A mathematician reflecting on the International Olympiad in Informatics

Benjamin A. Burton*

Abstract

In July 2013, students from 80–90 countries will descend upon Australia to take part in the International Olympiad in Informatics (IOI). On the surface the IOI is a computer programming competition, but in fact it involves a great deal of both mathematical technique and mathematical creativity. In this short article we introduce the readers to the IOI and the mathematics within.

1. Introducing the IOI

The International Olympiad in Informatics (IOI) is one of the five broad-brush Science Olympiads for high-school students, which also cover biology, chemistry, physics, and of course mathematics. Founded under the auspices of UNESCO in 1989, the IOI is one of the youngest Science Olympiads, but it has grown quickly to now include over 80 countries, making it the second-largest (behind only mathematics).

Despite informatics being roughly synonymous with computer science, the IOI has always had strong associations with mathematicians. Locally, the Australian team is trained by the Australian Mathematics Trust, and all of Australia's team leaders over the past decade have been trained mathematicians[1]. Internationally, the first IOI was organised by Petar Kenderov, a highly-respected Bulgarian mathematician, and it is common to find fellow mathematicians amongst the myriad of team leaders and deputies.

It was recently announced that Australia will host IOI 2013, with the event to be held at The University of Queensland in partnership with the Australian Mathematics Trust. This is a great honour for both the mathematics and computer science communities in Australia, and readers will doubtless hear more about the event as it draws nearer. In the meantime, this short article aims to (i) introduce the IOI to readers with whom it is unfamiliar, and (ii) illustrate the mathematics that runs throughout the competition.

*School of Mathematics and Physics, The University of Queensland, Brisbane QLD 4072.
Email: bab@maths.uq.edu.au

[1]Robbie Gates (leader from 1999–2000) holds a PhD in category theory, the author (leader from 2001–2008) holds a PhD in geometry and topology, and Bernard Blackham (leader since 2009) holds a BCM with a major in pure mathematics.

2. The structure of the IOI

At its core, the IOI is about algorithm design and computer programming. To solve an IOI task, students must design an algorithm that is both *correct* and *efficient*. They must then code this algorithm into a computer program, which they submit at the end of the contest for judging.

Algorithm design has a long history in mathematics — consider, for instance, Euclid's algorithm to find the greatest common divisor, a clever and extremely fast algorithm designed thousands of years ago and still in common use today. For a more modern example, the author, Rubinstein and Tillman have recently resolved an old topological conjecture [3]; although this is a theoretical result, it would not have been possible without significant work on efficient algorithms to support the underlying computations.

Although entry-level competitions in this field might focus more on computer programming (particularly where high-school students have little algorithmic training), the focus of the IOI lies squarely on algorithms — tasks are chosen to avoid excessive amounts of coding, and a good student will typically spend much of the contest designing algorithms using pen and paper. During training, students will often deem a task solved once they find a suitable algorithm, without having touched the computer at all. In a sense, computer programming acts as the means of communicating a solution in the IOI, much as a written proof does in the International Mathematical Olympiad. This has both benefits and drawbacks, as discussed further in [1].

The format of the IOI is as follows. Students sit two five-hour exams in front of a computer, each with 3–4 tasks to solve. A sample task is illustrated in Figure 1, taken from the inaugural Asia-Pacific Informatics Olympiad. Here we see the typical components of an IOI task: a description of a mathematical problem with some surrounding story, descriptions of the input and output requirements for the student's computer program, bounds on the size of the input, and limits on the program's running time and/or memory usage. Figure 1 is merely a brief summary of the full task description, which can be downloaded from http://apio.olympiad.org/.

Evaluation of IOI solutions is also done by computer. The judges prepare an *official data set*, consisting of input files ranging from small to large, and from simple to pathological. For each input file, a student scores points if their program gives the correct output within the given time and memory limits. In this way, students with optimal algorithms will score 100%, and students with correct but slow algorithms will score partial marks according to which input files their programs can solve within the time limit.

