Olympiads in Informatics, 2007, Vol. 1, 79–89

New Task Types at the Canadian Computing Competition

Graeme KEMKES
Department of Combinatorics and Optimization, University of Waterloo
200 University Ave West, Waterloo, Ontario, N2L 3G1 CANADA
e-mail: gdkemkes@waterloo.ca

Gordon CORMACK, Ian MUNRO, Troy VASIGA
David R. Cheriton School of Computer Science, University of Waterloo
200 University Ave West, Waterloo, Ontario, N2L 3G1 CANADA
e-mail: {gvcormack, imunro, tmjvasiga}@cs.uwaterloo.ca

Abstract. In the 2006 competition workshop held at Dagstuhl, Germany, there were many fruitful discussions about the difficulties facing computer science competitions today. Our competitions have several purposes: to foster interest in the discipline, to create a community, and to promote achievement, for example. Balancing these various purposes may require many tradeoffs. Several participants identified areas where we need to improve our competitions. Tom Verhoeff (2006) discussed the problem of giving a meaningful ranking to incorrect solutions. Maryanne Fisher and Tony Cox (2006) pointed out that some groups of students are disadvantaged by the present format. Many participants made suggestions for improving the competitions. One of the suggestions in our paper (Cormack *et al.*, 2006) was open-ended tasks. A task is *open-ended* if there is no known optimal solution to the problem. Points are awarded for correct submissions in proportion to how well they do. A vast number of real-world applications, such as pattern recognition, information retrieval, and compiler optimization appear suitable for this purpose. At Canada's national informatics olympiad, the Canadian Computing Competition, we have been exploring several of these suggestions. In this paper we describe the experiments we have performed and we analyze whether the objectives have been achieved.

Key words: computing competitions, open-ended tasks, informatics in Canada.

1. Introduction

Computing competitions have generally been focused on determining the top students who can solve algorithmic problems. For various reasons (efficiency, quantitative bias, competitiveness) the focus has been on tasks that are known to have an optimal solution: the student must find this optimal solution to get full credit on the given task.

This narrow goal, however, should not be the only goal of computing competitions. In particular, computing competitions should also focus on other goals, if they are to serve the informatics community as a whole and cultivate continued growth in the field. These other goals should include fostering interest in the discipline in order to ensure a contin-

ued source of future informatics students; to promote achievement of *all* sorts, in order for students to have a sense that informatics is an accessible field of study; and to create a community amongst students that have an interest in the field of informatics, since most students (in the Canadian context, at least) are geographically isolated from other students who have an interest in informatics. Some of these goals have been highlighted by Verhoeff (2006) (to award students who do not have the "best" solution) and Fischer and Cox (2006) (to promote achievement amongst students who are generally disadvantaged).

One idea that the Canadian Computing Competition (CCC) has adopted to address some of these other goals of computing competitions is the use of open-ended tasks. We call a task *open-ended* if there is no known optimal solution to the task. It follows from this definition that all NP-complete problems are open-ended, but there are also problems that do not fit into the NP-complete model that also have open-ended features about them (such as text compression, spam recognition, character recognition, etc.). These latter open-ended problems tend to be focused on current research areas, and thus provide students with "real-world" problems, that can, in turn, be a motivating factor to consider the field of informatics as worthwhile to pursue in higher education.

This paper outlines the Canadian experience of computing competitions to place the "typical" computing experience of secondary school students in context. We then outline the benefits of open-ended problems as a contrast to the typical experience students have in computing competitions or computing education. Following this, we outline two open-ended tasks which have different features. We conclude this paper by analyzing the effectiveness of these particular tasks and outline issues to consider when creating other open-ended tasks.

2. The Canadian Experience

2.1. *Informatics in Canada*

In Canada, the educational system is controlled by each provincial or territorial unit (there are 10 provinces and 3 territories), and the amount of informatics education (typically called "computer studies" in Canadian secondary schools) varies greatly between regions. If there is a computer studies course in secondary schools, the curricula focuses on learning programming languages (especially at the Grade 11 level, where Java is the prevalent language of instruction). Most secondary schools have at least one computer lab which is connected to the internet. The challenge in creating a national competition is to manage the diversity of the competing students: tasks should be approachable to (almost) all levels of ability and yet challenging enough for students with a very high level of experience and ability.

2.2. *Organization of the Canadian Computing Competition*

The Canadian Computing Competition (CCC) is divided into two stages: Stage 1 and Stage 2.

The Stage 1 competition is further subdivided into two separate contests: the Junior level (intended for students with limited programming experience, typically Grade 9 or 10 students, who are typically 14 or 15 years old) and the Senior level (intended for students with more programming experience, or students who have competed in the CCC before, typically Grade 11 or 12 students, who are typically aged 16 or 17 years old). This competition is written in secondary schools across Canada, and is composed of 5 algorithmic problems to be solved in 3 hours, with the score based exclusively on the student code providing the correct output on given input. Any programming language is allowed (students have used Visual Basic, Pascal, Turing, PHP, C, C++, Java, Perl) and the contest environments are heterogeneous. The evaluation of the student programs is done by the individual teacher, and results are sent into the CCC office for final processing. The top students in five geographical regions of the country are awarded prizes.

The Stage 2 competition gathers the top 20 students who competed in the Senior competition and provides a week-long program of activities (social activities, lectures by faculty members) and two competition days (similar to the IOI: three problems per day in a three-hour time period) where the allowed programming languages are C, C++ and Pascal (the current "official" IOI programming languages). The top four competitors are selected for the Canadian IOI team, again, based on the evaluation of their programs in terms of providing the correct output on given input.

