lightweight Small Set of Boolean*

Bit manipulation (bitmask) of integer values is usually a tricky subject for new Computer Science students. Therefore, having a bug-free visualization on bitmask like the one shown below will help both the teachers in showing various examples and the students in understanding the concept. For example, 42 | (1 « 2) = 46. This text-based visualization (see Fig. 1) will instantly show the necessary bit manipulations. We currently cannot find similar visualization in the Internet.

3.2. *Binary Heap Data Structure (Priority Queue)*

Binary Heap is a classical data structure commonly used to implement priority queue. There are (many) other similar visualizations in the Internet. The only strong point of having this visualization (Fig. 2) in our project is the similar unified look-and-feel as with the other visualizations in our project.

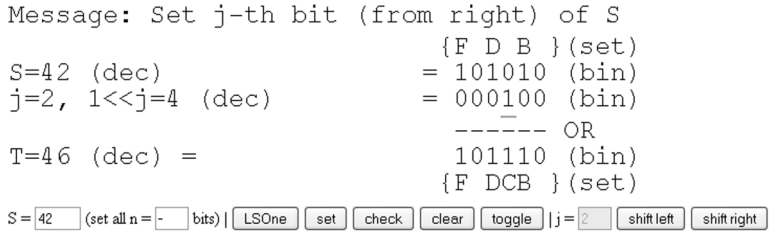


Fig. 1. Bitmask visualization.

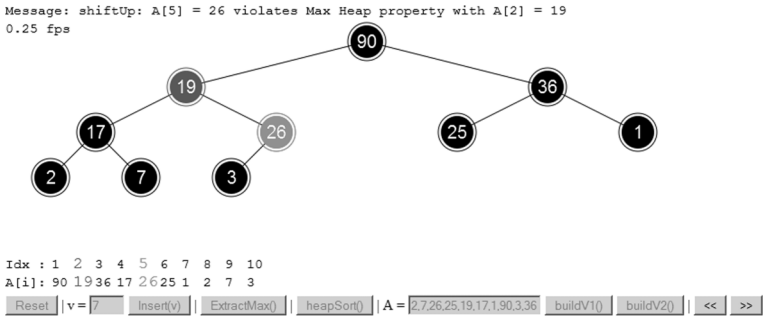


Fig. 2. Binary heap visualization.

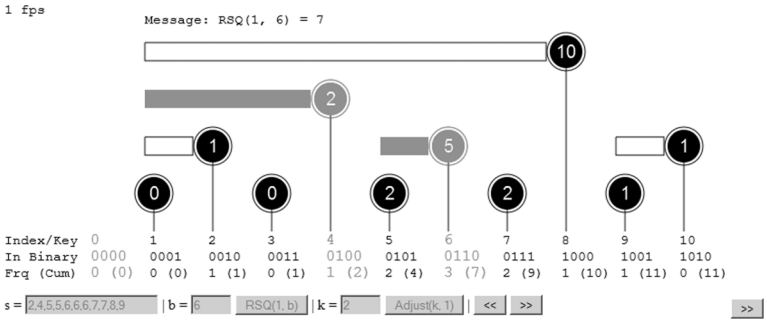


Fig. 3. Binary indexed (Fenwick) tree visualization.

3.3. *Binary Indexed Tree (BIT) or Also Known as Fenwick Tree*

BIT was invented by Fenwick (1994). This data structure is used to efficiently compute dynamic range sum queries (RSQ/cumulative frequency table). It has interesting usages in programming competitions. This data structure is included in the International Olympiad in Informatics (IOI) syllabus (Forišek, 2009), making it one of the studied data structure for IOI (high school) students. BIT operations can be cryptic for new users. For example, RSQ(1, **6**) is a summation of RSQ(1, **4**) + RSQ(5, **6**) because integer **6** ("1**1**0" in binary) becomes integer **4** ("1**0**0" in binary) if we strip off its least significant one. If we continue another step, integer **4** becomes integer **0** ("**0**00" in binary) and we stop here. This process is clearly shown and animated in Fig. 3. There is no existing BIT visualization that we can find on the Internet before ours.

3.4. *Graph Data Structure*

Graph visualizations are abundant in the Internet. However, almost every such visualizations uses *hardcoded* input graphs which limit its usage. We speculate that this is because it is hard to provide the input data for a graph and graph layout is another big subject by its own. Our visualization allows users to *draw their own* graph (Fig. 4). By having the users draw the graph themselves, we eliminate the need of using sophisticated graph

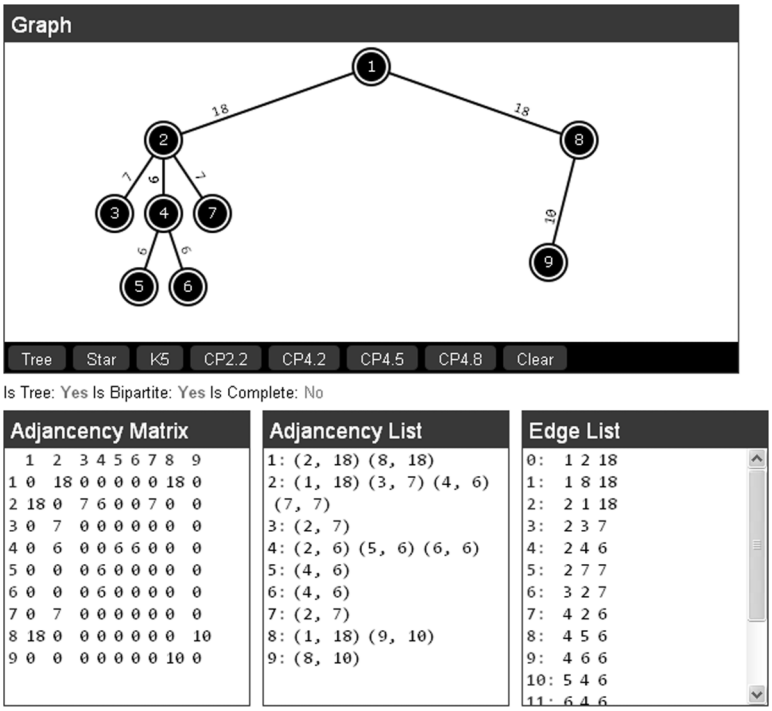


Fig. 4. Graph data structure visualization.

layout algorithm to layout the graph. Users can add/delete new vertex, add edges between vertices while simultaneously specify its direction and weight, and the visualization will be automatically updated with the correct Adjacency Matrix, Adjacency List, and Edge List representation of the user's input graph. As an additional feature, we also check if the graph is a tree, is a bipartite graph, or is a complete graph.

3.5. *Graph Traversal: Depth-First Search and Breadth-First Search*

In this visualization, users can draw their own graph as above and run one of two classical graph traversal algorithms (DFS or BFS). Many other graph traversal visualizations just stop at this point. However, we have incorporated extra features as these DFS and BFS algorithm can yield extra information about the graph structure. For DFS on undirected graph, we also compute vertices/edges that are classified as articulation points (cut vertices)/bridges. For DFS on directed graph, we also compute its Strongly Connected Components (SCCs) using Tarjan's (1972) algorithm as shown in the static figure blow (Fig. 5). Visualizations of these two DFS variants are currently not found elsewhere on the Internet. For BFS on unweighted graph, we also compute the shortest paths from starting vertex.

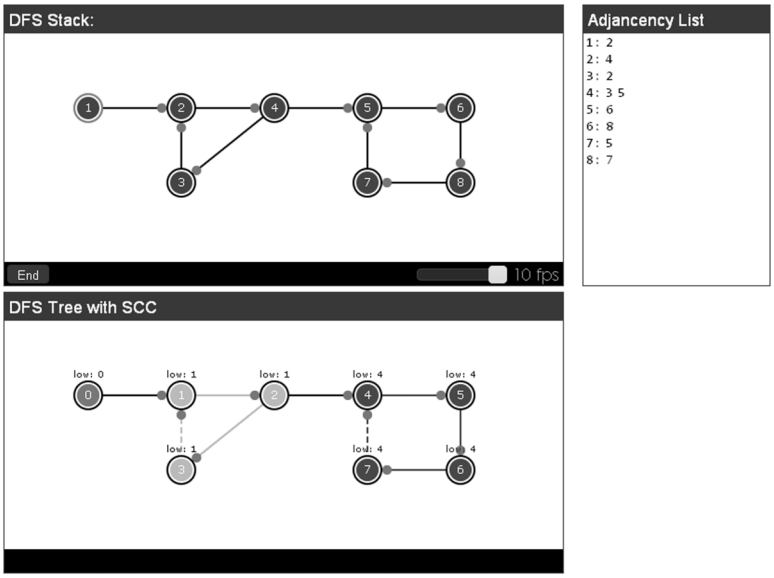


Fig. 5. Graph traversal visualization (showing the execution of Tarjan's SCC/DFS algorithm).

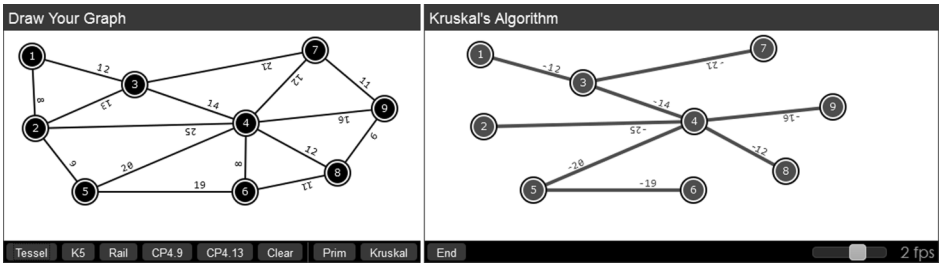


Fig. 6. MST visualization (showing the result of Kruskal's on input graph with all edge weights *negated*).

3.6. *Minimum Spanning Tree (MST): Prim's and Kruskal's*

There are several MST algorithm (Prim's and Kruskal's) visualizations on the Internet. The main strength of this visualization is that the users can draw their own graph (similar user interface as with the other visualizations above) and see how the classical Prim's or Kruskal's algorithm works on that graph. We added an extra feature of finding the *Maximum* Spanning Tree by simply negating all the edge weights (Fig. 6).

3.7. *Single Source Shortest Paths (SSSP): Dijkstra's and Bellman Ford's*

There are many other SSSP algorithm (especially Dijkstra's) visualizations on the Internet, although we have not managed to find the less frequently used Bellman Ford's algorithm visualization yet. This visualization maintains the same look and feel with the other visualizations in this project. It also inherits the same strength: Students can draw

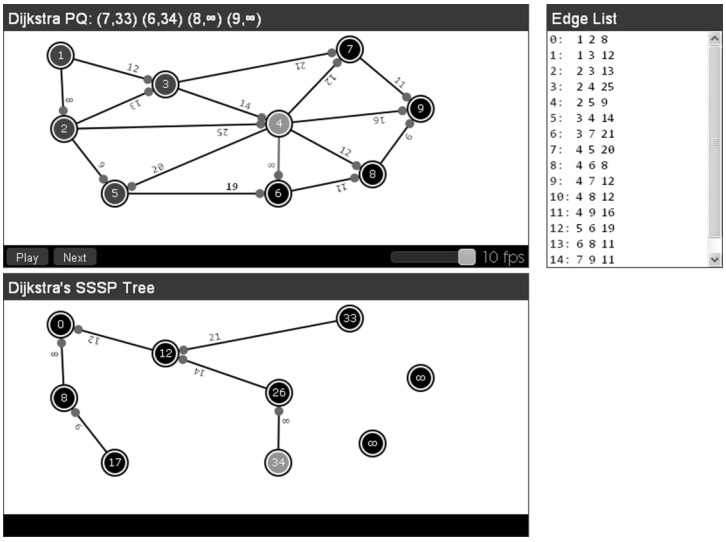


Fig. 7. SSSP visualization (showing *intermediate* Dijsktra's algorithm execution with source = vertex 1).

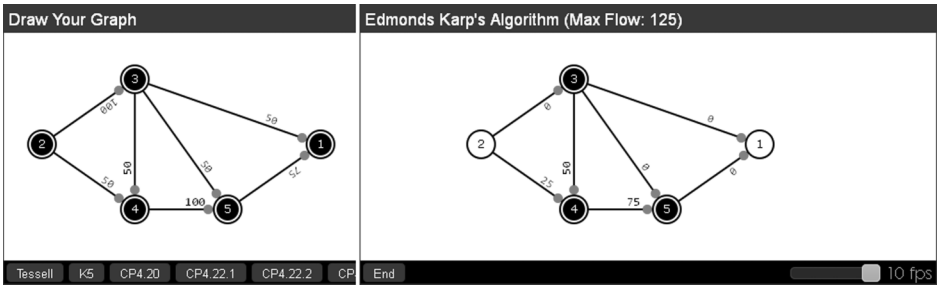


Fig. 8. Max flow visualization (showing the result of Edmond Karp's, source vertex 2, sink vertex 1).

their own graph. We also provide the SSSP spanning tree for students to easily visualize the shortest path information.

3.8. *Maximum Flow on a Network: Edmonds Karp's*

Visualization for computing maximum flow on a network is rare. Our visualization (Fig. 8) maintains the same look and feel and graph drawing capability with the other visualizations in this project. In the near future (Section 5), we will add various network flow modeling and variants to enhance the strengths of this visualization.

3.9. *Algorithm on Polygon: IsConvex, ConvexHull*

There are several algorithm visualizations for finding the Convex Hull of a set of points in the Internet. However, there are mostly standalone visualizations. Our visualization sup-

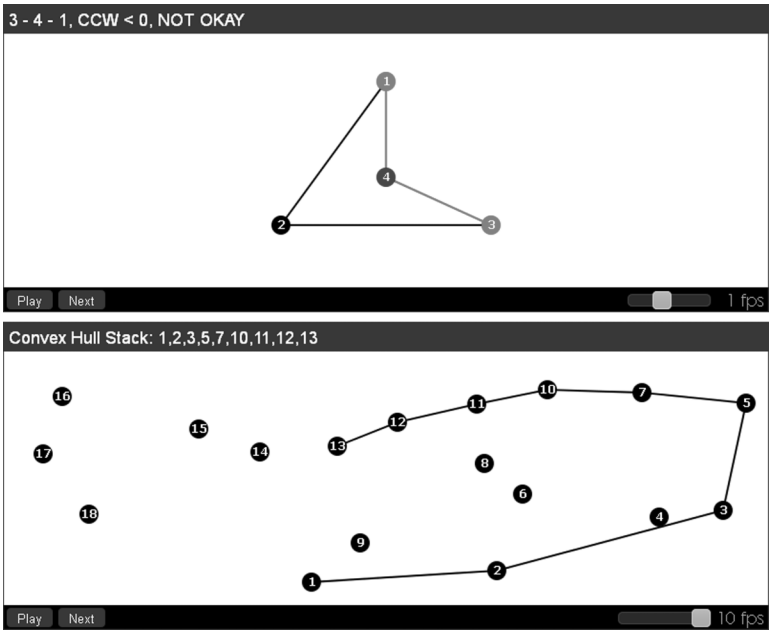


Fig. 9. Intermediate computations of IsConvex() routine (top) and Graham's scan algorithm (below).

ports three Convex Hull algorithms: Graham's Scan (Graham, 1972) as shown in Fig 9. bottom, Jarvis's March (Gift Wrapping) algorithm (Jarvis, 1973), and Andrew's Monotone Chain algorithm (Andrew, 1979). With this feature, users can compare the behavior of these three algorithms on the same set of points.

Additionally, we can do several other things with a polygon, like an algorithm to test whether it is convex or concave (Fig. 9, top), computing the area and perimeter of an arbitrary polygon, checking if a point is inside or outside the given convex or concave polygon, or cutting a polygon with a straight line. Some of these features have already been implemented whereas the rest will be added in the near future (Section 5).

4. Students' Feedbacks

The visualizations in Section 3 were used for the first time in two modules offered in semester 2 (January–April 2012) at National University of Singapore:

1. CS3233 – Competitive Programming (24 students), taught by the first author. Visualizations used: bitmask, bit, representation, dfsbfs, mst, sssp, maxflow, and polygon/convexhull.
2. CS2020 – Data Structures and Algorithms Accelerated (42 students), taught by a colleague of the first author. Visualizations used: heap, representation, dfsbfs, mst, sssp.

To measure the subjective quality of these visualizations, we present the following survey to the students from these two classes. The survey was conducted in March 2012. At that time, all visualizations listed above have been shown to respective students. All 24 (100%) of CS3233 responded as these visualizations were used extensively in the first author's class. However, only 7 (16.7%) of CS2020 students responded as the visualizations were not used extensively in this module. We separate the results of these two student groups into two columns below. However, the general responses from the two groups are rather similar.

4.1. *Multiple Choice Questions (MCQ)*

Table 2
MCQ results

Multiple Choice Questions	CS3233 (24 responses)	CS2020 (7 responses)
1. Does the visualization help you understand the algorithm taught *better*?	Yes (13/54.1%) No (0/0.0%) Neutral (11/45.8%)	Yes (5/71.4%) No (0/0.0%) Neutral (2/28.5%)
2. When do you use the visualization? (select all that applicable to you)	In class (8/33.3%) After class (4/16.6%) Before test (14/58.3%) Others (6/25.0%)	In class (1/14.2%) After class (0/0.0%) Before test (5/71.4%) Others (3/42.9%)
3. How do you access the visualization? (select all that applicable to you)	Desktops (14/58.3%) Laptops (14/58.3%) Tablets (3/12.5%) Smartphones (5/20.8%)	Desktops (4/57.1%) Laptops (6/85.7%) Tablets (0/0.0%) Smartphones (1/14.2%)

Several selected abridged reasons for question 1:

- **Yes**: Intuitive as I am a visual person; The visualizations help clarify concepts; I can play with various input values/scenarios; Useful to get initial feeling of how the algorithm works; I used to draw all these manually by myself.
- **Neutral**: Already understand the algorithm from the lecture/text book/reading the source code directly; Already know/master the algorithm beforehand; I prefer analysis (and proofs) rather than visualization to convince me that the algorithm works; Visualization bugs (in the version shown during the class) affect my understanding of the algorithm.

It is worth mentioning that none of the 31 respondents chose "No, the visualization does not help me understand more about the algorithm". The presence of these visualizations can thus be seen as a complementary pedagogical tool that will help a certain group of students immensely and will not have a negative effect on the other group of students.

From the second MCQ, we can see that some of the students indeed follow the discussions of the algorithm during the lesson. The most common usage pattern however is before tests/exams. This suggests that the primary usage of the visualizations is for self-learning and revision.

From the third MCQ, we can see that some students use their smartphones and/or tablets to access this visualization. While the number is still low at this point of time as not all respondents own these devices, we believe it will go up in the near future.

4.2. *Open Ended Essay Questions*

We also ask the students several open ended questions. We first ask them to list down the positive and negative aspects of having such algorithm visualizations. Some of the representative feedbacks are listed in Table 3.

We also ask them about their opinion on having a unified website that hosts not just the nine visualizations as listed in Section 3, but much more in the near future. Selected responses are listed below:

- **Positive**: It will be beneficial not only for this course (CS3233) but also other algorithm courses in NUS or other universities.
- **Positive**: A unified interface will be excellent. When students visualizing a certain algorithm in their mind or on paper, it is usually the same steps as shown in the visualizations.
- **Neutral**: To have them is a good idea. But if it does not come true, then it is not a big problem either.
- **Negative**: I personally prefer tracing algorithm by hand as it is easy to manipulate and help me remember the algorithm better, but I am sure there are others who prefer to have this tool.

5. Preliminary Conclusions, Future Works, and Acknowledgements

In this paper, we have presented a set of nine algorithm visualizations. They have a consistent look and feel, and are built using the same HTML5 canvas and Javascript code-base,

Table 3

Positive and negative aspects of having algorithm visualizations

Positive aspects	Negative aspects
The visualization always gives correct answer (after bug fixes).	Visualization bugs during the early rollout affect understanding.
Simple interface that is easy enough to use.	Some features are not user friendly yet (it still lacks proper documentation).
Ability to create usage scenarios on my own: Enter my own input values, draw my own graph, draw my own polygon.	It needs more than just nine algorithm visualizations as the first author teaches more algorithms in his CS3233 class.
Help me understand the algorithm better.	The presence of such visualizations may cause us not to practice on our own.
Faster than drawing by hand.	The steps shown need to be coupled with the highlight of algorithm (pseudo)codes.

and allows for interactivity with its users. More than half of the first 31 students who use these visualizations reported enhancement of their learning process whereas the other half reported neutral impact.

This visualization project is still an ongoing project. At the moment, this visualization project is hosted at: `www.comp.nus.edu.sg/~stevenha/visualization`. The URL may change in the future but the following keywords should help re-locating this visualization project: "algorithm", "visualization", "competitive programming", "SoC", "NUS", and the author names.

In the near future, we have these plans:

1. Improve the features of existing visualizations: Add *Directed* MST (Arborescence) problem solvable with Chu-Liu/Edmonds' algorithm (Chu and Liu, 1965; Edmonds, 1967), add various network flow modeling and variants (e.g., bipartite matching, min cost max flow, etc.), add several algorithm on polygons.
2. Add more classical algorithm visualizations to strengthen the library of algorithm visualizations that we currently have, e.g., algorithms on DAG (e.g., shortest/longest paths, counting paths), tree (e.g., Lowest Common Ancestor, etc.).
3. Add more non-classical algorithm visualizations that currently cannot be found elsewhere on the Internet, e.g., Segment Tree, Edmonds' matching (Blossom Shrinking) algorithm (Edmonds, 1965), Hopcroft Karp's bipartite matching algorithm (Hopcroft and Karp, 1973), Suffix Array and its Suffix Tree representation (Manber and Myers, 1991), etc.
4. Improve the user interface to make the visualizations more user-friendly (especially by adding source code highlights side by side with the visualization, adding the 'step backwards' feature) and add proper documentations.
5. Provide more pedagogical usage scenarios of the visualizations like having pop-quizzes on what will be the next step shown in the visualization, etc.
6. Make the visualization extensible to external developers by exposing our APIs.

We hope that our visualization project will be adopted by various Computer Science departments of various Universities worldwide.

The authors acknowledge Centre for Development, Teaching, and Learning at National University of Singapore (CDTL, NUS) for providing the initial funding for this visualization project.

References

Andrew, A.M. (1979). Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9, 216–219.

Chu, Y.J., Liu, T.H. (1965). On the shortest arborescence of a directed graph. *Science Sinica*, 14, 1396–1400.

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17, 449–467.

Edmonds, J. (1967). Optimum Branchings, *Journal of Research of the National Bureau of Standards*, 71B, 233–240.

Edmonds, J., Karp, R.M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2), 248–264.

Fenwick, P.M. (1994). A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24(3), 327–336.

Forišek. M. (2009). *IOI Syllabus*. `http://people.ksp.sk/~misof/ioi-syllabus/`. Last accessed 11 April 2012.

Graham, R.L. (1972). An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1, 132–133.

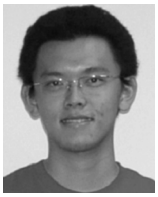
Halim, S., Halim, F. (2011). *Competitive Programming 2: This increases the lower bound of programming contests. Again*. `http://www.lulu.com`.

Hopcroft, J.E., Karp, R.M. (1973). An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4), 225–231.

Jarvis, R.A. (1973). On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2, 18–21.

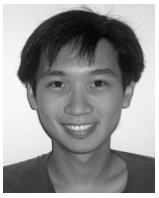
Manber, U., Myers, G. (1991). Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5), 935–948.

Tarjan, R.E. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2), 146–160.



S. Halim is a lecturer in SoC, NUS. He teaches several programming courses, ranging from basic programming methodology, intermediate data structures and algorithms, and up to the "Competitive Programming" module. He is the coach of both NUS ACM ICPC teams and Singapore IOI team. So far, he and other trainers @ NUS have successfully groomed two ACM ICPC World Finalist teams (2009–2010; 2012) as well as two gold, four silver, and six bronze IOI medalists (2009–2011).

Z.C. Koh is a second year computer science undergraduate at the National University of Singapore School of Computing. He has represented Singapore in the IOI in 2005–2007 and is now training for the ACM ICPC World Finals 2012. Zi Chun is passionate about software, design, and coffee. He is also interested in the pedagogy of computer science education. You can find out more about him at `http://zichun.i-xo.net/`.



V.B.H. Loh is an undergraduate in Faculty of Science, National University of Singapore. His interest lies in data structures and algorithms and mathematics (such as algebraic geometry and number theory). He likes to spend his free time solving challenging problems and is an avid GeoCacher.

F. Halim is a PhD student in SoC, NUS. He was IOI 2002 contestant (representing Indonesia). His ICPC teams (at that time, Bina Nusantara University) took part in ACM ICPC Manila Regional 2003–2004–2005 and obtained rank 10th, 6th, and 10th respectively. Then, in his final year, his team finally won ACM ICPC Kaohsiung Regional 2006 and thus became ACM ICPC World Finalist Tokyo 2007. Today, he actively joins TopCoder Single Round Matches and his highest rating is a yellow coder.

Olympiads in Informatics, 2012, Vol. 6, 69–85

A Self-Paced Learning Platform to Teach Programming and Algorithms

Mathias HIRON, Loïc FÉVRIER
France-IOI Organization
www.france-ioi.org
e-mail: mathias.hiron@gmail.com, loic.fevrier@france-ioi.org

Abstract. Self-paced learning platforms have received a lot of attention recently, with the promise of fundamentally changing education. For more than 10 years, the members of the France-IOI non-profit organization have been developing such a platform for teaching programming and algorithmics to self-motivated users ranging from beginners to advanced programmers. With the introduction of algorithmics in the French mathematics curriculum of high school, it is now starting to be used by teachers in classrooms. In this paper, we present and explain the choices we made when creating the contents and tools that constitute this platform.

Key words: self-paced learning, programming, algorithmics, exercises, e-learning platform.

1. Self-Paced Learning

Providing an education perfectly tailored to the needs of each student in a classroom has long been a dream for teachers as well as students. The practical constraints of giving an education to the vast majority of children in a country, while trying to give an equal chance to every child, can often make teaching and learning a very frustrating endeavor. Teaching children by batches of 25 or sometimes much more, with the criteria for forming the groups being mostly based on age and place of residence, makes it very hard for teachers to provide an effective educational experience to every child in the group.

Even if we could somehow form ideal groups where every student had the same background, the same level at the beginning of the year, the same set of skills and learning style, there would still be no way of giving a lecture in front of this group so that the pace and content of that lecture would fit every student's needs. Anyone can get distracted at some point and miss a key concept that is essential to understanding the rest of the lecture, and students who understood it well the first time will get bored and miss other parts if the teacher repeats it one more time to make sure everyone gets it.

With a self-paced approach to teaching, each student learns at her own speed, so at any time, every student in a group could be studying something different. The teacher is no longer the main provider of knowledge, the time usually spent broadcasting a lecture to the whole group is freed, and the teacher becomes available to guide students and help them individually. For a self-paced approach to work well, the external source of

educational content must be of a very good quality, very clear and engaging, so that each student stays focused and continues learning at a good pace while the teacher is helping others.

This need for a high quality source of educational content adapted to self-paced learning is one of the main reasons this approach has rarely been implemented in classrooms until recently. Books are not interactive enough, or engaging enough to be suitable for self-paced learning. New technology has long been expected to be the key, with videos or educational software promising big changes, but while the technology has already been available for many years, progress has been slow until recently. The lack of high quality resources made it hard for teachers to adopt this new style, and the lack of teachers using the self-paced model reduces the incentive for content creators to build the resources that are needed.

The trend is now changing, and changing fast. A single educational resource, initially created by one dedicated man, Salman Khan, who authored around 3000 short educational videos and put them on YouTube, seems to have been the key resource that was needed for self-paced learning to become possible and popular (Khan 2011). With millions of students now using the Khan Academy to learn at their own pace, this non-profit with now 30 full-time employees, has become the source of inspiration for several other significant initiatives in this field. One such example is Udacity, an online university co-founded by Sebastian Thrun, a former Stanford teacher, who after an experiment of successfully providing a Stanford course on Artificial Intelligence as an online interactive course to tens of thousands of students around the world, realized that he could have a much broader educational impact by creating online educational resources available to everyone, as opposed to teaching smaller groups of Stanford students in classrooms every year (Eddy, 2012).

As many other educators, the members of the France-IOI non-profit organization have been convinced by the potential of self-paced learning for many years, and worked hard on creating suitable educational content and teach programming and algorithmics to thousands of students. We learned a lot along the way about what makes an online resource effective for self-paced learning, and would like to share what we have achieved so far, and explain the reasons behind our choices.

2. Fast-Paced, Mastery Based Learning

2.1. *Mastery Based Learning*

One of the main issues with the traditional lecture-based teaching approach is that students move on from learning one concept to learning the next not when they are ready to do so and have mastered the previous concept, but when the teacher decides that it's time to move on. This time might be right for a small subset of her students, but is either too early or too late for the rest of the class (Gomes and Mendes, 2007). When the pace of learning is imposed, students quickly accumulate gaps in their knowledge of the domain that is being taught, and keep piling up new concepts on unstable foundations. Fast

learners are also victims of this approach, not only because they could probably move on faster and feel like they are wasting time, but also because no matter how good they are, there are always times when they get distracted or bored and miss some key part of the lesson.

The key idea of an effective self-paced learning approach is that a student doesn't move on to the next concept when the teacher tells him to, or even when the student feels like it, but simply when he has mastered a concept well enough to be ready for the next one. Therefore, an essential part of a good self-paced learning platform is a system that can determine whether a student is proficient enough with a given concept, so that he's ready to move on and use that concept as a stable foundation to keep building up his mastery of the domain.

Automatically assessing the proficiency of a student in a given concept is not an easy task, and the most common way to do that is to use multiple choice questions, which is far from being ideal. In the field of programming and algorithmics however, we have a much more effective way to automatically assess a student's proficiency: ask the student to write a program that involves the concept in question, and automatically grade that program against a series of tests, in the way it's being done in many programming competitions (Forišek, 2006).

The evaluation of programs is at the core of our platform. For each new concept, students are asked to write programs that read some input and provide their output either by using standard I/Os or through a library. The program is then evaluated on our server, where it's compiled and run in a sandboxed environment, against a series of test cases, within certain limits of time and memory. We use one that is part of the Moe grading system (Mares, 2009), which is the sandbox that has been used at recent IOI (International Olympiad in Informatics) competitions. Using this evaluation system, we built a comprehensive set of learning material, where each student learns a concept, practices it on exercises while proving his proficiency, then moves on to the next concept.

2.2. *Short Courses Motivated by Exercises*

When giving a lecture, there is a strong incentive for the teacher to present several new concepts in a row, or to go through a variety of examples of applications of these concepts during the same broadcasting session, before letting the students practice them on some exercises. Lectures often last an hour or more, without any activity on the part of the student, other than taking notes. Most students are not able to stay focused and fully engaged on the lecture for one or more hours at a time and often lose track at some point, not being able to connect again because they missed some of the earlier concepts.

The reason for this tendency is simply that it isn't very convenient to present one concept to the whole class in a couple of minutes, let everyone practice and wait for the majority to finish a couple of exercises to make sure they master that concept, before moving on to a new concept. Too much time is wasted switching back and forth from a broadcast mode to practice sessions, and it is much more comfortable to spend half of the time lecturing, and the rest practicing, or even just let students practice at home or during dedicated practice sessions.

With a self-paced learning platform and the knowledge being provided not by the teacher but by an external source, this incentive doesn't exist, and the cost of switching from learning a new concept to practicing mostly disappears. We see no more reason to group the presentation of several concepts together, before switching to practicing on this batch of concepts.

Each presentation should be short enough, that there is no time for the student to get distracted and miss part of what is being taught. Furthermore, if the student knows that she will need to apply the concept right away, it gives her a much stronger incentive to pay attention right now, and not wait until she really needs it for an exercise or the exam.

We go even further, by providing the reason for learning the concept before even presenting the concept itself: instead of providing a short course followed by an application exercise, we start by describing the exercise and follow on the same page with a short course that introduces the new concept needed to solve that exercise. The student reads the short course not because it is pushed to her, but because she is looking for a way to solve the exercise. Instead of being a passive receiver of a lecture, the student becomes active and seeks the information.

Typically, a content item on our platform is a combination of an exercise, and a very short course introducing the new concept needed to solve that exercise. This is the format we use for programming concepts that a student can't possibly discover by herself, typically a new element of the syntax of a programming language. When we teach more advanced concepts such as classical algorithms, we switch to an approach detailed in (Charguéraud and Hiron, 2008), where we give the student a chance to discover the new concept by himself, and provide the course in the form of optional hints, and detailed solutions.

2.3. *Many Small Exercises*

Our experience with traditional lectures in computer science is that fundamental programming concepts such as the notions of instructions, variables and affectations, conditions and loops, as well as the corresponding syntax, are taught in a very short amount of time. As an extreme example of this tendency, one of the authors witnessed a 4 hours lecture where all of these concepts and more were taught at once, before the students got any chance to write their first program. It is common to see students study advanced concepts such as object oriented programming and inheritance, while they are still uncomfortable with writing programs involving basic loops.

The issue is that there is not a lot that can be said about a concept such as the principle of the conditional instruction, without being tempted to immediately add more advanced concepts like boolean operators. The concept itself is very easily defined and explained, but mastering it is a very different story. Learning to program means learning a new way to think, and learning how to build all kinds of things using a very small number of building blocks. Even though a student can seem to understand and even be able to explain such a fundamental concept as the conditional instruction after a few minutes of introduction, it takes a lot of practice before he can really be considered to be proficient with that concept.

As a consequence, our introductory content is composed of many small exercises involving all types of uses of every programming concept, as well as many combinations of these concepts. At the time of writing, our platform offers no less than 18 exercises to practice basic if-else conditional instructions, and their combination with other basic concepts taught earlier. Continuing with the concept of boolean operators used within a conditional instruction means solving 10 more exercises.

Students learning on our platform write close to 80 small programs before starting to learn about concepts such as arrays and functions. By that time, they are getting very familiar with the basic building blocks of programs. It takes no less to build strong enough foundations and be ready to learn to use more advanced concepts on top of them.

2.4. *Detailed Solutions as an Essential Part of the Course*

The fact that a student was able to solve an exercise does not necessarily mean he understands every aspect of his solution. Students may have written parts of that solution without being entirely sure of what they were doing, or may have been helped by a fellow student or a teacher. They often have doubts and questions about what they just did, even when their solution is correct.

We try to make sure these doubts and questions are resolved by giving a detailed solution for each exercise, that includes a reference solution and explanations on how and why it works. The solution often resembles what the student has written, which gives her confidence and motivates her to continue. The explanations reinforce what the student has learned, and alleviate any doubts she may have about what she just did.

By providing reference solutions that are as clear and elegant as possible, we teach good habits by example. We observed that after reading our reference solutions for each exercise they solve and the justification for our choices, students often pick up our style and don't form too many bad habits.

To make sure the students really understand what is going on when their programs run and don't form an incorrect or blurry model of the execution of their solution, we try to show that execution from different points of view. We use detailed illustrations to present a visual model of the execution flow or of the state of the variables at a given time of the execution.

To go even further, we created a "simulator" (see Fig. 1), a tool that is inspired by the debuggers that can be found in many IDEs, and that allow users to run programs step by step. The IDEs that include these debuggers tend to have an overwhelming number of features, and asking students to use them can be counter-productive. Our simulator was created with only one goal: helping students to build a good mental model of what is going on during the execution of a program.

Students can run the reference solution step by step on any input data they provide. At each step, the current line of code that is executed is highlighted, as well as any changes to variables. These are shown within a representation of the heap and the stack, so that students start forming a good model of how memory is used, even before anything is explained to them about it.

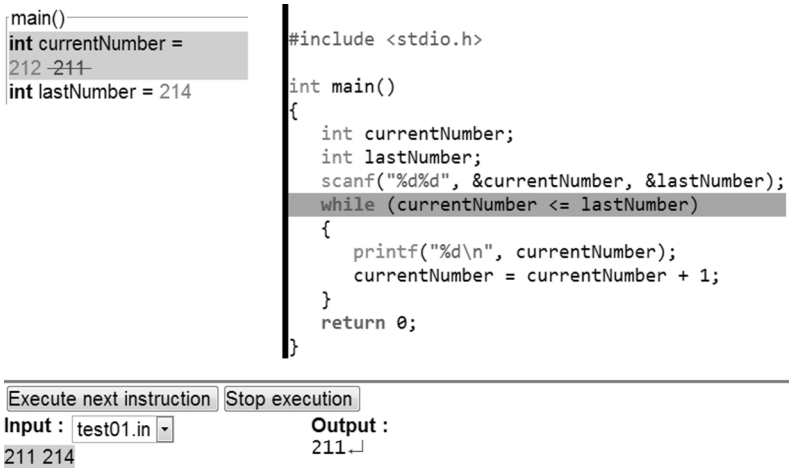


Fig. 1. Step by step execution simulator.

This simulator is far from being an actual debugger and can only be used on our reference solutions, but does a good job at what it is designed for. It is integrated directly on the solution page of the exercises, which makes it very convenient to use. Little by little, this tool helps students to learn to do mental simulations of the execution of their own programs.

Another ingredient of our solutions is the presence of reference solutions in other programming languages. Students are not expected to understand or study them in detail, but curious students who want to get a feeling of what each language looks like or students who have some previous experience with some of these languages can have a quick look by clicking on the corresponding tabs. This can help them to understand that the fundamental concepts of programming are independent from the syntax of the language they use.

As the exercises move from being purely about understanding the fundamental aspects of programming towards problems of algorithmic nature, the focus of our solutions becomes the thought process that can be applied from reading the problem statement to finding a working algorithm. For more information about this aspect of our approach, see (Charguéraud and Hiron, 2008).

3. Avoiding Frustrations

People have a natural tendency to enjoy learning, as it brings all kinds of good feelings: wonder, pride, self-confidence, self-respect and more. Learning is not always easy and the challenges are part of what makes people love it, but challenge often comes with some failures. Disappointment also happens when the material being taught is already known, and not much new is learned. Failing too often and for too long brings all sorts of bad feelings: frustration, disgust, low-confidence, shame, boredom, etc. Students learn well

when the good feelings associated with learning far overcome the occasional bad feelings. To be successful, a good teaching resource has to minimize these negative feelings while making sure students learn well and fast.

On top of creating good quality content and exercises, one of our priorities is to avoid the frustrations a student feels when getting stuck while trying to learn, either because he can't solve a given exercise, or because he doesn't know what to do next. Most of our users are using our platform from home, so when they get stuck and the platform doesn't help them, they are likely to just give up and never come back. Even if they are in a classroom, the teacher might not be available, or the student might just not feel like asking for help. We present several techniques that we employ to minimize the amount of frustration students will encounter while studying.

3.1. *Anticipating Errors*

Syntax errors are one of the main causes of frustration at the early stages of learning programming. Missing semicolons, parentheses, or double quotes are not obvious at all to beginners, and the error messages they generate often make things even harder to understand. While projects such as Scratch (Scratch, 2012) aim to avoid this issue completely by offering visual languages where no syntax error is possible, we believe that this approach only postpones learning to be careful and rigorous, which is an essential part of what programming is about. Execution errors such as infinite loops or divisions by zero can also be frustrating, and are not avoided by these tools.

We also noticed that a big part of the actual learning happens precisely when errors happen: as the student faces an error, all the details that they have doubts about are put into question, and he then actively tries to figure out the answer to his own questions. As a consequence, we don't eliminate errors completely, but reduce the risk that students get stuck for too long because of them.

The key idea of our approach is to anticipate errors, and teach students to recognize their own errors as well as the corresponding messages, before they even encounter them while solving an exercise. We do this in two different ways:

1. By providing courses that present different common errors associated with a recently learned programming concept, and show the corresponding error messages. Giving different examples and highlighting where the error is brings their attention to the types of errors that are possible. By showing and explaining the error messages associated with them, we reduce the amount of surprise and frustration the student might get in the future, if he encounters these same error messages. We don't necessarily expect students to remember each message, but merely recognizing something familiar and getting a rough idea of what types of errors can produce certain types of messages makes things much easier.
2. We give exercises whose only goal and difficulty is to find and fix syntax errors in one or more example programs. By solving these exercises, students learn to look for and fix the most common syntax errors. They are then more likely to notice their own errors in the future.

3.2. *Providing Help, through the Community or Automatically*

One way to make sure students never get stuck on a given exercise and always keep learning could be to give them access to the solution when they can't solve the exercise by themselves. Our concern with such an approach is that if it's too easy to get access to the solution, many students will be too tempted to read it before trying hard enough to solve the exercise by themselves. Some might even believe that they can save time and learn faster, if instead of really solving each exercise entirely, they only think about how they would do it, then look at the solution and see if they had the right idea. This might give them the illusion of learning for a while, but this learning will only be superficial.

Instead of giving them the solution, our goal is to give them the minimum amount of help that they need to continue by themselves. When possible, we avoid giving them even part of the solution and try to tell them something that will help them find the solution by themselves.

In the past, students who needed help contacted us by e-mail and sent us their source codes or ideas. The advice we gave them was often the same for a given exercise, so we created a system that provides these hints automatically, at the request of the students. Depending on the reason why students get stuck on a given exercise, these hints can help out very well, or be of no help at all, but this saved us and the students a lot of time, as they only needed to contact us in the latter case.

More recently, as the number of our users increased, and our free time diminished, we decided to set up a system so that other users could help each other. The principle of this system is quite basic: when stuck on an exercise, a student can ask for help on a forum where any student who has already solved that exercise can see the question and provide an answer. We try to make sure people who help others have access to all of the necessary information, by showing them the history of submissions of the student on that specific exercise (see Fig. 2). We make it very clear that helping a student doesn't mean giving him the solution directly, but only hints that make sure they don't stay stuck too long.

3.3. *A Clear and Well Designed Learning Path*

Making sure each concept is well explained and providing help to students when they get stuck is important, but there is another aspect that is essential to self-paced learning and that is too often neglected: making sure students always know what to do next, and making sure that what is offered to them corresponds to their level.

Online educational resources too often take the form of a multitude of courses and exercises accumulated in a big online database. With all these choices offered to them, students are expected to be able to find what they need. The paradox of choice (Schwartz 2004) showed that too much choice can generate anxiety for shoppers in a store, and the same is true for a student faced with a multitude of learning materials. Although we believe a student should have some choice for what to learn next, as she might be more in the mood to study one aspect of the domain rather than another, this choice should be limited to a very small number of clearly identified options, so that the student never feels lost and overwhelmed with the multitude of things to learn.

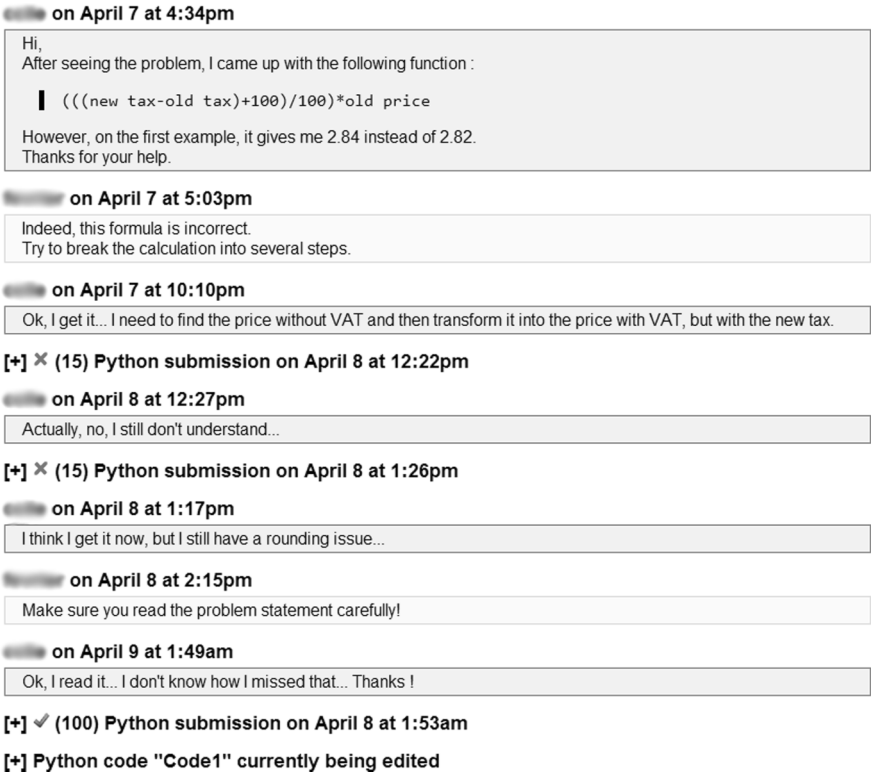


Fig. 2. Students helping each other on our forum.

The order in which different concepts are presented is an essential part of the quality of a curriculum. Not much is more frustrating to a learner, than studying material or trying to solve an exercise that assumes that some other concepts are mastered, if the student doesn't actually master them. Making sure the student is proficient in the prerequisites before letting her dive into some material is therefore key to avoiding this type of frustration.

On top of enforcing these constraints, we spend a lot of time thinking about the best order in which to present new concepts. Too many books or other introductory resources on programming focus on presenting the syntax of a language in a quasi-systematic way. As an example, after introducing integer variables to the student, they often continue with the other types (char, boolean, float, double, . . .). Learning these too early gets in the way of mastering the fundamental concepts associated with using variables. Our approach is to focus first on a small subset of the language features, that is needed to learn the fundamentals, and to keep additional elements of syntax for much later. We don't teach the syntax of a language, the language is only there as a means to teach programming, so we don't worry about making sure the student knows every aspect of that language.

This focus on teaching concepts in the right order is very apparent on our website. The content is split into a sequence of levels, and each level contains a sequence of num-

bered chapters, each focusing on a given theme. The courses and exercises in a chapter are also numbered, one of the many indications that students are expected to work on them in order. Students have the possibility to skip some exercises or chapters, as well as a whole level, but to skip a whole level and access to the next one directly, a student needs to solve a small set of challenging exercises to prove he is reasonably proficient with the concepts presented in the previous level. The first level starts at the very basics of programming, while at the time of writing, the last levels present relatively advanced algorithmic concepts, such as network flow algorithms.

4. Keeping Students Engaged

Learning can be fun, and has many other similarities with games. It can be easy at times and challenging at others. There is a sense of progress while playing a game and while learning, and both sometimes come with failures. In some cases, both can even become addictive.

Making learning more fun doesn't mean replacing some of the learning with games, or putting it inside a game. A better approach is to think about learning as already being some sort of game, and to try to make it a better game. For this, we can take inspiration from what works in successful games, and adapt it without changing the nature of learning. We have implemented a few ideas in this direction.