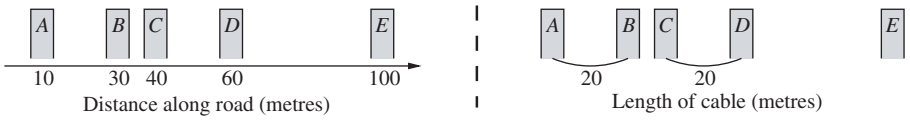
It is essential that the official data set be constructed carefully, and contest organisers typically put a great deal of time into ensuring that the spread of cases is fair, thorough and discriminates well. Computer evaluation also has benefits and drawbacks [1], [5], [7], and the IOI community is continually seeking ways to improve the experience.

Task: 'Backup'

Asia-Pacific Informatics Olympiad 2007 (by Mathias Hiron)

You are given the locations of n office buildings along a single road, and you wish to join these buildings in pairs with network cables so that the offices can back up each other's data. You are only able to lay k cables (thereby pairing off $2k$ buildings in total, with the remaining $n - 2k$ buildings left isolated).

Your task is to decide which buildings to join so that you use the *smallest possible total length of cable*. You must join $2k$ distinct buildings (i.e. you cannot have more than one cable running out of a single building).



For instance, consider the $n = 5$ buildings on the road above, and suppose you are able to lay $k = 2$ cables. The best option is to join A–B and C–D, giving a total length of $20 + 20 = 40$ m of cable as illustrated on the right hand side.

Input and output: The input to your program will be the integers n and k, followed by the locations of each of the n buildings along the road. Your output should be the smallest possible total length of cable that you can use.

Limits: The input will satisfy $2 \leq n \leq 100\,000$ and $1 \leq k \leq \frac{n}{2}$. Your program must run within 1 second, and may use up to 32 Mb of memory.

Figure 1. A task from the Asia-Pacific Informatics Olympiad

Not all IOI tasks follow the general format illustrated in Figure 1. Other examples include *output-only tasks* (where students are given all of the judges' input files and they must generate their output offline), or *interactive tasks* (where students' algorithms must interact with software provided by the judges). See [9] for details and examples.

3. The mathematics of algorithm design

To solve an IOI task can require a great deal of mathematical skill. This includes not only working with mathematical concepts (such as combinatorics, geometry, graph theory and recurrence relations), but also creative application of mathematical techniques (such as case analysis, complexity analysis, invariant analysis, and, of course, proof and disproof).

We shall not attempt to discuss the full range of relevant mathematical concepts and techniques in this short article; instead see [1] for an overview or [10] for a detailed syllabus. Here we simply offer an illustration, namely the solution to the task *Backup* from Figure 1.

3.1. A brute-force solution

A simple 'brute force' algorithm might be to run through all possible ways in which the $2k$ cables could be laid, and choose the layout with the smallest total length. However, there is a *very* large number of possibilities to consider. We can reduce this number through the following observation (proof left to the reader).

Lemma 1. *In the optimal solution, every cable must join two adjacent buildings.*

Although this helps, it does not help nearly enough. Even with Lemma 1, there are still $\binom{n-k}{k}$ choices of cables to consider (exercise!). With just $n = 100$ and $k = 25$ this gives around $5{\times}10^{19}$ possibilities, which a modern computer could not process in a millennium (let alone a second). The maximal case $n = 100\,000$ simply does not bear thinking about. Clearly we must find a more clever solution.

3.2. A dynamic programming solution

It often helps to decompose a large problem into a family of smaller, similar problems. With this in mind, let $f(b, c)$ denote the shortest possible cable length if we restrict our attention to the first b buildings and lay precisely c cables (where $0 \le b \le n$ and $0 \le c \le \frac{b}{2}$). The final solution that we seek is $f(n, k)$, and at the other end of the spectrum we clearly have $f(b, 0) = 0$ for all b. Unfortunately, we know little about the values in between.

Our challenge then is to find a recurrence relation that links together the different values $f(b, c)$, so that we can find our solution by incrementally computing values of $f(b, c)$ for all b, c. This is an example of a technique known as *dynamic programming*. With a little thought we arrive at the following formula.

Lemma 2. *Suppose that* $1 \le c \le b/2$. *Then* $f(b, c)$ *is the smaller of* $D + f(b-2, c-1)$ *and* $f(b-1, c)$, *where* D *is the distance between the* $(b-1)$*th and* b*th buildings.*

Why does this hold? If the bth building has a cable attached, then by Lemma 1 this cable goes to building $b-1$ with length D, and we are left to lay the remaining $c-1$ cables amongst the first $b-2$ buildings. If the bth building does not have a cable attached, then we must lay all c cables amongst the first $b-1$ buildings.

Our algorithm is now to compute $f(2, 1), \ldots, f(n, 1)$, then $f(4, 2), \ldots, f(n, 2)$ and so on, each time using Lemma 2 for the computation. Eventually we arrive at $f(n, k)$ and the algorithm is complete.

This is certainly faster than brute force, but is it fast enough? Our algorithm requires (roughly) up to nk computations. For $n = 100$ and $k = 25$ this total is 2 500, which fits easily within one second. However, for our maximal case

$n = 100\,000$ and $k = 50\,000$ we have up to 5 billion computations, which is still too much to squeeze into our time limit[2]. We must do better still.

3.3. Fixing the greedy solution

It is tempting to try a *greedy solution*, where we repeatedly lay the shortest allowable cable until all k cables have been used. A little experimentation shows this to be incorrect — in the Figure 1 example, for instance, laying the shortest cable *B*–*C* (10 m) would then force us to lay the much longer cable *D*–*E* (40 m), giving a suboptimal solution with 50 m of cable in total.

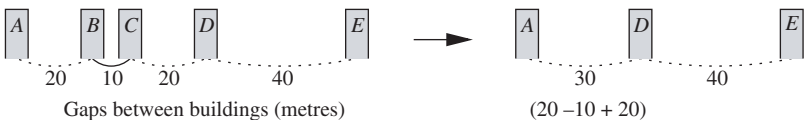
However, what if we allow ourselves to undo a bad decision? That is, we lay the shortest possible cable but also create a method for removing it later on. Further thinking along these lines leads to the following observation.

Lemma 3. *Suppose we have not yet laid any cables, and that the shortest possible cable runs from building b to $b+1$. If the optimal solution does* not *lay this cable, then it must lay two cables immediately on either side (i.e. from $b-1$ to b and from $b+1$ to $b+2$).*

We prove this by showing that, if we do not lay both cables on either side, then we can swap some cable in our solution for a shorter cable (yielding a contradiction). Again the details are left for the reader.

We therefore begin our algorithm by choosing the shortest possible cable; suppose this runs from building b to $b+1$ as before. We are now free to lay a cable anywhere within the range $1, \ldots, b-1$ or within the range $b+2, \ldots, n$. In addition, we are free to *replace* our original cable with two cables joining $b-1$ to b and $b+1$ to $b+2$.

With some creative rearrangement of our diagram, we can make this look like a new version of the original problem. Remove buildings b and $b+1$, and squeeze the surrounding buildings together so that $b-1$ and $b+2$ are separated by a gap of size $D_{b-1,b} - D_{b,b+1} + D_{b+1,b+2}$ (where $D_{i,j}$ is the old distance between buildings i and j). If we ever try to lay a cable across this new 'artificial' gap, we remember that in reality we must delete the old cable from b to $b+1$ and replace it with two on either side. The full rearrangement operation is illustrated below, where we begin by choosing the 10 m cable *B*–*C*.



[2]These numbers might seem arbitrary, but with experience students quickly gain an order-of-magnitude feel for how many computations can fit into one second. A rough figure nowadays is around 50 million.