4.1. *Presenting Each Exercise in the Context of a Small Story*

Solving programming exercises can be fun even if the question is expressed in the most pure form. Writing a program that displays numbers from 10 to 0 can be interesting and motivating by itself, just for the motivation of successfully solving a task, and getting better at writing loops. However, it can be even more fun if the student does that while imagining that she is about to launch a rocket into space and needs to write the program that handles the countdown. When adding fun stories to exercises, the goal is not to hide the actual task and pretend that we're not learning, or even to justify that writing a program that counts from 10 to 0 can be useful, but rather to add to the motivation and diversify the stimulation associated with solving a problem.

The main drawback of putting exercises in the context of a story is that it means more text for students to read, which can be a serious obstacle for many students. When we write a fun story for an exercise, we try to keep the text as short as possible, and we make sure that students who don't like to read these stories or don't have the time to do so won't have to. We do this by clearly separating the story part from the rest of the exercise, and making sure nothing essential will be missed by students who skip the story. This is similar to games where some players like to skip videos and jump right to the action, while others really enjoy watching them.

To write interesting stories, we get some inspiration from various sources, going from Greek mythology to modern literature, scientific discoveries, or even noteworthy events of our history. We then use this opportunity to introduce students to the source of our

stories, and provide small excerpts describing the original context. For students who are interested in finding out more about the myth of Sisyphus, or "The man who planted trees" from Jean Giono, after solving an exercise that relates to these, we provide pointers to online resources about these subjects.

4.2. *Making Each Exercise Part of a Big Adventure*

Increasing the motivation and fun of solving each exercise is a first step, but to make sure students keep solving exercises and don't give up, we need to make them curious about what's next. A big part of it is simply to make the content interesting enough that students just want to learn more. Another dimension can be added in the form of a story that spans multiple exercises and chapters. The same characters can be present in different exercises, making each exercise part of a big adventure. Many aspects of a good adventure book or game can be applied, and we try to write the stories of our exercises in that spirit.

All of the exercises we created to introduce programming to beginners take place on an imaginary planet, with its own inhabitants, culture, plants and animals, and each story is part of a longer adventure. What we have done is far from being at the level of best-sellers, but students already enjoy it, and we can improve this aspect little by little.

4.3. *Recognizing Short Term and Long Term Accomplishments*

One key aspect that can make games and learning addictive, is the sense of accomplishment that one feels as he makes progress. This aspect of what motivates players has been clearly recognized in games, and sometimes pushed to a ridiculous level, where you get awarded all sorts of achievements for performing the most trivial tasks. Without going to such extremes, recognizing short term and long term accomplishments during the learning process can be a very good way to keep students motivated.

A very simple but surprisingly efficient way to achieve this is to clearly display a green check mark in front of every exercise, chapter or level that a student has fully solved (see Fig. 3). Students are proud to accumulate all these proofs of their accomplishments, and really enjoy having a full column of green marks rather than one with some gaps, or worse, red marks showing exercises that they have failed to solve yet.

A longer term recognition of the achievements of our users is the "level" that is associated with their user name. Users start at level 1, then become a "level 2" user once

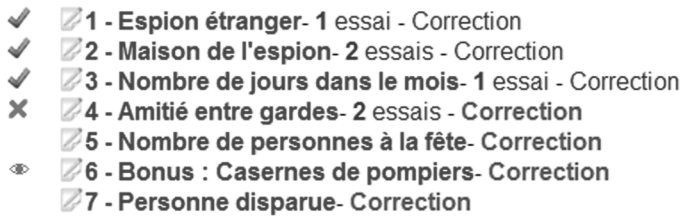


Fig. 3. Green check marks to recognize achievements.

they solved the great majority of exercises proposed in the first level of our content. We also work on measuring various parameters that we plan to associate with the "avatar" of each user: the experience (number of exercises solved), precision (based on the average number of attempts needed to solve exercises), persistence, and others. This is similar to characteristics that are associated with characters in role playing games, but in this case, these scores correspond to actual skills displayed by the student.

4.4. *Challenges*

While we teach the basics of programming to students who use our platform, we keep in mind that one of our goals is to teach algorithmics. The first few exercises that we use to teach each fundamental programming concept don't require to think much about what the algorithm should be, and are mostly focused on the implementation aspect. However, we often try to use exercises where some algorithmic thinking is needed, and more generally, problem solving aspects.

As an example, soon after introducing the concept of a sequence of instructions, we give students the following challenge: you are given a 5 liter and a 3 liter water containers. Using the instructions "fill(container)", "empty(container)" that fill or empty the given container completely, and "pourInto(source, destination)" that pours the content of the source container into the destination container until either the source is empty, or the destination is full. Using a sequence of these instructions, write a program such that at the end, the 5 liter container contains 4 liters. The solution is a sequence of 6 instructions, but takes some time to discover.

We noticed that even weak students tend to be interested in this type of exercises, and try really hard to find the solution by themselves. By solving this type of challenges once in awhile, we hope they will learn that if they try hard enough and long enough, they are able to solve problems that they had no idea how to solve initially, and without ever beeing told how to solve that type of question before. By giving them this kind of challenges very early, we try to show them how fun algorithmics and problem solving can be. We make the hardest of these challenges clearly optional, so that students don't feel too frustrated when they get stuck on one of them.

5. Content and Tools for High School Teachers, and Others

The very first version of our platform was created in 2001, as a tool to help us select and train students for the IOI. Computer Science was not taught as a subject in French schools at the time, and merely selecting the most talented self-taught French programmers was not sufficient for us to get good results. We decided to focus most of our efforts on developing this platform, and on training in general. This training platform had a very significant impact on our results, and our teams would from then on be awarded medals every year at the IOI.

In 2009, the French ministry of education decided to introduce algorithmic thinking as a part of the mathematics curriculum in high schools. By school year 2011–2012, it

was clear that most math teachers who were put in charge of teaching this subject had not found a good way to do so, having received no specific training to prepare them for this change. In August 2011, one of these teachers contacted us, to bring to our attention the fact that our platform had the potential to become a great tool for teachers to use in their classrooms. Until then, our target had mostly been self-motivated and passionate students, and even though we already had some programming courses used by thousands of users, these were not ready to be used by average high school students with no prior interest in the subject.

With the help of this teacher, Guillaume Le Blanc, who soon became an active member of our organization, we decided to dedicate a significant amount of time to creating content and tools specifically with high school students and teachers as our primary target. Although it was designed with a specific public in mind, we are convinced that these tools and content are suitable for anyone willing to learn programming, and any teacher who needs to teach programming to her students, at all kinds of levels.

5.1. *Ready to Learn Within a Few Minutes*

The first constraint a teacher is faced with before his students can start practicing programming on the classrooms computer, is to have the proper environment set up, with all the tools needed to edit, save, compile, and run programs. To make sure that installing such tools and teaching students how to use them doesn't take most of the first hour that the teacher can allocate to teaching this subject, we made sure to avoid this step entirely. Here is how the first programming class starts when using our platform:

- directly log into the computers; one student per computer is highly recommended,
- open a modern browser and go to france-ioi.org,
- create a new account, or log in using a facebook or google identity,
- click on the high school training section, and start working on the first exercise.

About 5 to 10 minutes after students enter the classroom, they have usually written and run their first program (the classical print "Hello World"), and are working on the second.

Programs are written in a very simple text editor directly integrated within the web platform. Programs written by students are automatically saved, compiled and run on our servers, and the results are shown in the browser, right below the editor. If a student has to leave while in the middle of an exercise, all he needs to do is to log out. He will be able to continue right where he was the next time he logs in. All of the time spent in the classroom is spent learning and practicing, and getting individual help from the teacher.

5.2. *Available Languages*

The official curriculum lets high school teachers free to use the language of their choice in their classes. We therefore offer our courses and exercises in some of the most popular programming languages among teachers. We started with Python because we and many teachers consider this language as being a good choice for beginners. We later adapted our

content to Java as well as Java's Cool, a simplified version of Java developed specifically for high school teachers by the INRIA (a French public research body fully dedicated to computational sciences).

Our exercises can also be solved using other languages available on our platform, such as C, C++, OCaml and Pascal, and we plan to adapt the associated courses for at least the first three of these languages. As our content is focusing on the fundamental concepts of programming, rather than on the details of the syntax of each language, the structure of the different versions is very similar.

5.3. *Adapted to the Official High School Curriculum*

The new official math curriculum (B.O., 2009) defines the programming concepts that every French student is expected to learn by the end of high school. One of the expressed goals is for students to be able to formalize some of the algorithms that they have already encountered, such as the Euclid algorithm (the computation of a GCD), and implement them using a programming language. Other algorithms such as statistical algorithms (mean, median, ...) or the binary search algorithm should also be taught.

To achieve these goals, the most fundamental programming concepts are needed, and students are expected to learn the concepts of sequential instructions, variables, basic calculations, input and output manipulations, as well as conditional statements and loops. Basic knowledge of arrays is not explicitly listed, but is implicitly required, as arrays are needed to implement most algorithms involving statistics.

The programming concepts listed in this official curriculum are no different than the first concepts any person interested in learning programming has to learn. As our users come from a wide range of backgrounds, we decided to make as few assumptions as possible on their proficiency with mathematical concepts. As a consequence, we have clearly separated our introductory exercises into two categories:

1. Exercises and courses dedicated but limited to introducing the programming concepts. They form a clearly identified linear progression that constitutes the first chapters of the 5 levels of our core content The chapters that correspond to the official curriculum contain around 100 small exercises, and an average student is expected to be able to solve most of these by spending about 20 hours of work.
2. Exercises and courses where these concepts are combined with mathematical concepts. These exercises are grouped by mathematical subject, and the algorithmic prerequisites are always given. For each subject, the underlying mathematical concepts are introduced and progressively mastered, from an algorithmic point of view. This category now has 40 exercises, and we are planning to greatly increase this number with the help of teachers, and hope to cover most of the mathematics curriculum in high school.

5.4. *Group Management Tools*

One of the great advantages of a self-paced learning platform is that, as students make their way through the set of exercises offered by the platform, data is collected about their

attempts, successes and errors. This data can be very useful to a teacher, who can get a good idea of which concepts each student has mastered, and which concepts they might need help with.

Our platform gives teachers the possibility to create groups that their students can then join. For each of the group he manages, the teacher has access to a summary of what exercises each student of the group has worked on or solved, at what time, and after how many attempts, and get access to the details of each submission. The current version of these tools is relatively basic and we have plans to improve it based on teachers' feedback.

5.5. *Early Feedback*

Our high-school ready content is available since the end of 2011, and several teachers have started using it as the primary way to teach programming and algorithmics to their students. The results are already very promising, and both the teachers and the students appear to appreciate this approach very much. A number of teachers are practicing on our platform, to get ready to use it for the next school years. The following is a testimony from one of the first teachers who have adopted our platform in their classroom:

I have started using the France-IOI platform to improve my own algorithmic knowledge and to complete the small formation I had received. I then used it both in the computer science club I animate, and with my regular classes, with small groups of 15 students at a rate of approximately one session every 2 to 3 weeks. I generally ask them to do at least one exercise each week at home, in order to maintain their knowledge.

Students love this way of learning, and this became their favorite activity. The overwhelming majority of my students are strongly engaged and study hard. They frequently ask me to do more of these sessions, but due to time constraints, I cannot satisfy their request.

There is no other activity for which I can obtain the same level of engagement from them, although I am not able to pinpoint the reasons for this. One thing that is certain, is that this is not purely due to working with computers. I organize other math-related activities on computers, but students appreciate them less than working with this platform, and sometimes would rather do them without computers.

6. Conclusion

In this paper, we present the features of the self-paced learning platform created by the members of the France-IOI organization and available on france-ioi.org, where anyone can learn programming and algorithmics from the beginning to an advanced level. We describe many of the choices we made when designing the current version of the tools and contents that are made available, and explain the motivations behind these choices.

We show how students learn the fundamental concepts of programming by solving many small exercises accompanied with very short courses, in a web-environment where they can directly edit, compile and run their programs. Following the principles of mastery-based learning, the solutions written by students are validated automatically

before they move to the next problem, and this system works with a variety of popular programming languages. The detailed solutions that are provided help them to reinforce their understanding of the concepts, and make use of a step by step simulator that helps students to form a mental model of how the computer executes a program.

We then explain how we try to avoid the frustration that often comes when faced with errors by anticipating them and explaining the most common ones, and how they can be recognized, and we describe how we prevent students from getting stuck by providing them automated hints, as well as a community based help system. We talk about our work on offering a very clear learning path to students, which we consider as being as important as the individual resources themselves.

We describe how we present each exercise in the context of a short story, with each of these stories being part of a big adventure that makes learning even more fun and engaging. We take inspiration from successful games while presenting this adventure, and motivate students by recognizing and clearly displaying their accomplishments. Easy exercises help students to quickly gain confidence, while harder exercises clearly identified as challenges make sure they practice their problem solving skills early.

We made sure that the first chapters of our content and the tools that come with them were perfectly adapted for the use by teachers in high school classrooms, in response to the introduction of notions of programming and algorithmics in the official high school math curriculum. We made sure students can start learning on this platform within minutes of entering the classroom, and can stop at any time without losing their work, and continue from home. With our group management tools, teachers who are freed from giving a lecture can follow what each student is doing and provide personalized help accordingly.

All this is work is done by a team of volunteers, including former IOI contestants and teachers. We do this because we are passionate about algorithmics and teaching it, and we believe all students have a tremendous potential that can they can reach if they have access to the right content and tools. As we don't want to restrict the access to any particular group, we are looking for volunteers to help us extend the reach of this platform, by translating existing content into as many languages as possible. The platform is ready for internationalization, and translation efforts have already started in various languages, such as English, Spanish, Lithuanian and German.

References

B.O. (Bulletin Officiel) (2009). Number 30, July 23. `http://media.education.gouv.fr/file/30/52/3/programme_mathematiques_seconde_65523.pdf`. Accessed March 2012.

Charguéraud, A., Hiron, M. (2008). Teaching algorithmics for informatics olympiads: The French method. *Olympiads in Informatics,* 2, 48–63

Eddy, M. (2012). Sebastian Thrun, Mastermind of Stanford's Free AI Course, Forms Free Education Website. `http://www.geekosystem.com/sebastian-thrun-udacity/`. Accessed March 2012.

Forišek, M. (2006). Security of programming contest systems. In: *Informatics in Secondary Schools, Evolution and Perspectives*. Vilnius, Lithuania.

Gomes, A., Mendes, A.J. (2007). Learning to program – difficulties and solutions. In: *Proceedings of the International Conference on Engineering Education*.

Khan, S. (2012). Let's use video to reinvent education TED talk.
`http://www.ted.com/talks/`. Accessed March 2012.
Mares, M. (2009). Moe – design of a modular grading system. In: *Olympiads in Informatics*, 3, 60–66.
Scratch (2012). `http://scratch.mit.edu/`. Accessed March 2012.
Schwartz, B. (2004). *The Paradox of Choice: Why More is Less*. ECCO, New York.

M. Hiron is the French IOI team leader, and is the cofounder and president of France-IOI, the organisation in charge of selecting and training the French team for the IOI. He creates tasks on a regular basis for national contests and online training programs, as well as for the French–Australian Regional Informatics Olympiad, and occasionally for other international contests such as the IOI or the APIO. He is a business owner working on projects ranging from web development to image processing and artificial intelligence.

L. Février is a PhD student in the Computer Science Department at the University Pierre and Marie Curie of Paris. Deputy leader of the French team at IOI 2011. His research interests focus on serious games applied to the learning of programming and algorithmics.

Olympiads in Informatics, 2012, Vol. 6, 86–99

Introducing CMS: A Contest Management System

Stefano MAGGIOLO[1], Giovanni MASCELLANI[2]
[1]*Mathematics Area,* SISSA/ISAS
Via Bonomea 265, 34136 Trieste, Italy
[2]*Faculty of Sciences, Scuola Normale Superiore*
Piazza dei Cavalieri 7, 56126 Pisa, Italy
e-mail: maggiolo@sissa.it, giovanni.mascellani@sns.it

Abstract. We present *Contest Management System* (CMS), the free and open source grading system that will be used in IOI 2012. CMS has been designed and developed from scratch, with the aim of providing a grading system that naturally adapts to the needs of an IOI-like competition, including the team selection processes. Particular care has been taken to make CMS secure, robust, developed for the community, extensible, easily adaptable and usable.

Key words: CMS, contest management system, grading system, IOI, IOI-like competitions.

1. Introduction

Programming contests have a long-standing tradition in computer science education, where the basic principle is to get young students interested by letting them compete on their skills alone. The main technical issues of organizing a programming contest can be roughly divided into three parts:

1. creating tasks and developing all the data required (statements, solutions, testcases, information on how to grade submissions, etc.);
2. when the contest is on-site, configuring the machines that the grading system and the contestants are going to use, in particular with respect to network security;
3. managing the actual contest (accepting and grading submissions, giving feedback, displaying a live ranking, etc.).

The aim of this paper is to describe a tool for dealing with the third point when contests follow the spirit and share the main features of the IOI and similar contests, such as CEOI, ICPC, national selections for the IOI, etc. To this end, we introduce *Contest Management System* (CMS), the free and open source grading system that will be used in IOI 2012.[1]

We designed CMS keeping some important keywords in mind.

- Secure: even though the main security measure for a contest is to isolate the contestants and the grading networks, obviously there must be at least one point of contact between the two; this must give the fewest possible ways of attacking the system.

[1] CMS is available at `https://github.com/cms-dev/cms`.

- Robust: an error or a critical condition in one part of the system must not take down the whole system; hot swapping of services and machines is not an exceptional condition but a standard procedure; the coherence of the state of the contest must always be ensured.
- Developed for the community: CMS must be easily accessible, free and open source (it is licensed under the GNU Affero General Public License (Free Software Foundation, Inc., 2007)); feature requests, bug reports and patches must be considered and applied whenever possible and without long delays; it must support localization of the contestants' interface.
- Extensible: new or rare task types and scoring methods must be easy to implement in the form of plug-ins; currently based on the current competition rules for the IOI, CMS should leave the possibility of reflecting the modifications of such rules by future hosts.
- Adaptable: CMS must not interfere with the first two points listed previously for organizing a contest, namely it must not mandate a minimum number of grading machines, or special network configurations (apart from the ones that prevent security issues), and must not require a particular method of preparing the tasks for the contest.
- Usable: CMS must be well-documented for contests administrators, developers, and contestants, and it must not require insights into internals of the system for running a contest.

The design and development of CMS started in the second part of 2010. Even though the authors had several years of experience using and contributing to the grading system developed for the Italian selection process, the design of CMS started afresh. Alpha versions started circulating in the first part of 2011 and were used in the selection process of the Italian team for IOI 2011. After that, more refined and stable versions have been used for official contests: OII 2011 (Italian Olympiad in Informatics) in October 2011, AIIO 2012 (Australian Invitational Informatics Olympiad) and FARIO 2012 (French-Australian Regional Informatics Olympiad), respectively in February and March 2012.

The paper is organized as follows: in Section 2 we motivate our decision to develop a new grading system; in Section 3 we describe the structure of CMS and explain the main design choices; in Section 4 we describe the process of setting up a contest using CMS; in Section 5 we show how data is manipulated inside the system to eventually produce a ranking; in Section 6 we briefly sum up our results and hopes for the future. Of course, this paper is just an introduction to the usage of CMS. Additional documentation, together with the source code, can be found on the CMS website (Boscariol *et al.*, 2010).

2. A Brief History of IOI Grading Systems

The earliest editions of the IOI did not use any grading system at all (Heyderhoff *et al.*, 1992): contestants were required to handle diskettes with their solutions, that were manu-

ally judged by the team leaders compiling, running, and scoring the solutions, according to some precise instructions by the Scientific Committee.

The progress of technology, together with higher reliability and lower costs of networks and equipment in general, made possible developing grading systems that are:

- automatic: where solutions are graded without human intervention;
- distributed: where solutions are submitted via a remote interface;
- live: where solutions are graded as they are submitted, providing a public ranking and/or feedback to the contestants.

The first semi-automatic system was used in IOI 1994 (Verhoeff, 1994), even if it still used diskettes to submit solutions. A fully automatic and distributed system was used in IOI 1999 (Piele, 1999), and IOI 2010 offered the first live evaluation (IOI 2010 Host Scientific Committee, 2010).

To the best of our knowledge, several different grading systems have been used during the period 1999–2011. Several times, the host country adapted its own grading system, used in national selection, to handle the tasks given in the IOI. Other times these systems were "borrowed" from other host countries. Finally, in IOI 2010 and 2011 the Marmoset grading system (Spacco *et al.*, 2006) has been used. It comes from outside the IOI community, thus it differentiates from the previous approaches.

The need for a system specifically designed for the IOI, with certain goals that we tried to achieve with CMS, has been obvious for many years (Verhoeff, 2002). Evident advantages are the transfer of expertise, the possibility to achieve a more polished product, the reusability of invested resources, and more. We decided to design CMS basing upon the previous experience of similar systems, which we analyzed in terms of the requirements and expectations from the ISC and the IOI community, as discussed below.

- Adapting the system used for the national selection of the host country is often the easiest solution, because of the experience of the country using its own software. But, often national selections and the IOI have slightly different needs; even more important, the IOI cannot expect that all host countries have a sufficiently developed or tested grading system. In these situations, having a standardized grading system could effectively lower the number of issues that a host country has to consider, possibly increasing the number of potential hosts.
- Using the system developed for the national selection of a different country (possibly a previous host, or not, as happened with the USACO system in IOI 2001; Kolstad and Piele, 2007) solves the issue of having a tested system ready to be used, but introduces some other problems: the current host may have no experience with the system, and often it is difficult to synchronize the adaptations needed for the current IOI with the main codebase, requiring additional work in the following years. It also often happens that these systems are privately developed, therefore they cannot be inspected by the community, nor used freely in other competitions such as national selections.
- Marmoset has been free since December 2009 (Pugh and Sims, 2009; Spacco *et al.*, 2012). It is a "system for handling student programming project submission, testing

and code review", and has been developed at the University of Maryland. Since it was born in the academic setting, and later applied to IOI 2010, it needed to be adapted to the context of the IOI. As far as we know, nobody has maintained the set of patches needed to be applied to Marmoset's codebase to comply with the competition rules of the IOI. On the other hand, Marmoset offers many more features that are related to teaching, thus increasing its size and complexity without any benefit for an IOI contest.

3. General Structure

CMS is organized in a modular way, with different services running, potentially, on different machines. When applicable, services can be replicated to allow scaling for larger contests.

CMS *Services*

The services, listed in Table 1, are Python programs built on top of AsyncLibrary, a custom-made RPC library using the asyncore framework (Sam Rushing and Python Software Foundation, 1996). Some of the services, called servers, provide also a second interface thanks to the web framework Tornado (Facebook, Inc., 2009), so they are able to both talk to the other services and to serve web pages to administrators or contestants. We require the services to be always responsive; hence remote requests must be completed in a short amount of time, or spawn a thread to handle the request. The only place where this behaviour is needed is in *Worker*, where a compilation or an evaluation cannot be split into multiple fast requests. All other services are single-threaded. The interactions among the services are shown in Fig. 1.

Table 1

The list of services composing CMS. Replicable is "Yes" when multiple instances of the service can cooperate to increase the capacity. The last two services (*ContestWebServer* and *AdminWebServer*) are also servers.

Name	Replicable	Duties
LogService	No	Receiving, aggregating and displaying all the logs of the system.
Worker	Yes	Running compilations and evaluations of submissions in a safe environment.
EvaluationService	No	Maintaining the queue of the jobs to be assigned to the *Worker*s.
ScoringService	No	Transforming the evaluation results into scores, and communicating them to the live rankings.
Checker	No	Calling the heartbeat function of all the services.
ResourceService	Yes	Collecting resource usage information about the machine in which it is running, and starting all the services on a machine.
ContestWebServer	Yes	Serving web pages to the contestants, accepting submissions and providing feedback.
AdminWebServer	No	Serving web pages to the administrators, configuring and managing the contests.

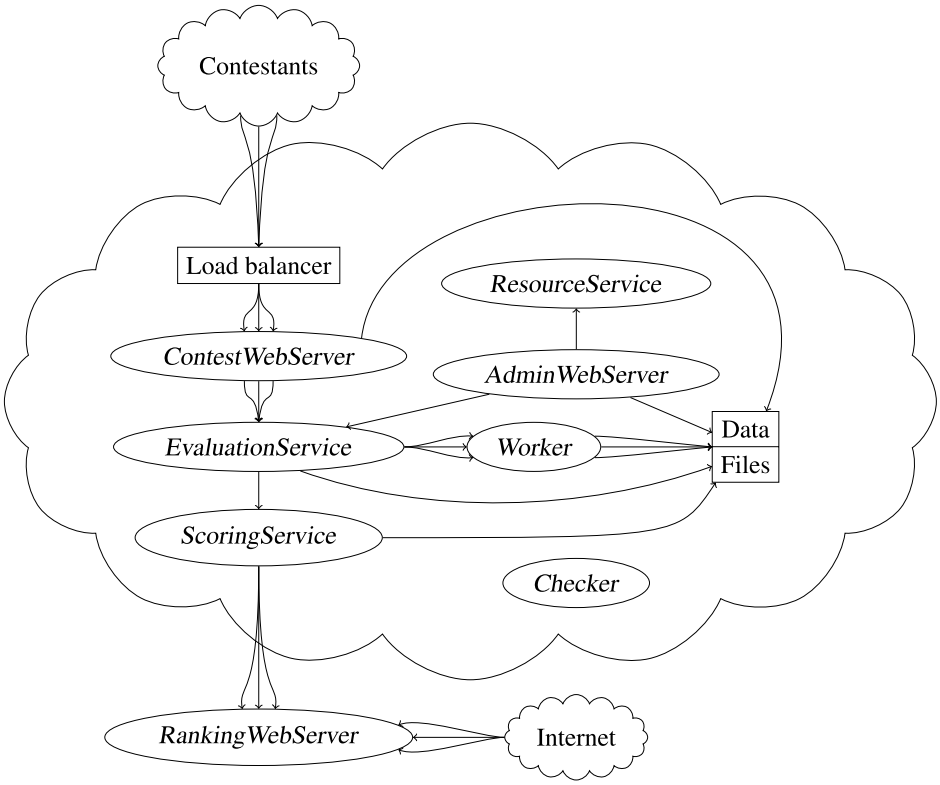


Fig. 1. Services and their interactions in CMS. An arrows represent a request of some kind, and multiple arrow heads mean that the service can be replicated. The services *Checker* and *ResourceService* talks with all the other services, and the relative arrows are omitted.

The nature of a grading system forces it to run untrusted code. To prevent contestants from cheating or even compromising the grading environment, all the evaluations and even the compilations of sources coming from contestants must run inside a sandbox. We use a (slightly modified) version of Martin Mareš's sandbox, which is part of the Moe contest environment (Mareš and Gavenčiak, 2001).

A special discussion is warranted for the program responsible for serving live rankings. Its name is *RankingWebServer* and can run in multiple instances on different machines. It is built on top of Tornado, but it is not part of the core structure of CMS, in the sense that it is allowed to run on machines that are not in the network used by CMS. This is because live rankings and the core system have different needs: the former needs to be accessed via a public network and may generate a large amount of traffic, whereas the latter must be private. CMS pushes data to *RankingWebServer* using HTTP queries, so the assumption that *ContestWebServer* is the only point of contact between CMS's network and the outside is preserved, since *RankingWebServer* does not query CMS.

Data Model and Implementation

From the viewpoint of CMS, the state of a contest is described by two kinds of objects: the *files* and the *data*.

- By files we mean something really similar to the usual concept of file: the only significant difference is that CMS does not consider them as part of a directory structure. They are used to store things that naturally look like a file, such as test-cases, task statements, submissions, compiled programs, etc.
- By data we mean a collection of objects implementing a set of classes defined in CMS; they are used to "glue" together the files and combine them in the actual description of the contest.

Files are addressed via their checksums using a hashing algorithm (SHA1 is used at the moment, but it would be easy to change it); in particular, they are immutable. This choice is justified by the huge simplification it gives: there are no concurrency problems, there is no need to care about local cache synchronization, it is easy to detect data corruption (although to save on CPU time, this is not done by default) and there is an automatic layer of data deduplication (if a file gets saved twice in the storage, only one copy is retained).

The file storage is implemented using PostgreSQL's "Large Objects", that can contain up to 2 GB of data each (PostgreSQL Global Development Group, 2012a). PostgreSQL supports accessing them using streams, so they can be copied to the local disk of a machine without loading them in the RAM. The previous and now discontinued implementation of the same service employed a custom service that kept files on the filesystem.

Data are stored in an SQL database using SQLAlchemy (Bayer, 2005) to do ORM (Object-Relational Mapping, i.e., automatically mapping the Python objects to their description in an SQL database). Most of the common SQL engines can be used with SQLAlchemy, although at the moment using anything other than PostgreSQL would require some minor modifications of CMS code, as explained below.

Files and data are stored in CMS using two different methods, hence, in theory, one could use two independent databases. At the moment this is not really possible, because CMS uses the same configuration line for both databases. This forces SQLAlchemy to be used with PostgreSQL, since it is the only database supported by the file storage. Changing this behaviour is not difficult and will be done as soon as there is evidence of its usefulness.

Handling Failures

It is important to notice that the database is the only unique point of failure of the whole system. It is thus essential that administrators can use standard procedures of replication and clustering to provide reasonable guarantees of uptime and resilience of data. The choice of using a widely-used database like PostgreSQL is supported, among other reasons, by the several mature solutions it offers to handle such situations; see for example (PostgreSQL Global Development Group, 2012b). CMS does not mandate the use of any of these solutions (it does not mandate the use of a replication or high availability solution at all), and, actually, the CMS authors have not yet settled on a preferred solution.

On the other hand, services can be started and stopped independently without affecting the system, of course apart from the downtime of the service itself. This allows administrators to quickly replace faulty hardware or software with identical copies, without caring about losing data or having to transfer information between the old and the new copy of the service, as long as the database is working correctly.

4. Walk Through

In this section, we will examine the procedure to run a contest using CMS, in order to get accustomed to its terminology and key concepts. We will assume that an on-site contest in the style of the IOI is being organized. The model we use is the competition rules for IOI 2011 (IOI 2011 Host Scientific Committee, 2011).

Network Setup

As mentioned, CMS does not interfere with the setup of the network. The only (sane) requirement comes from the fact that all the communication between services are not encrypted and there are no authentication methods for services; hence, the network in which CMS runs must be separate from the network of the contestants.

The only point of contact should be the HTTP port of the *ContestWebServer*, if the contest is small enough that one instance suffices, or the load balancer that splits the traffic amongst the instances of the *ContestWebServer*s (we provide a sample configuration to use nginx (Sysoev, 2004) as a load balancer).

To display a public live ranking, CMS's network must be able to access the (possibly remote) machine in which the *RankingWebServer*s are running; no other forms of communication are needed nor encouraged.

Task Preparation

Again, this is not an issue particularly relevant to CMS: the tasks can be set up using any method and any format. When everything is ready, the administrator should run an *importer*, that is, a program that translates the structure of the contest as it is in the file system to the internal format in the CMS database.

Of course, the importer depends on the particular format of the tasks. We provide an example importer, that works starting from the format used by the Italian Olympiads, and should be easy to adapt to different formats. We hope that in the future there could be a common task format for the IOI, that could be taken as a reference format for all the national Olympiads.

It is also possible to tweak the tasks after the import, or even to create the tasks from the interface of *AdminWebServer*. Nonetheless, using *AdminWebServer* to create contests and tasks is complicated by the large amount of different data needed to specify a task. To avoid mistakes and missing data, we encourage administrators to write their own importers.

There are some requirements that CMS enforces on the tasks: the *task type* (that is, the "instructions" on how to compile and evaluate the submissions) and the *score type* (how to translate the outcomes for each testcase to a score for the task) must be supported by CMS. All task types and score types, even the most standard ones, are in the form of plugins, loosely coupled to the core structure. In particular, it is easy to tweak types already defined or to add new ones.

Setting Up CMS

The administrators must install CMS on all the machines that will be running some services, and create a PostgreSQL database. They must decide, depending on the contest size and on the available resources, how many instances of a service to run. There are three replicable services: *ResourceService*, *ContestWebServer* and *Worker*.

We recommend to run an instance of *ResourceService* on every machine that is "interesting", that is, for which one would like to know the state in a given moment; in particular, for all machines where services are running.

For *ContestWebServer*, one can take advantage of multiple cores and provide as many instances as cores. In our experience, we reliably handled a contest with about 80 participants (OII 2011) with a single instance of *ContestWebServer*.

Finally, to provide the maximum fairness, the *Worker*s should be ideally one per machine (instead of one per core). A good rule of thumb is to allocate one *Worker* every 20 contestants, but this can significantly vary among contests.

After these decisions, the administrators need to make the relevant changes to the configuration file, that must be replicated on all the machines. In the future, this could be made easier by means of a configuration service. At the time of writing, a solution to deploy CMS to multiple machines by mean of network booting is nearly ready. This will allow administrators to avoid the process of installing and configuring the machines dedicated to the grading system.

Running CMS

When the configuration is synchronized on all the machines, the services can be started. Some of them need to work with a specific contest (*EvaluationService*, *ScoringService*, *ContestWebServer*), so it is also needed to have the contest stored in the database. Note that *AdminWebServer* does not need a contest, hence can be used to define new contests.

The services can be started together using *ResourceService*, which has an option to start automatically all the services in the given machine, and restart them automatically in case they die. As mentioned before, services can be killed and restarted without causing problems; one can also use the *AdminWebServer*'s "Resource usage" page to restart a specific service or disable the automatic restarting.

To ensure that everything works as expected, CMS offers to the administrators the possibility of running a test suite covering as many code paths as possible. The test suite is also a useful tool during the development, to ensure that new changes do not break existing code.

During the Contest

Once all the services are running, the contest is ready to start. The contest description contains the starting and finishing times (that can be changed on the fly, for example to delay a few minutes the beginning of the contest). Moreover, CMS supports a "relative timer" mode, where each contestant is given a certain amount of time starting from their first login into the system. This can be useful for online contests, to let contestants from different countries choose their preferred time slice to participate in the contest.

During a contest, contestants connect to *ContestWebServer* to submit their solutions, request feedback and check their statuses. The web interface also enables them to send questions and receive answers from the administrators, that can also send per-user messages and public announcements. Contest administrators can also use *AdminWebServer* to check that the contest is going smoothly, looking at statistics about the submissions and the services and their statuses; in case, they can request new compilations or evaluations of submitted solutions.

Once the contest is over, *ContestWebServer* stops accepting solutions and *EvaluationService* finishes processing the compilation and evaluation queue. Using *AdminWebServer*, administrators can decide to perform another evaluation round for either all or some selected submissions. In the end, *RankingWebServer* shows the final ranking of the contest. The complete state of the contest can be exported in a format that allows importing to another instance of CMS; moreover, administrators can also use an exporting tool to save the contest state in the format they prefer, without having to dump the whole database content. An example exporter is provided, that produces an output in the format used by the Italian Olympiad.

5. Internals of CMS

In this section we will describe in more detail the duties of the services, elaborating on how they interact and how the data flows through the system. Figure 2 represents this flow.

Submission of a Solution

Contestants interact with *ContestWebServer*; especially in case of national contests, *ContestWebServer*'s web interface can be translated in the local language using the standard tool gettext. Apart from read-only functions, such as downloading task statements and attachments, or inspecting the results of the submissions, there are two main functions contestants use: submitting solutions for a task, and test-releasing a submission.

Suppose that a contestant submits a solution. He has to provide exactly the files requested by the task, that are specified in the submission format table in the database. Output only (offline) tasks are an exception since the administrators may choose to let CMS "fill" the missing files using the ones from the previous submission.

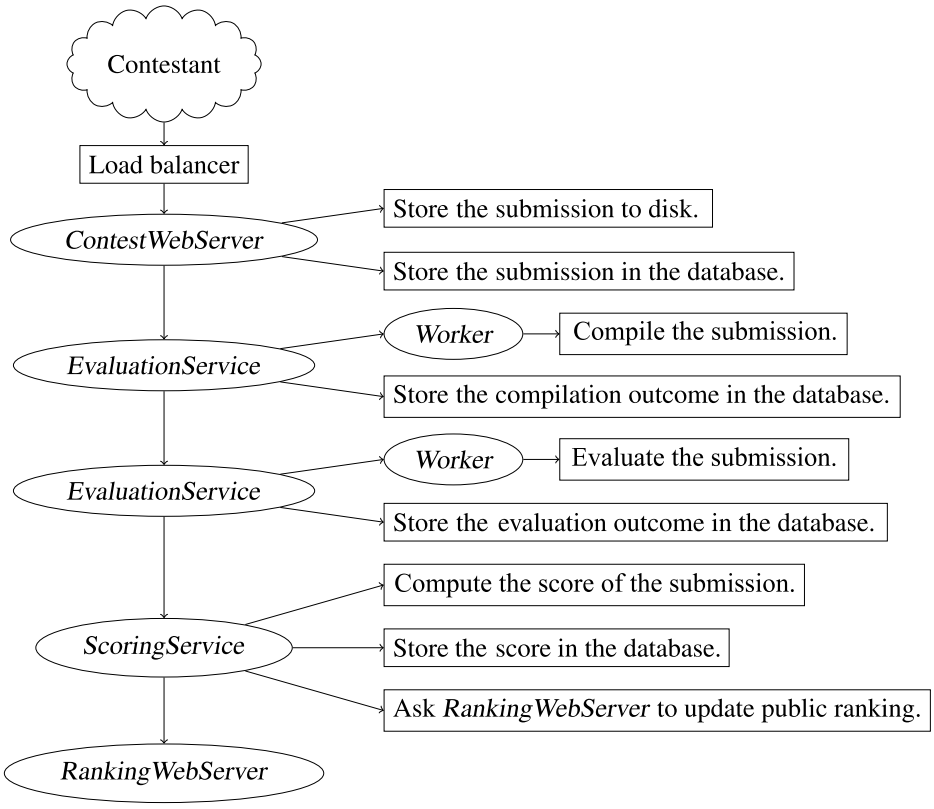


Fig. 2. Data flow of a submission through CMS.

Note that the task type, that is the plug-in in charge of specifying how to compile and evaluate the submission, uses the handlers provided by the submission format. For example, the submission format may force the contestant to submit a file with basename "encode" and one with basename "decode"; the task type may specify that they should be compiled into a single executable, and that the evaluation should run two instances of the program connected by a pipe. The task type can refer to another category of files, the *managers*, that are associated with the task and can be used, for example, to allow contestants to write just a function instead of taking care of the input/output, or to assess the correctness of the output when a plain diff is not enough.

If the submitted files respect the submission format, and some other predefined constraints are respected (files are not too large, submissions are not too frequent), *ContestWebServer* stores the submitted files in the database, and in a backup directory in the server, to allow recovery in case of a dramatic database failure. After that, it informs *EvaluationService* that a new submission is ready to be compiled.

Compilation and Evaluation

As a rule of thumb, services in CMS are informed that there is an action to perform. However they do not rely on the notification message, and also periodically check for actions that need to be done. For example, *EvaluationService* checks if there are submissions that need to be compiled or evaluated.

Either by receiving the message from *ContestWebServer*, or by checking the database, *EvaluationService* finds the submission to be compiled, and pushes it onto the job queue. A job is an action that a *Worker* performs with a single call: the compilation of a submission, or the evaluation on all testcases. *EvaluationService* ensures that, at any given moment, no more than one job per submission is in the queue.

When a *Worker* is available for a job, *EvaluationService* asks it to perform the job with the highest priority in the queue. We prioritize the compilations over the evaluations since they are often faster, and so they give an almost immediate feedback to the contestants.

As mentioned earlier, a job is usually long enough not to fit in the asynchronous framework of CMS, which requires the services to be responsive all the time: thus, the *Worker* spawns a thread dedicated to running the requested job. The main thread of the *Worker* does not accept other jobs before the current one is completed (nor *EvaluationService* should ask it to perform another job). The second thread instantiates the correct task type object and delegates the job to it, dying just after the job is completed and reported to *EvaluationService*.

A task type class needs to specify two operations: the compilation and the evaluation of a single testcase. Helper functions are provided to ease the writing of a task type: for example, it is easy to create and manage a sandbox, to run a program, or to move files from and to the sandbox. In the current version there are three task types already defined, for batch, output only, and communication tasks; the longest consists of about 60 Python statements. We plan to make it even easier to define new task types.

When the task type object has completed the job, *Worker* collects the result and send them back to *EvaluationService*, that writes them into the database. Indeed, *EvaluationService* is the only service that has "writing rights" on the parts of the database relevant to the compilation and the evaluation of submissions. If the job was a compilation, *EvaluationService* queues the evaluation; otherwise, it informs *ScoringService* that a submission is ready to receive a score.

Scoring

In the evaluation process, an *outcome* is assigned to each submission and testcase. An outcome is just a floating-point number, whose meaning depends on the score type of the task. When *ScoringService* receives the request from *EvaluationService*, or when it detects an evaluated submission without a score, it instantiates a score type object for the submission's task, and asks it to translate the outcomes of the submission to an actual score. When the result is ready, it writes it into the database, being the only service with writing rights to the scoring tables.

A score type class must implement two methods: the first returns the maximum score obtainable in the task, while the other returns the score actually obtained by a submission, plus some explanatory details string that will be passed directly to the contestant. Each of these methods returns data in two different flavours: the complete data, to be displayed to administrators and also to contestants that have test-released the submission, and a partial summary, for contestants that have not test-released the submission. Indeed, each task specifies a list of *public* testcases, intended to be shown to contestants regardless of the releasing status of the submission. Score types are usually simpler than task types, with an average score type taking around 25 Python statements.

Within the structure of CMS, the score computation is influenced only by the outcomes of the specific submission. It would be easy to use some other data regarding the submission (for example, the time); on the other hand, using data from *other* submissions generates interesting problems (both on the development side and on the competition side) that the authors did not want to address in the first iteration of the system. For instance, a development problem is that the number of changes of scores could be quadratic in the number of submissions, and not linear; a competition problem is that the relative position of two contestants in the ranking could change due to a submission from a third contestant.

Publishing the Scores

The second duty of *ScoringService* is to send all the information needed by the (possibly multiple) *RankingWebServer*s. This information includes the details about the contest, and the lists of users and tasks, that are transmitted when *ScoringService* starts. The other class of information includes submissions, together with their scores, and test-releasing times.

The internal structure of CMS works around the concept of a contest; for example, a row in the contestants table in the database does not refer to a physical person, but to the participation of a person in a contest. Competitions that are distributed in more than one day should use a CMS contest per day, replicating the contestants. Nonetheless, *RankingWebServer* can display a complete ranking for such contests; indeed, being completely detached from the information stored in CMS's database, it can choose its own format: it merges all the entities received by the *ScoringService*, so that using the same *RankingWebServer*s for all the days of a competition effectively results in having a ranking for the whole competition.

RankingWebServer can handle and display other information about the contest and its participants, namely a picture of each contestant, and partition them into teams. This information is not currently provided by CMS itself, so they must be manually loaded into the *RankingWebServer*s beforehand.

6. Conclusion

We presented a grading system specifically designed for the needs of a programming contest in the style of the IOI. Nonetheless, we took care of writing a system that is

extensible and configurable, in order to accommodate both possible future changes to the competition rules of the IOI, and different competitions (national selections, online contests, etc.).

The focus of the early development, that still carries on, has been directed towards providing a safe, tested infrastructure and a pleasant experience, in particular from the point of view of contestants. Still, facilities for administrators have been developed, and will be further expanded in the future. An agreement on a common format for developing and storing IOI-related programming tasks would be a great enhancement in this respect.

We hope that such a system can be further developed collaboratively. We are pleased that these collaborations with future hosts have already started, hoping that they will be fruitful for the IOI community.

Acknowledgments. The work in this paper has been partially supported by AICA (Associazione Italiana per il Calcolo Automatico) and MIUR (Ministero dell'Istruzione, dell'Università e della Ricerca). We would like to warmly thank Matteo Boscariol, developer of CMS together with the authors of this paper. We also acknowledge the important contributions of organizations and individuals that helped in all stages of the development of CMS: Luca Wehrstedt and Bernard Blackham for contributing to the design and development of substantial parts of CMS, the teams supervising the Italian and Australian selection process for IOI 2012 for providing testing and useful suggestions. A special thank you goes to Roberto Grossi (Chair of the IOI 2012 Host Scientific Committee) for his constant support and trust in our work. Finally, we would like to thank Bernard Blackham and Roberto Grossi for reading draft versions of this paper and giving valuable suggestions.

References

Bayer, M. (2005). *SQLAlchemy: The Database Toolkit for Python*. `http://www.sqlalchemy.org/`.

Boscariol, M., Maggiolo, S., Mascellani, G. (2010). *CMS, a Contest Management System*. `https://github.com/cms-dev/cms`.

Facebook, Inc. (2009). *Tornado Web Server*. `http://www.tornadoweb.org/`.

Free Software Foundation, Inc. (2007). *GNU Affero General Public License*. `http://www.gnu.org/licenses/agpl-3.0.html`.

Heyderhoff, P., Hein, H.-W., Krückeberg, F., Miklitz, G., Widmayer, P. (1992). *Final Report International Olympiad in Informatics 1992 Bonn/Germany*. `http://www.ioinformatics.org/locations/ioi92/report.html`.

IOI 2010 Host Scientific Committee (2010). *Competition Rules for IOI 2010* (Draft 1.0). `http://www.ioi2010.org/rules.shtml`.

IOI 2011 Host Scientific Committee (2011). *Competition Rules for IOI 2011* (Draft 1.0). `http://www.ioi2011.or.th/rules`.

Kolstad, R., Piele, D. (2007). USA Computing Olympiad (USACO). *Olympiads in Informatics*, 1, 105–111.

Mareš, M., Gavenčiak, T. (2001). *The Moe Contest Environment*. `http://www.ucw.cz/moe/`.

Piele, D. (1999). *Report from Turkey*. `http://www.ioinformatics.org/locations/ioi99/don99.shtml`.

PostgreSQL Global Development Group (2012a). *PostgreSQL 9.1 Manual. Large Objects*. `http://www.postgresql.org/docs/9.1/static/high-availability.html`.

PostgreSQL Global Development Group (2012b). *PostgreSQL 9.1 Manual. High Availability, Load Balancing, and Replication*.
`http://www.postgresql.org/docs/9.1/static/high-availability.html`.
Pugh, W., Ryan, S.W. (2009). *Marmoset*. `https://code.google.com/p/marmoset/`.
Sam Rushing and Python Software Foundation (1996). *Asyncore: Asynchronous Socket Handler*.
`http://docs.python.org/library/asyncore.html`.
Spacco, J., Pugh, W., Ayewah, N., Hovemeyer, D. (2006). The Marmoset project: An automated snapshot, submission, and testing system. In: *OOPSLA Companion*, 669–670.
`http://dl.acm.org/citation.cfm?doid=1176617.1176665`.
Spacco, J., Pugh, W., Ayewah, N., Hovemeyer, D. (2012). *Marmoset*.
`http://sourceforge.net/projects/marmoset/`.
Sysoev, I. (2004). *Nginx*. `http://nginx.org/`.
Verhoeff, T. (1994). *The Sixth International Olympiad in Informatics: A Trip Report*.
`http://www.ioinformatics.org/locations/ioi94/rprt-nl/index.html`.
Verhoeff, T. (2002). *Minutes of the International Scientific Committee Meeting 6, 17th May 2002*.
`http://ioinformatics.org/admin/isc/iscmeetings/meet-06/minutes06.txt`.

S. Maggiolo is a PhD student in geometry at SISSA/ISAS, Trieste. He participated in IOI 2002, winning a bronze medal and in IOI 2003. Since 2006 he collaborates with the training and selection process for the Italian team at IOI, and has been an observer in IOI 2009 and the deputy leader of the Italian team in IOI 2011. He is a member of the HSC for IOI 2012 and one of the main author of CMS, the contest system that will be used in IOI 2012.

G. Mascellani has been a contestant in IOI 2007 and 2008, winning a silver and a bronze medal. He is a student of mathematics in Pisa at the Scuola Normale Superiore and collaborates in training the Italian team for IOI. He is a member of the HSC for IOI 2012 and is one of the main authors of CMS, the contest system that will be used in IOI 2012. He is a Debian Developer.

Olympiads in Informatics, 2012, Vol. 6, 100–109

A New Contest Sandbox

Martin MAREŠ[1], Bernard BLACKHAM[2]
[1]*Department of Applied Mathematics, Faculty of Mathematics and Physics*
Charles University in Prague
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
[2]*School of Computer Science and Engineering, University of New South Wales*
Sydney NSW 2052, Australia
e-mail: mares@kam.mff.cuni.cz, bernardb@cse.unsw.edu.au

Abstract. Programming contests with automatic evaluation of submitted solutions usually employ a sandbox. Its job is to run the solution in a controlled environment, while enforcing security and resource limits. We present a new construction of a sandbox, based on recently added container features of Linux kernel. Unlike previous sandboxes, it has no measurable overhead and is able to handle multi-threaded programs.

Key words: automatic grading, sandbox, containers, threads, computer security.

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. The program must also finish each test run within given time and memory limits, so that it is possible to distinguish between correct solutions of different efficiency.

Additionally, proper security measures must be taken to avoid cheating – e.g., the program must not be allowed to access files to steal the correct answer, kill other processes, nor communicate over the network.

To achieve both security and limits on resources, programs are usually run within a controlled environment called a *sandbox.* In recent years, most programming contests seem to be run on Linux systems, so we will study Linux sandboxes only.

By far, the most common kind is a tracing sandbox; see, e.g., Mareš (2007) or Kolstad (2009). It uses the `ptrace` system call to ask the kernel to stop the sandboxed program whenever it is about to execute a system call. A monitoring process then examines the arguments of the system call and either lets the call proceed, or kills the program for committing a security violation. Time and memory limits are usually enforced by a combination of system call monitoring and the standard resource limit infrastructure inside the kernel (the `setrlimit` system call).

Recently, the overhead of system call tracing has received lots of attention. Merry (2010) has shown that especially in the case of interactive tasks, the number of system

calls can become very large and so can the time spent monitoring them. Mareš (2011) has published a more careful analysis, which shows that while the overhead is significant, it does not affect contest fairness much. He also evaluates several techniques for decreasing variance of time measurements, like pinning of tasks to specific processor cores, or real-time scheduling.

Another common problem with ptrace-based sandboxes is that they are unable to enforce security of programs running in multiple processes of threads. At the first sight, it seems not to matter, since programming contests rarely involve writing parallel programs. However, the run-time environments of several programming languages automatically create several service threads. This happens for example in the Java Virtual Machine, and in Mono, which is an implementation of the Microsoft Common Language Infrastructure.

Ptrace-based sandboxes are also highly architecture-specific. Supporting both 32-bit and 64-bit execution environments on x86 requires different implementations of several aspects of the sandbox, including the list of allowable system calls. This is complicated even further on 64-bit x86 machines, as the evaluated program may be either a 32-bit or 64-bit binary, or even a 64-bit binary using 32-bit system calls.

In this paper, we present a new design of a contest sandbox. It is based on namespaces and control groups, which are new features of the Linux kernel. These are primarily intended for partitioning a large machine with many processors into multiple nodes, but have turned out to be useful for our purposes, too. The advantages of this approach include a much smaller overhead and the ability to reliably sandbox multi-threaded programs.

In Section 2 we give a brief overview of some related state-of-the-art sandboxing techniques. In Section 3 we outline our requirements of a sandbox for programming contest environments. In Section 4 we describe the kernel features which we use for our sandbox, followed by a description of our implementation in Section 5. Finally, we measure the performance overhead of our sandbox in Section 6.

2. Related Work

Research on code isolation did not stop with simple tracing of system calls. Let us review currently known sandboxing techniques first.

Merry (2009) has suggested a contest sandbox based on the Linux Security Module infrastructure. Instead of monitoring the system calls by a user-space program, it inserts a kernel module, which uses kernel security hooks to check parameters of system calls. The time and memory overhead is almost zero, but Merry's implementation does not support multiple threads and it would be complicated to do that. The main drawback of this approach is the instability of the security module interfaces between kernel versions – thus the module has to be updated for every new kernel release.

It is also possible to use hardware virtualization or para-virtualization, so that the contestant's program can run on its own virtual machine with its own instance of the operating system (OS). All interaction between the program and the rest of the world can be easily limited by the configuration of the virtual machine. Unfortunately, the overhead

of contemporary virtual machines is very high and suffers from a large variance, so it does not seem possible to use them in a fair contest.

Software fault isolation (SFI) is a method of enforcing the secure execution of arbitrary code, as described in Wahbe *et al.* (1993). SFI is typically used to provide isolation between modules running in the same address space (e.g., dynamically loaded plug-ins). This approach is overkill for batch-processing type tasks used by many contests, as a contestant's program executes in a separate address space, enforced by the processor. It could be applicable to interactive tasks where the context switching overheads are high. However, many of these techniques have high overheads, and require recompilation of code using special compilers to intercept all potentially harmful operations.

Native Client (NaCl) is another approach to sandboxing native executables proposed by Yee *et al.* (2009). It implements software fault isolation, performing static analysis to ensure that all user-supplied code can be safely executed, but also makes use of hardware support to enforce memory protection. Once checked and loaded, user code runs at native speed. NaCl enforces specific requirements on the executable's structure to ensure the static analysis is sound. Like other SFI-based approaches, binaries must be compiled with specific compilers and libraries.

Linux also offers a security module known as *seccomp*, which can limit the system calls accessible to a program. This is similar in spirit to ptrace-based approaches, but in its original version, it was limited to allowing a very restrictive, fixed set of system calls. Programs had to be significantly modified to execute under seccomp. More flexible forms of seccomp have been proposed in the past, but as of the time of writing, none of these have made it into the upstream Linux kernel.

TxBox, developed by Jana *et al.* (2011), supports sandboxing of arbitrary processes through the use of operating system-level transaction support. Using transactions can limit the impact of untrusted insecure code, as system state can be "rolled back" after execution. This provides strong isolation properties and can work with arbitrary executables, but requires significant modifications to the OS kernel, and is not (yet) supported in popular Linux distributions.

3. Requirements for a Contest Sandbox

Most programming contests require students to submit the source code for their solution, which is automatically compiled and graded on a server. We wish to minimize the work required for a contest organiser in setting up such a server, and as such we aim to support a modern unmodified Linux distribution. We also wish to support as many languages as is practically feasible. This precludes the use of special customized OS kernels, distributions or custom language-specific toolchains. We aim to sandbox binaries compiled using standard compilers, and still ensure the integrity of the system.

In contest environments, the only untrusted part of the system is the program submitted by the contestant. All security risks therefore involve interaction between this program and the rest of the system. Any such interaction requires the use of system calls. Let us

review the list of all Linux system calls and look for their potential security issues. (We refer the reader to Kerrisk *et al.* (2012) for a concise description of Linux system calls and to Kerrisk (2010) for a more comprehensive treatise on Linux system interfaces.)

We note that our list includes only the security risks known to us (including those mentioned by Forišek (2006)) and we do not have a formal proof that it is complete. (In fact, such a proof would necessarily include a proof of correctness of the whole Linux kernel.)

The problematic system call groups are:

- *Access to files* (`open`, `read`, `write`, `stat`, ...). The set of files available inside the sandbox can be easily restricted by using traditional filesystem permissions. The sandboxed program runs under its own user and group ID and all directories outside a designated area are made inaccessible or read-only for that user ID. Additionally, we can change the root directory of the process to some sub-directory. We must not forget that the amount of data written to disk by the program has to be limited, which can be achieved using disk quotas.
- *Allocation of memory* (`brk`, `mmap`, `mlock`, ...). We must restrict the amount of memory available to the program. Even if we do not want to grade memory complexity of solutions, we must avoid exhaustion of system memory, which could cause the other parts of the contest system to fail. In case of a single process, `setrlimit` can be used to limit the total amount of address space available. There is no traditional UNIX mechanism for limiting total memory consumption of a group of processes.
- *Creation of processes and threads* (`fork`, `clone`, `vfork`). In a traditional Pascal/C environment, only a single process or thread (Linux does not distinguish between them internally) should be allowed. If we want to support multi-threaded runtimes, we must place a limit on the total number of processes to avoid overloading the task scheduler. Fortunately, there is a system resource limit on the total number of processes run by a given user. Also, a program may attempt to evade time limits by splitting the calculation into several processes, each running on different physical processors. To prohibit this, we have to measure the sum of execution times of all processes within the sandbox.
- *Sending of signals* (`kill`, `killpg`, `tkill`, ...). Signals are asynchronous events delivered to processes or threads. Sending of signals to our own process is harmless, but we must forbid sending signals to other processes. Otherwise, a part of the grader or even an unrelated process can be killed or confused. As expected, UNIX already disallows signalling processes run by other users, so running the process under a unique user ID suffices.
- *Inter-process communication* (`shmget`, `msgget`, `mq_open`, ...). There are several APIs for sending messages and sharing parts of memory. They are controlled by filesystem-like permissions, but a process can create a message queue or shared memory object accessible to everybody. This could be used for cheating, so we want to forbid such operations, or even better put a barrier between communication objects of our process and the rest of the system. Alas, there is no traditional mechanism for that.

- *Networking* (`socket`, `bind`, ...). All attempts to communicate over the network must be stopped. This includes not only TCP/IP networking, but also other address families, including local sockets addressed as a part of the filesystem. The traditional UNIX socket API does not offer any configurable limits.
- *Executing other programs* (`execve`). The program can run other applications and let them handle a part of the competition task. For example, it could replace calculations with big integers by calls to `bc` or `python`. While we do not necessarily consider such tricks unfair, we would still like to support contest organizers who do so. We do not see a way how to disallow `execve`, but it is easy to keep the directory tree available inside the sandbox free of all foreign programs. Optionally, some parts of the directory tree can be mounted with the `noexec` option, so that binaries stored in them cannot be executed at all.
- *Sleeping* (`pause`, `sigsuspend`, `wait`, `nanosleep`, `poll`, ...). There are multiple system calls which suspend the program until some event occurs. This may be for example an arrival of a signal, exit of a child process, readiness of data on a file descriptor, or simply the expiration of a timer. None of these operations compromise security, but they can lock up the grading system for an indefinite amount of time. To avoid that, we limit not only the execution time, but also the time elapsed on a "wall clock" (i.e., an independent clock measuring real time). The wall-clock time limit is usually set somewhat higher than the run-time limit, so the program does not time out if it shares the processor with other programs.
- *Accessing system time* (`alarm`, `gettimeofday`, ...). The rules of some contests explicitly forbid reading of system time. This limitation is hard to support without explicit system call tracing and we do not consider it important as there are many side-channels which can be used to estimate elapsed time. One example is the timestamp counter (TSC) inside the processor, which is essentially a register containing the number of CPU cycles since system boot. We have therefore decided not to limit time-related system calls.
- *Flushing buffers to disk* (`sync`, `fsync`, `sync_file_range`, ...). These operations do not compromise any secrets, but can result in added disk I/O activity. This extra disk activity can slow down the execution of both the contestant's program and other processes on the system. In our opinion, imposing wall-clock time limits is sufficient to limit the effects. For those contest administrators who are still concerned, one possible work-around is to put the writeable part of the directory tree on a RAM-disk.

4. Kernel Compartments

Traditional virtualization focuses on running multiple instances of Linux using a hypervisor, whereas OS-level virtualization (or compartments) offers lower overheads and greater timing predictability. There exist several projects to add OS-level virtualiza-

tion into the Linux kernel, including OpenVZ[1], LXC[2], and the Linux-VServer project[3]. Thanks to these projects, there is now sufficient infrastructure in the official Linux kernel to support compartments for running arbitrary untrusted code.

4.1. *Namespaces*

The Linux kernel provides the ability to isolate groups of processes through the use of *namespaces.* A namespace is a context used for specific kernel operations, such as networking, filesystem access or process control. Specifically, each Linux process belongs to:

- *a process namespace:* this is the set of all processes which are visible to a compartment. An isolated process should live in its own empty process namespace, preventing it from signalling or otherwise interfering with any other process on the system. Process namespaces form a hierarchy when created, such that a process is visible in its own namespace and all parent namespaces. Furthermore, when the top-level process of a namespace exits, all other processes spawned by it are automatically terminated. (Without this feature, reliably terminating a hierarchy of possibly malicious processes is almost impossible.)
- *a networking namespace:* this determines the set of networking devices available to the compartment, which for isolated processes should be none. Although isolated processes can still create IP sockets, they cannot use them as there are no network devices (not even the loopback interface).
- *a filesystem namespace:* this defines the set of mounted filesystems that can be accessed by processes. An isolated process may have its accessible filesystems stripped down to the bare minimum – this may simply be a single RAM-disk. Unlike traditional chroot jails, filesystem namespaces cannot be easily circumvented.
- *an IPC namespace:* this namespace protects the inter-process communication system calls which provide shared memory and message-passing (`shmget`, `msgget`, `mq_open`, ...). In a typical UNIX system, all processes on a machine may potentially communicate. By placing an isolated process in an empty IPC namespace, it is unable to interact with any other processes on the system using IPC.

When a new process is created using the `clone` system call, it can be optionally placed into new empty namespaces in each of the above categories (or otherwise inherit its parent's). These can be used to effectively limit what kernel resources are accessible to a process.

4.2. *Control Groups*

In the previous section, we showed that namespaces can tightly control access to kernel-provided resources and limit interaction between processes. However, they do not enforce any control over CPU time or memory resources. Traditional UNIX mechanisms

[1] `http://www.openvz.org/.`
[2] `http://lxc.sourceforge.net/.`
[3] `http://www.linux-vserver.org/.`

only allow CPU and memory to be limited on a per-process basis. They do not scale for enforcing these limits for groups of processes.

Linux has recently introduced the concept of control groups to achieve this. The administrator can define a hierarchy of control groups and place processes at arbitrary points within it. New processes automatically inherit their parent's control group.

A hierarchy can have several *controllers* connected to it. A typical controller manages some resource. For each group, it tracks the usage of the resource by all processes inside the group and its subgroups. Each group can also have its own limits on the maximum allowed usage.

The following controllers are interesting for our purposes:

- *CPU set controller:* processes inside a group can be tied to a subset of available processors, processor cores or memory nodes. We want to reserve such a subset for exclusive use by the sandbox. This helps us minimize timing noise caused by context switches and related caching effects. This timing noise can add significant variance to the measured execution time, as shown by Merry (2010).
- *memory controller:* total memory used by the group can be tracked and limited. This includes not only explicit memory allocations by processes, but also everything implicitly allocated by the kernel on behalf of these processes, for example cached parts of files.

  The accounting of memory pages is complicated by the presence of pages shared between processes belonging to different control groups. The memory controller accounts for them on one more-or-less randomly chosen group. Fortunately, we have already limited inter-process communication, so the only shared pages which can really occur are parts of commonly used files (e.g., shared libraries). When the memory gets tight, such pages are reclaimed, so while they can affect reported memory use of the sandbox, they should not influence memory limits.
- *CPU accounting controller:* this controller does not impose any limits. It just tracks total CPU time used by processes inside the group. Unlike traditional task timers which are based on statistical sampling, this controller works with nanosecond-precision timestamps of context switches used internally by the process scheduler. We periodically monitor the CPU time used and when it exceeds the limit, we terminate all processes.

5. Implementation

We have implemented a new sandbox based on the ideas described above and checked that it withstands all attacks on security known to us.

The sandbox has been incorporated in the Moe modular contest system (see Mareš (2009) for an overview), but it does not depend on any other modules, so it can be easily used in other contest systems, too. Source code and a test suite are available at `http://www.ucw.cz/moe/`.

Please note that a recent Linux kernel is required (we have used version 3.2.2) and that it must be configured to support namespaces and control groups. This is not always the

case with default kernels supplied by Linux distributions. We refer to the documentation accompanying the source code for more details on system configuration issues.

5.1. *Features*

By default, our sandbox runs in a light-weight mode, which uses only namespaces and user identity separation to achieve security. In this mode, programs are limited to a single process or thread only. (In fact, more processes can be allowed, but time and memory limits then apply to each individual process, which is seldom useful.)

When use of control groups is switched on, the sandbox uses the CPU and memory controllers to limit overall consumption of resources. In this mode, an arbitrary number of processes and threads may be run.

Both modes support time and memory limits. There are separate time limits for real execution time and wall clock time. It is possible to keep the program running for a while after the time limit expires, so that the exact run time is known even for programs which have timed out. Memory limits in the control-group mode affect the total memory usable by both by the OS and processes, while in the light-weight mode they affect all virtual memory allocated by the process only.

All processes are started with a custom root directory stored in a RAM-disk. This directory contains only a set of mount points used for bind-mounting selected portions of the system directory tree. These usually include standard libraries mounted read-only and a read-write working directory, which holds input and output files of the program and it is limited by a disk quota. The contents of the custom root are configurable.

Unlike ptrace-based sandboxes, we do not require any knowledge of the CPU architecture. Our sandbox uses only isolation primitives provided by the Linux kernel, and is therefore portable to any host CPU architecture that is supported by Linux. This also avoids the complications associated with 32-bit and 64-bit executables, described earlier.

5.2. *Achieving Better Reproducibility*

As Linux is a very complex system, running on inherently complex hardware, there are an exponential number of states a system may be in when judging a solution. The state of the system when a program commences can directly affect not only the execution time of that program, but also its behaviour. There are a number of measures that must be taken in order to improve the consistency of the system across executions. They are essential to the fairness of any contest environment, not only of our sandbox. (See Mareš (2011) for further discussion.)

Address-space randomization is a feature used to protect system binaries from code-injection exploits. It does this by altering the address-space layout of a process each time it is started. Although this poses no concern for correct programs, buggy programs with memory management errors may behave inconsistently as the bugs might occur only for certain address-space layouts. This is clearly undesirable for a programming contest, and can be turned off by writing `0` to `/proc/sys/kernel/randomize_va_space`.

Table 1

The measured execution times of a process performing 1,000,000 system calls under different sandboxing techniques. The results are the average (and standard deviation) of 50 executions

Sandbox	Execution time [seconds]	Slowdown against native
None	3.56 (0.014)	–
Ptrace	9.26 (0.015)	160%
Namespaces	3.56 (0.010)	0%

Modern computer systems are designed to save energy when unutilized. One of the ways in which this is achieved is by scaling the CPU's frequency to best suit the current workload. This directly affects the run-time of processes and its behaviour depends heavily on what other activities are executing on the system. For a contest environment, frequency scaling must be disabled to ensure consistency across executions. This can be done on Linux by forcing the CPU to the highest available frequency, by writing `performance` to `/sys/devices/system/cpu/cpu*/cpufreq/scaling_governor` for each CPU on the system.

6. Evaluation

We compare the improvement in run-time of our sandbox over a traditional ptrace-based sandbox as used in Moe. Processes that perform very few system calls have negligible impact from either sandboxing approach. The overheads of ptrace-sandboxing are only seen when many system calls are performed. The results for a process performing 1,000,000 system calls are shown in Figure 1. These experiments were performed on an Intel Core-2 Duo T8300 CPU running at 2.4 GHz, using a 3.2.2 Linux kernel.

We measured the overhead introduced by a ptrace-sandbox to be 5.6 μs per system call. For many batch tasks, where the number of system calls performed is in the order of 10,000 at the most, this is negligible. However, interactive tasks which communicate with other processes could easily demand over 1,000,000 system calls. Here, the ptrace sandbox would add over five seconds to the evaluation time.

In contrast, we measured no overhead using our sandbox and all standard deviations were less than 0.5% of the mean. This is not surprising, as all isolation is performed inside the kernel using its standard mechanisms.

7. Conclusion

We have described a new type of sandbox, primarily intended for, but not limited to, use in programming contests. Unlike other sandboxes, it is able to isolate multiple processes and has very small overhead, while its implementation is very simple and architecture-independent.

Thanks to the small overhead, it gives significantly more precise execution timing for programs involving lots of system calls, especially for interactive contest tasks.

Support for multiple processes allows us to evaluate programs written in programming languages with multi-threaded runtimes, like Java, C#, or Erlang. Furthermore, it can be easily used to isolate execution of compilers and graders.

We are aware that the mechanism we use is not completely secure with respect to new kernel versions. A new kernel may include additional system calls to operate on new types of kernel objects, which do not belong to namespaces known to the sandbox. We however expect that such objects will be accompanied by new namespaces, so it will be possible to trivially extend the sandbox to handle them.

References

Forišek, M. (2006). Security of programming contest systems. In: Dagiene, V., Mittermeir, R. (Eds.), *Information Technologies at School*, 553–563.

Jana, S. *et al.* (2011). TxBox: building secure, efficient sandboxes with system transactions. In: *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, 329–344, Austin, TX, USA.

Kerrisk, M. (2010). *The Linux Programming Interface*. No Starch Press.

Kerrisk, M. *et al.* (2012). *The Linux Man-Pages Project*. Available on-line at `http://www.kernel.org/doc/man-pages/`.

Kolstad, R. (2009). Infrastructure for contest task development. *Olympiads in Informatics*, 3, 38–59.

Mareš, M. (2007). Perspectives on grading systems. *Olympiads in Informatics*, 1, 124–130.

Mareš, M. (2009). Moe — design of a modular grading system. *Olympiads in Informatics*, 3, 60–66.

Mareš, M. (2011). Fairness of time constraints. *Olympiads in Informatics*, 5, 92–102.

Merry, B. (2009). Using a Linux security module for contest security. *Olympiads in Informatics*, 3, 67–73.

Merry, B. (2010). Performance analysis of sandboxes for reactive tasks. *Olympiads in Informatics*, 4, 87–94.

Wahbe, R. *et al.* (1993). Efficient software-based fault isolation. In: *Proceedings of the 14th Symposium on Operating Systems Principles*. 203–216, New York, NY, USA.

Yee, B. *et al.* (2009). Native client: a sandbox for portable, untrusted x86 native code. *IEEE Symposium on Security and Privacy*, Oakland, USA.

M. Mareš is as an assistant professor at the Department of Applied Mathematics of Faculty of Mathematics and Physics of the Charles University in Prague, organizer of several Czech programming contests, member of the IOI Scientific Committee, and a Linux hacker.

B. Blackham is a PhD student with the Software Systems Research Group at NICTA and the School of Computer Science & Engineering at the University of New South Wales in Sydney, Australia. He was formerly the team leader of the Australian IOI team and has been involved with training the Australian team for 10 years.

Olympiads in Informatics, 2012, Vol. 6, 110–114

Tasks of a Priori Unbounded Complexity

Pavel S. PANKOV[1], Kirill A. BARYSHNIKOV[2]
[1]*International University of Kyrgyzstan*
[2]*OJSC "Finance Credit Bank", Bishkek, Kyrgyzstan*
e-mail: pps50@rambler.ru, baryshnikov.kirill@gmail.com

Abstract. We consider the following types of tasks: (*) Is there an element meeting some conditions in an infinite set? (**) If such element a priori exists, find any element or the first of such elements. By the condition, the number of operations (steps) to solve the task is unbounded, and the first aim is to guess any upper estimation for this number and further to improve this estimation. In general, tasks (*) are non-resolvable because they are reduced to the well-known problem of words identity in Markov algorithmical language; tasks (**) are resolvable because an infinite discrete set is countable. We hope that some classes of such tasks may be of interest for using in informatics olympiads of various levels. A survey of such tasks in various branches of informatics and some ideas to create and to solve them are presented.

Key words: informatics olympiads, tasks, unboundedness.

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

Most of tasks offered at informatics olympiads are such that any (non-effective) algorithm (of exponential or polynomial complexity) and a (rough) estimation of steps during its execution are obvious and this algorithm can be implemented immediately.

Remark 1. We advise participants, who are not so experienced, to implement such an algorithm first and to check better algorithms (with small initial data) with it.

We consider another type of tasks: a priori, the number of operations (steps) to solve the task is unbounded (in other words, the search is in an infinite set). In general, such tasks are non-resolvable because some of them are reduced to the well-known problem of words identity in Markov algorithmical language or to Diophantine equations.

Nevertheless, we hope that some classes of such tasks may be of interest for use in informatics olympiads of various levels. A survey of such tasks in various branches of informatics and some ideas of to solve them are presented. Certainly, the jury is to know existence of a solution and any algorithm which can solve the task in appropriate time. This knowledge is the main information for the participant of the contest.

In general, such tasks may be put as follows:

General task 1. Find an element meeting some conditions in an infinite set.

General task 2. Find the first elements meeting some conditions in an infinite ordered set.

Here are some initial ideas:

Idea 1. Guess any finite set containing (or an upper estimation for) such element and further to improve this estimation.

Idea 2. Reduce the search on the infinite set to one on any sparse subset.

Idea 3. Distinguishing of "basic" or "unit" solutions to build a general solution of them (or an upper estimate for the solution).

In its turn, Idea 1 may be attained by

Idea 1a. Proof or guessing of some periodicity. Then the first period only is to be considered.

Idea 1b.Using the least common multiple for proof of periodicity.

To generate such tasks, the following ways are proposed:

Way 1. Use any known mathematical result (in number theory, theory of finite groups, combinatorics, graph theory etc.). Some examples are given below. It may be used for training but not for high level olympiads.

Way 2. Set an interesting task and try to solve it yourself.

Way 3. Take (occasionally) any element or notice any interesting element of the finite set, formulate its properties and try to construct an algorithm which can find this element in appropriate time.

Remark 2. Tasks of search on a finite set (easier) also can be constructed in such a way. After defining the element to be found we announce any a priori upper boundary greater than this element.

2. Number Theory

Let us begin with classical results (Way 1). All numbers are assumed to be integer and non-negative.

Task 3 (Chinese theorem of residues). Given prime numbers $p_1 < p_2 < \ldots < p_n$ and numbers $r_1 < p_1, r_2 < p_2, \ldots r_n < p_n$, find a number K such that $K \bmod p_i = r_i$, $i = 1..n$.

First step of solution (Idea 1b). If K meets these conditions then $K \pm p_1 p_2 \ldots p_n$ meets them too. Hence, to check numbers $1..p_1 p_2 \ldots p_n$ is sufficient ($O(p_1 p_2 \ldots p_n)$ operations).

An effective algorithm is also obvious: Let $K_1 := r_1$. Find such $d_{i+1} (0 \leqslant d_{i+1} < p_{i+1})$ that

$(p_1 p_2 \ldots p_i d_{i+1} + r_i) \bmod p_{i+1} = r_{i+1}$ and let $K_{i+1} = p_1 p_2 \ldots p_i d_{i+1} + r_i$, $i = 1..n-1$.

($O(p_1 + p_2 + \ldots + p_n)$ operations).

Task 4 (amicable numbers). Find such numbers u and v that the sum of proper divisors of u equals v and the sum of proper divisors of v equals *u.* (The first pair is small: 220 and 284, so an interesting task is the following: Find two pairs. . .).

Adding numbers themselves to the list of divisors we obtain a more convenient formulation:

Find such numbers u and v that the sums of their divisors equal $(u + v)$; and the formula for solving: sum of divisors of a number with prime factor decomposition $p_1^{k_1} p_2^{k_2} \dots p_n^{k_n}$ is

$$(1 + p_1 + \dots + p_1^{k_1})(1 + p_2 + \dots + p_2^{k_2}) \dots (1 + p_n + \dots + p_n^{k_n}).$$

Task 5 (confutation of Fermat's hypothesis).

Find the least number t such that the number $2^{2^t} + 1$ is not prime.

Task 6 (confutation of Euler's hypothesis).

a) Find the least number t such that there exists a solution of the equation $x^4 + y^4 + z^4 = t^4$ in positive numbers.

b) Find the least number u such that there exists a solution of the equation $x^5 + y^5 + z^5 + t^5 = u^5$ in positive numbers.

To solve this task successfully the technique of omitting the inner cycle is applicable.

Task 7 (Ramanujan's problem).

Find the least number t such that there exists more than one solution of the equation $x^3 + y^3 = t$ in positive numbers.

Way 3: (the simplest example) calculate powers of natural numbers and combine them to obtain small numbers. Maybe, we have found the equality $984^2 + 2^{12} = 5 + 99^3$. Thus, we have obtained the following tasks:

Task 8. Find a solution of the equation in positive numbers: a) $x^2 + y^{12} = 5 + z^3$; b) $x^2 + y^{12} = u^2 + 1 + z^3$; c) $x^2 + y^{12} = u^2 + w^4 + z^3, \dots$

If the jury finds an effective algorithm to solve any of these tasks then this task may be proposed at an olympiad.

3. Geometry

Task 9 (classical). One can jump F feet forward and B feet backward along the straight line (one-dimensional grid). How many steps (at least) are necessary to reach the point located X feet from the initial one?

Idea 3. Certainly, if X is not divisible by $G := GCF(F, B)$ then the task has no solution. Otherwise, solve the Diophantine equation $F/G * U - B/G * V = 1$ in positive numbers (the most effective algorithm for this task is using continuous fractions). Then the answer is not greater than $X/G * (U + V)$.

A similar task for a two-dimensional grid is more interesting.

Task 10. A hare is at the origin of coordinates and can jump only to integer points (with integer coordinates). Each of its jumps is of length 5. How many jumps (at least) are necessary to reach a given point *(X, Y)*?

Idea 3. Make some attempts. Jumps *(3,4)* and $(3, -4)$ result *(6,0);* shift *(6,0)* and jump $(-5, 0)$ result shift *(1,0)* and the task is reduced to Task 9.

Many tasks can be put on configurations (see Gardner, 1988, chapter 22). To make tasks more determined, we will consider configurations on grids only.

Way 3. Choose any finite set on a two-dimensional grid. Some subsets of (more than 2) points lie along straight lines (rectilinear subsets). Take into account cardinal numbers of these subsets. We obtain the following task on affine configurations.

Task 11. Find any (or with the least cardinal number), (or of the least size) configuration having prescribed numbers of rectilinear subsets of prescribed cardinal numbers.

Way 3. Choose any finite set on a two-dimensional grid. Compute squares of distances between pairs of points within the set and detect numbers representing equal distances. We obtain the following task on metric configurations.

Task 12. Find any (or of the least size) configuration such that values of distances between its points form a set of a prescribed cardinal number and numbers of distances of same value are also given. (Values themselves are not given; otherwise any a priori upper estimation of size of the set can be derived).

Remark 3. Some tasks on configurations arise from arrangements of stars on the flag of the United States (see Gardner, 1988, chapter 22).

Remark 4. Triangular and hexagonal grids may also be involved.

The following unique task on affine configurations without given quantities was discovered by S.N. Alekseenko (Pankov *et al.*, 2005).

Task 13. Find any configuration (set) of points fulfilling the following conditions:

A1) If two segments with ends belonging to the set have only one mutual point then it belongs to the set too.

A2) No point can be added to the set, preserving the property A1).

4. Acceleration of Algorithms

Most of tasks listed above can be transformed to the following type: given a (slow) algorithm. Write a program yielding the same result in appropriate time. For instance, Task 6b:

$u := 0$; *equal := false*;
Repeat
$u := u + 1$;
for $x := 1$ *to* u *{ for* $y := 1$ *to* u *{ for* $z := 1$ *to* u *{ for* $t := 1$ *to* u
{if $x^5 + y^5 + z^5 + t^5 = u^5$ *then {equal := true;* $x_1 := x$*;* $y_1 := y$*;* $z_1 := z$*;* $t_1 := t$ *} }}}}*
until equal;
Output x_1, y_1, z_1, t_1, u.

5. Conclusion

We hope that successful application of methods proposed above would yield new tasks with "short and elegant formulation" (Dagiene *et al.*, 2007); solving of such tasks created by Way 3 would yield interesting "minimal" objects.

References

Dagiene, V., Skupiene, J. (2007). Contests in programming: quarter century of Lithuanian experience. *Olympiads in Informatics: Country Experiences and Developments*, 1, 37–49.

Gardner, M. (1988). *Time Travel and Other Mathematical Bewilderments*. W.H. Freeman and Company, New York.

Pankov, P.S., Alekseenko, S.N., Asanov, T.D. (2005). Interactive computer presentation of a configuration closed with respect to intersection of convex hulls. *Bulletin of the Kyrgyz-Russian Slavic University*, 5(7), 85–88 (in Russian).

P.S. Pankov (1950), doctor of physical-math. sciences, professor, corresponding member of Kyrgyzstani National Academy of Sciences (KR NAS), is the chairman of jury of Bishkek City OIs since 1985, of Republican 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 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 IOIs in forthcoming years. Graduated from the Kyrgyz-Russian Slavic University in 2007.

Olympiads in Informatics, 2012, Vol. 6, 115–132

Type of Questions – The Case of Computer Science

Noa RAGONIS
School of Education, Computer Science Department, Beit Berl College
Department of Education in Technology and Science
Technion – Israel Institute of Technology, Israel
e-mail: noarag@beitberl.ac.il

Abstract. In this paper, I explore and discuss the variety of types of questions that can be used by computer science educators in different teaching situations and processes: in classroom lessons, in the computer lab, as homework, or in tests. The use of various types of questions offers many advantages, both for learners and for the teaching process. Twelve different types of question are discussed. Each is presented by its classification title, a short description of the specific type of question, a concrete example or an example pattern, and a short pedagogical discussion that includes remarks on cognitive aspects. Three general types of questions are presented and discussed: combination questions, narrative questions, and closed questions. These types of questions relate to "programming-like" assignments since they are the most common subject encountered in the teaching of computer science (Java is used as the implementation language). However, as discussed here in brief, most of the question types are also suitable for most other contents in the teaching of computer science.

Key words: computer and information science education, algorithms, human factors, types of questions, cognition in computer science, problem solving.

1. Introduction

One of the traditional problem-solving scenarios in computer science starts with the presentation of an open problem described as a "story", continues with its analysis and planning of its solution, and ends with the presentation of a solution as an algorithm in either pseudo-code or in a specific programming language. It is important, however, that computer science educators be aware of the fact that there are many additional types of questions, each of which requires a variety of thinking processes that enable to use a wider spectrum of cognitive skills.

Questions types within the computer science (CS) discipline appear in textbooks and on teaching websites, and are less frequently discussed in the contents of question patterns used to broaden the spectrum of CS educators' teaching tools. This paper aims to present a collection of CS question types as a tool for CS educators in preparing teaching artifacts to be used in different teaching situations and processes: classroom lessons, the computer lab, homework, or tests. The focus is on pedagogical issues intended to expand the educators' perspective with respect to question design. CS educators at all teaching levels, middle schools, high schools and universities, can find this presentation helpful

when planning their teaching process. The presentation of the question types and pattern versions includes recommendations on the use of the questions while teaching.

Three main pedagogical targets can be achieved by integrating different types of questions in the teaching of computer science: (1) they illuminate different aspects of the learned content; (2) they require students to use different cognitive skills; and (3) they enable educators to vary the teaching tools they use. The design of questions that require different cognitive skills is significant, first, in order to present each individual cognitive skill and illuminate different aspects of the learned content; second, since different students express their knowledge differently and it is important to give each of them the opportunity to articulate that knowledge; and third, to develop and enrich the cognitive skills of all students. Furthermore, the use of combinations of different question types throughout the teaching process helps maintain the students' interest, attention, and curiosity.

The examples or example patterns used here to demonstrate each of the question types relate to "programming-like" assignments, since they are the most common subject encountered in the teaching of CS. Yet, most of the question types can be assimilating into any CS content, as will be discussed in Section 6 below, in the context of Automata Theory, for example.

2. Background

Most of the discussion on questions and cognition within the teaching process of CS relates to problem-solving strategies and illuminates strategies such as stepwise refinement (e.g., Batory, *et al.*, 2004; Vasconcelos, 2007), algorithmic patterns (e.g., Muller, *et al.*, 2007; Ginat 2009), roles of variables (e.g., Sajaniemi 2005; Sorva, *et al.*, 2007), or tracing strategies (e.g., Venables, *et al.*, 2009). Other discussions relate to strategies adopted by experts versus novices (e.g., Brand-Gruwel, *et al.*, 2005], and other aspects like misleading problem-solving passes such as the design-by-keyword syndrome (Ginat, 2003). I believe that using different types of questions consistently throughout the teaching process enhances learners' skills and abilities with respect to problem-solving processes, since they investigate different analysis tools and expand their ability to examine a problem from multiple points of views.

Another relatively broad discussion in the context of questions in the CS domain is about the evaluation of student artifacts and the progression of learning (e.g., Chamillard and Braun, 2000; Byckling and Sajaniemi, 2006), as well as automatic assessing systems (e.g., Spacco *et al.*, 2006; English and Rosenthal, 2009). In this paper, I will not address assessment aspects but rather the thought processes that arise when students actively solve different types of problems.

Cognition is the process of thought. Cognitive skills are basic mental abilities we all use to think, to study and to learn. Cognitive processes can be analyzed from different perspectives within different contexts. In psychology or philosophy, for example, the concept of cognition is closely related to abstract concepts such as mind, reasoning, perception,

intelligence, learning, and many others that describe the mind's capabilities. The field of cognition focuses on the study of specific mental processes, such as comprehension, inference, decision-making, planning, and learning. Specifically, with respect to CS, we are familiar with the advanced cognitive skills of abstraction, generalization, concretization and meta-reasoning. As mentioned above, it is important to develop the learners' cognitive skills. The use of a variety of question types is one way to achieve this goal. The literature on CS education reveals only a restricted offering of research work related to the cognition aspects of different types of questions. Some research has been done with the intention of adapting Bloom's Taxonomy to the domain of CS. Bloom's Taxonomy was first described in 1956 as a hierarchical model of the cognitive domain (Bloom *et al.*, 1956), followed by several significant changes made by Anderson *et al.* (2001). Thompson *et al.* (2008) reviewed the work done throughout the past decade so as to apply Bloom's taxonomy to CS course design, evaluation, and assessment, and to provide an interpretation of the taxonomy that can be applied to introductory programming exams. This interpretation focused on the cognitive skills involved in addressing several types of questions, whereas Jones *et al.* (2009) focused explicitly on written examinations. These researchers determined the difficulty level of each question in an examination paper based on the criteria of keyword/s found in the question, and presented a cross-analysis across student performance, cognitive skill requirements, and module learning outcomes. Different kinds of studies, that combined computer science questions and cognition, relate to different automata systems aimed at question-answering processes. Pomerantz (2002), for example, evaluated four taxonomies of question types to determine the expressiveness of each for questions received by digital reference services. He offered a faceted classification scheme that can be used as a basis for automating parts of a reference question-answering process. The work of Yang *et al.* (2008) is another example of automated question-generating systems. This research introduced a method of designing a test question database management system based on the three-tier B/S structure, analyzing the features of the test question database structure, and expatiating the grouping algorithmic calculation, focusing on the main control parameters of knowledge points and question-type data. The research results indicate that the application of the system not only improves the work efficiency of teachers, but also positively boosts the teaching reform. Those results support my point of view that emphasizing different types of questions improves the learning-teaching process.

The main aim of the current paper is to broaden CS educators' tools for designing different types of questions. The presentation of each question type focuses on possible ways of using them in teaching processes. Emphasize is on pedagogical approaches and the discussion involves informal cognition considerations.

3. Presentation Structure of Question Types

This paper focuses on question patterns and presents twelve types of questions. I suggest several fundamental types and describe several variations for each. Clearly, additional

types of questions, as well as combination of different types of questions, exist and can be developed and used according to need.

Each type of question is presented in the following structure: a title that reflects the ***classification*** of the question type; a short ***description*** of the specific type of question; a concrete ***example*** or ***general pattern*** that demonstrates the said type of question; different ***variations*** of the question type; and a short pedagogical and cognitive ***discussion*** about the said type of question. Since my purpose here is to present a variety of questions in the context of computer science education, most of the examples will be quite simple to solve. For each type of question, it is clearly possible to develop a range of questions on different complexity levels, from both the algorithmic and the cognitive points of view. In addition, within each type of question, additional variations may be presented that require different cognitive skills.

After presenting the twelve question types (Section 4), three global aspects of question types are presented (Section 5), and the assimilation of the different types of questions into different CS contents is discussed (Section 6).

Following are several remarks about the actual use of this collection of questions in the computer science class:

- There is no specific rule for the presentation order of the twelve questions types.
- No specific rule or guidelines can be formulated with respect to the order in which the different types of questions should be used. Each CS educator should select the appropriate type of question and its complexity level according to the specific characteristics of the learners in the specific class.
- The suggested types of questions sometimes overlap each other and cannot be separated completely from one another. This point is further addressed when relevant.
- A question can combine several types of questions in its different sub-paragraphs, as illustrated by an example given in the sub-section entitled "Combining several types of questions" in Section 5.
- Questions can literally be divided into two types: Pure algorithmic tasks and story-based algorithmic tasks (narrative). This perspective will be discussed in Section 5.

4. Types of Questions

Type 1: Developing a Solution

Description. A development question presents an open problem for which students are required to develop a solution in the form of a verbal algorithm, pseudo-code, or by using a specific programming language.

Example. Write a method that accepts an integer n as input and returns the number of (integer) divisors of n.

Variations. A development question can address (a) a sequence of instructions; (b) a single method (as in the above example); (c) a whole program; or (d) a method with a specific efficiency (in the above example this can be $O(\sqrt{n})$.

Discussion. This type of questions invites different solutions. In some cases, the differences are not meaningful; in others, the different solutions represent different algorithmic approaches. Variation (d) is not entirely an open question since a significant constraint must be met – the requested efficiency. In this case, students cannot just choose a solution to solve the problem and therefore, it is considered to be more difficult than the other variations and requires wider considerations.

Type 2: Developing a Solution that Uses a Given Module

Description. In this case, the development question relates to a predefined module. The student must present a solution to a given problem while considering and using a given module. Documentation of the module is included in the question and the student must use it in the developed solution.

Example. Write a method that accepts an integer value n as input and returns the integer between 1 and n with the largest number of divisors. Use the method *numberOfDividers(n)*, which accepts an integer value n as input and returns the number of its divisors.

Variations. A development question that relates to a given module can be presented, among other ways, in one of the following forms: (a) write an instruction that invokes a given method; (b) write a method that uses a given method (like in the given example); (c) write a method that uses a given module a specified number of times; or (d) write a method that uses several different given methods.

Discussion. The fact that students must relate to a given module influences the development process of the solution. For example, in the case of a given method the student must match the developed method to a specific sub-task that the given method implements. This type of questions is considered more difficult than Type 1 questions because students must satisfy a constraint – the use of the given sub-task. The given module should not be a method but rather it can be, for example, a specified data structure (like Linked list) or class.

Type 3: Tracing a Given Solution

Description. A given code is presented and the students are asked to follow the code execution.

Example pattern. Present a tracing table that follows the execution of a given method. The table should include a column for each variable and for the code output.

Variations. A tracing question can involve following, for example on: (a) a complete program; (b) a single method; (c) a recursive method; or (d) the creation of objects. In addition, the following instructions can be used in each of the above variations: (1) follow the code execution with a given specific input; (2) follow the code execution with the student choosing the input; (3) follow the code execution with different specified inputs that are selected so as to guide the student to reveal what the given code does; (4) find different sets of inputs so that each set represents a different sequence by which the code is executed; or (5) find a set of inputs that yields a specific output.

Discussion. Variations 1–3 of the instructions can be considered to be closed questions. The student is required to trace a given code with a specified (given or self-chosen) input, and there is only one correct solution. Instructions (4) and (5) require the students to involve additional, deeper considerations and to examine the presented code more closely. It is not sufficient to understand different instructions; rather, it requires code analysis – what is the purpose of the code and how is it achieved. Clearly, more advance cognitive skills are needed in order to address these instructions in a meaningful manner.

Type 4: Code Execution Analysis

Description. A given code is presented and the student is asked to analyze some aspects of the code execution. This kind of questions is more complicated than tracing a given code since it requires the student to analyze code execution.

Example pattern. The following code includes a loop. Examine the code and answer the following questions:

(i) For what values of x and y will the loop not be executed at all?
(ii) For what values of x and y will the loop be executed exactly once?
(iii) For what values of x and y will the loop never end?

Discussion. This type of questions requires the student to understand the given code as a whole. Therefore, a higher level of thinking is needed in order to solve such questions than that needed to solve a tracing question. "Code execution analysis" questions relate mainly to two cognitive skills: (1) understanding programming structures; and (2) understanding the logic of a given code. Note that instructions (4) or (5) presented in the discussion of Type 3 questions – Tracing a given solution – can also be viewed as code execution analysis tasks.

Type 5: Finding the Purpose of a Given Solution

Description. A given code or algorithm to an unknown problem is presented and the student is asked to state the purpose of the solution – to determine what problem it solves.

Example pattern. Examine the given method and find the target of the method, that is, what is the problem that the method solves?

Variations. A "Finding the purpose of a given solution" question can relate to either: (a) a sequence of instructions; (b) a single method (like in the above example pattern); (c) a full program; or (d) a class of objects.

Discussion. Solving this type of questions requires a set of cognitive skills. In addition to an understanding of the code execution and the ability to trace it, a unique understanding and a unique skill, are required. These questions are considered harder than developing a solution for the same problem that the code solves. One reason for this is the need to comprehend someone else's way of thinking. To help students and guide them in solving this type of question, questions can contain scaffolding sub-questions. For example, a question can include several "Trace a given solution" sub-questions in the form of the instructions presented in the discussion of Type 3 questions, aimed at guiding the students to discover what the purpose of the code is.