Our new diagram now looks like another empty road, but with only $n-2$ buildings and $k-1$ cables to lay. We continue choosing the shortest possible cable and adjusting our diagram until all k cables have been laid, whereupon we have our final solution.

Returning to our example, the shortest gap on this new road is the 30 m gap A–D. We lay this 'artificial' cable, which entails replacing the old B–C with both A–B and C–D. With no more cables to lay, we have a final (and optimal) solution of total length 40 m.

But what of running time? Our algorithm takes k iterations, each of which requires us to find the shortest possible cable and then adjust the diagram accordingly. This can be made faster if we store the gaps *between* buildings, not the locations of the buildings themselves. Using techniques from computer science, we store the gaps in a priority queue where each find-and-adjust operation has worst-case running time proportional to $\log_2 n$. The entire algorithm then has running time proportional to $k \log_2 n$, which for $k = 50\,000$ and $n = 100\,000$ is under a million, fitting comfortably within our one second time limit.

4. Further reading

Australia has a well-established program of events leading up to the IOI; this program is detailed by the author in [2]. On the international front, Cormack *et al.* [5] give more detailed descriptions of the IOI and related competitions.

Skiena and Revilla [8] offer a problem-filled book aimed specifically at students training for the IOI and related contests. For this year's 20th anniversary of the IOI, Verhoeff [9] has published a paper covering the history and evolution of IOI tasks over the years.

For high-school students with no training in programming or algorithm design, the Australian Informatics Competition [4] and the Beaver Contest [6] are accessible entry-level events that aim to encourage and develop algorithmic thinking.

Anyone is welcome to try their hand at solving IOI-style problems (and easier tasks also!) on the Australian training site at http://orac.amt.edu.au/aioc/train/, or the USACO training site at http://www.usaco.org/. The official IOI website is http://www.ioinformatics.org/.

References

[1] Burton, B.A. (2007). Informatics olympiads: approaching mathematics through code. *Mathematics Competitions* **20,** 29–51.

[2] Burton, B.A. (2008). Informatics olympiads: challenges in programming and algorithm design. 31st Australasian Computer Science Conference (ACSC 2008) (Wollongong, NSW, Australia). Dobbie, G. and Mans, B. (eds). CRPIT, Vol. 74, ACS, pp. 9–13.

[3] Burton, B.A., Rubinstein, J.H. and Tillman, S. (2009). The Weber-Seifert dodecahedral space is non-Haken. Preprint, arXiv:0909.4625.

[4] Clark, D. (2006). The 2005 Australian Informatics Competition. *Australian Mathematics Teacher* **62,** 30–35.

[5] Cormack, G., Munro, I., Vasiga, T. and Kemkes, G. (2006). Structure, scoring and purpose of computing competitions. *Informatics in Education* **5,** 15–36.
[6] Dagienė, V. (2006). Information technology contests — introduction to computer science in an attractive way. *Informatics in Education* **5,** 37–46.
[7] Opmanis, M. (2006). Some ways to improve olympiads in informatics. *Informatics in Education* **5,** 113–124.
[8] Skiena, S.S. and Revilla, M.A. (2003). *Programming Challenges: The Programming Contest Training Manual*. Springer, New York.
[9] Verhoeff, T. 20 years of IOI competition tasks. *Olympiads in Informatics* **3,** 149–166.
[10] Verhoeff, T., Horváth, G., Diks, K. and Cormack, G. (2006). A proposal for an IOI syllabus. *Teaching Mathematics and Computer Science* **4,** 193–216. Current version maintained by Michal Forišek at http://people.ksp.sk/~misof/ioi-syllabus/.

Benjamin Burton has been involved with the Informatics Olympiad since Australia became a regular participant a decade ago. He currently sits on the international scientific committee that guides and oversees the IOI, and will chair the host scientific team when the Olympiad comes to Brisbane in 2013. He finds the IOI a stimulating counterpoint to his mathematics research in computational geometry and topology at the University of Queensland.