Type 6: Examining the Correctness of a Given Solution

Description. A given problem and its solution are presented. The student is asked to determine whether the given solution is a correct solution to solve the given problem.

Example. The following method was written by a student as a solution for the following problem: Write a method that accepts an array of integers as input and returns *true* if all of the array's values are identical or *false* if they are not. Is the method correct?

```
public static boolean equalsValues (int[] arr) {
    for (int i = 0; i <arr.length; i = i + 2) {
        if (arr[i] != arr[i + 1])
            return false;
    }
  return true;
}
```

Variations. This type of question can be presented for different types of tasks: (a) state whether a given solution to a given problem is correct (as in the above example); (b) check whether a given solution is correct and explain your answer; (c) if the given solution is incorrect, give an example of an input that demonstrates its incorrectness; (d) if the given solution is incorrect, give an example of an input that will lead to a correct output, which could then lead to the conclusion that the given solution is correct; (e) if the given solution is incorrect, correct the solution by implementing the minimal required changes (without the "minimal" restriction, students may present a totally different solution); or (f) the given solution may contain more than one mistake, and the question may or may not state that explicitly.

Discussion. In order to solve this type of question, students should apply logic algorithmic thinking skills. Here, as in Type 5 questions, students must analyze a solution that may not correspond with their own way of thinking had they been asked to suggest a solution. However, since the purpose of the solution is given, these tasks are considered to be easier than Type 5 questions.

In the given example, the two minimal required corrections are: (i) change the increment of variable i to 1 (instead of 2); and (ii) change the range of variable i to $i <$ *arr.length* -1. Correction (i) is based on a logical consideration of the solution, while correction (ii) involves addressing the array index, which is a more technical consideration.

Additional variations of correctness questions may address syntactic mistakes. Such questions should be posed while introducing new instructions or structures. I do not recommend, however, using them at more advanced stages since they do not reflect an understanding of the algorithmic problem, and the compiler in fact directs the debugging of such mistakes.

Type 7: Completing a Given Solution

Description. A given problem and an incomplete solution of it are presented; some of the solution instructions are missing. The students are asked to complete the missing instructions so that the solution actually will solve the problem.

Example. The following method was written by a student as a solution for the following problem: Write a method that accepts an array of integers as input and returns the number of array elements that are greater than their two neighbors (the previous element and the subsequent element in the array).

```
public static int numberOfBiggers (int[] arr) {
    _____ ;
    for (int i =_____ ; i<_____ ; i++) {
        if ( _____ )
            _____ ;
    }
    return _____ ;
}
```

Variations. A "Completing a given solution" question can vary in the extent to which instructions are missing. The number of missing instructions should be decided on, taking into consideration the effect on the question's difficulty and complexity. For example, if the objective is to focus on the use of a Boolean flag, the missing instructions should only be those that relate to the flag; or, if the loop limits are the target, the limits should be missing and perhaps the increase in the loop control variable as well. The extent of missing instructions is quite significant in the above example.

Discussion. This type of question also requires students to understand the logic of the given solution. Question difficulty is determined according to the students' level and stage of learning although for each subject there is a relatively simple completion of a given solution for which the logic of the solution is straightforward and not so difficult to understand. Still, other, more challenging questions exist and instructors should be aware of this potential complexity. For example, asking students to complete instructions for a given bubble sort algorithm with missing meaningful instructions without introducing the rational of this sorting approach, is considered a difficult question.

In general, the missing instructions can relate to one or more aspects of the algorithm and the instructor should consider whether to focus on one or more aspects. In the above example, the missing instructions involve three aspects of the algorithm: the counter control (initializing, increasing, and returning); the range of the loop (the first and the last array elements should not be accessed in the loop because they do not have two neighbors); and the specific condition to be checked.

Type 8: Instruction Manipulations

Description. A problem and its solution are given. Students are asked to address different manipulations performed on the solution.

Example. The following method is a version of a selection sort solution.

```
public static void selectionSort (int[] arr) {
    int p, temp;
(1) for (int i = 0; i < arr.length −1; i + +) {
(2)     p = i;
(3)     for (int j = i + 1; j <arr.length; j + +) {
            if (arr[j] < arr[p])
                p = j;
        }
(4)     if (p! = i) {
            temp = arr[i];
            arr[i] = arr[p];
            arr[p] = temp;
        }
    }
    }
```

Answer the following questions and explain your answers:

(i) Will the algorithm correctness be affected if the instruction in line (2) is removed and the instruction in line (3) is replaced by the following instruction:

(3) `for (int j = i; j <arr.length; j + +) {`

(ii) Will the algorithm correctness be affected if the instruction in line (4) is removed and the contents of the two array elements are exchanged anyway?

(iii) Will the algorithm correctness be affected if the entire body of the loop (1) (lines 2–4) is replaced by the following instructions?

```
for (int j = i + 1; j <arr.length; j + +) {
    if (arr[j] < arr[i]) {
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```

Variations. A "Manipulation of instructions" question can involve: (a) adding instructions; (b) removing instructions; (c) changing instructions; or (d) replacing instructions. The question can address the target of a specific code or the tracing of the changed code, or it can examine differences between outputs.

Discussion. Questions that manipulate an existing solution enable students to focus on the more meaningful aspects of the algorithm. Course instructors can then lead a discussion of such manipulations in order to clarify the essence of a given solution, as well as other computer science topics. For instance, a discussion about a change made to instructions can also highlight the concept of generalization. Students can be instructed to make a slight change to a given method so as to generalize the original method and solve a broader task. For example, a method that sorts array can be changed slightly so that it

sorts a section of the array between two given indexes. After the change, the method is more general and can sort different sections of an array, as well as the entire array (with the indexes 0 and the array length -1).

Type 9: Efficiency Estimation

Description. Students are required to estimate the efficiency of a given solution.

Example pattern. Estimate the efficiency of a given method in terms of Big-O notation. Explain your analysis.

Variations. This type of questions can be presented on different levels of cognitive complexity: (a) an example pattern that enables an early discussion on efficiency is: Focus on the loop in a given method, how many times is it executed? The following variations are more complicated: (b) estimate the efficiency of a specific method (as in the example pattern); (c) estimate the efficiency of a method that invokes another method, taking into account the efficiency of the invoked method; (d) compare the efficiency of different methods that solves the same task; (e) estimate the efficiency of a recursive method; or (f) develop a solution with a required specific efficiency.

Discussion. Questions that require students to relate to efficiency can be integrated quite early on in the teaching and learning processes. Instructors should not wait till they teach complicated algorithms in order to teach the concept of efficiency. Questions, such as that presented in variation (a), demonstrates the basic idea of the efficiency concept, which, in general, is considered to be abstract and difficult to understand.

Note that variation (f) is actually a development question (Type 1) with an efficiency restriction that requires a combination of cognitive skills. Students should not be satisfied with finding an algorithmic idea that solves the given problem; rather, they should estimate its efficiency and if it does not satisfy the restriction mentioned in the question, they should seek a different solution.

Type 10: Question Design

Description. Students are asked to design a question on their own.

Example 1. Design a question that checks the understanding of the sort-merge algorithm.

Students' answers to this question (which are questions) can be based, for example, on tracing regular and/or extreme cases.

Example 2. Design a question whose solution requires the use of a method that finds the most frequent value in an array.

An example of such a question is: Write a method that for each of a school's 10th grade classes, accepts student grades on a computer science test as input and returns the frequent grade in each class.

Variations. A design question can relate, among other things, to: (a) clauses in a given question. Students are asked to compose additional clauses for a given question that, in their opinion, help clarify some extreme case; (b) a question whose solution requires the use of a given method (see above Example 2); (c) a question that checks the understanding

of a specific concept or algorithmic idea (see above Example 1); or (d) an entire test or worksheet that examines a specific learning unit.

Discussion. This type of question changes the learners' point of view. In addition to the experience gained by examining a question from an instructor's point of view, it encourages the students to think about the learning concepts. It leads learners to reflect on what was learnt, the main involved concepts, the sub-contents of the concepts, and furthermore, to think about checking their own understanding. In addition, the design of questions is a kind of active learning that encourages creativity.

Type 11: Programming Style Questions

Description. Students are asked to examine the programming style of different solutions proposed for the same task.

Example pattern. Look at the different, correct solutions for a given problem. Examine the solutions and state, in your opinion, which is the best solution. Explain your choice.

Variations. The different solutions given for the problem should differ in one aspect only, according to the instructor's decision, such as: (a) different kinds of loops; (b) the need to use an array for the solution; or (c) different algorithmic approaches (for example: giving two correct solutions to a problem and asking the students which of them is "nicer" and why). If the instructor decides to integrate several aspects into the question, the different aspects can be presented explicitly in the question and the students can be asked to address the solutions according to each aspect. Alternatively, the students can be asked, first to address the different aspects and then discuss the solutions themselves.

Discussion. This type of question enables to discuss various aspects of programming styles, while comparing different solutions. The different aspects can be for example: modularity, complexity, programming style, readability, or memory uses. Such a discussion can increase the abstraction level of student thinking.

Type 12: Transforming a Solution from One Representation to Another

Description. A problem and its solution are presented to the students in a specific syntax or paradigm, and the students must transform the solution into a different syntax or paradigm.

Example pattern 1. The given loop is implemented using a *while* loop structure. Transform the loop so that it is implemented by a *for* loop structure.

Example pattern 2. The given method is implemented using a *while* loop structure. Transform the method into a recursive method that achieves the same target.

Example pattern 3. The following method sorts an array of integers according to the imperative approach implemented in Java. Transform the method so as to reflect the functional approach and implement it in Scheme.

Variations. The different representations can be: (a) between programming paradigms (as in Example pattern 3); (b) within the same programming paradigm but between programming languages; (c) within the same programming language but between structures

(as in Example pattern 2 or, for example, the transformation from a nested if statement to a switch-case statement); or (d) within the same programming language but between different algorithmic approaches (as in Example pattern 2).

Discussion. The focus of this kind of question should be placed on qualitative aspects rather than syntactical aspects, where qualitative aspects mean, for example, problem analysis according to two different programming paradigms or the transformation of an imperative solution into a recursion solution in the same programming language. Such tasks require abstract thinking processes.

Similar to the "programming style questions" (Type 11), qualitative transformation questions enable to concentrate on core computer science concepts. Such questions lead students to explore different ways of thinking in problem-solving situations. Since this kind of question demands a high level of abstraction, it is not necessary suitable for all students. Obviously, transformations between two programming paradigms can be carried out only after the two said programming paradigms have been learned.

It is my opinion that transformations that involve only syntactic issues, for example, from pseudo-code to any formal language, do not involve problem-solving skills. Such tasks may be required in order to practice ways of writing, but they do not involve meaningful CS concepts.

5. Global Aspects of Question Types

In this section, I present three additional approaches to questions. The first involves combining several types of questions, the second refers to story questions, and the third to closed questions. The three approaches can be applied to most of the twelve question types presented above.

Combining Several Types of Questions

Despite the attempt to uniquely classify computer science questions in the above list of question categories, in most cases questions are a combination of several types, as the following example illustrates (while others may not be classifiable by the presented collection at all).

Example. The target of Methods A and B presented in Fig. 1 is to determine whether or not an integer n is a prime number.

Following is a list of questions that can be asked separately or in any combination according to the pedagogical purposes of the instructor.

(i) Check the correctness of the solutions. Do they solve the problem?
Type6: correctness.

(ii) What is the purpose of each method? (in case the problem is not indicated).
Type5: find the purpose.

(iii) Trace each method given $n = 19$.
Type3: trace.

```
//Version A
    public static boolean prime (int n) {
        for (int i = 2; i < n; i++) {
            if (n%i == 0)
                return false;
        }
        return true;
    }
-------------------------------------------------
// Version B
    public static boolean prime (int n) {
        if (n%2 == 0)
            return false;
        for (int i = 3; i < n; i = i + 2) {
            if (n % i == 0)
                return false;
        }
        return true;
    }
```

Fig. 1. Combining several types of questions.

(iv) For each method, determine how many times the loop is executed for $n = 19$.
Type4: code execution analysis.

(v) Find a value of n for which the loop in Version B is executed 10 times. Is there only one answer?
Type4: code execution analysis.

(vi) What is the efficiency of each of the two methods?
Type9: efficiency.

(vii) Is the solution still correct if you change the upper loop limit in Version B to n/2 instead of n? If it is, what is the method efficiency after the change?
Type8: manipulation; Type6: correctness; Type9: efficiency.

(viii) Is the solution still correct if you change the loop limit in Version B to $\sqrt{n}$ instead of n? If it is, what is the method efficiency after the change?
Type8: manipulation; Type6: correctness; Type9: efficiency.

Story Questions

Questions can literally be divided into two types: pure-algorithmic tasks and narrative-algorithmic tasks. ***Pure-algorithmic*** tasks are problems that directly and explicitly address program structures and program variables, and present the task in that context. ***Narrative-algorithmic*** tasks are problems that do not directly address either the required program structures or the required program variables; the problem to be solved is embedded in a story and in order to solve it, learners must recognize both what is given and

Table 1

Tasks presented as pure-questions and as narrative-questions

Task	Pure question	Narrative question
Find the maximum of a list of numbers.	Write a method that accepts a list of integers as input and returns the maximum value of the list.	During a sports day, each of the 30 students in 5 classes participated in two competitions, the high jump and the long jump. Write a program that inputs, for each class, each student's two results, and outputs the best high jump result and the best long jump result for each class.
Check whether a given array is sorted.	Write a Boolean method that accepts an array as a parameter and returns a Boolean value if the array is up-sorted.	A teacher wishes to encourage her or his students, and so gives them special certificates if their test scores improve. Write a method that accepts the list of each student's grades as input and determines whether he or she deserves a certificate of recognition.
Exchange characters with their successive characters according to the Unicode table.	Write a method that accepts an array of characters as a parameter and changes the array so each character is replaced by its successive character according to the Unicode table.	A message that is to be sent between financial partners must be encoded. The message includes words, spaces, and periods. Write a method that accepts a string with a message as a parameter, and returns a coded message in which each letter is replaced by its successive letter according to the ABC. The letter Z is to be replaced by the letter A. Spaces and periods remain unchanged.

what the target of the problem is. Specifically, learners should decide which elements are relevant to solving the problem and which are irrelevant. Most of the examples presented in the list of question types are pure examples in which the task is presented directly. Table 1 presents several tasks as both pure-questions and as narrative-questions.

It is important that CS educators are aware of the differences between the problem-solving skills required to solve pure-questions as opposed to narrative-questions. A pure-algorithm question directs the learner to the core of the task; in narrative question, on the other hand, students must reveal the task and determine what the specific assignments are. Since in the real world, most problems are based on narratives, solving this type of questions is an important skill that computer science students should acquire. Still, these questions are usually more complicated.

When teaching new computer science content, I recommend that several stages be followed in which questions of the two types are addressed. First, present a story that embeds the new learned topic so that the class grasps the essence and target of the new topic, which will enhance their programming tool arsenal. Second, focus for a while on pure-questions to enable a gradual knowledge construction process of the new tool or structure in the context of programming. Finally, integrate narrative questions into the subsequent stages of the teaching process.

Closed Questions

The common concept of ***closed questions*** refers to questions that present a list of possible answers of which the learner is expected to choose and mark one. The most frequently encountered types of closed questions are multiple choice questions and true/false questions. In fact, what really is "closed" are the answers, not the questions. The twelve question types presented above could be discussed in terms of questions: (a) that can be presented naturally only as open questions, where learners must present their own answers; or (b) that can be presented naturally as either open questions or closed questions, where the learners must choose and mark an answer from a set of given answers.

In what follows, the 12 types of questions is divided into groups with relation to the option to be presented as closed questions or not.

- *Types of questions that can be presented as closed questions*
  The types of questions that can be presented naturally as closed questions are: Type3 – Tracing a given solution; Type4 – Code execution analysis; Type5 – Finding the purpose of a given solution; Type6 – Examining the correctness of a given solution; and Type9 – Efficiency estimation.
  Example. A closed question of Type6 can give a list of methods that aim to solve the same task. The learner is asked to indicate for each such method whether or not it is correct.
- *Types of questions that can not be presented as closed questions*
  The types of questions that can not be presented naturally as closed questions are: Type1 – Developing a solution; Type2 – Developing a solution that uses a given module; Type10 – Question design; Type12 – Transforming a solution from one representation to another.
  These types of questions obviously require that the learner develops a solution that satisfies the instructions.
- *Types of questions that can not be presented naturally as closed questions*
  The remaining types of questions can be presented as closed questions but this is not their natural form: Type7 – Completing a given solution; Type8 – Instruction manipulations; Type11 – Programming style questions.
  Example. A closed question of Type7 can give a list of optional instructions to be added to a given code that solves a given task in a specific place. Students are asked to indicate which of them is suitable to be added.

6. Applying the Different Question Types to Different CS Contents

As stated above, the question types presented in this paper relate to programming-like questions; still, most of these types can be implemented on other CS contents as well. Table 2 displays specific variations of the twelve question types as could be implemented, for instance, in the field of Automata Theory.

Table 2

Question types and representative examples in automata theory

Type of question	Example pattern
Type 1: Developing a solution	Design a finite automaton A that recognizes regular language L.
Type 2: Developing a solution that uses a given module	Given finite automaton A_1 that recognizes language L_1 and finite automaton A_2 that recognizes language L_2, design a finite automaton that recognizes the language $L_1 \cup L_2$.
Type 3: Tracing a given solution	Given a pushdown automaton P and the word w, show the sequence of states that P goes through when processing w.
Type 4: Code execution analysis	Given a finite automaton A, find: – a word whose processing will terminate in an acceptable (final) state; – a word whose processing will terminate in an unacceptable (not final) state; – a word whose processing will terminate in the trap state.
Type 5: Finding the purpose of a given solution	Given a Turing machine *T,* determine what language it processes.
Type 6: Examining the correctness of a given solution	Does Turing machine T recognize language L?
Type 7: Completing a given solution	Complete the pushdown automaton P so it recognizes language L.
Type 8: Manipulating instructions	Given a Turing machine T, what language does the machine recognize if the path from state q_1 to state q_2 is replaced by the following given path?
Type 9: Estimating "efficiency"	Given a finite automaton A that recognizes language L, find a different finite automaton that recognizes the same language with fewer states.
Type 10: Designing a question	Design a question that requires the presentation of a BNF grammar for an irregular language.
Type 11: "Programming" style questions	Given three different pushdown automatons that recognize language L, examine the automatons and state which of them, in your opinion, is more "qualified".
Type 12: Transforming a solution from one representation to another	Given a Turing machine T, present a BNF grammar that expands the same language.

7. Summary

This paper explores and discusses variety of types of questions that can be used by computer science educators in different teaching situations. The paper illuminates the important role computer science educators have in introducing their students to different types of questions that can be used alongside learning-teaching processes. The exploration of different types of questions and their variations deepens students' understanding of the learnt computer science concepts and help them acquire a variety of cognitive skills. It also provides different learners with the opportunity to express their knowledge, which is

elicited by different sorts of questions, and to refine their understanding of complex concepts. Furthermore, the use of a variety of question types provides intellectual challenges and maintains learners' concentration, interest, and motivation.

Further work can be done to analyze the types of questions identified according to known taxonomies.

Acknowledgments. Many thanks to Prof. Orit Hazzan and Dr. Tami Lapidot, my colleagues at the Technion – Israel Institute of Technology, for their helpful review and for encouraging me to publish this work.

References

Anderson, L.W., Krathwohl, D.R., Airasian, P.W., Cruikshank, K.A., Mayer, R.E., Pintrich, P.R., Raths, J., WITTROCK, M.C. (Eds.) (2001). *A Taxonomy for Learning and Teaching and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Addison Wesley Longman.

Batory, D., Sarvela, J.N., Rauschmayer, A. (2004). Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, 30(6), 355–371.

Bloom, B.S., Engelhart, M.D., Furst, E.J., Hill, W.H., Krathwohl, D.R. (1956). *Taxonomy of Educational Objectives Handbook 1: Cognitive Domain*. London, Longman Group Ltd.

Brand-Gruwel, S., Wopereis, I., Vermetten, Y. (2005). Information problem solving by experts and novices: analysis of a complex cognitive skill. *Computers in Human Behavior*, 21(3), 487–508.

Byckling, P., Sajaniemi, J. (2006). A role-based analysis model for the evaluation of novices' programming knowledge development. In: *Proceedings of the Second International Workshop on Computing Education Research (ICER'06)*, Canterbury, United Kingdom, 85–96.

Chamillard, A.T., Braun, K.A. (2000). Evaluating programming ability in an introductory computer science course. In: *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education (SIGCSE '00)*, Austin, Texas, United States, 212–216.

English, J., Rosenthal, T. (2009). Evaluating students' programs using automated assessment: a case study. In: *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '09)*, Paris, France, 371–371.

Ginat, D. (2003). The novice programmers' syndrome of design-by-keyword. In: *Proceedings of the 8th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '03)*, Thessaloniki, Greece, 154–157.

Ginat, D. (2009). Interleaved pattern composition and scaffolded learning. In: *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '09)*, Paris, France, 109–113.

Jones, K.O., Harland, J., Reid, J.M., Bartlett, R. (2009). Relationship between examination questions and Bloom's taxonomy. In: *Proceedings of the 39th IEEE International Conference on Frontiers in Education Conference*, San Antonio, Texas, USA, IEEE Press, Piscataway, NJ, 1314–1319.

Muller, O., Ginat, D., Haberman, B. (2007). Pattern-oriented instruction and its influence on problem decomposition and solution construction. *ACM SIGCSE Bulletin*, 39(3), 151–155.

Pomerantz, J. (2002). Question types in digital reference: an evaluation of question taxonomies. In: *Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL '02)*, Portland, Oregon, USA, ACM, New York, NY, 404–404.

Sajaniemi, J. (2005). Roles of variables and learning to program. In: Jimoyiannis, A. (Ed.), *Proceedings of the 3rd Panhellenic Conference "Didactics of Informatics"*, , University of Peloponnese, Korinthos, Greece. Available at: `http://cs.joensuu.fi/~saja/var_roles/abstracts/didinf05.pdf` [Last access at March 29, 2012].

Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J.K., Padua-Perez, N., (2006). Experiences with marmoset: designing and using an advanced submission and testing system for programming courses. In: *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITICSE '06)*, Bologna, Italy, ACM, New York, NY, 13–17.

Sorva, J., Karavirta V., Korhonen, A. (2007). Roles of variables in teaching. *Journal of Information Technology Education*, 6, 407–423.

Thompson, E., Luxton-Reilly, A., Whalley, J., HU, M., Robbins, P., (2008). Bloom's taxonomy for CS assessment. In: *Proc. Tenth Australasian Computing Education Conference (ACE 2008)*, Wollongong, NSW, Australia. CRPIT, 78. In: Simon and Hamilton, M. (Eds.), ACS, 155–162.

Vasconcelos, J. (2007). Basic strategy for algorithmic problem solving. Retrieved from: `http://www.cs.jhu.edu/~jorgev/cs106/ProblemSolving.html` [Last access at June 2, 2010].

Venables, A., Tan, G., Lister, R., (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. In: *Proceedings of the Fifth International Workshop on Computing Education Research Workshop (ICER '09)*, Berkeley, CA, USA, ACM, New York, NY, 117–128.

Yng, A., Wu, J., Wang, L. (2008). Research and design of test question database management system based on the three-tier structure. *WTOS*, 7(12), 1473–1483.

N. Ragonis is the chair of the Curriculum Committee and academic advisor, School of Education, Beit Berl College. She is a lecturer in the Department of Computer Science, and served for ten years as the head of the Department. Teaches courses related to computer science (e.g., OOP, graph theory, computational models), to the didactics of computer science and to information technologies such teaching and learning in online environments and query learning with spreadsheets. She is also adjacent senior lecturer, Department of Education in Technology and Science, Technion – Israel Institute of Technology. Her activities in the past twenty years concerns: educational research mainly focused on cognitive aspects of teaching and learning of computer science and integrating tutoring activities in teachers education; in-service teachers training; development of high school text books as well as teachers guides; and serve as a member of the management staff of "Machsava" (Thought), the Israeli National Center for High School Computer Science Teachers, Technion – Israel Institute of Technology and Weizmann Institute of Science.

Olympiads in Informatics, 2012, Vol. 6, 133–147

REPORTS

An Experience on the Organization of the First Spanish Parallel Programming Contest

Francisco ALMEIDA[1], Vicente BLANCO PÉREZ[1], Javier CUENCA[2], Ricardo FERNÁNDEZ-PASCUAL[2], Ginés GARCÍA-MATEOS[3], Domingo GIMÉNEZ[3], José GUILLÉN[4], Juan Alejandro PALOMINO BENITO[4], María-Eugenia REQUENA[4], José RANILLA[5]

[1]*Departamento de Estadística, I.O y Computación, Universidad de La Laguna 38201 Tenerife, Spain*
[2]*Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia 30071 Murcia, Spain*
[3]*Departamento de Informática y Sistemas, Universidad de Murcia Campus de Espinardo, 30071 Murcia, Spain*
[4]*Centro de Supercomputación, Fundación Parque Científico, Ctra. Madrid km. 388, Complejo Espinardo, 30100 Murcia, Spain*
[5]*Departamento de Informática, Universidad de Oviedo Campus de Viesques, 33204 Gijón, Spain*
e-mail: {falmeida,vicente.blanco}@ull.es, jcuenca@um.es, r.fernandez@ditec.um.es, {ginesgm,domingo}@um.es, {jguillen,jpalomino,mrequena}@parquecientificomurcia.es, ranilla@uniovi.es

Abstract. The first Spanish Parallel Programming Contest was organized in September 2011 within the *Jornadas de Paralelismo*, in La Laguna, Spain. The aim of the contest is to disseminate parallelism among the participants and Computer Science students who can use the material generated in the contest for educational purposes. The contest is similar to other sequential and parallel programming contests in which teams participate by solving a set of problems in a given time. But the Spanish contest has characteristics which distinguish it from other contests: an automatic tool (Mooshak) is used to validate the solutions and the tool has been modified to send the solutions to a cluster of nodes and to obtain the classification based on the speed-ups achieved; candidates can participate both in situ and online; a classification with the records for each problem is maintained on the web page of the contest, with explanations and codes of the record solutions so that the page can be used for educational purposes. This paper summarizes the experience and perspectives of the contest.

Key words: programming contests, online judge, parallel programming.

1. Introduction

With the evolution of technology parallel computing is becoming increasingly popular. At present, the basic computing systems are parallel (laptops and desktops are dual, quadcore or even hexacore, possibly with hyperthreading and programmable graphics cards) and so are clusters and supercomputers, which are composed of multicore nodes. This situation has led to a continuous increase in parallel computing courses in computing curricula (Meder *et al.*, 2008), and to some initiatives of the IEEE Technical Committee on Parallel Processing (IEEE TCPP, 2011), such as a proposal of parallel computing curriculum with a number of topics to be adopted in Computer Science studies.

In this context, the first Spanish Parallel Programming Contest (SPPC) was organized in September 2011 in the *Jornadas de Paralelismo* (JP, 2011). Our goal is to spread parallelism and at the same time that the material generated in the contest can be used for educational purpose in parallel computing courses.

Computing competitions can be organized in different ways, in a variety of fields and with different goals (Dagienė, 2010; Hakulinen, 2011). In particular, programming contests are useful for increasing the students interest for programming and for enhancing their programming abilities. So, there are a number of Computer Olympiads all over the world, with the main events being the International Olympiad in Informatics (IOI, 2011) and the ACM Programming Contest (ACM ICPC). Furthermore, contests are successfully used for educational purposes in programming courses (Fernández Alemán, 2011; García-Mateos and Alemán, 2009; Lawrence, 2004).

There are also a number of tools for the organization of programming contests (Kolstad, 2009). Some of them maintain a set of problems to practice with (UVa Online Judge) or allow the creation of contests (Leal and Silva, 2003).

Associated to the increasing popularity of parallelism, a number of competitions devoted to the topic have emerged in the last years, for example the Student Cluster Competition (SCC) and the Marathon of Parallel Programming (MPP). The Spanish Parallel Programming Contest follows this line, but it has some distinguishing features, which we explain below:

- The teams can participate in two ways (in situ and online) and there are two different classifications. This allows the teams to participate without needing to physically attend the *Jornadas de Paralelismo*.
- The tool Mooshak (Leal and Silva, 2003) has been modified to send the solutions provided by the teams to a cluster of four nodes, each with eight cores, and to obtain the classification considering the speed-ups achieved.
- A classification of records for all the problems of the different editions is maintained on the web of the contest (SPPC), with the explanation and code of the best solutions. In this way the page can serve as an educational resource for parallel programming courses.

The rest of the paper is organized as follows. Section 2 details the general organization of the contest. The problems in the first edition are commented on in Section 3. Section 4 describes how the first edition of the contest ran. Finally, future perspectives are discussed in Section 5.

2. General Organization

The contest has some organization aspects similar to those of other sequential and parallel programming contests:

- As in other contests, teams of three students and a teacher who acts as a coach are encouraged to participate. In the ACM Programming Contest (ACM ICPC) and the Marathon of Parallel Programming (MPP) the teams are of three components, and in the Student Cluster Competition (SCC), with a longer running time, the teams have six components and are supported by a computing company.
- As in the ACM competition and the Marathon, the Spanish contest lasts a short time (four hours), and the teams must solve a number of programming problems in that time.
- In our case (and also in the Marathon), a sequential solution of each problem is provided, and the teams generate parallel solutions with the aim of reducing the execution time of the corresponding sequential solution. The programs are developed in C, and the parallel environments OpenMP (Chandra *et al.*, 2001) and MPI (Snir and Gropp, 1998) can be used to develop the corresponding shared-memory and message-passing versions. MPI and OpenMP can also be combined to further reduce the execution time using hybrid parallelism.
- The problems and the sequential solutions are selected to cover different algorithmic paradigms and a variety of computational costs. The task selection is explained in the next section.

The main difference of the Spanish contest with respect to others is the evaluation system, which is done automatically and in real time, and allows in situ and online participation.

The cluster Arabí of the Supercomputing Centre of the Scientific Park Foundation of Murcia (SCC) is used for the evaluation of the solutions. The cluster comprises 102 nodes, each with eight cores, and four nodes (a total of 32 cores) are used in the contest. The SCC facilitates the occasional use of this subcluster for training, and it is used in the contest for the preparation and evaluation of the solutions provided to the teams, for a warm-up session and for the celebration of the contest. A job queue system is used to ensure only one program runs in the subcluster at a particular moment, which is necessary for the calculation of speed-ups and the classification.

The tool Mooshak (Leal and Silva, 2003) is installed in a virtual machine from which the solutions generated by the teams are sent to the queue in the cluster. It has been necessary to make some modifications in Mooshak to adapt it to the characteristics of the contest: Mooshak is used in conjunction with the subcluster of Arabí, and a new form of obtaining the classification based on speed-ups has been added to Mooshak.

The teams send their solutions to Mooshak, and it connects to a host associated to Arabí, where the programs are compiled and sent to the queue, from where the jobs are sent to the part of the subcluster specified in the submitted job. The solution is validated in the host by comparing it with the solution given by the sequential program provided by the organization. Finally, the host sends back to the Mooshak acknowledgement of the

correctness of the solution. In case of error, extra information is provided (compilation or execution error, too many resources required, etc).

The classification is computed based on the speed-ups: the execution time of the sequential program divided by that of the parallel program ($S_p = t_s/t_p$) (Grama *et al.*, 2003). In our case t_s is the time obtained for the test input with the sequential solution provided by the organization, and t_p the execution time for the same input with the program sent by the contestants. In that way the speed-up would be one for solutions which do not improve the sequential program. For each problem, the mark assigned to a correct solution could be $\max\{S_p - 1, 0\}$, so that solutions which do not reduce the sequential execution time have no positive mark, and positive marks begin from zero. The teams can send a maximum of ten solutions to each problem without penalization. After that, each additional submission means one point penalization, and the mark for the problem is $\max\{\max\{S_p\} - 1 - \max\{s - 10, 0\}, 0\}$, where s stands for the number of submissions for that problem and $\max\{S_p\}$ represents the maximum speed-up achieved in the s submissions. For a problem for which some team has a mark higher than 15, the marks of each team are linearly scaled so that the maximum mark is 15. This is to avoid very high marks (which could be obtained with an efficient use of the system combined with an improvement of the sequential solution) and to avoid some problems having an excessive weighting in the final score (problems with different algorithmic complexity have different parallelisation complexity). The final score for each team is obtained by adding up the marks in the problems for which they have provided some valid solution.

For each problem a brief description of the problem together with an example input and an execution scheme and the sequential solution are provided. The input provided is similar in number and form of the entries and in the execution time to that of the input used for automatic validation and scoring. So, the contestants can use this entry to evaluate their solutions in their laptop or in a parallel system to which access is provided by the local organization. The execution scheme is a C program (file `scheme.c`) which can not be modified. The I/O are performed via this program, which has a limit for the execution time and generates the solution in a file which is compared for validation with the output of the sequential program. The scheme is compiled and linked by the system with the file with the sequential function (`sec.c`). The resulting executable is run through MPI, with only one MPI process and one OpenMP thread, so it is considered the sequential version to compare against. The file `sec.c` has a heading of the form:

```
/*
CPP_NUM_CORES = 1
CPP_PROCESSES_PER_NODE 1
CPP_PROBLEM=mm
*/
```

where `CPP_NUM_CORES` establishes the number of cores to use (maximum 32), `CPP_PROCESSES_PER_NODE` the number of MPI processes to run on each node, and `CPP_PROBLEM` the name of the problem. The example corresponds to the heading of the sequential program, and so the number of cores is 1 and the number of

processes per node is 1. The teams modify the sequential function to make it parallel and send the file with the new function and the modified heading to use more cores and MPI and/or OpenMP. The number of nodes reserved to run the program is $NUM\_NODES = \lfloor (CPP\_NUM\_CORES - 1)/8 \rfloor + 1$, and the number of MPI processes $NUM\_PRO = NUM\_NODES * CPP\_PROCESSES\_PER\_NODE$. Inside each MPI process, the number of OpenMP threads (NUM_THR) is set with the function $set\_omp\_num\_threads$. So, a maximum of 32 cores can be used by a parallel program, and it is possible to use message-passing parallelism ($CPP\_NUM\_CORES > 1$ or $CPP\_PROCESSES\_PER\_NODE > 1$ or both, and $NUM\_THR = 1$), shared-memory parallelism ($CPP\_NUM\_CORES \leqslant 8$ and $NUM\_THR > 1$), or hybrid MPI+OpenMP parallelism. Several combinations can be used to attempt to achieve the highest speed-up.

3. Problems in the First Spanish Parallel Programming Contest

When designing a programming contest it is necessary to carefully select the problems to work with. There are some papers dedicated to tasks selection in computing competitions (Burton and Hiron, 2008; Vasiga *et al.*, 2008; Hakulinen, 2011). For a parallel programming contest the problems selection has some particularities that differentiates it from other computing contests. In this section the problems used in the First Spanish Parallel Programming Contest are shown and the criteria for problem selection are discussed, comparing them with the recommendations in the literature.

Five problems were proposed to be solved in four hours (to generate parallel solutions and adapt them to the computational cluster). They can be found on the web page of the contest (SPPC). Next we enumerate and discuss the problems:

A Multiplication of matrices with rectangular holes: Two square real matrices are multiplied. The matrices have rectangles of zeros. The rectangles can overlap.
The sequential solution provided uses the zeros structure of the matrices to accelerate the computation. The contestants can parallelize that sequential version or develop the parallel program from a different sequential version. For example, they could use a dense or a sparse matrix multiplication version, but the matrices are not dense or sparse, and with those approaches the parallel version may be far from satisfactory speed-ups. Furthermore, the sequential version provided does not optimize memory access, and in a multiplication AB, matrix B is accessed by columns, which can be improved just by transposing matrix B and accessing it by rows.

B Live game with variable neighborhood: This is a live game where the value in each position depends on the values in the neighboring positions in the previous generation, but in different generations the neighborhood varies, with the neighbors being the positions at a given Manhattan distance.
The sequential program follows an iterative scheme, and parallelisation can be achieved only inside each iteration. The computational cost and the memory ac-

cess in each iteration have order $O\left(n^2\right)$, which makes it difficult to obtain highly efficient parallel versions.

C Obtain values in given positions after sorting: An array of integers is given, together with a set of positions. The problem is to obtain the values in these positions when the values in the array are sorted.
The sequential solution sorts initially the positions, and then obtains the values in those positions by applying a partition scheme recursively. The quicksort pivoting strategy is applied to obtain the element in the middle position, and then the same method is applied to the left and right parts of the integer values and with the left and right parts of the positions. Obviously, it is possible to obtain a parallel solution just by sorting the array of integers, but the execution time would be higher than that of the solution provided, and consequently the speed-up achieved (if any) would not be very high.

D Multiply four dense square matrices.
This is the easiest problem in the contest. Three typical and naive matrix multiplications are performed. It is possible to optimize the memory access as indicated for problem A, and the parallelization of the matrix multiplication gives high speed-up (the computational cost is $O\left(n^3\right)$ and the access cost is $O\left(n^2\right)$). Furthermore, two of the multiplications can be done in parallel, which would allow a better use of the cluster, and consequently higher speed-up.

E Knapsack problem with affinities: We have a number of knapsacks with a certain capacity each, and a set of objects with a certain weight and with affinities between the objects. The objective is to obtain the assignation of objects to the knapsacks with the highest total affinity. The weight restrictions must be fulfilled, and the total affinity is the sum of the affinities between objects assigned to the same knapsack.
The sequential solution follows a backtracking scheme. It is possible to obtain better sequential solutions, with a better backtracking or with branch and bound algorithms, but the best solution depends on input, and the entries to be solved are not very large because it would produce a very long execution time. So, possibly the best approach is to parallelize the backtracking algorithm by generating a set of subproblems with all the possible assignations of some objects, and to assign a number of subproblems to different processes or threads.

The five problems follow well known algorithmic schemes and can be parallelized with parallel schemes which are explained in parallelism books (Almeida *et al.*, 2008; Grama *et al.*, 2003; Quinn, 2004). The targeted contestants (final years undergraduate and master and doctoral students) should know the sequential schemes and the basic parallel algorithms. So, the parallelization is not a big problem ... if they did not have a time limit of four hours. Furthermore, access to internet was allowed in the contest for a number of reasons: the possibility of participation online, the use of the tool Mooshak through internet, the need for access to a remote parallel system and because at present most of the bibliography is consulted on internet. So, the problems should not be typical problems or they can be well known problems but that should be modified to achieve the maximum performance in the system where the contest runs. The solutions must be

Table 1

Estimated maximum speed-ups achievable, the maximum with sequential, message-passing and shared-memory optimization, and the records in the contest and at February 8, 2012

	A	B	C	D	E
Sequential	3	1	1.2	4	2
message-passing	3	1.5	1.5	3.5	3.5
shared-memory	6	6	4	7	6
maximum estimated speed-up	54	9	7.2	98	42
Record in the contest	17.09 seq. shared	2.68 shared		25.88 seq. message shared	
Record at February 8, 2012	19.5 message shared	6.23 seq. shared	2.55 shared	45.13 seq. message shared	5.9 message shared

correct (they are checked automatically), but the goal is to achieve a high speed-up, and for that it is necessary not only to solve the problems in parallel, but also to optimize the sequential program and to adapt the parallel program to the computational system. Even though all the sequential programs follow well known algorithmic schemes, the solutions for A, B and D use a regular scheme (a number of loops) while the solutions of C and E have a more complex structure, and consequently it should be more difficult to obtain a parallel program for them.

Roughly speaking, we can estimate the ease of parallelism of each problem (the maximum expected speed-up) by multiplying the speed-up expected by sequential optimization, by the use of multiple nodes with message-passing and by the use of all the cores in a node. The maximum estimated speed-up is shown in Table 1. The values in the table represent estimations based on the empirical knowledge of the members of the organizing committee (they are not experimental speed-ups obtained by running efficient programs). The record speed-up for each problem in the contest and at February 8, 2012 are also shown, together with the combination of optimizations used in each record. Due to the difference between the estimated speed-ups and those of the records, there is space for further improvement. The different sources for improvement are commented:

- The sequential speed-up corresponds to improvements in the sequential program. As mentioned, the matrix multiplications (problems A and D) can be improved by changing the data access, transposing one of the matrices or designing an algorithm by blocks. In problem A the matrices are not dense, and so the estimated improvement is lower. In problem C the level of recursion can be changed, and so slightly reduce the execution time. For problem E the source of improvement is non predictable, because it depends on the input, but some changes can be done in the backtracking, or alternative methods can be used, and so a value of 2 is assigned to the sequential speed-up.

- The maximum message-passing speed-up is 4, because the cluster comprises 4 cores which work together with message-passing. Of course, the complete system (32 cores) can be used with MPI processes and message-passing, but the speed-up obtained with the use of cores in the same node is included in the shared-memory speed-up.
  Problems A, D and E have the highest computational cost, and so the highest speed-up is assigned to them. The value assigned is less than 4 due to the cost of communications. Problem A is a matrix multiplication, which is easily parallelizable, but the structure of the matrices, with rectangles of zeros, reduces the computational cost and hence the achievable speed-up. Problems B and C have lower costs, and it will be more difficult to obtain high speed-up with message-passing programs.
- The number of cores per node is 8, and this is the maximum shared-memory speed-up.
  The speed-up in shared memory is relatively easier to obtain, particularly for the dense matrix multiplication (problem D). The lowest achievable speed-up has been assigned to problem C due to its low computational cost.

Next we comment on some of the tasks generation recommendations in Burton and Hiron (2008) and indicate how they apply to the problems in the contest:

- The problem statements are short and easy to understand, so that the participants can concentrate on the solution (the parallelization) of the problem. Some of the statements are very short, as for example that of problem D: "Multiplication of four dense square matrices."
- The problems are modifications of classical problems, and the sequential solutions provided follow typical sequential algorithmic schemes, but to obtain a highly efficient parallel solution it may be necessary to modify the sequential program and to design the parallel version bearing in mind the target computational system.
- For each problem there are different possibilities of parallelization, of varying difficulty and efficiency. And for some problems satisfactory solutions can be easily programmed or taken from internet (this is specially true for problem D). So, it should be easy to gain some marks in some problems, and more difficult to obtain high speed-up.
- There are no official solutions apart from the sequential ones, which follow well known schemes.
- The problems on which the proposed problems are based are well known, and also the schemes solving them, but the best solution is not clear, and modifications of the basic schemes and adaptation to the computational system are needed.

4. The Competition

The contest was celebrated in the *Jornadas de Paralelismo* in September 2011 in La Laguna, Tenerife. The *Jornadas de Paralelismo* is an annual event which attracts most of the people working in parallel computing in Spain, so this is an appropriate framework

Fig. 1. Participants in the First Spanish Parallel Programming Contest.

for a parallel programming contest. The contest is conceived for undergraduate students in their last years or for master or doctoral students, and some of them participate in the *Jornadas de Paralelismo* by presenting their initial research results.

To facilitate participation, the teams can participate in situ or online, with two classifications: one for teams participating in the *Jornadas de Paralelismo* and the other for all the participants (in situ and online participants). Eight groups from six universities participated (Fig. 1, four in situ and four online). The number of teams is not large, but we consider it satisfactory for the first edition, moreover considering La Laguna is far from most of the Spanish Universities. Anyway, our main goal with the contest is to spread parallelism, and with this first edition we hope to have prepared the base for a higher participation in successive editions.

Before the contest there was a warm-up session in July, to check the modifications made to Mooshak and the correct response of the computational system, and to allow the participants to get familiar with the mechanics of the contest, the tool Mooshak and the cluster. That session lasted four days (to facilitate participation and experimentation), and it consisted of two very simple problems: mergesort and matrix multiplication. For the matrix multiplication, in addition to the sequential solution, OpenMP, MPI and hybrid MPI+OpenMP programs were provided, so that the teams could gain experience of the behavior of the system with different types of parallelism.

The contest runs on Mooshak, which validates the submissions and calculates the speed-ups and the classification in real time. Furthermore, it allows guests to be invited into the contest, and that way the evolution of the contest can be followed by non participants. The last moments of the contest were followed by about 25 guests, which gives an idea of the interest the contest aroused. Mooshak provides a classification in the form shown in Fig. 2, which shows the final classification. For each team and problem the mark obtained is shown, and between brackets the lowest execution time from all the submissions (0 if no correct solution is obtained), the maximum achieved speed-up and the number of submissions. The last two columns show the number of problems for which the team has sent a correct solution and the total points. Problems C and E were not

Team	Problems A	B	C	D	E	Total Points	
1 UAM	15.000000 (1103 15.792384 2)			15.000000 (626 25.880192 9)		2	30.000000
2 UALM	8.241741 (1914 8.677116 3)	0.000000 (0 0.000000 1)	0.000000 (0 0.000000 2)	8.963288 (1022 15.464775 4)	0.000000 (21954 0.000000 2)	3	17.205029
3 UMU	5.651547 (2665 5.950094 3)	1.681804 (6386 1.681804 2)		2.297346 (3390 3.963717 4)		3	9.630697
4 UCLM	0.000000 (0 0.000000 1)			7.421084 (1219 12.803938 4)		1	7.421084
5 ULL	0.000000 (0 0.000000 4)	1.553071 (6708 1.553071 2)		2.807980 (2879 4.844738 3)		2	4.361051
6 UAC	0.014791 (18238 0.015572 9)	1.634769 (6500 1.634769 6)	0.000000 (18477 0.000000 1)	1.517337 (4651 2.617932 5)	0.000000 (21222 0.000000 1)	5	3.166897

Fig. 2. Final classification of the First Spanish Parallel Programming Contest.

solved correctly by any team in a time lower than the sequential time, which is in concordance with the higher difficulty we considered for these two tasks. Furthermore, the highest speed-up has been obtained for problem D, followed by problem A and finally problem B. This coincides with the easiness estimated in Table 1. In problems A and D the maximum speed-ups were obtained by sequential optimization (optimization of the access to memory by transposition of the second matrix in the multiplications) combined with shared-memory or message-passing parallelization. No team combined both types of parallelism, which means the speed-ups obtained are far from the maximum estimated.

The evolution of the classification is shown in Fig. 3. The minutes at which modifications in the classification happen are represented. There are some points where the marks of some teams decrease (minutes 198 and 234). This happens when a speed-up higher

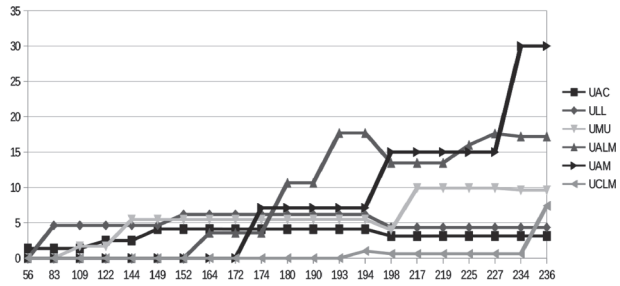


Fig. 3. Evolution of the classification throughout the First Spanish Parallel Programming Contest.

than 16 is obtained, because in this case the marks of all the teams for that problem are recalculated. All six universities were at some moment in the first position, and the last hour has the maximum number of submissions, with four changes in the first position.

5. Conclusion and Perspectives

The paper shows the experience of the first Spanish Parallel Programming Contest, held in September 2011. This event aims to disseminate parallelism among Scientific Computing students. The contest is similar to other programming competitions, and there are other parallel programming contests, but there are some differences: this one has two classifications, in situ and online; the tool Mooshak is used for an automated and real time evaluation of the submissions and for the classification, for which a new classification scheme has been implemented along with the system to connect Mooshak to the cluster where the contest is carried out; and a record table is maintained in the web page of the contest, with explanations and codes of the fastest solutions obtained in the contests or submitted outside them, so that the page can be used for preparing the participation in successive editions and for educational purposes, all of which are being used in various parallel programming courses in Spanish universities.

Additionally, the task generation process has been described. Recommendations of other authors for task generation have been considered, and the adaptation to a parallel programming contest are commented on.

We can consider the first edition of the contest has been successful, with a small number of participants, which is normal for the first edition, due to the specialization of parallelism (which is becoming more and more popular, but at present is not studied by all Computer Science students), and to the celebration of the contest at the University of La Laguna, which is far away from most of the Spanish universities.

The perspectives of the contest are:

- We are now working on the preparation of the next edition, in September 2012. We plan to organize an open session in the *Jornadas de Paralelismo* to follow the last minutes of the competition, and to discuss the solutions provided by the contestants and other possible solutions.
- The web page and the contest will also be in English, so that non Spanish students can participate online, and the use of the web page as an educational tool will be more visible.
- Some additional modifications can be included in Mooshak to better adapt it to the contest. For example, it could be interesting to include the generation of classifications with the format of Fig. 3, and not only in table form (Fig. 2).
- The inclusion of a CUDA competition is being considered. This supposes additional organizational work, because the programming paradigm changes, and problems which can be solved with a SIMD approach should be generated, and the expected speed-up estimated. Furthermore, some small modifications should be included in Mooshak to addapt it to the job queue system in a multicore+GPU environment.

Acknowledgements. This work has been funded in part by the Spanish MCYT under Grant TIN2008-06570-C04-02 and by the Fundación Séneca, Consejería de Educación de la Región de Murcia, 08763/PI/08. The authors gratefully acknowledge the computer resources and assistance provided by the Supercomputing Center of Fundación Parque Científico of Murcia.

References

Almeida, F., Giménez, D., Mantas, J.M., Vidal, A.M. (2008). *Introducción a la programación paralela*. Paraninfo Cengage Learning.

Burton, B.A., Hiron, M. (2008). Creating informatics olympiad tasks: exploring the black art. *Olympiads in Informatics*, 2, 16–36.

Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. (2001). *Parallel Programming in OpenMP*. Morgan Kauffman.

Dagienė, V. (2010). Sustaining informatics education by contests. In: Hromkovic, J., Krlovic, R., Vahrenhold, J. (Eds.) *Teaching Fundamentals Concepts of Informatics*. Springer, 1–12.

Fernández Alemán, J.L. (2011). Automated assessment in a programming tools course. *IEEE Trans. Education*, 54(4), 576–581.

García-Mateos, G., Fernández Alemán, J.L. (2009). A course on algorithms and data structures using on-line judging. In: *ITiCSE*, 45–49.

Grama, A., Gupta, A., Karypis, G., Kumar, V. (2003). *Introduction to Parallel Computing*. Addison-Wesley.

Hakulinen, L. (2011). Survey on informatics competitions: developing tasks. *Olympiads in Informatics*, 5, 12–25.

IEEE Technical Committee on Parallel Processing.
`http://www.cs.gsu.edu/~tcpp/curriculum/index.php`.

International Olympiad in Informatics.
`http://www.ioinformatics.org/index.shtml`.

Jornadas de Paralelismo (2011).
`http://jp2011.pcg.ull.es/`.

Kolstad, R. (2009). Infrastructure for contest task development. *Olympiads in Informatics*, 3, 38–59.

Lawrence, R. (2004). Teaching data structures using competitive games. *IEEE Transactions on Education*, 47(11), 753–759.

Leal, J.P., Silva, F.M.A. (2003). Mooshak: a web-based multi-site programming contest system. *Softw. Pract. Exper.*, 33(6), 567–581.

Marathon of Parallel Programming.
`http://regulus.pcs.usp.br/marathon/current/index.html`.

Meder, D.J., Pankratius, V., Tichy, W.F. (2008).
`http://www.multicore-systems.org/separs/downloads/GI-WG-SurveyParallelismCurricula.pdf`.

Quinn, M.J. (2004). *Parallel Programming in C with MPI and OpenMP*. McGraw Hill.

Snir, M., Gropp, W. (1998). *MPI. The Complete Reference*. The MIT Press.

Spanish Parallel Programming Contest
`http://cpp.fpcmur.es`.

Student Cluster Competition.
`http://sc10.supercomputing.org/?pg=studentcluster.html`.

Supercomputing Centre of Murcia.
`http://www.cesmu.es/inicio/`.

UVa Online Judge and Contest System Developed by the University of Valladolid (Spain).
`http://online-judge.uva.es/problemset`.

Vasiga, T., Cormack, G., Kemkes, G. (2008). What do olympiad tasks measure?. *Olympiads in Informatics*, 2, 181–191.

F. Almeida received his degree and the MSc in mathematics from the University of La Laguna in 1989 and 1992 respectively. He obtained his PhD in Computer Science in 1996. Currently he is professor in the Department of Statistics and Computer Science in the University of La Laguna. His research interests are primarily in the areas of parallel computing, parallel algorithms for optimization problems, parallel systems performance analysis and prediction, skeleton tools for parallel programming and web services for high performance computing and grid technology.

V. Blanco Pérez received his degree in physics and his MSc in physics from the University of Santiago de Compostela in 1992 and 1993 respectively. He obtained his PhD in physics in 2002. In October 2000 he became an assistant professor in the Department of Statistics and Computer Science in the University of La Laguna. Since 2009 he is associated professor in the same department. His research interests include performance analysis of parallel codes, parallel algorithms for dense and sparse algebra, Grid technology, and GPGPU tecnology.

J. Cuenca is an associate professor in the Computer Engineering Department at the University of Murcia, Spain. He received his BSc (engineering in computer science) from the University of Murcia in 1994, and his PhD in computer science from the University of Murcia in 2004. He was director of the Computer Engineering Department from 2008 until 2011. Since 1998 he has taught several subjects: "Fundamentals of Computer's', "Fundamentals of Operating Systems" in the computer science degree, and "Parallel Programming" in the computer science master. His research interests include issues related to parallel computing, linear algebra software and software autotuning techniques.

R. Fernández-Pascual received his MSc and PhD degrees in computer science from the University of Murcia, Spain, in 2004 and 2009, respectively. In 2004, he joined the Computer Engineering Department as a PhD student with a fellowship from the regional government. Since 2006, he has been an assistant professor at the University of Murcia. His research interests include general computer architecture, fault tolerance, chip multiprocessors and performance simulation.

G. Garcia-Mateos is a professor working at the Computer Science Faculty of the University of Murcia, Spain. He received his PhD degree in 2007, and is a member of the Computer Vision Research Group. Since 1998 he has been teaching algorithms and data structures, and has written two textbooks and several scientific papers on computer science education and programming contests. In 2002 he participated in the creation of the Programming Olympiad in Murcia for computer science students, and in 2008 the Informatics Olympiad in Murcia for high school students. Currently, he is the director of both olympiads. These contests are the Murcia

local stages of the ACM International Collegiate Programming Contest and the International Olympiad in Informatics, respectively.

D. Giménez is an associate professor in the Computer Science Department at the University of Murcia, Spain. He has been a faculty member of the university since 1988, where he teaches algorithms and parallel computing. He received his degree in mathematics from the University of Murcia in 1982, and his PhD in computer science from the Polytechnic University of Valencia in 1995. In 2002 he participated in the creation of the Programming Olympiad in Murcia for computer science students. His research interests include scientific applications of parallel computing, matrix computation, scheduling and software auto-tuning techniques.

J. Guillén is a telecommunications engineer from the Polytechnic University of Valencia and executive MBA in the Escuela de Organización Industrial. Since 2009 he has been project manager for the Supercomputing Center of Fundación Parque Científico de Murcia, managing R&D collaboration projects involving public research centres, universities and companies in different sectors such as industry, naval, biotechnology and engineering. Previously, for more than 4 years, was project manager for Ericsson, coordinating international projects for telecom operators from different countries such as Ireland, Hungary, Egypt and Nigeria, mainly related to network rollouts and systems integrations in all project phases. First job position since 2003 as analyst and team leader for the international IT and business consulting company Everis for the customer Telefónica Spain.

J.A. Palomino Benito (1983) holds a computer engineering degree from the University of Alicante. Currently he is doing his PhD thesis research since he presented his dissertation about parallel linear systems solvers in the Department of Science of the Computation and Artificial Intelligence at the University of Alicante. Since 2008, he has been working as application area manager in the Supercomputing Center of the Murcia Science Park in Spain. Hence his interests focus on HPC and algorithms in general, and he is a member of the Spanish Parallel Programming Contest organizing committee.

María E. Requena studied telecommunications engineering at the Polytechnic University of Valencia and did her PhD at the Polytechnic University of Cartagena (UPCT). She did her final project in the Repsol-YPF refinery in Cartagena. She began her career in Madrid in the SDB ALTRAN consulting involved in reengineering projects like power switching Moviline at Motorola and the design and implementation Imagenio platform at Telefonica R&D. She returned to Cartagena where she completed doctoral courses and worked in several R&D projects. While in the UPCT she published several books, international and national papers. Currently she is a head of the Supercomputing Center of the Science Park Foundation of Murcia a position she has held since March 2008.

J. Ranilla is a PhD in computer science and associate professor at the Faculty of Informatics, University of Oviedo (Spain). His research interests include information retrieval, knowledge management, parallel computing, and machine learning. Contact him at the Computer Science Department at the University of Oviedo, 33271, Campus de Viesques, Gijón, Spain.

Olympiads in Informatics, 2012, Vol. 6, 148–157

Learning Computer Programming as an Extra Curriculum Activity, the Challenges

Francis DOGBEY

Advanced Information Technology Institute, Ghana-India Kofi Annan Centre of Excellence in ICT
PMB, State House, Accra, Ghana
e-mail: francisd@aiti-kace.com.gh

Abstract. This paper describes the i2CAP project for senior high school students in Ghana. The project promotes and demystifies computer programming through programming contests. It is run in two distinct divisions: inter-schools programming contests and the National Olympiad in Informatics (in preparation towards IOI participation). This project has developed the algorithmic thinking and computer programming capacity of about 10,000 students and 303 ICT teachers from 257 senior high schools throughout the Ghana. This paper describes selection, training and organization of the project as well as the challenges and successes of running a fairly balanced programming contest among digital divided senior high schools in Ghana.

Key words: i2CAP, inter-schools programming contest, algorithms, national olympiad in informatics.

1. Introduction

The programming contest at high school level in Ghana was established by the Ghana–India Kofi Annan Centre of Excellence in ICT (AITI-KACE) in 2004 in preparation towards the country's participation in the International Olympiad in Informatics (IOI). The contest, run under the project name 'I TOO CAN PROGRAM' (i2CAP), aims at demystifying computer programming, and promoting interest in computer science. The project provides students the opportunities and resources needed for exposure to computing careers.

Ghana has a three level educational system namely: free compulsory basic education, senior high school education and tertiary education. Pre-tertiary education is made up of nine years basic education (i.e., primary and junior high) and a three years senior high school/technical/vocational education. There have been several educational reforms especially at the pre-tertiary level with the sole objective of improving the standard of education in the country. Recently, the educational institutions have incorporated teaching of Information and Communication Technology (ICT)/ computer literacy classes into their teaching curricula. Like other African countries, the use of ICTs in the Ghanaian schools is generally increasing and noticeably growing (Amenyedzi *et al.*, 2011). This was further strengthened by the enactment of the ICT for Accelerated Development (ICT4AD) policy into law in 2004 by the Parliament of Ghana. This law seeks to promote ICT education

and to support in the development of research capacity in ICT among others (National ICT Policy and Plan Development Committee, 2003).

The quality and quantity of ICT infrastructure and human resources at senior high schools in Ghana varies significantly. The so-called 'endowed' schools have more computing resources (i.e., PCs, Internet Access, ICT Teachers, etc.) compared to the 'less endowed' ones, creating a digital divide (Merry *et al.*, 2008) among the schools. The interest of the project is to encourage as many students as possible to participate in the project in order to develop the algorithmic thinking and basic programming skills that are useful for their career. The question is how to run the contest among the digitally divided schools with some level of fairness?

There is no specialised school in Ghana for talented young students. All students at the pre-tertiary level, except those from the international schools, study a common curriculum and sit the same examination. Time is one of the critical resources required to develop exceptional programming skills and to develop the necessary algorithmic thinking skills required to compete at IOI. Another question is how to convince parents or guardians, teachers and heads of schools to embrace computer programming contests, and more importantly, the students, to invest time into this time consuming activity. A barrier is that the universities and the polytechnics in Ghana make undergraduate admission decisions strictly based on performance of the applicants at the West African Senior School Certificate Examination (WASSCE), the standardized test for the English speaking West Africa countries.

This paper is structured as follows: Section 2 describes our training methodologies; Section 3 describes organization of the contests; Section 4 summarizes our observation, experience and challenges and Section 5 our conclusion.

2. Training

2.1. *Training Material*

Our first step in implementing the project was developing training materials for the participants. As a result we developed a training manual (both in hard and soft copies) that were distributed to participants and interested students during the training workshops. We also created an online resource (`http://i2cap.aiti-kace.com.gh`) to engage and to develop interested senior high school students around the country. In order to achieve fairness, we ensure that tutorials on the website are not significantly different from tutorial provided in the hard or soft (on CD) copies. Our aim is to involve the *students in a computer programming contest that is fun, educational and beneficial to the society*. As a further aid, contact details of professional programmers with excellent algorithmic thinking and thorough understanding of computational theories were made available to the interested students. The experienced programmers serve as both mentors and role models for the students.

The website also provides programming tips and pointers to other relevant learning materials. Practice questions as well as preliminary contest questions are uploaded onto

the website for students. Students are allowed to answer the problems and submit the answers for constructive comments or review by mentors either by post, email or uploading it at the website. Major flaws identified in the student solutions are included in the frequently asked questions. Unfortunately, because of resource constraints including monetary constraints, information at the website is revised and updated regularly in contrast to what can be done with the hard copies send to the schools. We also observed that many of the students prefer to download the materials from the website directly and share it among their colleagues rather than waiting for the hard copy from us. Also, many of the students involved in downloads are either from endowed schools or have affluent family background. This partially defeats our aim to provide equal opportunity to project resources.

2.2. *Capacity Building*

In Ghana, computer programming is not taught as a subject or part of a subject at senior high school level. There are steps being taken to introduce ICT (i.e., computer literacy) into the senior high school examination curriculum and to make it an examination subject. Currently, some of the endowed schools engage the services of computer hardware technicians to maintain the computers at their computer laboratories and to train their students to acquire basic computer literacy skills.

We recognize that there is a very limited number of ICT teachers with excellent programming or algorithmic thinking background to support the project. As a result we adopted a capacity building approach in implementing the project by organising intensive two weeks programming training workshop for interested mathematics and science teachers. Our training assumed that teachers are familiar with basic-to-intermediate mathematical concepts and have the capacity to think logically. This supports Gulati's argument that several open learning initiatives in developing countries have focused on educating and training their unqualified teaching force (Gulati, 2008).

Since many of the participants of the train-the-trainer boot camp were relatively new to computer programming, we adopted hands-on training, pair programming (where necessary) and real world scenarios based on classical algorithms during the training workshop. This combined with game technique such as error spotting, output prediction, etc., generated and sustained much interest, making programming fun among the participants. Our learner-centred approach enhanced the problem solving capacity of the teachers. This observation buttresses the Elizabeth Sklar, et al arguments for research in technology education that student-centred learning environments support design (Martin, 1996; Leper, 2000), constructionism (Papert, 1980; Piaget, 1972) and team work (Garner, 1983) and therefore generates the strongest outcome (Eguchi, 2004).

The training was conducted using BASIC, Ruby and C/C++ programming languages. The choice of programming language was hugely influenced by ease of use, learner ability and preference. For easy facilitation the entire country was divided into localities. A locality consisted of all selected schools from a region or parts of a region depending on the geographical dispersion of schools in the region. Also, each of the

participating teachers agreed to identify and train between 40 to 50 intelligent students, out of which a team is formed to represent their schools. The decision on the number of students was influenced by the voluntary nature of training and the fact the project is organized as an extra curriculum activity.

Our capacity building technique seeks to support our objective of providing near-balanced resources to enhance fairness and minimise the possible knowledge gap. At the same time it introduced another problem of severe under-representation of women during the mentoring session for ICT teachers, and therefore supports the well known fact that, women are underrepresented in computer science and this trend is worsening (Doerschuk, 2004). In our attempt to improve the underrepresentation, at all time a woman mentor was part of the mentoring programme for teachers and during our inspection visits to the school. Our invitation letters to the school authorities also encouraged them to nominate qualified women for the mentoring programme. In spite of all these measures, we had less than 10% female participation in the programme.

3. The Contests

Unlike other participating countries at the IOI, we organized two distinct contests: the inter-schools contest (see Fig. 1) and the National Olympiad in Informatics (see Fig. 2) as described below.

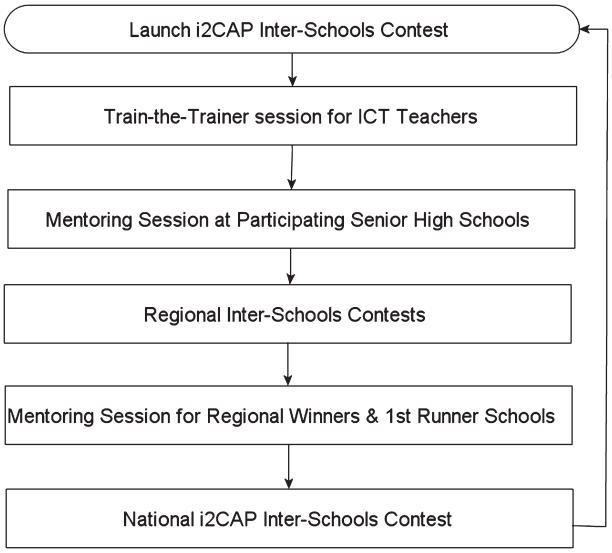


Fig. 1. National i2CAP – Inter-Schools Contest.

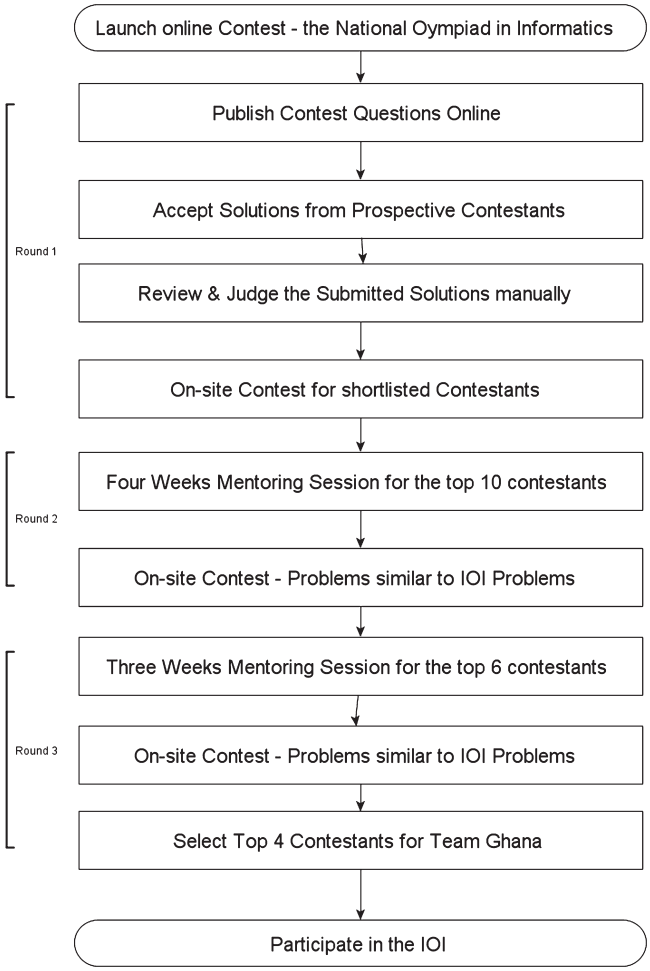


Fig. 2. National Olympiad in Informatics.

3.1. *Inter-Schools Contest*

The first step of the inter-schools contest is the selection of participating schools. This is done through advertisement in the daily Graphic (i.e., national daily), project website and formal invitation to all schools that participated in the train-the-trainer workshop. We use these combinations of outreach methods to obtain larger coverage and to ensure our sponsors also got the necessary media publicity they deserve.

The inter-school contest is different from the regular IOI in many respects. Contestants represent their schools rather than themselves. Each school selects up to four contestants to represent it at the regional contests. The contestants work in teams to submit one solution per question. They are assigned only one computer for programming. There are ten regions in Ghana and so we organized 10 regional contests and a national inter-

schools contest. At the national inter-schools contest, each region is represented by the regional contest winner and 1st runner-up. Our first aim at this contest is to promote students interest in computer science and programming. Second, is to ensure that parents, teachers and the heads of schools accept the contest as a necessary capacity building project and to give their consents and supports for their childrens' participation. Both aims were partially achieved as some of our stakeholders refused participation due to time constraints and for the fear that it would compromise their academic performance at the WASSCE. The disappointed brilliant students who could not represent their schools were encouraged to enrol for the National Olympiad in Informatics.

The tasks for this contest are less challenging compared to that used for the IOI and Ghana's National Olympiad. The tasks are aimed at testing students' understanding of mathematical concepts, and basic algorithms and programming concepts taught during the mentoring session organized by their teachers. Another aim is to fairly nurture algorithmic thinking in the younger students across Ghana.

The inter-schools contests are usually held on Saturdays to minimize interruption of regular classes. The regional contests are held at the secondary schools with functioning and large computer laboratories. However, the National i2CAP events are held at our (AITI-KACE) laboratories in Accra. Each PC used for the contest has BASIC, Ruby or C compilers readily installed. Each team submit one solution for evaluation. There are four questions with varied difficulties to be solved within three hours by contestants. There are five judges made up of professional software engineers, computer science lecturers and mathematicians who manually evaluate the contest. Since many of the judges had their high school level education in Ghana, we conduct background checks to ensure that no two judges graduated from same institution particularly if that institution is competing. The solutions submitted by the contestants are evaluated based on the quality and efficiency of the codes or the algorithms implemented.

During the inter-schools contests, computer science and mathematics book prizes are awarded to all participating schools. In addition, the top three schools are awarded PC prizes donated by our sponsors. By awarding prizes to top performing schools we hope to achieve the goal of supplementing the efforts of ICT education in the country.

3.2. *National Olympiad in Informatics*

The inter-schools contests were run for about three years prior to the first selection process for IOI event. As a result the National Olympiad in Informatics benefited immensely from the publicity generated by the inter-schools contests among the students. Our national olympiad event is open to any senior high school student below 19 years by 1st July of the preceding year to the IOI event. However, only Ghanaian students are selected as part of the Ghana delegation. Like the inter-schools contest, the national olympiad contest is launched through the media and project website.

The first step of the selection process for the national olympiad team is the publication of the preliminary contest questions at the project website. We also run adverts in the national dailies to direct interested students to the website. All submissions for the

preliminary contest questions are done online. Students participating in this contest are expected to implement their answers in C/C++, BASIC or Ruby. Usually, 80 to 100 submissions are received by the deadline. The solutions submitted are then evaluated by a panel of judges manually. About 30 to 40 students are short-listed for an on-site contest (i.e., *round 1*).

The top 10 best contestants from round 1 participate in 4-week boot camp training. The training focuses on classical algorithms used for IOI tasks. In addition, students are introduced to basic or advanced programming concepts using C/C++. Typically, the students are given 2 to 4 weeks break for self study after which another on-site contest (i.e., round 2) is then held to eliminate four more students. Assessment at this stage includes a 5% to 10% of continuous assessment and the remainder of the scores obtained during the contest. The continuous assessment is based on the number of questions successfully solved during the boot camp session, the observed algorithmic support to other students and the student attitude. Questions at this stage of on-site contest are based on previous or similar to the IOI tasks.

The best six students from *round 2* participate in the final three-week problem solving boot camp. Afterwards, another onsite contest (*i.e., final round*) is run to select up to four students to represent Ghana at the IOI depending on funding.

4. Observations/Results

Generally, the project has been very successful but there are some challenges as well. There were both unanticipated challenges and benefits.

4.1. *Observation*

Although, just about 18% of the participants were females at inter-schools i2CAP events, some of the regional contests were won by all-female teams. This is an encouraging development and a good start for development of women for computer science careers. It is also interesting to note that some of the winning schools actually had no computer laboratory and had to rely on a PC in an instructor's home for the training towards the contest. This suggests that some of the potential leaders for the ICT industry are disadvantaged, but with appropriate support their talents can be realised. Another observation is that many of the schools that won regional contests are among the so-called 'less endowed' schools in the country.

4.2. *Results*

The i2CAP project has sparked a lot of interest among senior high school students nationwide with many considering a career in ICT. Also, several of the past participants at the inter-schools or National Olympiad in Informatics are pursuing computer science or related disciplines at the tertiary level. There is also a limited number of them who have built on the exposure and now own and run their small IT businesses. Others are also

employed as trainee software developers. Many of the high school authorities have also learnt to appreciate and embrace ICT, and now support the project.

About 10,000 (estimated) students were mentored nation-wide on programming concepts and algorithmic thinking through the 257 senior high schools that participated. A total of 573 students tested their knowledge and skills through the inter-school contests. In addition, about 100 students received advanced programming in C/C++, algorithms and data structures through the National Olympiad in Informatics events.

The computer programming and problem solving capacity of 303 ICT coordinators/instructors nation-wide were enhanced.

All the participating schools and contestants (national olympiad only) were provided with computer science and discrete mathematics books to support their skills development in ICT. In addition, the winner, 1st runner-up and 2nd runner-up of the inter-schools contests were awarded with computers and their accessories to boost ICT education in their respective schools.

Ghana has been represented at IOI since 2008. Previous IOI participants have also developed the skills and now support the mentoring sessions for IOI. The performances of the contestants have also improved over the years.

In addition, there were some intangible but real successes of the project. Both the ICT coordinators and the students who participated in any of the activities of i2CAP project have become a community for continuous collaboration. Several of the participants have become more vocal, confident and supportive of one another. The inter-schools contests also provide the participating schools the opportunity to interact directly with educational authorities and to re-echo their needs to them.

4.3. *Challenges*

The i2CAP project failed to cover every single senior high school in the country due to budgetary constraints. As a result, participation in the inter-schools contest was restricted to only schools with installed and functioning computer laboratories or having access to computers to practice. This suggests that there may be some savvy youngsters with potential to develop excellent algorithms that are unused throughout the contest.

Currently, teaching and learning of ICT (i.e., programming) as a subject at senior high schools is optional and non-gradable. As a result some intelligent students are not interested in studying a subject that has no contribution to their final grades at school.

It is also very difficult and challenging to obtain permission from parents for their children especially daughters to participate in the mentoring sessions, although the entire project is fully funded.

For some unknown reasons, many of the contestants are in the final year of their high school studies making them not eligible for the next contest. Several attempts to encourage first year students to participate in the contests failed to yield positive results.

The poor teacher remuneration makes the staff turnover of ICT teachers very high. This is because ICT teachers have a higher chance of getting other high paying jobs and

move out of the schools (Leliveld, 2002). This is attributable to the shortage of skilled ICT professionals in country. As a result, the project is forced to organize the train-the-trainer workshop all the time.

5. Conclusion

The i2CAP project provides senior high school students with the opportunity to experience and develop computer programming skills whilst working in teams. This provides the contesting teams with an invaluable experience of teamwork, leadership and communication skills. The project also strengthens and develops algorithmic thinking capacity required for software development and undergraduate computer science education.

Acknowledgements. The Finatrade Foundation Ghana was the main sponsor of i2CAP events from 2007 till 2010; the Star Assurance Company also provided travel insurance cover to the Ghana team to the IOI in 2010. AITI-KACE is the technical sponsor of the project.

References

Amenyedzi, F., Lartey, M., Dzomeku, B. (2011). The use of computers and internet as supplementary source of educational material: a case study of the senior high schools in the tema metropolis in Ghana. *Contemporary Educational Technology*, 2(2), 151–162.

CIA. (2012). *The World Fact Book*. Retrieved April 16, 2012, from `https://www.cia.gov/library/publications/the-world-factbook/geos/gh.html`.

Doerschuk, P. (2004). A research and mentoring program for undergraduate women in computer science. *34th ASEE/IEEE Frontiers in Education Conference*.

Eguchi, E.S. (2004). Learning while teaching robotics. *American Association for Artificial Intelligence*.

Garner, H. (1983). *Frames of Mind: Theory of Multiple Intelligences.* Basic Books.

Gulati, S. (2008). Technology-enhanced learning in developing nations: a review. *The International Review of Research in Open and Distance Learning*.

Leliveld, M. (2002). *Science, Mathematics and ICT (SMICT) Education in Senior Secondary Schools in Ghana*.

Leper, M.A. (2000). Turning "play" into "work" and "work" into "play": 25 years of research on intrinsic versus extrinsic motivation. Academic Press.

Martin, F. (1996). Idea and real systems – a study of notions of control in undergraduates who design robots. *Constructionism in Practice: Designing, Thinking and Learning in a Digital World.*

Merry, B., Gallotta, M., Hultquist, C. (2008). Challenges of running a computer olympiad in South Africa. *OLympiads in Informatics*, 2, 105–114.

National ICT Policy and Plan Development Committee. (2003). *The Ghana ICT for Accelerated Development (ICT4AD) Policy.* Retrieved from The Republic of Ghana. `http://www.moc.gov.gh/moc/PDFs/Ghana_ICT4AD_Policy.pdf`.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books.

Piaget, J. (1972). *To understand Is to Invent*. New York, The Viking Press, Inc.

F. Dogbey holds an MSc in engineering (dependable computer systems) from Chalmers University of Technology, Sweden. He is a faculty member of the Advanced Information Technology Institute, Ghana–India Kofi Annan Centre of Excellence in ICT. He was deputy team leader in 2008 but a team leader, coordinator of the Ghana National Olympiad since 2009. He has been the technical facilitator since i2CAP project inception. His current research interests include machine learning, databases, fault tolerance and object oriented programming.

Olympiads in Informatics, 2012, Vol. 6, 158–169

IOI Training and Serbian Competitions in Informatics

Aleksandar ILIĆ[1], Andreja ILIĆ[2]
[1]*Facebook Inc*
1601 Willow Road, 94025 Menlo Park, California
[2]*Faculty of Sciences and Mathematics, University of Niš*
Višegradska 33, 18000 Niš, Serbia
e-mail: {aleksandari, andrejko.ilic}@gmail.com

Abstract. This paper summarizes the story of the Serbian competitions in informatics. The annual cycle of these competitions is described along with the selection and training process. The structure and work of the Serbian Scientific Committee for high school competitions in informatics, which is responsible for selection of Serbian IOI team, is given. We address content and problems of our general education in computer science in schools. Examples of problems are also provided in Appendix.

Key words: informatics competitions, Serbia, Serbian Olympiad in informatics, SIO.

1. Introduction

This paper summarizes the story of the Serbian competitions in informatics. Our annual cycle is divided in five stages. In the next section we described these stages and included both technical and organizational part of these competitions. We described the selection and training process for our national team. The structure and the work of the Serbian Scientific Committee for high school competitions in informatics, which is responsible for selection of Serbian IOI team, is given in Section 3. Selection process and preparation of the tasks is, probably, the hardest job for the Scientific Committee. This process is divided in stages and every stage is described in this paper. We addressed curriculum and problem of our general education in computer science in schools in Section 4. Some examples of problems are also provided in Appendix. The main web pages with all important information on Serbian competitions in informatics are `http://www.dms.rs/` and `http://www.yuoi.nis.edu.rs/` (currently only available in Serbian). For the national reports of other countries see references.

2. The Selection and Training Process in Serbia

The annual cycle of Serbian competitions consists of five levels, in the increasing order of the task difficulty: online Qualifications, Regional, National, Serbian olympiad in informatics (SIO) and IOI Team Selection competition. The fifth level is not mandatory and

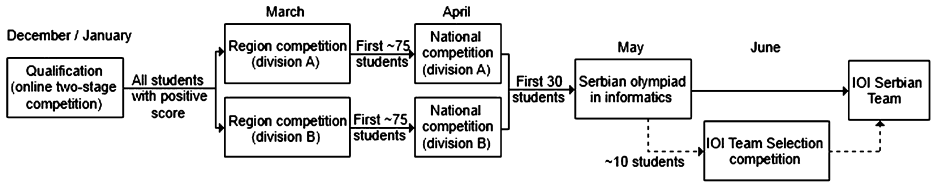


Fig. 1. Diagram of the annual cycle of Serbian competitions.

it is organized in cases where the selection of the national team for IOI is not clear from the previous rounds.

In the last four years, the Scientific Committee introduced one additional online competition – **Qualifications**. The tasks are set up on Z-training site (we will address this online-judge later in this report). This competition is a two-stage competition (December and January). In each stage, students are given five problems and they have one week to solve them. Difficulty of the problems for qualifications varies from trivial one (e.g. primality testing) to the ones similar to National competitions. The only condition that students have to meet to participate in Regional competition is to have positive score on Qualification. The main motivations for these online qualifications are popularization of the programming competitions and algorithmic type of problems to the beginners (input and output, testing on data sets, scoring . . .). Around 350 students have submitted at least one problem at last year's Qualifications.

For Regional and National competition students are divided in **two divisions: A and B**. The division A consists of students which are second, third and fourth grade in schools that are working by the program of the Mathematical Gymnasium. All the other students are in the division B. The main difference between the problems in these two divisions is difficulty. Also, the minimal number of points needed for advancing to the next stage can be different between divisions. The reason for two divisions is to give chance for everybody to win a national award – without this the awards will be mostly won by the students from division A. Usually the number of competitors that receive diplomas and awards on National competition and SIO is one half of the number of all competitors and the awards are distributed by the ratio $1 : 2 : 3$.

The Regional competition, usually held in the beginning of March, is the first stage of the Serbian competitions. It is organized in every region, around 30 of them, where students are struggling with the three algorithmic problems for which they have three hours to solve. Before the competition, the Scientific Committee provides the problem statements, test data and a simple console application for evaluation that works with executable files. Using this, the regional committee can evaluate contestants' sources and get the preliminary results for their students. Exactly one of the problems (the most difficult one in division B and the easiest in division A) is the same for both divisions. After the competition, all source codes are sent to the Scientific Committee for final evaluation and the results from this evaluation are final. These results are united and, approximately the first 75 students per division are selected and invited to the National competition.

The National competition has the same structure as Regional, except that the students have five hours to work on problems. The competition is held in three different

Table 1
Statistics of school grades for students in Regional competition and SIO

Grade	Percentage of students on the Regional competition	Percentage of students on SIO
1	19	14
2	22	23
3	35	30
4	24	33

sites (Belgrade, Niš and Novi Sad). In these university towns the Scientific Committee has representatives, so the communication and evaluation is much simpler. The first 30 students in this completion are invited to SIO, usually the first 24 from A category and 6 from B category.

SIO has a similar structure with that of the IOI. The competition is organized in the capital city of Serbia, so all competitors are in the same place. In this stage of competition there are no divisions – all students are solving the same tasks. This competition is a two-day competition (without break days), where students are given three problems and five hours each day. Solutions are graded in real-time, but students can not see the results until they leave the competition room.

Table 1 contains statistics for student grades in high school competitions from Regional and final level. After adding Qualification stage, we notice an increase of the number of contestants from the first grade and the number of contestants having positive score on the Regional competition.

For the Regional and National competition there is no global training. Some schools organize preparation, which are different from town to town. For SIO, depending on the budged, there can be additional competition during the training camp, where the national team is being selected.

IOI training camp for our national team, plus some additional competitors from the first or second grade, is the only official training that is organized every year. Most of the times IOI training camp is organized in some mountain camp where students have lectures all day. General structure of the lecture is divided into two parts: 4 hours of theoretical lecture (on a blackboard) in the morning and 4 hours of coding in the evening. Typically, this training lasts one week. We present part of the last year training schedule in Table 2.

All types of tasks on Serbian national competitions are designed to be of algorithmic nature and all of them have to be solved by directly programming on the computer. Similar to IOI's rules, there are three general categories: batch tasks, interactive tasks and output-only tasks. On the other hand, there are no restrictions for known algorithms on the last level of competition (SIO).

The knowledge needed for each stage of the annual cycle is presented to the students in the beginning of every year. This list is not strict and it is given to represent some sort

Table 2

Part of the last year training schedule

Day	Time	Lecture
01	10h – 14h	Graph theory
	15h – 19h	Graph theory (coding)
	21h – 23h	IOI tips and tricks
02	10h – 14h	Data structure
	15h – 19h	Data structure (coding)
	21h – 23h	Interview questions
03	10h – 14h	Game theory
	15h – 19h	Game theory (coding)
	21h – 00h	Output only problems
04	10h – 14h	Problems from the other national competitions
	15h – 20h	IOI practice competition

of guideline for students and their professors. Of course, if a student knows all of the given algorithms, that does not imply that he / she will do well on the competition.

- **Regional competition**
  - simple data structures (arrays, strings, matrices, sets, lists);
  - simple mathematical algorithms (number systems, prime numbers, Euclid's algorithm);
  - backtrack and recursion;
  - elementary sorting algorithms (selection sort, insertion sort, bubble sort, count sort);
  - basic geometric objects (points, lines, segments, circles) and their manipulation.
- **National competition**
  - fast sorting and searching algorithms (quick sort, merge sort, binary search);
  - representing simple graphs and trees (DFS, BFS, connected components, topological sort);
  - dynamic programming and combinatorial enumerations;
  - simple geometric algorithms (area of the polygon, point in polygon);
  - greedy algorithms.
- **Serbian Olympiad in informatics**
  - graph algorithms (Dijkstra's algorithm, MCST, SCC and BCC, Eulerian path, Flow algorithm, Bipartite matching, . . .);
  - data structures (Hash tables, Segment tree, Cumulative tables, Disjoint data set, . . .);
  - geometric algorithms (convex hull, closest points, triangulation, polygon intersection);
  - advanced algorithms for pattern searching (KMP, suffix tree, . . .).

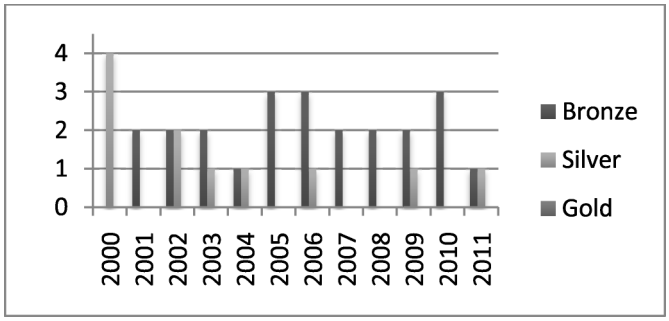


Fig. 2. The summary of IOI medals achieved by Serbian students.

All tasks on Regional and National competitions are evaluated offline in one centralized place and the results are afterwards distributed in each place where the competition was held. For Serbian Olympiad in Informatics we use online evaluating system written by Nikola Mihajlović in Java and we started to use it from 2004. The system supports programming languages C, C++, Pascal and Java. All mentioned types of problems are included in this system. Every problem is tested on some data corpus, where test case(s) from the problem statement is precondition for accepting the code for evaluation and the authors must provide checkers for testing. On these competitions, every problem has equal maximal number of points (100) and the number of test cases can vary.

3. Serbian Scientific Committee for the High School Competitions in Informatics

The Scientific Committee for the high school competitions in informatics, which is responsible for the technical organization and management, consists of the chairperson, Dragan Urošević, and Serbian former IOI and SIO competitors, which are mostly students or software developers in various companies around the world (around 20 active members). The Scientific Committee is a part of Mathematical Society of Serbia.

The task creation and selection are the most important jobs of the Scientific Committee. This process has three of stages:

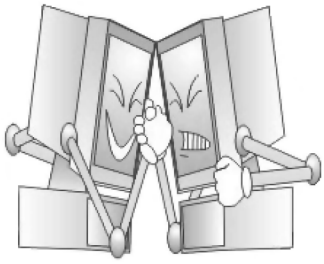


Fig. 3. The Serbian Scientific Commitee logo.

- *Submitting the problem proposals.* In this stage, more or less, every member of the SC submits some problem proposals without solution or hints. This is done before the actual formal meeting for any competition, so the members can think a little bit about the problems and get some intuition about their difficulty and possible approaches.
- *Problem selection phase.* After the discussion of the proposed problems, Scientific Committee choses the problem set (five or three tasks). The rest of proposals are saved for some other competitions or rejected as known or inadequate.
- *Problem creation.* For every problem we select one main author and one tester. The author of the problem is responsible for writing the task statement, creation of the test data and writing the official solution for this problem. On the other side, tester is writing another optimal solution (not necessarily the same algorithm) and some other possible approaches which are not optimized in terms of memory or time constraints. This way we want to be sure in the problem and test data correctness and have good distribution of points depending on the algorithms.

Besides this, Scientific Committee for the high school competitions in informatics is involved in many other events / projects. The members of SC are also active on various forums and prepare problems for other competitions, such as TopCoder. Here we will address the main three.

Z-Trening is an online resource `http://www.z-trening.com/` of challenging programming exercises (mostly compiled from the previous international programming competitions) that serves for training for the future contests. Z-trening is created by Serbian former competitor, Aleksandar Zlateski.

Each problem is accompanied by a series of tests that can be used to assess the correctness of a given solution. Most of the problems are from the Serbian competitions. Problem archive also includes problems from various other competitions with official data set. Users are free to submit their solutions in *Pascal*, Java, *C*, or *C++* and are graded based on the number of tests passed. Each user is associated with a history of his / her results and later awarded medals. Z-Trening also hosts online programming competitions, which usually last for 3–4 hours and include several new problems. Most of the material on this online judge is collected and organized by Scientific Committee of our National competition.

BubbleCup is a coding contest created by Microsoft Development Center Serbia in 2008. It is a team contest, similar to the ACM Collegiate Contest, aimed at university and high school students. The original goal of the contest was popularization of computer science among students and additional training for international competitions. Soon that idea was outgrown and the vision was expanded to attracting talented programmers from the entire region and promoting the values of communication, companionship and teamwork. The contest consists of two parts – qualifying rounds in the spring and the final round in September. Qualifications are organized in two stages as an online competition where students have twenty five days to solve 10 problems. Problems are selected from Timus online judge. The final is on-site round with the same structure as ACM finals and the best teams are rewarded with valuable prizes.

With each new iteration the contest has been growing. In its first year close to 100 participants (only from Serbia) took part and in 2011 it attracted more than 500 participants from 8 different countries in the region. Besides the competition itself, one of the great outcomes is BubbleCup Booklet. The booklet contains detailed analysis and solutions for all problems, both from qualifications and finals. The official BubbleCup website is `http://www.bubblecup.org/` and it contains detailed information about the competition and BubbleCup Booklets from previous years.

Junior Balkan Olympiad in Informatics is a new competition, established in 2007 in Serbia and the first JBOI was held in July 18 – 23. in Belgrade. The competition is open for students aged 15.5 and younger, residents of Balkan countries. The competition consists of two 3-hour rounds. All problems are algorithmic in nature and available programming languages for the competition are C, C++, C# and Pascal.

Each competing team consists of two leaders and four students – competitors. The following countries participated so far: Bosnia and Herzegovina, Bulgaria, Croatia, FYR Macedonia, Greece, Romania, Montenegro, Serbia and Turkey. The primary motive and goal of this competition is identifying talents among young students. We have invested much effort and resources to support young programmers in their education, personal development, work and on their road to success. Apart from popularization of informatics and programming among students, the goal of this event is to create friendly environment in which students can make new friends and exchange information in elementary school.

The third and fourth JBOI was held under International tournament in informatics in Shumen, Bulgaria. The fifth one in 2011 was held jointly with BOI in Bistrita, Romania.

In the last five years, Mathematical Gymnasium in Belgrade, Niš and Novi Sad started experimental class of students of seventh and eighth grade with some more intense courses in mathematics and informatics. Some of the best students from these classes also took part on high school competitions in division B with nice results (three students even qualified for SIO last year).

4. Short Overview of General Education of Informatics (Computer Science) at Schools

In Gymnasiums, informatics is thought with one or two classes per week, which is really low compared to other important courses. The aim of the course is the acquisition of basic computing literacy and training students to use computers in their further education and work. Unfortunately, the emphasis is far from the algorithmic nature. The tasks of teaching computer science and information technologies are (from the first to the fourth grade):

- introducing students to the internal organization of computer systems;
- introducing and training students to use the operating system;
- introducing and training students to use word processing programs and multimedia applications;
- introducing students to the principles of representation and processing of drawings on the computer;

- introducing students presentations on the Internet and Internet search principles;
- training for primary use of one of the programs to work with the tables;
- develop the ability to fully, accurately and concisely define the problem;
- introduction to the algorithmic way of solving problems;
- the basic data types and algorithms;
- introduction and practical use of Pascal programming language for solving problems on a computer;
- mastering the principles of creating modular and well-structured program;
- introduction to the windows-based programs;
- introduction and practical use of Delphi or Java programming environment for solving problems on the computer;
- learning and mastering the basic operations of databases.

On the other hand, in Belgrade there is a special school for gifted and talented students in mathematics and informatics – Mathematical Gymnasium. In other university towns, there are also branches of this school (one department per school class) that are working by this program. Most of the professors in these classes are professors at the Universities in Serbia. The program differs from the one mentioned above. Students gain knowledge in programming languages (Pascal, Delphi, C, C++, C#, Java; Fortran; Prolog), operating systems (Windows, Linux, Unix), databases and DBMS (Oracle Database, Microsoft SQL Server, MySQL), and in various IT and ICT fields.

5. Conclusion

In spite of the non-algorithmic content of programming courses in our schools, Serbian IOI team shows solid results. In our opinion, this is the largest problem that we are facing. This gap between school and competition type of problems is something that we want to put emphasis on in the next years. Achieving better results and higher interest of young students in this type of contest in the initial level can definitely reflect at the IOI results (that is why we introduced qualification round and involve students from seventh and eighth grade in high school competitions). Another big problem is a lack of the materials for the beginners. There are only few books of this type in Serbian. Collections of tasks with full analysis are very hard to find. This will enable us to train students for a longer period of time, which is the main point.

References

Cepeda, A., Garcia, M. (2011). Mexican olympiad in informatics. *Olympiads in Informatics*, 5, 128–130.

Combefix, S., Leroy, D. (2011). Belgian olympiads in informatics: the story of launching a national contest. *Olympiads in Informatics*, 5, 131–139.

Italiani, M. (2011). Italian olympiad in informatics: 10 years of the selection and education process. *Olympiads in Informatics*, 5, 140–146.

Kelevedjiev, E., Dzenkova, Z. (2009). Tasks and training the intermediate age students for informatics. *Olympiads in Informatics*, 3, 26–37.

Koivisto, J. (2011). The national computer olympiads and the IOI participation in Finland. *Olympiads in Informatics*, 5, 147–149.

Merry, B., Gallotta, M., Hultquist, C. (2008). Challenges in running a computer olympiad in South Africa. *Olympiads in Informatics*, 2, 105–114.

Malaivongs, K. (2011). Preparing students for IOI: Thailand country report. *Olympiads in Informatics*, 5, 150–154.

Pankov, P.S., Oriskulov, T.R. (2011). Kyrgyzstan national report on olympiads in informatics. *Olympiads in Informatics*, 5, 155–160.

Verhoeff, T. (2011). 20 years of IOI competition tasks. *Olympiads in Informatics*, 3, 149–166.

Yovcheva, B., Momcheva, G., Petrov, P. (2009). jBOI – one more possibility for increasing the number of competitors in informatics. *Olympiads in Informatics*, 3, 167–173.

Zur, E., Benaya, T., Ginat, D. (2010). IOI Israel – team selection, training, and statistics. *Olympiads in Informatics*, 4, 151–157.

Appendix

Problem: The Hamming distance

Here is an example of the problem from the **National competition 2011.** This was the second problem for students in division B. The average number of points for this problem was 26.3 (out of 100) and nine contestants solved this problem (out of 85).

Statement. The Hamming distance of two integers a and b is the number of bits in binary representation at which these numbers differ. For given n integer numbers $a[1], a[2], \ldots, a[n]$ calculate the sum of Hamming distances for all pairs of numbers, i.e.,

$$\sum_{i<j} \mathit{ham}(a[i], a[j]).$$

Input. In the first line of the input is the number of elements n ($1 \leqslant n \leqslant 100,000$), and in the second line are n integer numbers $a[i]$ ($1 \leqslant a[i] \leqslant 2,000,000,000$).

Output. In the first line of the output print the sum of the hamming distances among all pairs of numbers $a[i]$.

Sample input	**Sample output**
4 1 9 4 10	14

Explanation. The Hamming distances for all pairs of the given array are:
$\mathit{ham}(1,9) = 1$, $\mathit{ham}(1,4) = 2$, $\mathit{ham}(1,10) = 3$, $\mathit{ham}(9,4) = 3$,
$\mathit{ham}(9,10) = 2$, and $\mathit{ham}(4,10) = 3$. The sum of these Hamming distances is 14.

Naive solution. Traverse all pairs of numbers and calculate the Hamming distance using bitwise operations – one can do this in many ways: using bitwise operators, strings, arrays (bitwise operators are fastest). This solutions runs in $O(n^2 \log M)$, where M is the largest number among $a[i]$'s.

```
unsigned int sum = 0;
for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
    {
        unsigned int x = a [i] ^ a [j];
        while (x > 0)
        {
            sum++;
            x = x & (x - 1);
        }
    }
return sum;
```

Optimal solution. Note that each bit is independent. For ith bit, $0 \leqslant i < 32$, let it count represents how many numbers have 1 on ith position. Then on the aggregate sum, we just add $count \cdot (n - count)$. The complexity of this algorithm is $O(n \cdot \log M)$, since we traversed the array a exactly $\log M$ times.

```
unsigned int max_bit = 32;
unsigned int sum = 0;
unsigned int pow = 1;
for (int i = 0; i < max_bit; i++)
{
    unsigned int count = 0;
    for (int j = 0; j < n; j++)
        if ((a [j] & pow) != 0)
            count++;
    sum = sum + count * (n - count);
    pow = pow << 1;
}
```

Problem: Division Query

Here is an example of the problem from the **Serbian IOI Team Selection competition 2011.** The average number of points for this problem was 18.46 (out of 100) and, unfortunately, none of the contestants had maximal number of points. For this problem, because of its difficulty, we present only the main idea for the optimal solution.

Statement. You are given an integer array a of the length n. There are two types of operations that you can perform on this array:

- $1\ i\ x$ – sets ith element of the array to x ($a[i] = x$);
- $2\ i\ k$ – return the longest subarray (consecutive elements) of the array a such that it contains element $a[i]$ and all elements from it are divisible by k. This length can be equal to zero.

There are q operations of the above types in the input. Write a program that preforms these operations and for every query (an operation of the second type) outputs the corresponding result.

Input. In the first line of the input there are two numbers n and q ($1 \leqslant n,\ q \leqslant 200,000$) which represent the number of elements in the array a and the number of operations, respectively. The next line contains n positive integers, separated by one space –

the elements of the array a. The next q lines describe the operations. Values of the elements in array a and the value for k in queries are from 1,000,000,000.

Output. For every query in the input, print the longest subarray with the described property (in the same order as from the input).

Sample input	**Sample output**
6 4	3
2 25 20 30 19 5	1
2 3 5	5
2 4 6	
1 5 100	
2 3 5	

Remark. In 40 percent of the test cases, the number k for all queries in the input will be the same.

Naive solution. Trivial solution is to preform each query by traversing all elements, starting from $a[i]$ in both directions, and checking whether they are divisible by k. When we find the first element that is not divisible by k, we can stop. The complexity of this operation is $O(n)$, so overall complexity is equal to $O(n \cdot n)$. This solution gets only 10 points.

Subproblem solution. If the number k is equal for all queries, we can maintain an additional array *mark* such that *mark*$[i] = 1$ if k divides $a[i]$; otherwise *mark*$[i] = 0$. Now, we can translate the query for the ith element as follows: find the longest subarray of 1's containing the element with the index i. It suffices to find first-left and first-right element of the array mark, starting from ith, with value 0. This can be implemented with binary search and cumulative sums in $O(\log^2 n)$ per operation. Faster $O(q \cdot \log n)$ solution can be obtained by exploiting binary structure of cumulative table or disjoint set data structure rather than using binary search.

Optimal solution. For the optimal solution we are going to use a variation of the segment tree data structure. The structure is represented as a complete binary tree with the root node 1. The leafs correspond to the elements from the array a, starting from left to right in the last level. By definition, it follows that the number of nodes in segment tree is at most $2n$. For every node v we are going to store the greatest common divisor of values in its subarray. This way, when the element $a[i]$ is set to new value x, we are going to change only $\log n$ values in the segment tree. Using the property that every subarray can be covered with at most $\log n$ different nodes, the total complexity of this algorithm is $O(q \cdot \log^2 n)$, where we have $\log^2 n$ because of the complexity for finding the greatest common division and as every subarray can be covered with at most $\log n$ nodes of binary tree.

A. Ilić (1984) holds a PhD in computer science from Faculty of Science and Mathematics, University of Niš from 2011. He published over sixty papers on combinatorial and spectral graph theory and design of algorithms. From July 2011 works as a software engineer in Facebook on friends ranking and optimization. Aleksandar represented Serbia as a contestant on two IOIs and two IMOs wining two silver and two bronze medals, and wrote problems for IOI 2006, JBOI 2007, IOI 2008, BOI 2011, HackerCup 2012. He was the main organizer of international high school mathematics competition "Math Kangaroo" in Serbia and the first Junior Balkan Olympiad in Informatics for elementary school students in 2007.

A. Ilić (1987) is a final year student at Faculty of Sciences and Mathematics, University of Niš. Since 2007 Andreja is a member of the Serbian Committee for Competitions in Informatics and the main organizer of many training camps of Serbian national team. He won more than twenty medals as competitor in informatics and mathematics on national and international olympiads, representing Serbia. He wrote problems for BOI 2012, BubbleCup 2010, BubbleCup 2011. His areas of research are spectral graph theory, social networks and machine learning.

Olympiads in Informatics, 2012, Vol. 6, 170–177

Conducting Off-Line Informatics Olympiads with Individual Tasks

Jyldyz R. JANALIEVA
International University of Kyrgyzstan
e-mail: noledi@yandex.ru

Abstract. Conducting off-line olympiads is convenient and demands less resources than on-line olympiads. To prevent using results of other participants by a participant, we use individual generativity of tasks involving personal data. Also, uniqueness (all participants taking the test will obtain different versions of tasks) is provided. Winners obtain certificates related to their personal data only (not prizes), hence participants are not motivated to solve for others. On the base of experience of conducting such olympiads (including team olympiads with inverse tasks) in Kyrgyzstan since 2006, methods to develop tasks and pairs of inverse tasks, examples of tasks and instructions for organizing such competitions are proposed.

Key words: internet olympiads, off-line olympiads, individual tasks, team competitions.

1. Introduction

Undoubtedly, using the Internet to organize competitions for pupils yields large advantages and possibilities including involving participants of different countries, and in fact of the entire world. However, on-line competitions demand vast and complex equipment and use more resources. Hence, we propose to conduct some competitions off-line. Also, the main defect of Internet competitions is the opportunity to obtain "help" from other persons. To diminish this defect we propose to use individually generated tasks. Also, using the Internet for team competitions yields the possibility of forming commands of distantly separated members, as it was proposed (Pankov, 2006) in the form of necessarily-collective ones: each teammate works at a separate computer; they can communicate only by means of a computer network across a central computer-server; a goal of the competition can be achieved if and only if each teammate fulfils their own task.

To ensure these terms we also propose to use individually generated inverse tasks.

Section 2 contains definitions of extended tasks and individually generated tasks.

Section 3 contains definitions of inverse and semi-inverse tasks for team competitions, with examples.

Section 4 describes off-line individual competitions.

Section 5 reports about such competitions on applied mathematics (new genre) in Kyrgyzstan.

Section 6 stresses peculiarities of conducting necessarily-collective competitions.

2. Definitions of Extended and Individually Generated Tasks

In our observation, almost all young persons who could master programming begin to write testing programs performing the following standard "method of multiple choice": the student is shown a question and some (three-five) preliminary written answers and is to choose a right answer. This method is easy for programming but instigates the student not to solve the task but to "guess" the right answer. Also, implementation of this method does not give opportunity to evince programming skills and knowledge of subjects previously studied. Attempts of diversification of this method are reduced to a random choice in data base with ready tasks and to a random permutation of answers.

We, with our students, implemented some principles which gave the opportunity to develop software involving various peculiarities of the subjects studied, and examining various abilities of persons tested.

For brevity, we shall use the term "test" to mean any kind of control of quality of education, in a more general sense than a common one.

Other than common demands of tests (Validity, Objectivity and Reliability), to improve efficiency of testing, we offer the following ones:

Generativity: a complete text (content) of a task must not exist before testing, and must be generated randomly just before it.

Uniqueness: all students taking the test must obtain different versions of tasks (of the same level of difficulty or of different levels, according to the teacher's will).

DEFINITION 1. An extended task is an algorithm generating different logically correct and methodically proper tasks (of same level of difficulty) and corresponding right answers from initial data, (randomly) chosen from finite but sufficiently large sets (ranges).

This general definition was offered by Pankov, Janalieva (1995).

Remark. Extended tasks do not conform to a well-known class of "problems with parameters". Each participant receives a concrete task.

Remark. Random (occasional) tasks were used in various kinds of testing. For example, occasional commands are being given at military manoeuvres. We proposed to use this technique for all subjects and to implement it by means of computer.

DEFINITION 2. The number of ranges being used for generating the task in sufficiently different ways is said to be the dimension of the extended task.

DEFINITION 3. An algorithm permitting one (a teacher) to choose subsets of sets of initial data and generating different logically correct and methodically proper tasks by initial data, (randomly) chosen from these subsets is said to be an adjustable extended task.

EXAMPLE 1 of a mathematical task (well-known type of algebraic tasks).

1) Randomly choose $V \in 10 \ldots 20$ (boat's velocity), $W \in 2 \ldots 9$ (current velocity), $D \in 5 \ldots 10$ (time of boat's steaming downstream) and $U \in 5 \ldots 10$ (time of boat's steaming upstream).
2) Calculate $R := D(V + W) - U(V - W)$.
3) If $R = 0$ then go to 1.
4) Randomly choose $L \in 1 \ldots 4$.
5) Form the text of task using three of four values V, W, D, U. If $L = 1, 2, 3, 4$ then the unknown (right answer) A is V, W, D, U correspondingly, as follows:
«Boat steamed from the pier ... hours downstream and ... hours upstream and stopped at $|R|$ km below/upstream (by the sign of R) the pier. Boat's velocity is ... and/or current velocity is ...». «Find ...».
So, this extended task is five-dimensional.
Version for current testing by computer:
6) Output the text «Input answer».
7) Input $A1$.
8) If $A1 = A$ then output the text: Congratulation: you have solved the task."
else output the text: "Unfortunately, you are wrong".

Version for off-line competition:

6) choose random numbers B, C and calculate any complex expression $N = F(A, B, C)$ such that M can be found from this equation uniquely, for instance $F(A, B, C) = B \cdot M^3 + C \cdot M - A^2$;
   - output the text «Write your answer and numbers 'B', 'C', 'N' into the file of your answers».
   - stop.

Before the deadline of the competition the participant sends the file with answers to the jury. The deciphering program calculates the value of A by means of the values of B, C, N for the jury.

EXAMPLE 2 to create a "black box" type program for interactive solving.

1) Output the text:
«There is a cubic polynomial $P(X) = AX^3 + BX^2 + CX + D$ with integer coefficients. Requesting non-zero integer values of X and analyzing the corresponding values of $P(X)$ detect the polynomial. The fewer queries you make, the higher right answer will be estimated».
2) Randomly choose
$A \in \{-3, -2, 2, 3\}$, $B \in -9 \ldots -1 \cup 1 \ldots 9$,
$C \in -9 \ldots -1 \cup 1 \ldots 9$, $D \in -9 \ldots -1 \cup 1 \ldots 9$.
3) Let $M := 0$.
4) Output the text:
«Input non-zero integer value of X in the range $-100 \ldots 100$ or 0 for output»

5) Input X.
6) If $X \neq 0$ then let $M := M + 1$, calculate and output $P(X)$ and go to 4.
   Version for current testing:
7) If $X = 0$ then output the text «Input A, B, C and D».
8) Input $A1, B1, C1, D1$.
9) If $A1 = A$ and $B1 = B$ and $C1 = C$ and $D1 = D$ then output the text:
   "Congratulation: you have solved the task in 'M' queries."
   else output the text: "Unfortunately, you are wrong".

Version for off-line competition:

7) If $X = 0$ then
   - calculate any complex expression $N = F(M, A, B, C, D)$ such that M can be found from this equation uniquely, for instance $F(M, A, B, C, D) = B \cdot M^3 + C \cdot M + D - A^2$;
   - output the text «Write found values A, B, C, D and the number 'N' into the file of your answers».
   - stop.

Before the deadline of the competition the participant sends the file with answers to the jury. The deciphering program calculates the value of M by means of the values of A, B, C, D, N for the jury.

This class of tasks is four-dimensional.

DEFINITION 4. An individually generated task consists of two algorithms. The first algorithm composes different, logically and methodically correct tasks using personal information about a contestant. The second algorithm generates corresponding right answers using the same information.

Correspondingly, the contestant is to write their personal data and found answers into the file of answers and the second algorithm generates right answers for checking using the same data.

The same data are printed into the certificate.

3. Definitions of Inverse and Semi-Inverse Tasks

A task T in general can be presented as follows:

"Given $G(T)$ (with some explanations); find $A(T)$" (if the answer is single-valued)
and

"Given $G(T)$ (with some explanations); find any element of the set $A(T)''$ (if the answer is multiple-valued).

If it is an examination or a competition then "... in appropriate time" is to be added.

DEFINITION 5. A task T_2 is said to be inverse to the task T_1, if $G(T_2) = A(T_1)$ and $A(T_2) = G(T_1)$.

The first teammate solves the task T_1, the jury transfers their solution $A'(T_1)$ to the second teammate, the second teammate solves the task T_2' with the initial data $G(T_2') = A'(T_1)$ and the jury compares their solution with $G(T_1)$.

But the teammates can find opportunity to communicate $G(T_1)$ too (for instance, by mobile telephone). Hence, we propose also

DEFINITION 6. A task T_2 is said to be semi-inverse to the task T_1, if there is a (simple) algorithm M such that $G(T_2) = M(A(T_1))$ and $G(T_1) = M^{-1}(A(T_2))$.

The first teammate solves the task T_1, the jury calculates $M(A'(T_1))$ for their solution $A'(T_1)$ and transfers it to the second teammate, the second teammate solves the task T_2' with the initial data $M(A'(T_1))$: $A(T_2')$ and the jury calculates $M^{-1}(A(T_2'))$ and compares it with $G(T_1)$.

Example of an inverse task with single-valued answer:

EXAMPLE 3. *Task* T_1. Given three segments cut by the plane on the coordinate axes X, Y, Z. Find coordinates of the point on this plane being closest to the origin of coordinates.

Task T_2. Given the point on the plane being closest to the origin of coordinates. Find three segments cut by this plane on the coordinate axes X, Y, Z.

Example of an inverse task with multiple-valued answer:

EXAMPLE 4. *Task* T_1. Given three segments cut by the plane on the coordinate axes X, Y, Z. Find coordinates of three non-collinear points lying on this plane but not on coordinate axes.

Remark. In this task, the jury is to check the condition "not on the coordinate axes" before transferring.

Task T_2. Given three points lying on the plane. Find three segments cut by the plane on the coordinate axes X, Y, Z.

Example of an semi-inverse task with single-valued answer:

EXAMPLE 5. *Task* T_1. Given three segments cut by the plane on the coordinate axes X, Y, Z. Find three coordinates of the point on this plane being closest to the origin of coordinates.

Algorithm M. Multiply three coordinates by a non-zero number m.

Task T_2. Given three coordinates of a point on the plane being closest to the origin of coordinates. Find three segments cut by this plane on the coordinate axes X, Y, Z.

Algorithm M^{-1}. Divide three obtained numbers by the number m.

Combinations of above definitions yield also "extended inverse tasks", "individually generated inverse tasks", etc.

4. Organization of Off-Line Competitions

4.1. Some days before forthcoming competition, applications are collected.
4.2. 20–30 minutes before the beginning of the competition the executable file with individually generated tasks is sent to all participants and confirmations when obtained and successfully run are collected.
4.3. When the competition starts, the password for the opening the executable file is sent to all participants.
Comment. Sending of the executable file and communication of the password are separated because communication of the password requires less time and has less chance of any difficulties.
4.4. Participants solve the tasks and fill in files with answers.
4.5. Before the deadline participants send files with their personal data and answers.
Remark. Earlier submitting of these files is encouraged with additional points (it is announced in advance).
4.6. Jury inputs all participants' information and some complementary information into the deciphering and estimating second program (Algorithm 2 for all the tasks).
4.7. Jury sends total table with winners' list, notes about right way of 'tasks' solutions, best works, some mistakes and winners' certificates to all participants.

There is no reason to do other's work, because winners get certificates only.

5. Olympiads on Applied Mathematics

Using the methods discussed, since 2006 the Applied Mathematics and Informatics chair jointly with the Information and Computing Technologies chair of the Kyrgyz–Russian Slavic University has conducted open competitions in applied mathematics both for KRSU students and for students of other universities.

Here are some tasks. Letters denote individually generated positive integer numbers in appropriately selected ranges.

Task 1. Find the length of the arc of the parabola $y = Ax^2$ $(B \leqslant x \leqslant C)$ with accuracy 0.1.

Remark. This easy task gave the paradoxical, although predictable result. All students who knew the integral formula for arc length made mistakes in the calculations and the results were meaningless but most of the students, who did not know this formula, solved the task correctly.

Task 2. *KRSU_1.exe* program is given. It calculates one of the towns in Kyrgyzstan using personal data. Further the contestant inputs arbitrary shifts along the earth surface (receiving corresponding feedback), but not far from Kyrgyzstan until s/he reaches the capital Bishkek with the KRSU. «Guess where you were first, and write the answer either in the form: a) to the east (west) ... km and to the north (south) ... km from the KRSU (less points), or b) by the name of the town (more points)».

Task 3. A right-angled triangle with legs A and B rotates around a line passing through the apex of the right angle, and lying in the plane of the triangle. Find the greatest possible volume of the resulting solid of revolution with accuracy 1%.

Task 4. There is a mixture of two radioactive isotopes with different half-lives. For a given a) $0 \leqslant t < 50$ or b) $1 \leqslant t < 50$ the program (exe-file) gives the mass of the mixture at time t after the start. Find the mass of each isotope in the initial time with an accuracy of 1%.

Task 5. With a piece of plasticine of volume A cm^3 and a ruler, measure an approximate value of π by as many as possible sufficiently different ways. For each method, write the idea, giving results of measurements and calculations, and presenting one or two photos.

6. Organization of Off-Line Team Competitions

There are two types of such competition for pairs.

1) participants are of same educational institution;
2) participants are of different towns or countries.

In the first case, participants form teams by their own will.

All first teammates sit in the first computer class, all second teammates sit in another one.

Individually generated semi-inverse tasks are used.

Tasks are more difficult, only one tour (2–3 hours) is conducted.

In the second case, teams are formed randomly, no pairs from a same town. Participants do not know who their teammates are before the end of the competition.

Individually generated inverse tasks are used.

Tasks are easier; some tours (20–40 minutes each) are conducted. For each tour, new pairs are formed. Thus, the sum of points of every participant in some pairs shows their average level.

7. Conclusion

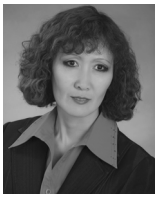
We hope that the organization of the competition by the proposed technique will expand the range of distance competitions in general, simplify their organization, make them more diverse, and evaluation of the results more objective. We are grateful to the KRSU administration for the support of olympiads in applied mathematics as a new genre of competitions.

References

Janalieva J.R. (2009). Development of software for the examination on the basis of generated tasks. In: *Proceedings of the VI International Scientific-Practical Conference. Intellectual Technologies in Education, Economics, Management*, Voronezh, Russia, 330–335 (in Russian).

Pankov, P.S., Janalieva, J.R. (1995). Experience and perspectives of using UNIQTEST complex of unique tests in the learning process. In: *Theses of Reports of Scientific-Practical Conference Education and Science in New Geopolitical Space*. International University of Kyrgyzstan, Bishkek (in Russian).

Pankov P.S. (2006). Necessarily-collective computer competitions for schoolchildren. In: *Information Technologies at Schools. Proceedings of the Second International Conference on Informatics in Secondary Schools. Evolution and Perspectives*, Vilnius, Lithuania, 585–588.



J.R. Janalieva (1969), doctor of pedagogical sciences, conducts various competitions on mathematics, informatics and languages including collective ones for students of Bishkek and Internet ones for students of Kyrgyzstan. Graduated from the Kyrgyz State University in 1991, she works as a docent of the International University of Kyrgyzstan.

Olympiads in Informatics, 2012, Vol. 6, 178–191

Competitions' Tasks in Informatics for the "Pre-Master" Group of School Students

Emil KELEVEDJIEV[1], Zornitsa DZHENKOVA[2]

[1]*Institute of Mathematics and Informatics, Bulgarian Academy of Sciences*
Akad. G. Bonchev str., block 8, 1113 Sofia, Bulgaria
[2]*Ivan Vazov High School*
54 Mitko Palauzov str., 5300 Gabrovo, Bulgaria
e-mail: keleved@math.bas.bg, zornica.dzhenkova@gmail.com

Abstract. Some features of informatics tasks given at the Bulgarian National Competitions for the "pre-master" school students group (9–10th grades in 12 grade school, i.e., 16–17 year old students) are discussed. The study covers the period 2004–2011. The considered group of students is important, because among them will appear candidates for the national team in the next one or two years. While tasks for the "master" group are complicated and not easily made into a system, the tasks examined here are more suitable to be analyzed. In the paper, an attempt is made to arrange these tasks using keywords, and to introduce a coefficient for measuring relative difficulty. The work continues two previously published papers of the same authors in this journal, devoted to the tasks for the intermediate and youngest students.

Key words: tasks in competitive informatics, informatics for the school students.

1. Introduction

In recent years, the competitions in informatics have been continually expanding and involving various age groups. This process can be observed in Bulgaria, as well in many other countries in the world. An example of this development is the establishment in 2007 at Belgrad, Serbia, of a new kind regional Balkan Youth Olympiad in Informatics for students up to 15.5 years old, as well the International Autumn Tournament in Shumen, Bulgaria, starting in 2009, where the competition includes two age groups – juniors and seniors.

In Bulgaria since 2001, several age group systems have been applied to divide school students for the national informatics competitions (the Autumn, Winter, and Spring Tournaments, as well for the three rounds of the National Olympiad in Informatics). Starting in 2002, groups were introduced with letter names: A, B, C, and D, which comprised 11–12, 9–10, 7–8, and 4–6th school grades, respectively. Now, we have a slightly modified system with 5 age groups A, B, C, D, and E, that cover 11–12, 9–10, 7–8, 6, and 4–5th school grades, respectively.

2. Systematization and Classification

After accumulating an archive of enough tasks (Websites *Infoman* and *Infos*) previously given at competitions, it becomes possible to start attempting classification using keywords. Information obtained in this way may serve the needs of teaching, and also for the preparation of future competitive tasks. The produced table is presented at the end of this article. We have to mention that, with the development of informatics competitions in the world, research on competition tasks has appeared (Verhoeff, 2009). The authors of the present article continue here their own ongoing research (Kelevedjiev and Dzhenkova, 2008a, 2008b, 2009), where they covered age groups from 4 to 8th school grades (groups E, D and C).

Tasks for the age group considered here have an intermediate character in the interval between the easy for the group C and the essentially more difficult, with Olympic features tasks, for the group A. While the A group tasks often have complex nature, which leads to difficulties in their classification, the tasks of the group B in most of the cases are more easily identifiable.

The curriculum for the master group A in the Bulgarian olympiads coincides with the curriculum of IOI, while topics expected for tasks of the group B are easier. The reader may get some pictures of themes for this group in Table 1, where the curriculum for the Bulgarian National Training Camps is presented.

For our classification, as seen from the Table 2, the main subdivision is made by the 9 main characteristics presented below. An important note is that this subdivision is not a strict classification, because many of the tasks of a given type have indications of the tasks to another type – e.g., tasks of type "graphs" are often solved by a method of dynamic programming. We are reflecting this multiformity in the "addition" column of Table 2.

1) *Graphs* (20 tasks). From the widely developed algorithmic area of the theory of graphs, prevalent methods in considered tasks are for finding the shortest path, determining the connectivity, cycles, topological sorting and minimum spanning trees. A part of these tasks relate to geometrical graphs, and methods of solving are dynamic programming, divide and conquer, and etc.

2) *Dynamic programming* (20 tasks). In this section we treat tasks, which are directly related to this basic approach to the compilation of algorithms. We mention that tasks of the other sections can also often be solvable with this approach.

3) *Combinatorics* (12 tasks). Contains tasks related to non-equivalent configurations, coding, representation of numbers, etc. One of the tasks requires "only an output file".

4) *Geometry* (12 tasks). Covers topics related to polygon's area, convex hull, crossing line segments, etc. Some of these tasks are solved with the approach of recursion and with elementary numerical methods.

5) *Search and sorting* (10 tasks). This group is related to a broad theme, containing binary search, exhaustive search, application of greedy approaches and some elementary numerical methods.

6) *Arithmetic* (5 tasks). These tasks require knowledge for divisibility of numbers, fractions, numerical systems, etc.

7) *Games* (4 tasks). Small group containing tasks to search for strategies in games with used the tabular methods, odd-parity methods, etc.

8) *Data Structures* (4 tasks). Here are allocated tasks, in whose solution the main approach is aimed at organizing the data, with which to improve response time for necessary operations. We see that in the given tasks are used the following data structures: Queue, Pyramid, Priority Queue and System of Disjoint Sets ("Union-Find").

Table 1

Curriculum for the Bulgarian National Training Camps, group B

	Topics	Year	Hours
1	STL containers and iterators.	1	4
2	Permutations.	1	4
3	Combinatorial configurations. Encoding and decoding.	1	4
4	Data structures for set presentations.	1	4
5	Algorithmic geometry. Intersections of points and lines.	1	4
6	Algorithmic geometry. Polygons. Convex hull.	1	4
7	Hash tables.	1	4
8	Graphs. Bi-connectivity. Strong connectivity.	1	4
9	Graphs. Euler's and Hamilton's cycles.	1	4
10	Graphs. Minimum spanning trees.	1	4
11	Graphs. Critical paths.	1	4
12	Catalan's numbers.	1	4
13	Dynamic programming tasks.	1	4
14	Games. Min-max strategies.	1	4
15	Recursion tasks. Recurrences.	1	4
16	Strings. Searching. Distances.	1	4
17	Data compression. Huffman's codes.	1	4
18	STL algorithms.	2	4
19	Combinatorial configurations on sets.	2	4
20	Gray's sequences.	2	4
21	Data structures for string algorithms.	2	6
22	Introduction to networks and maximal flows.	2	4
23	Partitions of numbers.	2	4
24	Partitions of sets.	2	4
25	Matching in graphs.	2	4
26	Algorithmic geometry. Closets and farthest points.	2	4
27	Special trees.	2	4
28	Strings. Syntax analysis.	2	4
29	Graph coloring. Planar graphs. Geometric graphs.	2	4
30	Introduction to formal grammars and automata.	2	6
31	Recursion tasks.	2	4
32	Games. Alpha-beta pruning.	2	4
33	Dynamic programming tasks.	2	6
34	System of linear equations.	2	4

Table 2

Classification of the tasks given to National Competitions in Informatics in Bulgaria (2004–2011) for the group B

	Task & competition			x	y	k	Type	Addition
1	fib	R2	2010	64	25	-0.44	aritmetic	
2	div	AT	2004	10	25	0.43	aritmetic	divisibility
3	seq	WC	2008	3	90	0.94	aritmetic	digits
4	even	ST	2009	2	94	0.96	aritmetic	number systems
5	dfrac	R2	2008	1	76	0.97	aritmetic	fractions
6	polygon	WC	2011	17	25	0.19	geometry	
7	abc	AT	2006	26	42	0.24	geometry	recursion, fractions
8	green	AT	2005	37	62	0.25	geometry	polygon area
9	rtri	AT	2006	20	75	0.58	geometry	integer points
10	lines	AT	2008	10	77	0.77	geometry	segment intersection
11	hull	WC	2004	4	95	0.92	geometry	convex hull
12	segment	R2	2006	3	91	0.94	geometry	
13	plate	WC	2008	3	93	0.94	geometry	numerical method
14	points	R2	2004	1	95	0.98	geometry	
15	lines	AT	2007	0	79	1.00	geometry	integer points
16	mov	ST	2007	0	70	1.00	geometry	segment intersection
17	walk	R2	2005	0	100	1.00	geometry	numerical method
18	ban	ST	2011	50	27	-0.30	graphs	connectivity
19	apples	WC	2010	60	34	-0.28	graphs	dynamics in graphs
20	alei	WC	2011	43	53	0.10	graphs	cycles in graphs
21	repair	ST	2010	25	45	0.29	graphs	MST
22	nails	R2	2011	26	67	0.44	graphs	connectivity
23	wght	ST	2004	24	65	0.46	graphs	
24	circle	ST	2008	24	68	0.48	graphs	shortest paths, geometry
25	recon	AT	2010	17	58	0.55	graphs	divide and conquer
26	circle	R2	2007	17	70	0.61	graphs	topological sorting, geometry
27	rings	AT	2008	10	57	0.70	graphs	shortest paths
28	folders	AT	2005	10	75	0.76	graphs	trees
29	race	ST	2010	10	75	0.76	graphs	shortest paths
30	triangles	R2	2009	11	81	0.76	graphs	topological sorting, geometry
31	trees	ST	2007	4	40	0.82	graphs	trees
32	trip	ST	2006	5	66	0.86	graphs	
33	avio	R2	2004	5	92	0.90	graphs	shortest paths
34	amb	R2	2009	2	92	0.96	graphs	shortest paths
35	edge	WC	2008	1	76	0.97	graphs	cycles in graphs
36	graths	WC	2005	1	97	0.98	graphs	shortest paths
37	obez	ST	2005	0	100	1.00	graphs	
38	shift	R2	2010	53	46	-0.07	DP	
39	salam	ST	2011	40	36	-0.05	DP	
40	calc	R2	2011	26	42	0.24	DP	
41	skok	WC	2007	34	61	0.28	DP	
42	stambs	R2	2008	33	66	0.33	DP	
43	tab	R2	2007	22	59	0.46	DP	
44	balls	ST	2006	22	69	0.52	DP	
45	brd	R2	2004	13	71	0.69	DP	
46	sum	AT	2004	8	55	0.75	DP	
47	jobs	R2	2010	10	75	0.76	DP	

To be continued

Continuation of Table 2

	Task & competition			x	y	k	Type	Addition
48	palind	R2	2011	9	82	0.80	DP	
49	flat	ST	2009	5	62	0.85	DP	
50	tre	ST	2007	4	61	0.88	DP	
51	knapsack	WC	2010	5	81	0.88	DP	
52	march	WC	2005	6	91	0.88	DP	
53	points	AT	2010	5	84	0.89	DP	bit masks
54	jumps	AT	2007	3	83	0.93	DP	
55	plate	R2	2005	0	71	1.00	DP	
56	ldist	WC	2006	0	96	1.00	DP	strings
57	king	WC	2009	0	97	1.00	DP	
58	handshakes	AT	2009	22	45	0.34	other	
59	average	AT	2004	20	60	0.50	other	algebraic transformations
60	fib	AT	2006	13	73	0.70	other	algebraic transformations
61	moves	ST	2004	9	86	0.81	other	integer points
62	irc	ST	2004	0	100	1.00	other	
63	inter	WC	2009	0	100	1.00	other	programming language
64	game	WC	2004	27	64	0.41	games	
65	phrope	ST	2006	5	69	0.86	games	
66	even	R2	2009	4	90	0.91	games	
67	decbin	AT	2008	2	87	0.96	games	number systems
68	sum	WC	2007	53	29	-0.29	combinatorics	
69	coins	AT	2009	45	32	-0.17	combinatorics	
70	toto	ST	2010	37	57	0.21	combinatorics	
71	code	WC	2010	23	55	0.41	combinatorics	output only
72	keokak	R2	2006	13	83	0.73	combinatorics	
73	square	ST	2005	11	79	0.76	combinatorics	
74	substr	WC	2011	10	82	0.78	combinatorics	strings
75	six	ST	2005	8	88	0.83	combinatorics	
76	squ	ST	2009	8	91	0.84	combinatorics	non-equvalent configurations
77	cart	R2	2005	5	90	0.89	combinatorics	presentations of numbers
78	signals	WC	2005	4	91	0.92	combinatorics	coding
79	perm	AT	2007	1	27	0.93	combinatorics	
80	y1984	WC	2006	31	51	0.24	DS	union-find
81	music	WC	2007	21	61	0.49	DS	priority queue
82	compATition	AT	2010	2	92	0.96	DS	pyramid
83	n23	AT	2005	0	82	1.00	DS	queue, divisibility
84	common	ST	2011	63	13	-0.66	srch. & srt.	
85	sqsum	WC	2009	37	24	-0.21	srch. & srt.	search in a table
86	table	WC	2004	30	50	0.25	srch. & srt.	exaustive search
87	dist	R2	2007	23	58	0.43	srch. & srt.	
88	riddle	AT	2009	12	65	0.69	srch. & srt.	greedy, binary search
89	points	R2	2006	12	72	0.71	srch. & srt.	
90	parATo	ST	2008	8	80	0.82	srch. & srt.	
91	triangle	ST	2008	7	73	0.83	srch. & srt.	geometry
92	plates	R2	2008	4	94	0.92	srch. & srt.	
93	Festb	WC	2006	0	100	1.00	srch. & srt.	interpolation, binary search

Legend: R2 – Round 2 of the National Olympiad (District round), AT – Autumn Tournament, WC – Winter Competitions, ST – Spring Tournament, DS – data structures, srch. – search, srt. – sorting, DP – dynamic programming, MST – minimum spanning tree.

9) *Other* (6 tasks). There are tasks of various types, and it is difficult to classify them to any of the above types. To note, for some tasks it was necessary to apply secondary school algebraic technique and for some other tasks it was necessary to apply modeling.

Using tasks' names in the Table 2, the reader can find in Websites *Infoman* and *Infos* the full descriptions of the tasks, which are representatives of the above described types (with problem statements, translated also in English, test examples, authors' solutions and, in most cases, detailed analysis).

3. Examples of Tasks

As examples, we present abridged formulations of typical tasks, which represent the above groups. Task's numbers are taken from Table 2. The coefficient k is explained in the next section; a values of about 0.5 mean that the task is "typical".

3.1. *Graphs*

Example 1. Task 24, $k = 0.48$ (This task includes also geometric graphs, shortest path). In the plane are given n circles: $C_1, C_2, \ldots, C_n$. Consider the undirected graph with vertices at given circles and there exits an edge between two circles, when both circles have exactly two common points. Write a program that computes the number of edges in the shortest path from C_1 to C_n. The first line of standard input contains $n (2 \leqslant n \leqslant 1000)$; each of the next n lines contains 3 integers $x,\ y,\ r$, giving the coordinates of the center and radius of a circle (x, y are in the range $-10000, \ldots, 10000$, and $0 < r < 10{,}000$).

Example 2. Task 25, $k = 0.55$ (This task includes also "Divide and conquer"). The company of Peter has designed a new space station, composed of N modules, labeled from 1 to N. Some pairs of different modules are linked by corridors, such that it is possible to go from each module to each other by a unique path of corridors. The length of each corridor is a positive integer. There is no more then one corridor linking two modules. The chiefs of Peter would like to keep in secret the project. That is why Peter encoded the topology of the station giving, for each two modules, the distance between them (i.e., the sum of the lengths of the corridors on the unique path between the modules). Write a program to decrypt the coding of Peter and to reconstruct the topology of the station. The first line of the standard input contains the number N of the modules ($3 \leqslant N \leqslant 1024$). Then $N - 1$ lines follow. On the first of these lines, the distances from module 1 to modules 2, 3,..., N are given. On the second line are given the distances from module 2 to modules 3, 4,..., N, and so on. The last line contains the distance from module $N - 1$ to module N. All distances are positive integers not grater than 1024. The program has to print N lines. The first line has to contain the list of the neighbors of module 1, i.e., the modules that are linked with corridors to it. The list has to start with the number L of neighbors, followed by their labels, sorted in increasing order. All numbers has to be separated by single spaces. On the second row of the output, formatted in the same way,

the list of the neighbors of module 2 has to be printed, and so on. The output has to finish with the list of the neighbors of module N.

3.2. *Dynamic Programming*

Example 1. Task 43, $k = 0.46$. Given is a table with M rows and N columns ($0 < M < 1000, 0 < N < 1000$). Each cell contains an integer in a range from -100 till 100. We consider all sub-tables with m rows and n columns that contain successive rows and columns ($0 < m < M,\ 0 < n < N$). Write a program that inputs M, N, m, and n, the integers in the given table, row by row, and outputs the maximum sum of integers, which may be exist in some of the considered sub-tables.

Example 2. Task 44, $k = 0.52$. There are given n balls, arranged in a row along a straight line in a hall. Opposite, there are placed m empty baskets in a row parallel to the row of balls. Each ball and each basket is colored with one of the following colors: r – red, b – blue, g – green, w – white, p – pink, y – yellow. We play the game by taking successive balls in the order which they are arranged from left to right. We can return back a ball taken to the same place in its row, but we cannot get a ball that we have already put back, and we cannot go back in the row of the balls. A ball, that we do not put back, we put in the basket of the same color. This basket should be located just right of the last nonempty basket at the current moment (if we are at the beginning of the game, we may put the taken ball in any basket of the same color). Each basket can take at most one ball. The aim of the game is to score the maximum number of balls. Write a program that inputs two strings, containing some of the letters $r,\ b,\ g,\ w,\ p,\ y$. Each of the strings is no longer than 999 characters. The first string specifies the position of the balls and the second – the position of the baskets. The program has to output the maximum number of balls that can be placed in baskets according to the described rules.

3.3. *Combinatorics*

Example 1. Task 71, $k = 0.41$ (Output only). Consider all 5-letter words with letters 0, 1 or 2: $00000, 00001, 00002, \ldots, 22222$. The distance between two words is the number of corresponding elements which are different. For example, the distance between 01201 and 11011 is 3. Select as many words, so that the distance between any two of them is greater than or equal to 3. Save results to file.

Example 2. Task 72, $k = 0.73$. An artificial language has 4 sounds: A, E, O, and K. Each word may contain only those sounds and is correct, if the following four requirements hold: 1. A word has at least one vowel sound; 2. One after another cannot be two identical sounds; 3. A word has no more than two consecutive vowels; 4. The letter A cannot be followed by a vowel. For example, strings A, OA, EO, OKO, KOAK, AKEO and KAKA are correct words, but strings K, OKKA, KEOA, KOOK and KAO are not. A word is called "reversible" if it is correct and also correct when it read from right to left, also. For example, the words A, EO, OKO, AKEO, KAKA are reversible, and the words KOAK, OA and are not reversible. Write a program that inputs an integer n ($2 \leqslant n \leqslant 20$) and outputs the number of words with at most n letters that are not reversible.

3.4. *Geometry*

Example 1. Task 9, $k = 0.58$. A rectangle with side lengths m and n (m and n are integers, $0 < m < 30,\ 0 < n < 30$) is divided into mn squares, each with side 1. Each point that is a vertex of at least one of these squares, we will call a node. Write a program that inputs m and n, and outputs the number of rectangular triangles, whose vertices are nodes.

Example 2. Task 10, $k = 0.77$. In the plane are given N different line segments. Write program that computes which is the maximum number of the given line segment, that can be crossed by a vertical or horizontal line. The program inputs N, then N lines, each containing 4 integers with coordinates of segment's endpoints ($0 < N < 300000$, coordinates are in a range: $-10^8, \ldots, 10^8$).

3.5. *Search and sorting*

Task 87, $k = 0.43$. Given are positive integers M and N ($1 < M < 3000,\ 1 < N < 3000$) and we consider all points with integer coordinates (x, y) in the plane, for which $1 \leqslant x \leqslant M$ and $1 \leqslant y \leqslant N$. We construct all segments that have endpoints at two different considered points. Given is a decimal number $d,\ 0 < d < 10000$), and with at most 4 digits after the decimal point. Write a program that finds out a segment among the constructed ones which length is the closest to the value of d. A line of the standard input contains values of M, N and d. The program should output the matching value as a number with fixed decimal point with exactly 4 digits in its fractional part, obtained with rounding by cutting.

3.6. *Arithmetic*

Task 2, $k = 0.43$. Write a program, which inputs two integers N and P, $0 < N < 1000000001,\ 0 < P < 50001$, and outputs the largest integer M, such that $N!$ is divisible by P^M.

3.7. *Games*

Task 64, $k = 0.41$. Candies are divided into two piles, containing respectively A and B pieces. Two persons play a game with alternating moves, where each player on his turn must choose one of the piles and divide it into two parts. The other pile is given away and the game continued with both new piles. Of course, new piles may not have equal number of candies, but each must contain at least one candy. The game ends when the two new piles each contain one candy, i.e., the players cannot make more moves. The winner is the one, who made the last move. Write a program that inputs the number of candies A and B before the next move of a player $1 < A < 3001,\ 1 < B < 3001$). The program must determine whether the player who is to move loses the game if the opponent plays optimally (i.e., the best possible for himself to win) or the player can win no matter how

the other player plays. If the player, who is to move, loses the game, the program should output number 0, but if the same player can win, the program should output values C and D, which is both new piles after his move. Under the rules, each of C and D is greater than or equal to 1 and $C + D$ is equal to one of A, or B. Because the player may have several different possible moves, the program has to output those for which $C + D$ is the least possible and then to choose a move for which C is the least.

3.8. *Data Structures.*

Task 81, $k = 0.49$. (Priority queue) Each participant in a casting has an initial rating, which is expressed as a decimal fraction from 0 to 100, with an accuracy of 4 digits after the decimal point. The jury has to evaluate all participants for N days, $1 < N < 200$. The organizers made the casting, so that participants arrived in town only the night before the working day of the jury (including the night before the first day) and accommodated in a hotel. Every day, the jury chose to examine a number of participants, which was accommodated in the hotel, and which had the highest rating, but if there was such with equal rating, they are chosen according to the earlier registration at the hotel. Once a participant is examined, he immediately has to leave the hotel. The hotel has 10000 beds and never overflows during the time of the casting. There was left unexamined participants in the hotel after the Nth day, although new candidates were not accommodated at the night before the $(N + 1)$st day. Write a program that finds the name of the first unexamined participant, i.e., one that should be examined first in the $(N + 1)$st day, if the jury would have decided to work one day longer. The program inputs the number of days N, followed by N lines, each with a number M of participants arrived at the night before that day, $0 \leqslant M < 500$, followed by M pairs consisting of the name of a participant and his rating. The last line contains the number of participants K that the jury actually can examine for a day, $0 \leqslant K < 300$. Each name contains no more than 10 letters.

4. Assessment of the Relative Difficulty of a Task

Let us denote by x a percentage of the contestants received a score over 60 points (of 100 maximum possible) for a given task (i.e., solved the task "successfully") and by y a percentage of the contestants received a score less than 30 points (i.e., solved the task "failed"). We define a coefficient for **relative difficulty**: $k = (y - x)/(x + y)$.

This relative value varies between 1 and -1, and takes the its maximum when $x = 0$ and its minimum – when $y = 0$. In the first case the value is $k = 1$ and there are no competitors, which have solved the task "successfully" (all have worse results), and in the second case, when $k = -1$, no competitor has not solved the task "failed" (all have better results). So, when the value of k is close to 1, we can consider that the task is relatively difficult, but when $k < 0$, the task is easy.

From the presented data and graphics we may conclude, that at the national competitions in informatics very rarely are there tasks with a negative coefficient k. This is a natural expectation – at these competitions there are not given easy tasks. The defined

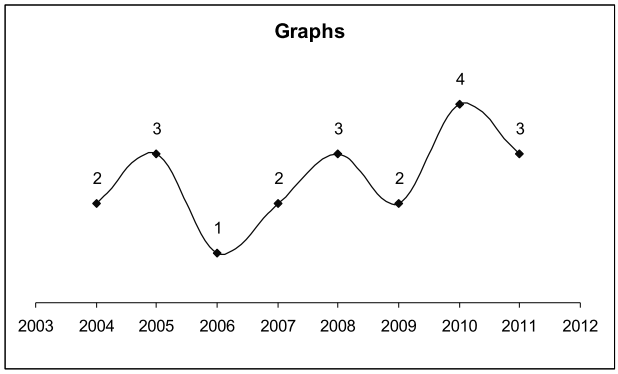


Fig. 1. Number of tasks of type "Graphs" in years.

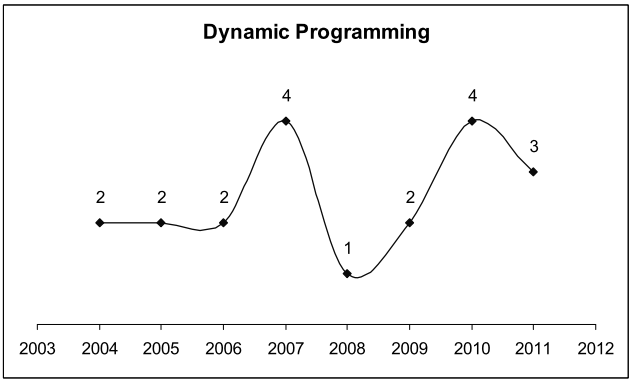


Fig. 2. Number of tasks of type "Dynamic Programming" in years.

coefficient has no direct connection with the absolute difficulty of the particular task, nor it is a measure of the feasibility of the competitors, but rather it shows to what extent the authors of the tasks are selected them in accordance with the age of the competitors.

We note that at the Bulgarian national competitions there is no rule that a tasks' set should contain a task that is easy (supposed to be solved by most participants) and a task that is intentionally chosen to be a hard.

5. Conclusion

The main topics in competition tasks for 9 and 10th school grades includes: Theory of Graphs and Dynamic Programming. To a lesser extent, there are tasks related to Combinatorics, Arithmetic, Search and Sorting, and Geometry. There is a trend to decrease in the number of tasks with geometrical nature, which is explained by the rare presence of these tasks in the international competitions.

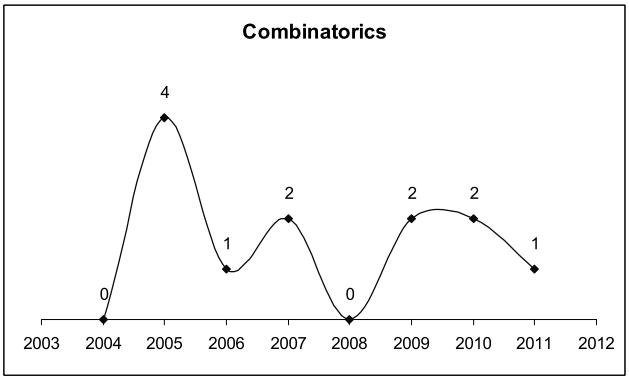


Fig. 3. Number of tasks of type ”Combinatorics” in years.

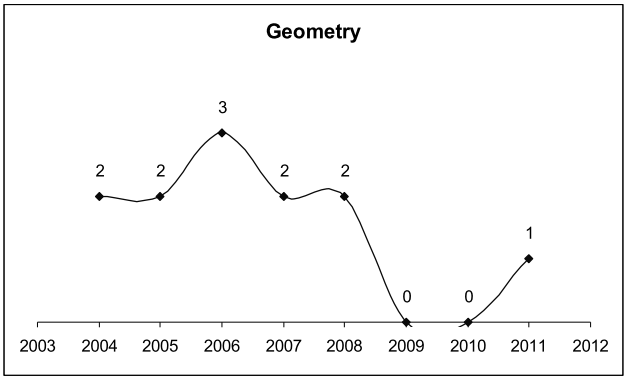


Fig. 4. Number of tasks of type ”Geometry” in years.

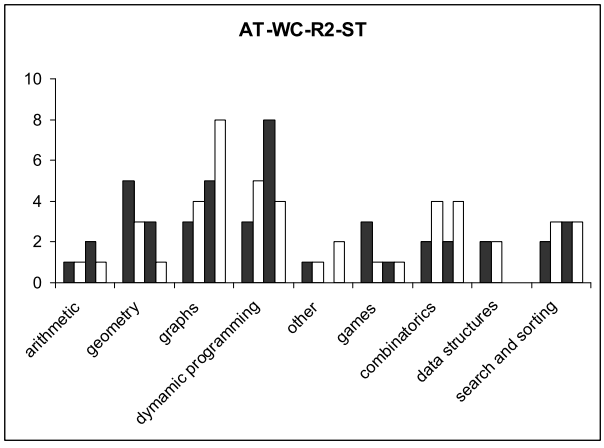


Fig. 5. Number of tasks according to their type, allocated in competitions (2004–2011).

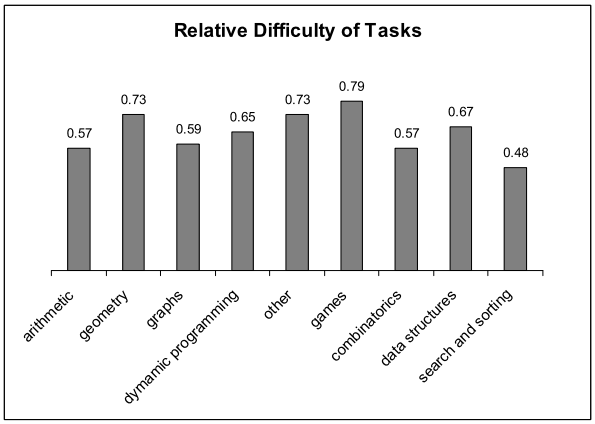


Fig. 6. Relative difficulty of the tasks, according to their type (2004–2011).

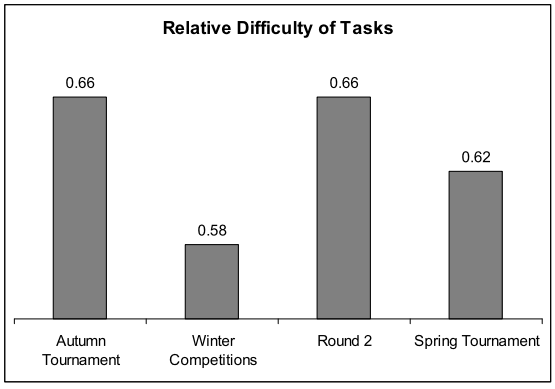


Fig. 7. Relative difficulty of the tasks, allocated in competitions.

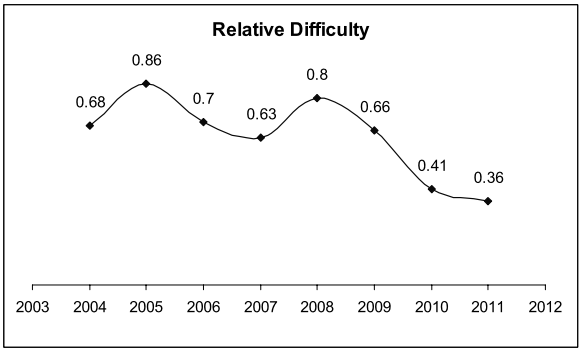


Fig. 8. Relative difficulty of all the tasks in years.

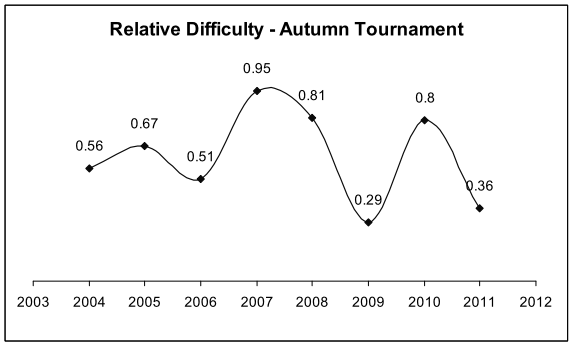


Fig. 9. Relative difficulty of the tasks, given at the Autumn Tournament in years.

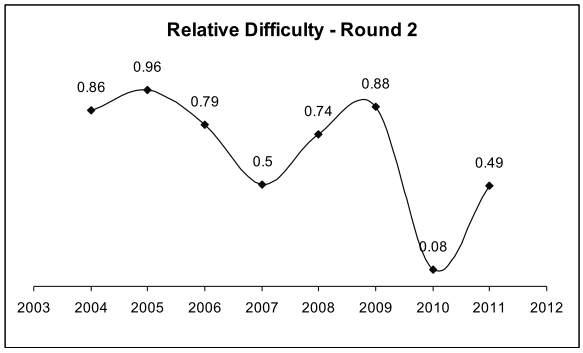


Fig. 10. Relative difficulty of the tasks, given at the Round 2 of the National Olympiad in years.

References

Kelevedjiev, E., Dzhenkova, Z. (2008a). Tasks and training the youngest beginners for informatics competitions. *Olympiads in Informatics*, 2, 75–89.

Kelevedjiev, E., Dzhenkova, Z. (2008b). Competition's tasks in informatics for the 4–7th school grades. *Mathematics, Informatics and Education in Mathematics and Informatics*, 37, 367–378 (in Bulgarian).

Kelevedjiev, E., Dzhenkova, Z. (2009). Tasks and training the intermediate age students for informatics competitions. *Olympiads in Informatics*, 3, 26–37.

Manev, K., Kelevedjiev, E., Kapralov, S. (2007). Programming contests for school students in Bulgaria. *Olympiads in Informatics*, 1, 112–123.

Verhoeff, T. (2009). 20 Years of IOI competition tasks. *Olympiads in Informatics*, 3, 149–166.

Website Infoman. `http://infoman.musala.com` (retrieved on 30 March 2012).

Website Infos. `http://www.math.bas.bg/infos` (retrieved on 30 March 2012).

E. Kelevedjiev is a research fellow in the Institute of Mathematics and Informatics at the Bulgarian Academy of Sciences. His field of interests includes algorithms in computer science, operation research, digitization techniques, etc. He was a member and now is the chairman of the Bulgarian National Committee for Olympiads in Informatics since 1993; leader of the Bulgarian teams for many IOI's and BOI's.

Z. Dzhenkova is a teacher in the Ivan Vazov High School in Gabrovo, Bulgaria. She is coauthor of a manual for beginner's training in competitions and Olympiads in Informatics. Her field of scientific interests includes Education in Informatics and Information Technology. She is a member of the Bulgarian National Committee for Olympiads in Informatics.

Olympiads in Informatics, 2012, Vol. 6, 192–198

Young Talent in Informatics

Preliminary Findings of an IOI Survey Launched by AICA in Cooperation with IT STAR

Plamen NEDKOV
Project Coordinator of the Survey
e-mail: nedkov@utanet.at

Abstract. The paper presents some preliminary results of the IOI-related Survey intended to examine the experience of several Central, Eastern and Southern European countries whose participation in competitions of the International Olympiad in Informatics is truly remarkable. It identifies several common features, which have contributed to this success. These include tradition in organizing and participating in informatics-related competitions, strong emphasis on mathematics in national education, further extra-curricular activities and individual training, early start in training and competing, dedicated and motivated individuals and organizations.

Key words: informatics olympiads, competitions, education, motivation.

1. Introduction

The International Olympiad in Informatics (IOI) is widely recognized as a leading international competition of algorithmic nature, in which national teams composed of secondary school pupils show such basic IT skills as problem analysis, design of algorithms and data structures, programming and testing.

On the occasion of the International conference on Young Talent in Informatics, organized in conjunction with the 24th IOI in Italy, and the 25th Anniversary of UNESCO's endorsement of IOI, AICA – the leading Italian Informatics Association in cooperation with IT STAR – the regional ICT Association in Central, Eastern and Southern Europe, launched a survey with the objective to examine and promote the experience of countries in Central, Eastern and Southern Europe whose IOI teams have shown remarkable results in IOI competitions. The findings, based on consultations, interviews and Internet research, are intended as contribution to the conference and will be further disseminated.

2. Approach

Competitions are embedded in the process of education. Whether in the classroom, or on a local, regional, national/international level, the constant monitoring of teaching and learning results is a useful practice to assess performance, gain new experience, introduce measures to improve the process and reward the best performers. In this regard, the

performance of national teams at IOI competitions could be viewed as indicative of ongoing processes in the broad fundament of the national education systems and the national network of mathematics and informatics related institutions.

In considering our project approach, we kept in mind that exact sciences are "difficult". There is a decline of ICT students in many European countries (Schagen, 2011). On the other hand, ICT competences are strategic assets for the development of Europe as a real knowledge society, and any broadening of the gap between the scarcity of ICT-Skills and the needs of informatics professionals and users by the economy might have serious consequences (Occhini and Nedkov, 2009). Policy-makers in Europe should take urgent notice of this fact and ensure the necessary measures and investments in education. We hope the findings of the project will be useful to this end.

This paper (and, indeed, the survey) is based on a questionnaire related to the selection, preparation and participation of national IOI teams of Bulgaria, Croatia, Latvia, Poland and Slovakia. These countries were chosen in order to have a representative sample of the whole of Eastern Europe – Latvia for the Baltics, Poland and Slovakia for Central Europe and Bulgaria and Croatia for South-Eastern Europe. With the exception of Poland, these countries are rather small in all counts but deliver remarkable results at international informatics competitions.

Their overall ranking, on the basis of IOI medals, is presented in Table 1.

Their IOI achievement is striking and they were our start-up point, however, the project is open to the experience of other countries in the region, which have done remarkably well in IOI. We hope to have on board in the near future material concerning Romania, 6th in the overall IOI ranking, the Czech rep. – 12th, Hungary – 15th, and other.

Chairpersons/leaders of the national bodies and IOI teams were invited to complete a questionnaire and to comment on such issues as the national team selection, coaching, communication and promotion, motivation and background for success. A section of the questionnaire was on informatics curricula in schools. In addition, personal interviews were organized in some of the countries. This was carried-out so as to gain a deeper understanding of the following:

Table 1

??? caption ???

#	Country	Gold	Silver	Bronze	Total
1	China	57	22	12	**91**
2	Russia	40	28	12	**80**
3	**Poland**	**31**	**27**	**23**	**81**
4	United States of America	30	30	14	**74**
7	**Slovakia**	**20**	**32**	**18**	**70**
8	**Bulgaria**	**18**	**32**	**30**	**80**
17	**Croatia**	**10**	**23**	**29**	**62**
23	**Latvia**	**5**	**19**	**33**	**57**

Source: `http://www.eduardische.com/ioi/`.

- How could countries of this region with small economies and tight budgets for education show consistently, within the IOI format, significantly higher results than the larger and richer economies of Western Europe?
- What are the driving forces of this achievement?
- What are the motivation factors?

A separate set of contacts was made with youngsters who are/were national team-members and have won medals in IOI competitions. The perspective of school teachers and trainers was sought to expand our knowledge of the successful organization and experience of these countries and the issues involved and we are of the opinion that the project could lead to further work with wider implications in education and beyond.

3. Findings

On the basis of our work so far we can identify the following groupings of factors contributing to the successful participation of the national teams of these countries in IOI competitions:

- *Tradition.*
- *Strong emphasis on mathematics in national education.*
- *Targeted extra-curricular activities.*
- *Early start and gaining experience by participating in competitions.*
- *Systematic management, dedicated people.*
- *Motivation and reward.*

3.1. *Tradition*

Math competitions were organized in the region as early as the end of the 19th century: a primary-school math competition was reportedly held in Bucharest, Romania in 1885 and in 1894 the Eotvos competition in Hungary set the model for math competitions of secondary school pupils. Mathematics journals were launched in both countries in the 90's of the 19th century. In 1934 a Mathematical Olympiad was organized in Leningrad, now St. Petersburg, Russia. The 1st International Mathematical Olympiad (IMO) was organized by Romania in 1959 with seven countries from Eastern Europe participating – the idea to organize such a competition matured during the 4th Congress of the Romanian Mathematicians in 1956 and provided a model for the organization of other competitions, including in the field of informatics. Today, IMO brings together competitors from over 90 countries (Kenderov, 2006).

The proposal for an International Olympiad in Informatics (IOI) was made by Blagovest Sendov[1] on behalf of Bulgaria, and endorsed by UNESCO's General Conference in 1987. The 1st IOI was held in Pravetz, Bulgaria in 1989 with the participation of teams from 13 countries (Kenderov, 2007).

[1] Mathematician, was rector of Sofia University and president of the Bulgarian Academy of Sciences

The respondents from the countries involved in our survey point out that they possess tradition and culture in organizing programming competitions – an important factor for the success of their teams, and an inspiration for further work.

3.2. *Emphasis on Mathematics in Schools*

The regular school curricula in mathematics and informatics are not sufficient to form good competitors in IOI related competitions, however, a solid mathematical knowledge base is certainly a tipping success asset – as a matter of fact, some of the competitors and winners of medals in past IOI competitions have done similarly well in IMO competitions.

The Eastern European model of secondary education during the 60s, 70s and 80s of last century has had a strong emphasis on the study of exact sciences. In addition, a network of gymnasiums specializing in mathematical education (similar to the model of foreign language schools in these countries) was established in the late 60s – early 70s. Such specialized math-oriented schools continue to function today, though, in some of our interviews it was made clear that the ongoing reorganizations of the educational field might negatively influence math education in secondary education in general, which might also reflect on the performance in math and informatics-related education and competitions. In the case of Slovakia, the reorganization of school education has led to a reduction of the weekly number of math teaching in basic and secondary school. On that backdrop, the results of the Slovak IOI teams suggest a decline with no gold medals won during the last 6 years.

3.3. *Extra-Curricular Activities*

There are some informatics related courses in the curricula of lower secondary (10–14 years) and upper secondary schools. This as mentioned above is not sufficient for the preparation of highly competitive participants in informatics related competitions. Several of the IOI competitors we interviewed said that on-line competitions, training, solving tasks from previous competitions, participation in correspondence seminars in programming, participation in special courses for math and informatics competitions, summer camps and other forms of preparation have contributed to their success.

There are many paths and activities to ensure a solid preparation for IOI-related competitions. We give the following for illustration:

In the case of Bulgaria and Latvia, private educational institutions are involved in preparing all who wish to compete: in Bulgaria, the A&B private school in Shumen was established with the objective to prepare pupils for national and international informatics related competitions (Mollov *et al.*, 2009), in Latvia, Progmeistars private school[2] has similar objectives. Both institutions have developed and introduced methodologies in delivering results and a confirmation of this is that pupils who have attended their courses have won gold medals during the last IOI-2011 in Thailand. A gold medallist who had

[2] `www.progmeistars.lv`

taken part in such courses said that this has helped him think as a programmer – an important detail related to the performance of these private schools.

One successful training activity in Slovakia is the Correspondence seminar in programming (CSP; Andrejkova, 2005). On the seminar's website one can find the problem sets and other information. The competition is of theoretical nature and is organized in rounds. Another form of training/competition for students from secondary schools is "PALMA" – Programming, algorithms, and mathematics. This is an on-line competition involving real programming and automatic evaluation. There is also PALMA Junior organized for pupils at basic school and prepared for one- or two-member teams.

The Polish Children's fund has an important function to work with talented children in various fields, including IT. Similarly, in Bulgaria the High School Student Institute (HSSI) was established by the Institute of Mathematics and Informatics, the Union of Bulgarian Mathematicians, Foundation "Eureka" and the International "St. St. Cyril and Methodius" Foundation with the objective to identify, develop and manifest the talent of pupils in mathematics and informatics. In Croatia, the programs of the Croatian Computer Science Association (CCSA), the main organizer of the selection and preparation of the national IOI teams, fully cover the IOI syllabus.

In most of these countries, the olympiad itself is an educational activity: materials are published after the events and contain detailed analysis, Internet portals are maintained and summer camps are organized.

3.4. *Early Start, Gaining Experience by Participating*

Most of the respondents to our questionnaire said there is no minimal age for participating in regional and national competitions, but in practice the youngest participants are from 10 to 14 years old. In our further interviews, opinions were expressed that, to be successful in IOI competitions, contestants should start at least at the age of 12–13 as at 15 it is already late.

Another aspect of this is that many contestants experience an initial "stage fright" related to their participation in international competitions. Gaining experience by participating is an important success factor and this is reflected in the record of top performers during the course of several consecutive years of participation in IOI competitions.

3.5. *Systematic Management, Dedicated People*

Formally IOI related matters are under the broad umbrella of the national ministries of education, yet in all countries there are specialized professional institutions, which have responsibilities to prepare and oversee the process of selection through internal competitions of the national teams, their additional coaching and participation in IOI competitions.

This is a tested process of meticulous organization based on several tiers of selection – school/local, regional and national competitions, summer schools, training camps and international competitions (Manev *et al.*, 2007; Diks and Madey, 2008). Teams from all

the countries involved in this survey take part in interregional competitions such as the Balkan Olympiad in Informatics, the Baltic Olympiad in Informatics and the Central European Olympiad in Informatics. For some countries (i.e., the Baltic OI for Latvia) these interregional competitions are an element in the selection process, for others (i.e., Poland in the case of the Baltic OI) they present opportunities for "younger" participants to gain experience.

In the core of this process are dedicated individuals – university professors and students, teachers, tutors, ex-competitors. Many of them have started their involvement as early as the first participation of their country in the IOI process. Some fears were expressed that it is hard to find "new recruitments" for this activity. At the same time, it was a pleasure to meet with the Latvian organizers during the recent 18th Baltic OI, 3–7 May 2012 in Ventspils, Latvia[3] and to observe that many ex-members of the Latvian IOI teams (currently students or young professionals) continue to be involved as organizers, deputy team leaders for recent IOIs, and as authors of tasks and other activities related to IOI.

The ministries for education provide some funds for the organization of competitions, training camps and international travel and associated activities, though these are reportedly far from sufficient. Additional funds are sought and sponsors attracted, very often on the basis of personal connections, in some cases also related to ex-competitors that have consequently done well professionally. Sponsorship from the IT Industry, foundations and other sources is an important aspect of the funding of IOI related activities in Bulgaria, Latvia and Poland.

3.6. *Motivation and Reward*

Motivation is in the core of success and all stakeholders in the process are motivated to take part and see opportunities in doing so. There are two types of participants – the pupils that go through the process and compete at IOIs, and their teachers, instructors, methodological leaders and organizers.

The competitors we interviewed are bright young men, yet no one of them considers himself as exceptional in any respect. After one session of interviews in Latvia I was asked jokingly by one of them whether this would make him famous. This reminded me of a reaction from Genadii Karatzkevitch (Belarus)[4], then 14 years old. Asked whether he has his own strategy of problem-solving his response was that he tries various ways and one works . . . followed by "I am no genius. I am simply good at it".

Yes, they are good at it and this is achieved with a lot of practice and exercises and the motivation comes from various sources including family and friends, teachers and tutors, computers and ICT, challenging problem-solving, learning from mistakes and improving, competing and winning, . . . One should also not forget the fun part of it – summer camps, meeting friends, local and international travel.

[3] `http://www.boi2012.lv/`

[4] `http://www.ioi2009.org/downloads/br8-3str-en.pdf`

When entering the path of international competitions, the link between success in IOI's and its impact on future professional development is hardly a fixation for anyone. But the possibilities are there and these competitors are gradually exposed to them – top performers are "noticed" and become heroes of sorts in their communities, travel more, have possibilities for internships in leading IT companies, receive awards and scholarships, gain easier access to top universities in their countries and internationally.

For the other type of participants – teachers, researchers, instructors, team-leaders – there are some financial awards, according to some of the respondents, though the highest reward for a good performance in our mind is professional satisfaction, accomplishment and recognition. Clearly, most of the persons involved in IOIs are professional academics and educators and their experience with IOI is directly reflected in their academic output.

References

Andrejkova, G. (2005). *E-Learning and E-Training of Students and Their Teachers in Informatics*. http://virtuni.eas.sk/rocnik/2005/data/program/64_15_Andrejkova.pdf.

Diks, K., Madey, J. (2008). *From Top Coders to Top IT Professionals*. http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/d/Diks:Krzysztof.html.

Kenderov, P.S. (2006). Competitions and mathematics education. In: *Proceedings of the International Congress of Mathematicians*, Madrid.

Kenderov, P.S. (2007). An insight of the first informatics olympiad. *IT STAR Newsletter*, 5(3). http://nl.starbus.org.

Manev, K., Kelevedjiev, E., Kapralov, S. (2007). Programming contests for school students in Bulgaria. *Olympiads in Informatics*, 1, 112–123.

Mollov, A., Yovcheva, B., Petrov, P. (2009). Internal contests as an element of the training of pupils for competitions in informatics. In: *Proceedings of the 38th Spring Conference of the Union of Bulgarian Mathematicians*.

Occhini, G., Nedkov, P. (Eds.) (2009). *B. Lamborghini in ICT Skills Education and Certification*.

Schagen, J.D. (2011). The ICCT-mindsets model – attracting young people to the world of ICT. *IT STAR Newsletter*, 9(4). http://nl.starbus.org.

P. Nedkov is chief executive of IT STAR and editor of the IT STAR Newsletter. His background is in international economics. He has edited many ICT-related publications and is involved in the organization of conferences. Plamen was head of Department for International Organizations at the Bulgarian Academy of Sciences and executive director of the Sofia Office, International Foundation for Survival and Development of Humanity. He served in various capacities in IFIP, including as contracts officer and member of the Congress Committee. He was IFIP executive director, delegate to many sessions of UNESCO's General Conference and elected representative to the NGO-UNESCO Liaison Committee.

Olympiads in Informatics, 2012, Vol. 6, 199–204

Comparisons of the IMO and IOI

Peter TAYLOR
Australian Mathematics Trust, University of Canberra
ACT 2601 Australia
e-mail: pjt013@gmail.com

Abstract. There are several International olympiads for secondary school students, but the five which are most widely recognised are those in mathematics, physics, chemistry, biology and informatics, in approximate order of founding. Most of these were originally founded under the auspices of UNESCO. Of these, the discipline most closely related to Informatics is mathematics, and in fact in several countries, for example Australia and Bulgaria (the latter of which can be credited as the founding country of IOI), the two olympiad programs are administered by the same organisation. The purpose of this paper is to compare the two olympiads in some detail, including tasks, topics, evaluation, etc. Despite the close relationship between the disciplines themselves, and the fact that there are similarities in the structure of the olympiads, in fact between all five, such as the medal structure, it is surprising also how different are the evolved traditions of IMO and IOI.

Key words: International Mathematical Olympiad (IMO), International Olympiad in Informatics (IOI), syllabus, task preparation, exam format, problem evaluation.

1. History

Of the five olympiads referred to, mathematics, through the IMO, is the oldest, having started in 1959, when the first IMO was held in Romania. This first IMO was only attended by Eastern Bloc countries, and this stayed the case for a few years until it gradually expanded. The first Western country to enter was Finland in 1965.

The IMO has grown strongly to the point where now more than 100 countries participate annually in most years. Rapid growth happened in 1988 when hosts Australia invited a number of new countries from the Asia Pacific, and in 1997 when hosts Argentina invited most Latin American countries in. The two strongest blocs by language are the Russian-speaking group and the Spanish-speaking group from Latin America.

By contrast the IOI is the equal youngest of the major five olympiads (with biology), having been first held in 1989 in Bulgaria. The IOI has grown to be the second largest, with over 90 countries now participating.

2. Exam Formats

The exam formats of mathematics and informatics are quite different, although the number of tasks is similar, over two days, each day 3 problems for IMO, and for IOI (although in 2009 and 2010 IOI added a fourth problem, designed to be very easy).

An informatics task will normally contain a lot of wording, and usually introduce the student to some new situation. Because IOI problems are solved on a computer, extra space is needed for descriptions say of data format. Informatics problems can be quite long and students need some language skills to comprehend the situations described.

By contrast IMO problem statements can be very short. A problem as short as two lines is possible. Of course for a short question question, if dominated by an equation, language might not be so much of an issue, and a problem can be understood in some cases by non-native speakers. However for most questions students will need to understand the language to understand the question.

Whereas informatics questions are worth 100 points, giving a maximum of 800, allowing for a wide spread of scores, IMO problems are always worth 7 points, enabling a maximum of 42 points. Marking schemes can be quite tight. If a geometry question is answered by a perceived unorthodox method, such as using coordinate geometry or complex numbers, a solution might be given 7 points if correct, but otherwise not be eligible for partial marks. In some questions, not all of the points between 0 and 7 are available.

3. Medal Structure

The most similar feature of all the olympiads, and certainly of the IMO and IOI is the medal structure. In both olympiads the 1:2:3 system applies with the same rules on ties. This does permit some meaning to distributions of medals across olympiads.

4. Privacy

The attitudes of the IMO and IOI are almost diametrically opposed when it comes to publication of results. Although earlier I recall the attitude of the IMO to scores was to emphasise that the IMO was an individual rather than team event, IMO now on its official web site `http://www.imo-official.org/` (see "results") publishes the results of every student question by question, even if they get no points, and they also provide official placings for each country each year. (Some earlier individual results are unknown.) This is accepted and completely non-controversial within the IMO community.

There are two possible ways of providing a premiership for countries. One would be to use every point won by the student (as IMO does), while the other, which I have seen done by Australian colleagues from other science olympiads, is to use a medal count based on something like 3, 2 and 1 points for Gold, Silver and Bronze. The IOI strictly leaves off the record scores of students who do not get medals in the longer term, but astute observers of the new dynamic scoring system could construct unofficial premiership tables while they exist.

5. Topics and Syllabus

The IOI maintains an evolving syllabus document. This is quite detailed and is discussed and evolves dynamically. The question of a syllabus for the IMO has not been seriously raised. The syllabus for the IMO is based on tradition and a question would be considered suitable if it fell into past patterns, although occasional experimentation takes place. For example the "windmill" question of 2011 (easily downloadable from `http://www.imo-official.org/problems.aspx`) composed by Geoff Smith of the UK was certainly in this category.

IMO questions have however been based on this unwritten principle since the beginning, and fall into one of the categories of algebra, combinatorics, geometry and number theory. The method of evaluation will be discussed in more detail below, however of the six problems it has always happened that each gets 1 or 2 problems in the final paper, and on almost all occasions (1997 and 2011 are exceptions) geometry is a topic with 2 problems. There is a strong support for geometry, partly because it is envisaged that training in geometry is more likely to yield scores for the students, and partly it is just seen as an important part of mathematics.

The IMO papers have adopted fairly similar structure over the years. In the last 15 years there has been a much greater tendency, as in the IOI for IMO papers to be graded. This means that the first problem of the day should be highly accessible for most students, the last problem very challenging indeed, while the middle one quite discriminating. I believe there is a similar approach in the IOI, although, sometimes as also in mathematics, the results can be a little different than expected.

Over the years the view of most is that the IMO paper has, especially by this structured grading approach, become much more difficult in recent years. There was one IMO, Hong Kong in 1994, in which members of the USA team all scored the maximum 42 points and the cut-off for a gold Medal almost a perfect score. This has not been repeated although it is not uncommon for strong teams such as China to have all students obtain say at least 38 points and Gold Medals.

I have detected little change in the unofficial IMO "syllabus" over the years, although I do note that it has been a long time since a genuine 3-dimensional Euclidean geometry problem was set.

6. Problem Evaluation

Problem evaluation methods of the various olympiads can vary quite markedly and the comparison between the IMO and IOI is quite sharp. Whereas the preparation of IOI papers is mainly entrusted to host and international scientific committees which operate impartially, a huge amount of the design of the IMO paper is in the hands of the Jury (team leaders, meeting before the students arrive, the Jury being the IMO term for the IOI General Assembly).

However before the IMO gets to this point much of the heavy lifting is done by a problem selection committee formed by the host country. Each country is asked to submit

to this committee 3 problems. Of course they do not all do this but many do with the result that in excess of a hundred questions can be submitted to this host committee.

If the host country is a country of strong mathematics traditions, as say Romania was in 1999, the evaluators are from the host countries. But there has been a growing trend in recent years, especially among host countries which are not so strong, for them to bring in well-known strong mathematicians from other countries.

Such mathematicians will live together in the time leading up to IMO for typically a month, working through the submitted problems individually, meeting to compare notes, and eventually toward the end of that time forming a shortlist of say 28 problems, approximately 7 from each of the four categories, to hand over to the jury.

So the Jury (team leaders) arrives about 4 days before the students (who remain with deputy leaders) and spends its own time at a "retreat", a secret location where they are not to communicate with the outside world. This tradition has been in place since the USA hosted the IMO in 1981. This Jury takes the short list (and has the benefit of the presence of the committee who formed the shortlist) and during this retreat it selects the paper and translates it into various languages.

During this time another large group of mathematicians arrives and observes. This is the group known as the coordinators, who decide the final marks of the students. In a strong country the coordinators will all be local. There can be 60 of them. In weaker countries these coordinators will include invited outsiders.

Towards the end of the retreat, the coordinators (a separate team for each of the six problems, each with its own captain) becomes aware of the problem they deal with and determines a draft marking scheme. Finally at the end of the retreat, each captain announces its proposed marking scheme to the Jury, and debate takes place. Sometimes the Jury raises issues which result in change to the marking scheme, but despite the fact there can be rigorous debate, a consensus is achieved.

The leaders remain in retreat until the second day of exams, after which they can communicate with their deputies and the students.

The conclusion of the exams sees all scripts photocopied, the deputy joins the leader, and collectively they go through all the scripts, including rough working, and try extract every mark they can (they might also have further advisors, sometimes known as brains trusts, usually very clever people to help the leader and deputy and who are staying at the IMO as Observers, at the expense of their team organisations).

This is a very demanding time on the leader and support group. They need to read the student papers very thoroughly, including rough working, and extract whatever marks they can see. They then have a meeting with the coordinators, who have already read the duplicate scripts and have formed their own opinions. Mostly they agree readily and sign off on scores, but there are many times they do not and need to adjourn. The coordinators are in the stronger negotiating position as they need to be consistent. They do look at matters which are raised. In the end if the leader will not sign off, the matter can go back to the jury, but this is very rare to reach this point. Probably most IMOs do not have such a situation.

So the contrasts with task preparation and evaluation at the IMO and IOI are quite different. This is mainly because of the nature of the subject, despite the intellectual aspects of the subjects being closely related.

The IOI leaders have almost the same responsibilities as IMO leaders technically in problem creation as they can detect a good reason to reject a problem during the quarantine period, and have done so on rare occasions. But the scientific committees of the IOI do most of the work, which also includes a heavy workload in creating the automatic marking regime, mostly enabling scores to be simply notified to students not too long after the exam is completed.

In conclusion, I also note that one of the main differences is that IOI is marked completely objectively by a computer. There is some subjectivity in the coordination of IMO in that the student is reliant on their leaders and the coordinators to find every bit of their work which might lead to a mark.

7. Recent IMO Issues

The IMO is going through a major debate. It sees two vital issues. The executive of the IMO, known as its Advisory Board (very much the equivalent of IOI's International Committee, but with some powers relating to more matters which would transgress some powers held by IOI's International Scientific Committee) recently brought a proposal to the Jury related to two matters, both of which, despite being independent, lent to the same solution.

There was great concern that in the days of mobile phones that the current problem evaluation and setting regime led to a possibility of cheating which was impossible to police. There was also concern that the cost of holding the retreat was so significant that it was inhibiting many countries from bidding to host IMO.

So the IMOAB, as the Advisory Board is known, proposed to the 2011 Jury in Amsterdam that an independent committee be appointed to set the paper (very similar to the way it is done at the IOI) and basically dispense with the retreat, but possibly leaving a similar quarantine arrangement to the IOI to enable avoiding embarrassing mistakes, like selecting a well-known problem for some students, or one with some newly discovered technical problem.

The IMOAB was almost unanimous, but the proposal was rejected by something like a 60 to 30 margin. So considerable debate will continue. Hosts for the next two IMOs have planned on using the traditional format, with a retreat.

I have been to a number of IMOs and IOIs however I do have some experience of the other three main science olympiads as my colleagues in Australia, who work together with me for government support, have helped me understand some of these other traditions. These other three major olympiads also have variations in the way tasks are prepared and assessed and yield to similar comparisons to those discussed above.

8. Concluding Remarks

I have tried to compare some issues as they apply to IMO and IOI because I feel it is important that there is some cross knowledge across the disciplines. Whether this causes us to change anything is not so important, but it is useful to know how others deal with similar issues.

9. References

The official web site of the IMO, as given above, at

`http://www.imo-official.org/`

is the most comprehensive source of data of IMO. It is administered by Slovenia, has whatever scores are known, all the problems, and provides official placings. It is easy using this site to see how a country has performed over time.

There are several sources of solutions. The most definitive were published on the earlier IMOs by Murray Klamkin. See

Klamkin, M.S. (1986). *International Mathematical Olympiads 1978–1985*. Mathematical Association of America.

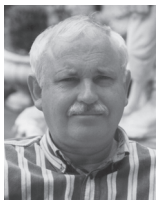
More contemporary collections of problems and solutions can be found on the Art of Problem-Solving site

`http://www.artofproblemsolving.com/Wiki/index.php/IMO_Problems_and_Solutions,_with_authors#2011`.

The 2011 windmill problem referred in the text can be found at

`http://www.artofproblemsolving.com/Wiki/index.php/2011_IMO_Problems/Problem_2`.

A solution is not on this site on 11 March 2012.



P. Taylor graduated with a PhD in applied mathematics at the University of Adelaide in 1972. Since then he has been a lecturer and professor at the University of Canberra. He is currently executive director of the Australian Mathematics Trust, which administers Australia's participation in both IMO and IOI. He also co-chaired ICMI Study 16 *Challenging Mathematics in and beyond the Classroom*, which was published by Springer in 2009.

Olympiads in Informatics, 2012, Vol. 6, 205–211

Putka – A Web Application in Support of Computer Programming Education

Jelko URBANČIČ[1], Mitja TRAMPUŠ[2]
[1]*Zavod za računalniško izobraževanje Ljubljana (Institute for Computer Education Ljubljana)*
p.p. 4716, SI-1001 Ljubljana, Slovenia
[2]*Jozef Stefan Institute*
Jamova 39, SI-1000 Ljubljana, Slovenia
e-mail: jelko.urbancic@guest.arnes.si, t.mitja@gmail.com

Abstract. In many countries the interest in pre-university computer programming education has been changed during the last decade. In the Slovenian case, the education reform, demographic and other reasons produced a big decrease of interest among young people. The paper describes our approach to increasing the interest for computer programming by integrating a web based tool "Putka" into the education process. Putka is, at its core, an online judge. Two facts make it stand apart from similar applications, however. First, its design and many additional features are influenced by its heavy use in support during long-term programming classes rather than only one-off competitions. Second, while its use in the classroom was the primary motivation for developing Putka, it has also hosted numerous competitions with varying rules. Out of necessity, this multifaceted nature bore what we believe to be a very flexible technical design.

Key words: computer programming education, web based education tool, online judge.

1. Introduction and Historical Background

The Slovenian story about the attitude of young population to the informatics is similar to that from some other developed countries. In the first period, during the introduction of computers into homes at the end of the 80s and the beginning of the 90s we had a very strong interest. The result of this boom was quite a large number of primary and secondary school contestants, 3000 in the country of two million inhabitants.

Computer education was used as a synonym for computer programming. The education was lead mostly as additional courses in primary and secondary schools and the rest was lead by private enterprise and the civil society movement. One organization is of particular interest for our story: Zavod za računalniško izobraževanje Ljubljana (abbr. ZRI, eng. Institute of computer education Ljubljana), a private organization founded in 1989. The institute expanded its activity fast and the number of primary and secondary school students rose up to 1000 by 1995. Remarkably, the Ministry of Education had no special strategy or program for the introduction of computers and informatics in primary schools and a very weak one for the secondary schools until 1992.

The second period, starting in 1995, saw a significant decrease of interest for computer programming among the young, similarly to other developed countries.

Besides all sociological, demographic and other reasons, the Ministry of Education affected the decline significantly. A large amount of money was spent in computer technology. Most of the primary schools in the country built a computer classroom. But the education program through which the computer classroom should be put to use was weak and the competence of the majority teachers was insufficient. A big loss of motivation for informatics and computer programming consequently occurred among young people. The number of young students taking computer programming classes and the number of contestants were both reduced by at least by factor of five in a few years. All private and civil society organizations either disappeared or switched to other business except ZRI, but it too was forced to decrease the number of students down to 250 by 2002.

The third and current period was marked by the introduction of another school reform that caused shock to many extracurricular activities including computer programming. Part of the reform is the introduction of elective courses for primary schools. While this has its benefits, the list of available courses is in reality often severely limited by staff and budget resources at individual schools. A key factor in deciding which courses to offer is the students' interest; computer programming as a somewhat fringe activity has, to our knowledge, never been realized as an elective course. On the other hand, the children each got their own schedule, causing them to be fragmented, with school courses finishing late in the afternoon.

The daily time frame for all children's out-of-school activities was mostly reduced to a two-hour interval between five and seven in the evening. As the demand for extracurricular programming courses shrank, there was no more room for full-time programming teachers. As a consequence, the quality dropped: a teacher who also teaches unrelated subjects cannot dedicate enough time to properly prepare for a programming course.

Many civil society organizations and private institutions for children out-of-school activities collapsed. The remaining ones faced stronger competition from all kinds of activities.

This was a very strong shock for computer programming. The number of contestants fell from 3000 in the first half of the 90s to less than 100 nation-wide in the period from 2005 till 2010, finally climbing to 200 in 2012.

All this time, ZRI was a leading institution in the field of computer programming education for the young population. It changed its business model in 2002 and at first only made its courses, taught by voluntary teachers, available to only about 20 most motivated students. The number of students has then been increased step by step to the present number of about 40. This is a part of our mission of preserving the organizational and methodological knowledge of teaching the young population computer programming. The second part of ZRI's vision is to prepare more or less competent contestants for the IOI (Urbančič, 2008). At the time when the number of young students practicing computer programming was at its minimum we were thinking about what we can change. We can influence neither the demographic factors nor the educational policy within the country. The students' motivation is the only factor that can be influenced by us. We believed a user friendly learning tool that provided students with instant feedback and progress reports would enhance their motivation. Although online judges with ample task collections already existed, very few of them met even one out of our three criteria:

- a large number of entry-level tasks systematically covering the basic programming concepts, e.g., simple loops or conditionals;
- tasks and user interface in Slovene (important for young students);
- the possibility of administering the system: creating user groups, tracking progress of students, creating tasks and possibly contests . . .

Surprisingly, this is still largely true today. Apart from the obvious Slovene language barrier, most judges (UVA toolkit, 2012) are not publically available as a software package and mostly support only ACM-style problems (fixed input, fixed output). We therefore decided to build a tool of our own as an institute project. The web was deemed to be the most suitable platform.

During our first attempt in 1999, we seriously underestimated the complexity of the task at hand: not only did we have little experience with both project management and web development; we also did not start off with a clearly defined idea of what the end result of the project should be. Once we realized all this, we cancelled the project, postponing the general idea of developing a web-supported learning platform for later.

During our next attempt, in 2005, we used an approach influenced by a previous project together with some good practices of project management. This resulted in a well-rounded program which was however still plagued by instabilities, the result of our overly-eager experimentation with new technologies.

Finally, the third attempt resulted in the first operational version and since July 2007 the application is available at www.putka.si.

2. Description of the Present System

Putka grew along with the scope at which it was being applied. At first designed to support classroom courses in a very basic manner, it got expanded over time with **numerous convenience features**, e.g., task search, tagging, difficulty ratings, task grouping and so on. While the improvements are mostly simple, they are the result of real needs identified during years of experience and therefore significantly improve the teaching experience.

In addition to the classroom setting, Putka soon supported and got used for organizing competitions. To date, it hosted several dozen competitions (including at national level; Putka, 2012), both on-site and online, with **varying rules**, among them ACM (ICPC) (ACM Putka, 2012) and IOI. Competitions bring not only diversity in rules, but also in **types of tasks**. We support classic input-output tasks, tasks with non-unique outputs, interactive tasks, tasks with manually-graded outputs and likely even scenarios we failed to foresee. This is due to an extremely flexible (but succinct) *test script* framework for specifying evaluation procedures (see Section 2.3).

2.1. *System Architecture*

In supporting this large set of use cases, a highly modular design was instrumental. There are four main components to Putka:

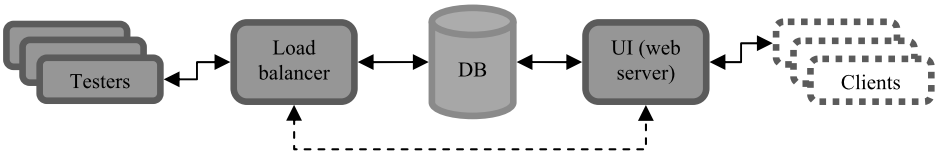


Fig. 1. The communications and data flow diagram between the main system components. The dashed line represents notifications of DB changes.

- *the database* – the single point of persistent data storage;
- *the manager* – distributes work among testers (load balancer);
- *the tester(s)* – evaluate user programs in a strictly controlled environment;
- *the user interface* – a web application.

Each of the components can be run on a separate computer. In practice, we tend to run everything but the testers on a single machine. The communication between the components is kept as streamlined as possible. Figure 1 illustrates the communication links.

2.2. *User Interface*

The user interface is realized as a web application, allowing students access to their learning environment from virtually anywhere.

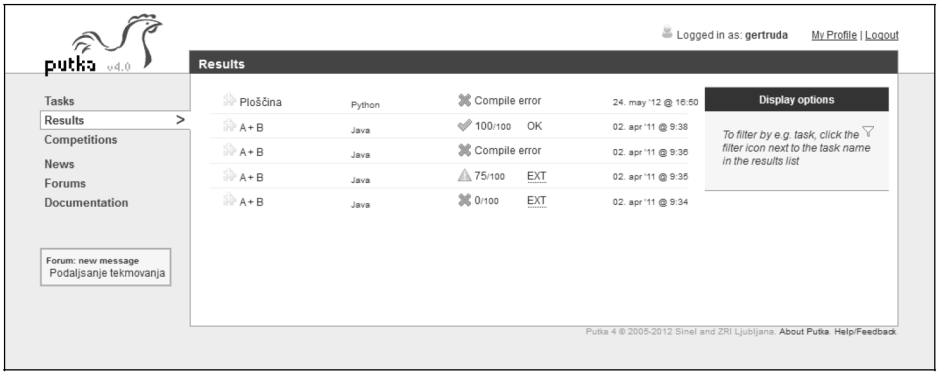


Fig. 2. A screenshot of the user interface showing the overview of user's submissions. The administrative/teacher's part of the UI is not shown.

Features. The UI consists of several modules, accessible through the main menu. Of central importance in Putka are the programming tasks. They are hierarchically organized into directories equipped with a full ACL (Access Control List) permission system, allowing fine-grained control over which groups of students can view or edit[1] which tasks.

[1] Some senior students are actually editors/authors of simpler problems (a valuable experience), but their access to the hard problems should clearly be limited.

A separate module shows aggregate and detailed upload results; this can be a very lively and informative view for administrators during a competition or a class session. The tasks and results view(s) are tightly interlinked and support various searches and filters – with hundreds of tasks and users, this is essential especially for the teachers. Competitions are managed in a small separate module. Forum is the last of the modules involving tasks relatively strongly. It features a general-purpose online forum with additional automatically created discussion threads about individual tasks. These threads are also linked to from the task pages and students involved with a task receive a notification on new posts; this is particularly useful during competitions when the forum is the only means of communication.

Users and their groups can be managed in a separate module. Administrators can get an overview of each user and track his progress by using this module. The **documentation** module hosts reference manuals and tutorials for several programming languages. All the pages are mirrored (hosted locally). This self-contained setting is crucial for on-site competitions with no internet allowed.

Technology. The UI is written in Django. We make use of Django's object model to keep the data cleanly organized and structured. We also tightly integrated the data model with a server-push framework running on top of Tornado/Tornadio, a non-blocking web server. This allows for pushing notifications to the web browser (e.g., "a task has finished its waiting/evaluation", "a competition has started", "new unread message" ...) without the user having to reload the page. A relatively small part of the UI code also runs on the client site, written in javascript. However, we made it a point to keep the interface usable, if less comfortable, even with javascript disabled.

2.3. *The Testing Backend*

By the backend, we refer to the part of Putka concerned with evaluating users' programs in a controlled environment. The backend is controlled by a single **manager**, an application that distributes work among **testers** and provides a low-level monitoring interface to them. The testers compile and run the user programs. This is performed in a **debugger-like** fashion: all system calls made by the user programs are intercepted and blocked or allowed based on the programming language (e.g., the python interpreter needs access to /etc/passwd) and/or the task. This prevents almost all forms of attacks on the backend. The time and memory consumption of user programs are of course also being monitored and limited.

The testers run on machines with different hardware configurations, the time limits get adjusted based on a "**speed factor**" computed for each computer. However, we have learned to be wary of such situations: it turns out that the speed ratio between two machines can **vary wildly** (5x and more) depending on the program being evaluated and its speed bottleneck (RAM, 64-bit operations ...).

Test scripts: Defining the Test Scenario. The tester is a C++ application which **embeds the python interpreter** and exposes to it some basic Putka-related functions (configuring the jail, executing programs in the "debugger", reporting results to the manager).

On top of these, a python layer provides **convenience functions** encoding the most common test scenarios (e.g., "compile, test on all input-output pairs, and sum the scores of test cases"). The evaluation procedure and scoring are fully defined by a python *test script* that is a part of each task and gets interpreted in this environment. The convenience functions enable the scripts to be very short in most cases (often a single function call), while the low-level functions and the power of the full python interpreter keep almost any conceivable test scenario expressible in our framework.

3. The Present Role of the Putka System

The mission of the Putka system is to motivate students and improve the quality of the courses on one hand and to provide a quality contest environment on the other. Very importantly, the system needs to be offered in the national language.

To this end, today the system is managed by the Putka team[2] and is widely used for courses at ZRI. An archive of around 300 problems is available to the students. The problems are only gradually revealed to the students according to their progress. The students can upload their solution from anywhere (ZRI, home etc.). Each student can access his archive of solutions. The coaches have access to student's archives to monitor their progress and to facilitate problem discussion. There is also a module for online discussion, used outside the classroom and during competitions.

The required competences of the users are set very low to include the Putka system in the **early education process** as soon as possible. Students must have basic knowledge of standard input-output operations and fundamentals like conditionals and loops – many tasks are formulated only using these simple constructs. The very easiest tasks do not even require loops; beginners aged 12-16 are normally introduced to Putka after ten hours of introductory courses.

To promote programming among the young, the whole system with a limited set of exercises is publicly available and anyone can access it upon registering at www.putka.si.

Besides the learning/classroom mode, the contest mode is being actively used as well, with multiple internal and national contests using IOI and ACM rules being executed each year.

4. Conclusion

From 2005/06 when the number of students attending ZRI and/or computer programming contests in Slovenia reached its minimum, the attendance almost doubled by 2012. There were some good promotional activities in Slovenia lead by the ACM in these years, but at least for the turn for the better at ZRI, the impact of systematic approach using the Putka system was crucial. The system's design and choice of features, both based on previous teaching experience, have been proven to be successful through five years of continuous use in the classroom and at competitions.

[2] A volunteer group of ZRI teachers and (former) students.

References

ACM Putka (2012). Available at: `http://putka.upm.si/tasks/`.
Putka (2012). Available at: `http://www.putka.si/`.
Urbančič, J. (2008). 20 mednarodna računalniška olimpijada, Presek, Društvo matematikov, fizikov in astronomov Slovenije, 36(2), 27–29.
UVA Toolkit (2012). Avaliable at: `http://uvatoolkit.com/links.php`.

J. Urbančič had been involved in computer programming at eighteen. He received PhD in atmospheric sciences (1988), two years later founded ZRI where he has been acting director until 2002. Since then, he has been the manager in the state administration and besides that he is volunteering at ZRI and national association as a part time teacher, coach, judge and organizer of the competitions. He attended IOI between 2006 and 2010 as deputy leader of national team.

M. Trampuš has been actively involved with computer programming education since he was 10 – first as a student and competitor and national and international levels (CEOI, IOI, CERC), later as an organizer and occasional teacher. Recently, he attended CEOI 2011 and 2012 as the national team leader. He is one of the three core software developers of the present Putka system. Currently, he is working on his PhD in the area of data mining.

From the beginning the **Putka team** has had following active members: Nino Bašič, Jan Berčič, Žiga Ham, Tomaž Hočevar, Mitja Trampuš, Jelko Urbančič and Andrej Veber. In previous versions Andrej Bukošek, Domen Blenkuš and Peter Koželj were also active. Recently, three new young colleagues joined the team. Presently nine members of the team cover the roles of system administration, softwaredevelopment, task administration and user administration. The name Putka is the acronym of "Verifying the effectiveness, thoroughness and correctness of algorithms" in Slovene language. It is also the expression for a lovely little hen, that's why it has a hen in its logo.

Olympiads in Informatics, 2012, Vol. 6, 212–217

Theoretical Tasks on Algorithms; Two Small Examples

Willem van der VEGT
Dutch Olympiad in Informatics, Windesheim University for Applied Sciences
PO Box 10090, 8000 GB Zwolle, The Netherlands
e-mail: w.van.der.vegt@windesheim.nl

Abstract. In the Dutch Olympiad in Informatics theoretical subtasks are used to tests some of the skills needed for algorithmic design. The results were somewhat discouraging. An analysis for future use of theoretical tasks is performed.

Key words: informatics olympiad, programming competition, task design.

1. Introduction

The second round of the Dutch Olympiad in Informatics (Dutch, 2012) is very selective; we want to identify the top ten students, but we also intend to offer a fair competition in which contestants can show what they are able to do in a few hours. It is usually created with two or three background stories, each leading to several subtasks (van der Vegt, 2009).

The system of subtasks also makes it easier to slip in a more theoretical subtask. Once we asked contestants to create a sample input file that could produce a specified output for a given algorithm. We only asked for this file, not for a program or a description how to find it. At the IOI we call this an output only task; we had just an output only subtask. In another contest a game was played in which you could get stuck. One of the subtasks was to output the minimal number of moves that was at least possible, whatever the initial position of the game, and however badly the game might be played. We even predicted the output for a specific case and asked the contestants to explain this strange output in a few words. In this case, they had to deliver a plain text file. Of course this file was examined and graded manually.

In this year's second round one of the background stories had to do with set partitioning. There were eight subtasks; four of them were rather easy programming tasks, two were very hard programming tasks and the final two were tasks where contestants had to find a sample set of integers, fitting with the problem statement.

This paper addresses the choice for such theoretical tasks, founded in the aims of a programming contest like the informatics olympiad. We will also discuss the results of last year's theoretical questions, and formulate a few recommendations for the future use of this kind of subtasks.

2. The Aims of a Programming Contest

Programming contests are of course about programming. What sense does it make to ask more theoretical questions, even questions that can be solved without using a computer or a computer program?

The goals of the IOI are to bring together, challenge, and give recognition to young students from around the world who are the most talented in informatics (computer science), and to foster friendship among these students from diverse cultures (IOI website, 2012). The competition tasks are of algorithmic nature; however, the contestants have to show such basic IT skills as problem analysis, design of algorithms and data structures, programming and testing.

This year's Call for Tasks specified this: IOI tasks are typically focused on the design of efficient, correct algorithms. Input and output are to be kept as simple as possible (Call for Tasks, 2012).

One of the central words in these goals is algorithm. The IOI Syllabus (IOI-syllabus, 2009) uses a quote to elaborate this word: "Algorithms are fundamental to computer science and software engineering. The real-world performance of any software system depends only on two things: (1) the algorithms chosen and (2) the suitability and efficiency of the various layers of implementation. Good algorithm design is therefore crucial for the performance of all software systems. Moreover, the study of algorithms provides insight into the intrinsic nature of the problem as well as possible solution techniques independent of programming language, programming paradigm, computer hardware, or any other implementation aspect" (ACM, 2001).

These remarks immediately raise the next question. How to test the design of algorithms?

3. Black-Box Testing and Alternatives

One of the problems of testing algorithms in a programming contest is that it is usually done by black box testing. The organizer presents a set of test data and checks the output of the submitted program. Success at the test is an indication of a proper designed and implemented algorithm. Constraints on memory use and run time are means for testing efficiency of the program. But this way of testing demands a lot of your test data.

Forišek (2006) discusses the suitability of programming tasks for automated evaluation. Several interesting tasks are definitely not suitable for IOI-style automatic testing, like planar graph coloring or substring search. Some of the other evaluation methods that are suggested are pen-and-paper evaluation, supplying a proof and code review (white-box testing).

Pohl (2008) gives an overview of the disadvantages of black-box testing. In Bundeswettbewerb Informatik, the German Olympiad in Informatics, manual grading is performed. Contestants need to write a description of the solution approach and to choose their own examples for test input. For each task a set of grading criteria is developed; grading is a joint effort of a team of jury members.

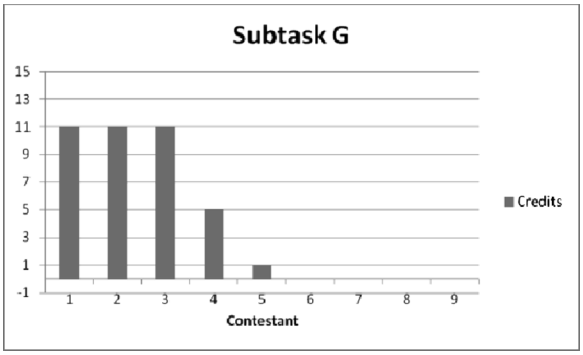


Fig. 1. Results for subtask “Strictly Inpartitionable”.

One of the aspects of designing algorithms is to be able to analyze the boundaries of a solution space. With the theoretical questions we use at the Dutch Olympiad of Informatics we want to check whether our contestants have a real understanding of the problem. We try to look into their thinking process, not by examining their algorithms, by one step earlier. We want to find out if they have a proper image of the problem their algorithms should have to deal with.

4. Two Small Problems

The task Fair Division (Dutch, 2012) used sets of different integer values. Most subtasks asked for a partition of such a set in such a way that the sum of the values of all subset were the same. All the values n_i were chosen with $0 < n_i < 1000$. For the theoretical subtasks the number of values is at most 30.

Subtask G introduces the term Strictly Inpartitionable. Contestants are asked to give an example of a set with different values, for which it is impossible to partition it in 2 or more subset with equal sums, in other words a strictly inpartitionable set. It should be a set with as many values as allowed (30 if possible) and the highest value should be minimal. Partial credits were allowed for solutions that did not meet these both conditions.

Out of 22 contestants, only 9 submitted a solution. And only 5 of them received any credits. 11 out of 15 was the maximum achieved.

What went wrong? Some of the contestants were not able to deliver solutions for all subtasks, so they did not enter any solution. Two contestants submitted a set with much larger numbers; they were rejected. Two others submitted a wrong solution. Two contestants submitted a solution with a set of far less than 30 values; they received a few points. All three contestants with 11 points entered a set with 30 different values, using the same way of thinking. They took the number 1, 2, 3..29 and the final number exceeded 435, the sum of the integers from 1 till 29. This way they were sure that the set was in no way partitionable; their 30th value was too large to fit in one of the subsets.

The solution the organizers had thought of had a much smaller highest value. We looked for the set in which the sum of all values was a prime number. For us it was

obvious that such a set is strictly inpartitionable. Since the sum of the integers from 1 till 30 is 465, we looked for the smallest prime number above 465. This number is 467. So if we remove 29 from the row and add 31, we have the optimal solution. One of the contestants must have been thinking in this direction, but he skipped 30 instead of 31, so his solution was partitionable in two subsets.

Subtask H introduced Easy Partitionable Sets. Contestants are asked to give an example of a set with different values, for which it is possible to partition it in m subsets with an equal sum, for all $2 \leqslant m \leqslant k$, maximizing the value of k. Partial credits were given for solutions with $k > 3$.

This is of course a very difficult task. Even checking the submissions is horrible and time consuming. We expected to get at least some solutions. But only three contestants submitted. One of the submissions was rejected, because the set was not partitionable in three subsets. The second contestant created the set {1, 119, 2, 118, 3, 117, 4, 116, 5, 115, 6, 114, 7, 113, 8, 112, 9, 111, 10, 110, 11, 119, 120}. It was easy to create 12 building blocks of subsets with a sum of 120. By combining these subsets, partitions in 3 and 4 subsets with equal sums could be made. A partition for $m = 5$ is not possible. This contestant has got 3 points for his solution out of a maximum of 35. The best result was for a young contestant with a background in the Mathematics Olympiad. He received 15 points submitting {1, 2, 3, 4, 5, 6, 7, 8, 9,10, 11, 12, 13, 14, 15}. His solution reached the k-value of 6.

There is simply not enough data to see if a mathematical problem analysis was performed by the contestants. If a set is m-partitionable, then m needs to be a divider of the sum of the set. If a set is easy partitionable for all values of m till a maximum of k, then k needs to be a common multiple of all integers for 2 to k. So it is a good idea to start with a set with as a sum the least common multiple of 2 to k. For $k = 8$ this least common multiple is 420. If you want to have a set with 30 values, the average value should be 14. We used this for our sample solution to get {13..27, 29..43} as an easy partitionable set for all m from 2 to 8.

Table 1

Best contestant solution for subtask H

2-partition	3-partition	4-partition	5-partition	6-partition
{1..5, 7..11}	{1..4, 6..9}	{1, 14, 15}	{1, 5, 6, 12}	{1..4, 10}
{6, 12..15}	{5, 10, 12, 13}	{2..4, 6..8}	{2..4, 7, 8}	{5, 15}
	{11, 14, 15}	{5, 12, 13}	{9, 15}	{6, 14}
		{9, 10, 11}	{10, 14}	{7, 13}
			{11, 13}	{8, 12}
				{9, 11}

5. Evaluation

With 5 and only 2 partial solutions in a field with 22 contestants, these two theoretical tasks did not deliver what they were developed for. In a discussion with contestants and fellow organizers, we found a few reasons.

1. These two subtasks were 2 out of 14 subtasks in a three hour contest. Only 6 contestants succeeded in submitting over 10 of these 14 subtasks.
2. These two subtasks were at the end of the task description. If contestants were not able to write a program for subtask F or even subtask E, they simply did not start looking at these two final subtasks.
3. Theoretical tasks are out of the comfort zone of the contestants. They are used to writing programs, rather than solving problems without using a computer program.
4. Contestants are not used to think about properties of numbers. No solutions used the notion of prime numbers. The least common multiple was not at hand in their mind.

The question remains whether these kind of questions can help to learn more about the ability of contestants to design proper algorithms. The constraints in subtask G were posed to guide the thinking of contestants in the direction of prime numbers. This idea failed. Some of the contestants however used another property of the natural numbers to reduce the maximum value within the set. With subtask H, we expected to see that contestants create a small working example and extend it.

The main problem we encountered is that asking for solution for theoretical questions is a very implicit way to find out about the skills on algorithms and design. Reducing the total number of subtasks and reordering the subtasks within the task description can improve the participation. Combining more theoretical questions with writing a program can enlarge the involvement with the subtask. And of course, focusing on the goals of the contest is still needed to design future tasks.

References

ACM/IEEE-CS Joint Curriculum Task Force (2001). *Computing Curricula 2001: Computer Science*, December. http://www.acm.org/sigcse/cc2001/

Call for Tasks (2012). `http://www.ioi2012.org/competition/call-for-tasks/`.

Dutch Olympiad in Informatics (2012). `http://www.informaticaolympiade.nl` (in Dutch only).

Forišek, M. (2006). On the suitability of programming tasks for automated evaluation. *Informatics in Education*, 5, 63–76.

IOI-syllabus (2009). `http://people.ksp.sk/~misof/ioi-syllabus/`.

IOI-website (2012). `http://www.ioinformatics.org`.

Pohl, W. (2008). Manual grading in an informatics contest. *Olympiads in Informatics*, 2, 122–130.

Van der Vegt, W. (2009). Using subtasks. *Olympiads in Informatics*, 3, 144–148.

W. van der Vegt is teacher's trainer in mathematics and computer science at Windesheim University for Applied Sciences in Zwolle, the Netherlands. He is one of the organizers of the Dutch Olympiad in Informatics and he joined the International Olympiad in Informatics since 1992. He was involved in the IOI-workshops on tasks in Dagstuhl (2006, 2010) and Enschede (2008). Currently he is one of the team leaders of the Netherlands at the IOI.

Olympiads in Informatics, 2012, Vol. 6, 218–225

Israel: The Regional Competition and Teacher Involvement

Ela ZUR[1], Tamar BENAYA[1], Oren BECKER[2], David GINAT[3]
[1]*The Open University of Israel, Computer Science Department*
108 Ravutzky st. Raanana, 43107, Israel
[2]*Einstein Institute of Mathematics Edmond J. Safra Campus, Givat Ram*
The Hebrew University of Jerusalem
91904 Jerusalem, Israel
[3]*Tel-Aviv University, Science Education Department*
Ramat Aviv, 69978 Tel-Aviv, Israel
e-mail: {ela, tamar}@openu.ac.il}, oren.becker@mail.huji.ac.il, ginat@post.tau.ac.il

Abstract. During the last year, the Israeli ministry of education increased its support to the Israeli IOI project. The primary goals of the increased support were to expose the project to a wider audience of high school students and to expand the team selection and training process. The team selection process involves several stages. The two early stages are: students' early preparation and a regional competition. We present here several questions of the regional competition, which require only numeric answers without algorithms or explanations. We then display the results of a survey, conducted among our high-school teachers, of their attitudes and involvement in the project's early stages. The results shed light on the teachers' views of the project's regional competition, task characteristics, school selection, student preparation, and more.

Key words: regional competition, teachers attitudes.

1. Introduction

The International Olympiad in Informatics (IOI) is the primary computer science (CS) competition for young (secondary school) students. In Israel, until 2010, the IOI project was composed of four stages: a self-study stage 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 (`www.tau.ac.il/~cstasks`; Hebrew) and in our previous paper (Zur *et al.*, 2010). For the past two years, an increased support by the ministry of education (`http://www.csit.org.il/`; Hebrew) enabled us to expand the project's four stages, and add a fifth stage. This stage consists of a regional competition, which precedes the national competition. This stage is aimed for a large audience, including students with very limited programming experience. The better students of this stage are invited to the national competition. In our previous, 2011 paper (Zur *et al.*, 2011), we described our initial experience with the new regional competition and its successive national competition, and provided some statistics regarding the students' backgrounds, motivation,

achievements and points of view. In this paper, we further elaborate on the regional competition and our teachers' attitudes and involvement. In Section 2 we elaborate on our recent regional competition, and in Section 3 we display the outcome of a questionnaire posed to our high-school teachers.

2. The Regional Competition Tasks

In the past year, the regional competition was held in November. It took place simultaneously in 140 high schools. Participation in the exam was voluntary, and approximately 1800 students, with orientation towards CS and math, participated in the exam. The exam was harder than that of the previous year. Only a few students obtained a grade higher than 90.

As one primary goal 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 tasks focused on mathematical and algorithmic characteristics, on which one had to capitalize her computation.

The regional exam included five tasks. Task 1 required parallel computations of shortest paths on a grid with obstacles. Task 2 involved arithmetic operations and backward reasoning. Task 3 involved the processing of a list, element by element, with some bookkeeping. This task is based on the task "Mean Sequence" of the 17th Olympiad in Informatics (`http://olympiads.win.tue.nl/ioi/ioi2005/`) held in Poland in 2005. Task 4 required recursion and dynamic programming; and Task 5 was based on computing a greatest common divisor.

We display below two of the five tasks. The first one – Task 2 – was the easiest for the students and the second task – Task 3 – was the hardest.

Task 2 of the regional exam

Given the two operators $+1$ and $\times 2$, which you may use repeatedly; compute the *minimal* number of operator invocations required to reach each of the two integers 417 and 794 from the integer 10.

For example, for reaching 21 from 10, two invocations suffice: $\times 2$ and then $+1$. For reaching 24, three invocations are required: $+1$, $+1$ and then $\times 2$.

Hint: The units-digit of the multiplication of the two answers (to 417 and 794) appears in one of the answers.

The objective of this task was to examine whether students turn to backward reasoning (as the suitable computation is that of starting from any of the given integers and computing backwards towards the integer 10). The hint was provided in order to assist

students in verifying their calculation results. We found in our previous study (of 2011) that the vast majority of the students do not err in their calculations, when the solution scheme is clear to them. The hint helps decreasing the likelihood of calculation error even further.

Out of the 1767 students, 1331 (75%) provided the correct answer for the integer 417, and 1090 (62%) provided the correct answer for the integer 794. A total of 1041 (59%) provided the correct answer for both integers. We may notice a difference between the two calculations (for 417 and 794). The calculation required for 794 is longer. We believe that the clearer one's view of the solution the lower the likelihood of error, particularly as the required computation "gets" longer.

Task 3 of the regional exam

The following increasing sequence, S, of 10 elements is as follows:

50 96 146 194 250 320 350 374 396 420

We would like to calculate the *number* of non-decreasing sequences of 11 positive integers such that each of the sequences keeps the following characterstic: the average of the 1^{st} and the 2^{nd} elements is the 1^{st} element of S, the average of the 2^{nd} and the 3^{rd} element is the 2^{nd} element of S, ... the average of the 10^{th} and the 11^{th} elements is the 10^{th} element of S.

For example, if S was composed of four elements: 4 7 10 12 the answer would have been 3, as each of the following sequences keeps the required characteristic:

4 4 10 10 14
2 6 8 12 12
3 5 9 11 13

1. What is the number of non-decreasing sequences for the given sequence S?
2. Can the number 350 in S be changed such that the answer to (1) will increase? If so, what number should it be changed-to in order to get a maximal answer for (1)?
3. Can the number 320 in S be changed such that the answer to (1) will increase? If so, what number should it be changed-to in order to obtain a maximal answer for (1)?

The objective of this task was to examine whether students demonstrate both inductive progression and abstraction. The inductive progression should be carefully employed upon advancing "through" the given sequence S. Abstraction should be expressed by conducting repeated projections of a (shrinking) range of (consecutive) values, during the inductive progression. A student who gains sufficient insight into the task should answer properly not only part 1 of the task, but also parts 2 and 3.

Out of the 1767 students, 238 (13%) answered part 1 correctly; 38 (2%) answered part 2 correctly; 52 (3%) answered part 3 correctly; and 20 (1%) answered all three parts

correctly. As with the previous task, we believe that the better insight one gained into the task, the lower the likelihood was for her to err. The low achievements in this task assisted us in an initial "pin-pointing" of the top students.

3. Teachers' Attitudes and Involvement

The teachers' involvement in the regional competition was vital for the project's success. The regional exam was held in the schools, and the teachers participated in their schools' local arrangements. They identified the talented students, notified them about the exam, and (some) advised them on how to prepare. The exam questionnaire was posted in the internet at a time that was announced a-priori. The teachers downloaded the questionnaire and posed it to the students. The exam lasted two hours.

As the exam answers involved only several integers, the teachers were asked to copy their student solutions to a spreadsheet and submit it to us. We graded the exam (automatically, with a short computer program) and posted the list of the top 15%, whom we invited to the national competition. We also posted the exam solution.

Due to the vital role of the teachers in the regional exam, we were interested in examining their views. Thus, we conducted a preliminary study among the CS teachers whose students participated in the regional competition, in an attempt to learn about their backgrounds, attitudes, involvement and perceptions of the Olympiad project.

We posed a 14-question questionnaire. The questionnaire was sent by e-mail to 122 CS teachers, and was answered by 45 of them (37%). The following gives some information about the teachers who answered the questionnaire:

- 85% were experienced teachers who teach CS for more than 7 years.
- 85% of the teachers were heads of the CS program in their schools.
- 20% of the teachers have been involved in the project for more than 7 years. 49% of the teachers have been sending students to the Olympiad project for 2-6 years. The rest (31%) were new to the Olympiad project, this being their first year of participation.

We present below the questions from the questionnaire along with a summary of the teachers' answers.

The type of questions posed in the regional and national competitions are different from the type of questions which students encounter in their CS studies. Do you find this type of questions interesting?

- The vast majority of the teachers (86%) claimed that they find such questions very interesting. Some teachers said that the questions stimulate thinking and creativity.
- Some teachers said that they use these questions to motivate talented students, and others said that they borrow ideas from these questions and use them in the classroom.

Do you think that the Olympiad project encourages students to select CS in high school?

- Most of the teachers said that the Olympiad project primarily interests talented students, and therefore does not affect most of the students' decisions whether to choose CS studies in high school.
- Contrary to that, some teachers said that the Olympiad project encourages the CS students and adds interest to the subject.

Are you satisfied with the way the regional and the national competitions are organized?

- Most of the teachers (62%) were satisfied with the format of the competitions. Some teachers mentioned that the fact that the regional competition takes place in the schools makes it easier logistically for the teachers, and it is also better for the students because of the location familiarity. They also mentioned that the regional competition is a good preparation for the national competition. In addition, some teachers said that because the regional competition takes place in the schools, it reaches a wider audience.
- 22% of the teachers thought that the format can be improved. Some suggestions were: to allocate time for the preparation of the students towards the competitions; to let students take the regional exam individually at their convenience; to organize the competitions earlier in the year so as to allow more time for the IOI team preparation.
- 16% of the teachers had no comments regarding the format of the competitions.

Are you familiar with the Israeli Olympiad website?

Almost all of the teachers (91%) said that they are familiar with the website. They said that they referred students to the website, and that they presented the students with sample questions from previous competitions (found in the website). Some teachers said that they borrowed ideas for questions they posed in their regular CS classes.

Do you prepare the students to the regional competition?

- 40% of the teachers said that they did not prepare the students for the regional competition because they do not have enough time. Yet, they did encourage the students to enter the website. One teacher said that she would like to have answers to the sample questions in the website in order to help her prepare the students.
- 40% of the teachers said that they did prepare the students to the regional competition. They conducted special lessons dedicated to sample questions. Some teachers let the students work individually on these problems in class. Others handed out sample questions with their solutions.

Would you like to receive help with the preparation of the students towards the regional competition?

- Most of the teachers (74%) said that they do want help with the preparation of the students. The teachers offered many suggestions, the main ones include: to conduct teacher training workshops; to provide additional learning materials and guidance for the teachers; to conduct local or regional training sessions for the talented students; to provide a discussion group in the website; and to add additional classroom

hours dedicated to the Olympiad project. One teacher suggested locating young talented students (8th graders) in math by posting weekly questions in the website, thus exposing them to algorithmics.

- The rest of the teachers said that they do not need help with the preparation. Most of them doubted students' spare time. Some said that it is enough for the students to self-study from answers to the sample questions. One teacher claimed that our evaluation may be conducted without prior preparation.

Do you collaborate with the math teacher in your school in the process of selecting and preparing talented students to the regional competition?

- Most of the teachers (75%) said that they do not collaborate with math teachers in their school. Some of the teachers said that they were unaware of this possibility and they will collaborate with the math teachers next year. Others said that there is no need to collaborate because they are familiar with all the talented students in their school. A few teachers claimed that the math teachers are not interested in collaboration because their students are too busy with their studies, and they have their own Olympiad project.
- The rest of the teachers claimed that they do collaborate with the math teachers in the process of selecting and preparing talented students.

Does the principle of the school support the Olympiad project and does he encourage the students' participation?

Most of the teachers (78%) said that the principle of the school supported the Olympiad project and encouraged the students' participation. They said that the principles viewed the Olympiad project as a worthy project for developing talented students. The principles helped with the logistics and the organization.

Did talented non CS students participate in the regional competition?

Only 18% of the teachers said that non CS students participated in the regional competition. The non CS students heard about the competition from friends, from science and math teachers and through a letter they sent to the students. The rest of the teachers (82%) said that non CS students did not participate in the competition.

Were you surprised with the number of students from your school that passed the regional competition (and invited to the next stage)?

38% of the teachers reported that no student in their school passed the regional competition. 48% of the teachers reported that 1 or 2 students passed the regional competition. 14% of the teachers reported that 4 to 14 students passed the exam. Most of the teachers (65%) were not surprised with the results of their students. The teachers that were surprised said they expected that more of their students would pass the exam, as they are very talented.

Is this the first time that your students encountered the type of questions which appeared in the regional competition?

The vast majority of the teachers (85%) said that the students encountered this type of questions for the first time. The other teachers claimed that they used such questions in their classrooms, or their students were familiar with such questions from previous years.

Describe the students' reactions about the questions in the regional competition.

- All students felt that the questions were challenging. Some students enjoyed the challenge and others felt that the questions were too difficult.
- Some of the positive remarks included: the questions were not standard questions; they were math and not CS oriented; the students enjoyed and expressed interest in the questions because they were stimulating and did not require coding; the questions were difficult, challenging and interesting; after the exam the students continued discussing the questions and their answers.
- Other, more frustrated remarks were: the questions were exhausting, hard to understand and there was not enough time to solve them; the questions were very difficult, they required non standard thinking; only a small number of students showed interest in the questions; the questions had many possible solutions, it was hard to find an algorithm; we did not have the required math background; some students continued despite the difficulty and others gave up; even the bright students were frustrated.

After the regional competition, did you solve the questions with your students?

Most of the teachers (86%) did not solve the questions with their students. The main reasons mentioned were: lack of time, lack of interest on behalf of the students, and difficulties in gathering the students together.

How did you cope with the students' questions during the regional competition?

Most of the teachers said that they did not answer questions because they did not feel involved with the professional aspects of the competition.

4. Conclusion

Our main conclusions from the study described in the previous section are:

- Most of the teachers claimed that they find the questions very interesting. In addition, they borrow ideas from the questions to their classrooms.
- Almost all of the teachers said that they are familiar with the website and that they refer students to the website.
- Preparation for the regional competition should not take place during school hours because teachers claim that they are very busy preparing the students for the CS matriculation exams and therefore do not have spare time to dedicate to the Olympiad project.
- The Olympiad project team should be involved in the preparation towards the regional (and the national) competitions because the teachers claim that they do not have the professional qualifications required in order to prepare the students for the competitions. The teachers suggested to conduct workshops both for teachers and students and to increase the utilization of the website with training materials and weekly questions and discussions.
- We should find ways to reach younger talented students prior to the time in which they start studying CS in secondary school. This will let us teach these students

for a longer period of time, and better prepare them for the Olympiad project. In addition, this would encourage students to choose CS studies in high school.

- We should encourage the CS teachers to collaborate with the math teachers, in order to locate talented math students for the CS Olympiad project.
- Most students felt that the questions were difficult, and some of them gave up. Perhaps we may include some easier (short?) questions in the regional exam in order to motivate a broader audience.

References

Burton, B.A. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.
Kubica, M., Radoszewski, J. (2010). Algorithms without programming. *Olympiads in Informatics*, 4, 52-66.
Ministry of Education Website. `http://www.csit.org.il/`.
The 17th International Olympiad in Informatics. `http://olympiads.win.tue.nl/ioi/ioi2005/`.
The Israeli IOI Website. `http://www.tau.ac.il/~cstasks`.
Zur, E., Benaya, T., Ginat, D. (2010). IOI Israel – team selection, training, and statistics. *Olympiads in Informatics*, 4, 151–157.
Zur, E., Benaya, T., Becker, O., Ginat, D. (2011). Israel – the regional and national competitions. *Olympiads in Informatics*, 5, 161–168.

E. Zur is involved in the Israel IOI project since 1997. 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. Her research interests include CS education, distance education and teacher preparation.

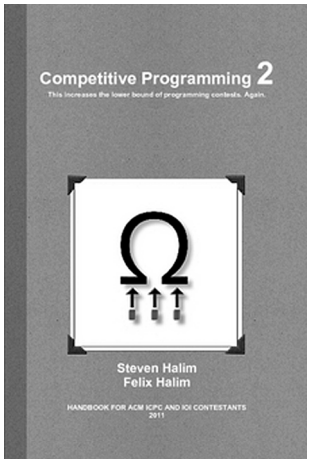
T. Benaya holds a MSc in computer science from Tel-Aviv University. She is a faculty member of the Computer Science Department at The Open University of Israel. She designed and developed several advanced undergraduate computer science courses. Her research interests include distance education, collaborative learning, computer science education, computer science pedagogy and object oriented programming.

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, IOI 2011 and is currently Israel's team leader to IOI 2012. He is currently in his first year of MSc studies in mathematics in the Hebrew University of Jerusalem.

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.

Olympiads in Informatics, 2012, Vol. 6, 226–228

REVIEWS, COMMENTS



Competitive Programming 2

Author: Steven HALIM, Felix HALIM
Publishing house: Lulu
Country: United Kingdom
Year of edition: 2011
Language: English
Number of pages: 262

There are many ways in which students can be prepared for programming contests and indeed training more generally in algorithms. Two of the more obvious areas are exposure to a wide variety of different algorithms and techniques, and practice by solving a good number of such problems. The former is important, not only at the initial levels where a student is introduced to the fundamental building blocks (e.g. dynamic programming or depth-first search) but at the progressive levels where a knowledge of more sophisticated methods gives a useful, if not essential, toolbox. Algorithmic programming contests have changed significantly since the 80s and, as this book mentions in passing, what were once the deciding problems are now basic requirements. The need for practice should be clear. Not only does it give the student the opportunity to confirm they understand the algorithms, it develops skills in finding appropriate algorithms, appreciating the different (test) cases an algorithm needs to solve, seeing ways in which multiple algorithms and data-structures can be combined and, from a purely contest perspective, helps in increasing a student's speed and accuracy.

Competitive Programming 2 is an excellent resource for both exposure and practice. It is the second edition (the first edition is just *Competitive Programming*) of this book, which grew originally from a similarly titled course that has been taught at the National University of Singapore since 2009. Its page count puts it at over 70% larger than the first edition. Readers of that edition, considering whether to also buy that edition can find a detailed comparison of the two editions (and the planned third edition) at the authors website [1]. The book is available in both A4 and A5 formats and an electronic version should be available by the time you read this. The A4 copy was review and worked well

as a format. This reviewer has heard from a colleague that the A5 edition is too small, although has not personally seen a copy at that size.

The format throughout the book is generally a brief overview of a given topic, followed by multiple algorithms, exercises, a long list of categorised problems to practise, and a brief conclusion. Starting with the categorised problems, this is a wonderful resource. Around 1300 problems appear in the book (almost exclusively from the University of Valladolid [2] online judge) and the bulk of these appear in these lists. These problems, which appear after an algorithm or group of related algorithms, typically have multiple sections each of which list a good number of problems and also highlights three *'must try'* problems. For example, the graph traversal problem list contains 56 problems (18 must try problems) over 6 sections: Just Graph Traversal; Flood Fill / Finding Connected Components; Topological Sort; Bipartite Graph Check; Finding Articulation Points / Bridges; Finding Strongly Connected Components. The ad-hoc section towards the front of the book contains 160 problems. Superb!

The description of algorithms through the book is a little more variable and varies from very detailed to incredibly terse. In general there is an okay amount of detail; enough for a motivated student to get the idea of the algorithm and start to play with it (or find details elsewhere) but not enough for the book to stand alone as a text for the weaker students. In fairness, the book does not set out to become another introductory algorithm textbook – the prerequisites suggests that it expects its audience to have passed a "basic data structures and algorithms course" – although contrary to this the very detailed sections tend to be the more basic material, for example in the dynamic programming explanation. When it goes to the other extreme it is often because of tendency to give the algorithm through code rather than explanation or just to drop in the name so that the algorithm is ticked-off.

The style used for each algorithm varies but the following are typical items that appear. There is often a motivating problem, discussion of specific tasks, examples of using the algorithm for other algorithmic tasks, exercises (as distinct from tasks, these are of the type more typical in a normal algorithm text; hints and solutions appear for many of these exercises), examples, discussions on complexity and short biographies of the people involved. Example code appears frequently in the text, and is available in both C++ and Java forms online. The coverage of the book is good, and focuses on what is used in programming contests, with the broad chapter headings covering pretty much what you would expect. The focus on contests works well and enables the book to offer focused advice on choosing an algorithm; weighing up the ease of implementation against the bounds specified in the problem or those generated by a specific sub-problem in a given solution.

The book, while applicable to all programming contests, is focused on the ACM ICPC and the IOI, and chooses its subject material accordingly. Where appropriate in the text the authors indicate where material is only relevant to one of these contests. The book is skewed towards the ICPC and the authors, whilst displaying a very broad knowledge of its problem set (the authors claim to have solved over 50% of the Valladolid problems), do not display similar knowledge of the IOI set (only 8 problems are discussed and only

one is pre-2008). This is not a problem in terms of presenting the material. In the terser algorithmic sections where no problems are discussed, and a good IOI task exists, it appears to be a choice of the authors since not only do good ICPC tasks exist but they are used by the authors in the corresponding problem lists. Furthermore, the bias means that the listed problems can be submitted to an online judging system which is invaluable. One does feel however, on occasion, as though the existing text has been hacked to include IOI references rather than having had it in mind from the beginning.

The majority of the book discusses algorithms (and data structures) though it does kick off with an introductory section on competitive programming. It makes a good introduction and one that might usefully be read by all students entering such contests, even those for whom most of the material in the rest of the book would be too advanced. It is debatable whether "Type Code Faster!" should have been the first tip – for the vast bulk of competitors this is not going to be an important factor – but the remaining tips are sensible and strong.

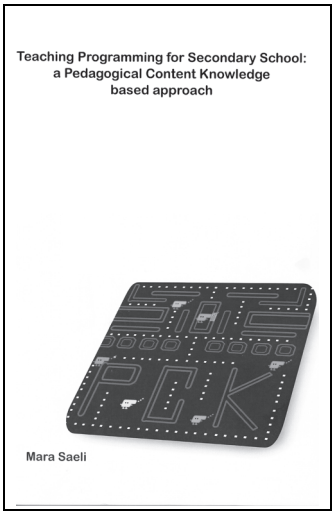
The book has two clear audiences; coaches and students. For the former the book provides a comprehensive set of categorised problems, and a far wider selection of problems that most are likely to have familiarity with themselves. The latter get the same list, as well as a good sampling of applicable algorithms. The book could be given "as is" to the more talented or motivated students, though for many the guidance of a coach along with exposure to other material would be advisable. It is not a substitute for an algorithms textbook but it is an excellent accompanying book and one which comes highly recommended.

References

1. https://sites.google.com/site/stevenhalim/
2. http://uva.onlinejudge.org

Richard Forster

Olympiads in Informatics, 2012, Vol. 6, 229–231



Teaching Programming for Secondary School: A Pedagogical Content Knowledge Based Approach

Author: Mara SAELI
Publishing house: Eindhoven University of Technology
Country: The Netherlands
Year of edition: 2012
Language: English
Number of pages: 164
ISBN: 978-90-386-3084-7

In this book (a PhD thesis) Mara Saeli reports her research about "teaching programming in secondary school with the aim of portraying the Pedagogical Content Knowledge of this subject", as is mentioned on the backside of it. The reason for a review in this journal is that it contains a meticulous description and analysis of why teaching of programming in secondary school is important, what should be taught, what problems and limitations students have while learning programming and how the teaching could be performed. All these issues have a great relevance for teachers who prepare students for Olympiads in informatics and for researchers who investigate this Olympiads. Problem solving, perhaps one of the most important aspects for Olympiad's participants turns out to be a very relevant part of the teaching and learning of programming.

The study of Saeli takes Pedagogical Content Knowledge (PCK) as a starting point. PCK is defined by Shulman (1986): it is the amalgam or combination of knowledge of a teacher of the content of a specific topic and the knowledge of the pedagogy of it: giving insights into educational matters relative to the learning and teaching of a specific topic. The PCK of a teacher starts during the education as a teacher in a particular field and grows by enhancing his theoretical background and teaching experiences. Teachers with good PCK are teachers who can transform their knowledge of a topic into something accessible for the learners.

In order to operationalize the concept of PCK for programming in secondary school, the author used the reformulation of PCK by Grossman (1990), namely the answers to the

four key questions: why to teach . . ., what to teach . . ., learning difficulties of . . ., and how to teach . . .? Saeli started to answer these questions by performing a broad literature review with respect to programming in secondary school. She found the following answers.

- Why: programming enhances students' problem solving skills and offers them a learning environment which includes aspects of different disciplines, gives opportunities to use modularity and transferability of the knowledge and/or skills, and to work with a multi-disciplinary subject.
- What: a list of concepts/aspects which a programming curriculum should include, for instance knowledge of data, instructions and syntax of a programming language, but also primitive expressions, means of combination and of abstraction, strategies, and programming sustainability which refers to the ability to create user friendly and attractive software that takes care of ethical and privacy issues.
- Learning difficulties: difficulty to instruct the machine about the solution of a problem, the tendency to converse with a computer as if it was human, the tendency the students maintain a local, limited point of view, failing to find a suitable solution.
- How: for instance offering a simple programming language so students can focus on the syntax, carefully choosing a diversity of problems in order to focus the students on algorithmic thinking.

In the literature these answers are not very well connected to each other. So, it was necessary to go into more detail with respect to each commonly taught topic. To this purpose Saeli organized six workshops in four different countries (Lithuania, Italy, Belgium and the Netherlands) with about five experienced informatics teachers in each workshop. Each workshop started with the question: What are the core concepts ("Big Ideas") of programming? Every participant answered this question individually. During the second part of the workshop the participants discussed one of more of the formulated Big Ideas using the following eight questions as a guideline:

1. What do you intend the students to learn about this Big Idea?
2. Why is it important for the students to know this Big Idea?
3. What else do you know about this Big Idea (and you don't intend students to know yet)?
4. What are the difficulties/limitations connected with the teaching of this Big Idea?
5. What do you think students need to know in order for them to learn this Big Idea?
6. Which factors influence your teaching of this Big Idea?
7. What are your teaching methods (any particular reasons for using these to engage with this Big Idea?
8. What are your specific ways of assessing students' understanding or confusion about this Big Idea?

Question 1 is about the *why*, question 2 about the *what*, questions 3, 4, 5, 8 about the *difficulties*, and questions 6, 7 about the *how*. This method to obtain data as a basis for portraying the PCK are based on CoRe (Content Representation), developed by Loughran *et al.* (2004) to give a narrative account providing an overview of how teachers approach the teaching of a specific topic in science in secondary school.

The results of the workshops turned out to be the following seven Big Ideas: control structures with focus on loops, data structures, arrays, problem solving skills, decomposition, parameters and algorithms. These results are in line with the research literature.

The next part of this study was to find out if and how the Dutch textbook support teachers with respect to the teaching of this seven Big Ideas. The reason for performing this and the following part of study is that the Dutch informatics teachers have almost all originally another disciplinary background and got a license for teaching informatics by a re-educating programme focused on informatics Content Knowledge, comparable to one year of a university Bachelor study in informatics. Using the developed portray of the PCK of programming as a referential framework the conclusion is, in most general terms, that the textbooks are helpful as far as the Content Knowledge is involved, but failed for the Pedagogical Knowledge.

The last part of the study was an investigation among the Dutch secondary informatics teachers: how do they assess their own PCK of programming? Here again the developed portray of the PCK was the basis for developing the online questionnaire that served as the research instrument. Saeli argued convincingly the validity of this instrument. About a quarter of all the almost 350 Dutch secondary informatics teachers filled in the questionnaire, but only 69 did so completely. (The reader should know that informatics is an elective course in the curriculum of the upper part of the Dutch secondary schools preparing to higher vocational and university education). The outcome of this study confirmed that the Dutch secondary informatics teachers have, in general, a poor PCK as a consequence of the fact that that they are re-educated teachers form other disciplines. As for their Pedagogical Knowledge, it turns out that this is sufficient, but not for extracurricular topics – this is at least problematic for a fast developing subject as informatics. This conclusion is of course closely related to the fact that their Content Knowledge must be qualified as low. These findings are even more problematic because of the conclusion that the Dutch textbooks do not really support the teachers with respect to the Pedagogical Knowledge.

Saeli is maybe not the first researcher who tried to portray the PCK of programming, but certainly she is the first who has done so in a systematic way. The four chapters of her thesis which report her studies are in the meantime all published in peer reviewed scientific journals. Interesting for the readers of this journal is her recommendation to use tasks borrowed from Olympiads or designed analogue to Olympiad tasks in order to improve the teaching of programming in secondary schools.

References

Grossman, P.L. (1990). *The Making of a Teacher: Teacher Knowledge and Teacher Education*. New York, Teacher College Press, Columbia University press.

Loughran, J., Mulhall, P., Berry, A. (2004). In search of pedagogical content knowledge in science: developing ways of articulating and documenting professional practice. *Journal of Research in Science Teaching*, 41(4), 370–391.

Schulman, L. (1986). Those who understand: knowledge growth in teaching. *Educational Researcher*, 15, 4–14.

Bert Zwaneveld

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).

Abstracting/Indexing

OLYMPIADS IN INFORMATICS is abstracted/indexed by

- Cabell Publishing,
- Central and Eastern European Online Library (CEEOL),
- EBSCO,
- Educational Research Abstracts (ERA).

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, 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, 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. `http://rice.edn.deakin.edu.au/archives/JITTE/j113.htm`.

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 (2007–2011). `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 Dagienė
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`