3. Open-Ended Problems

As outlined in Section 1, we define an open-ended task as one where there is no known optimal solution. In this section, we describe the benefits of this type of task in terms of addressing the computing competition goals of fostering interest in the discipline, creating a community and promoting achievement.

3.1. *Fostering Interest in the Discipline of Informatics*

One method to foster interest in the discipline is to provide students with problems that are manageable (i.e., they are able to develop at least a partial solution to them) and yet are interesting or appealing. To appeal to students, problems that they encounter should be realistic and meaningful. By asking open-ended problems, such as spam recognition or text-compression, which are both research-level problems, students can be motivated to consider computer science as an active field, where not all of the interesting problems have been solved. In other words, the problems are "real". Moreover, as with other discoveries and insights, it may be that the idea of a student that causes forward progress to be made on these problems. Finally, real data can be used to evaluate this type of task (such as real email messages, or real text that needs to be compressed), and the use of real data further underscores the applicability of computer science to real problems.

3.2. *Creating a Community*

Though the problems stated here were not posed as such, open-ended tasks can be done collaboratively. This more community-based version of the task utilizes the exploratory nature of the open-ended task, and is more likely to lead to future discussions which are more lengthy and exploratory in nature. This contrasts to the more typical closed-task, where once the optimal solution is known, there is nothing further to discuss, since all avenues of exploration have been exhausted. Furthermore, as noted by Fischer and Cox (2006), typical programming tasks may appeal to a certain subsection of the population, but with open-ended tasks, there is the potential to appeal to a broader group of students, since students with various levels of ability may be able to collaborate and create a meaningful and insightful solution.

3.3. *Promoting Achievement*

When assigning marks for open-ended tasks, evaluators can reward students on other dimensions of achievement, including the typical measurement of technical skill. For example, it is possible to measure originality of a solution (to some extent) to reward interesting use of data structures, or thought processes or consideration of a special subcase.

As well, since there is no optimal solution, there is no perfectly correct solution, and thus, there is no incorrect solution (other than solutions which do not compile, or do not meet the specifications). Therefore, "incorrect" solutions would not receive zero points, and thus, students who can follow the minimal requirements of the task can have a non-zero score and a sense of accomplishment. In particular, it is possible to give students a basic strategy which, if implemented correctly, gives students a minimum level of achievement. This ensures all students have some chance of providing a partial solution to the problem: frustration or hopelessness is lessened.

Since open-ended tasks are typically evaluated on a continuous scale, a wide diversity of achievement can be rewarded, and, as an added benefit, the probability of tie scores is significantly reduced. (The concept of continuous scoring is discussed more thoroughly in (Cormack *et al.*, 2006).)

4. Open-ended Tasks at the Canadian Computing Competition

In this section, we outline two open-ended tasks: Paint-by-Number (an NP-complete open-ended task) and Codec (a non-NP-complete open-ended task) that were used at the Canadian Computing Competition.

Codec is a problem to design and implement an algorithm to compress and decompress English text.

Paint-by-numbers is a logic puzzle (similar to Sudoku or other number-puzzle problems where a solution is a satisfying assignment to a collection of constraints) which is computationally difficult.

We first present the task descriptions as given to the students in the CCC Stage 2, then briefly discuss the student attempts and evaluate the success of these tasks. We conclude by offering some suggestions for future open-ended tasks.

4.1. *Gordon Cormack, Codec (CCC 2006 Stage 2)*

The problem of lossless data compression is to transform some data into a compressed form such that:

(a) the original can be reproduced exactly from the compressed form.
(b) the compressed form is as small as can reasonably be achieved.

You are to write two programs – `compress` that performs lossless compression and `decompress` that reproduces the original data from the compressed form. The data to be compressed will be plain English text represented using printable ASCII characters (i.e., all characters with ASCII values between 32 and 126 inclusive). The compressed form is a string of binary bits. For convenience, we will represent this string of bits as a character string containing only 0s and 1s.

`compress` reads the original data from the file `codec.in` and writes the compressed form `compress.out`. `decompress` reads the compressed data from a file called `compress.out` and writes the corresponding original data to `codec.out`. Of course, `codec.in` and `codec.out` must be the same file. Pictorially, we have the following flow of information:

`compress` must output only 0s and 1s and `decompress` must exactly reverse the effects of `compress`. That is, condition (a) above must hold for any English text. If `compress` and `decompress` meet these criteria, your score will be determined by the relative size of the input and output by

$$\text{score(as\%)} = 50 \cdot \sqrt{\frac{8c - b}{c}},$$

where c is the total number of characters in the original text and b is the number of bits in the compressed form. Note that scores may exceed 100%, but scores that are less than 0 will be given 0% (i.e., no negative marks will be given, but bonus marks may be awarded).

Discussion and Hints

It is well known that any ASCII character can be represented using 8 bits. Such a representation would achieve a score of 0 using the formula above. Since there are fewer than 128 possible symbols in the input, it is possible to represent each one with 7 bits. Such a representation would receive a score of at least 50%.

A smaller representation can be achieved, with high probability, by observing that some letters are more common than others. Suppose we estimate that a character α occurs with probability p_α in a given context. The best possible code will use $-\log_2(p_\alpha)$ bits to represent that character. If one estimates p_α one can construct a ***prefix*** code with about $-\log_2(p_\alpha)$ bits for each character in the following manner:

- build a binary tree with one leaf for each character α;
- organize the tree so that the depth of α is approximately $-\log_2(p_\alpha)$;
- use a binary representation of the path (0 = left, 1 = right) to represent α in the compressed data.

One way to estimate p_α is simply to compute the fraction of characters equal to α in a sample of data similar to that to be compressed. Another is to use an adaptive method, in which the data is compressed one character at a time, and the sample consists of the text already compressed. A sample of English text is available in the file `sampleText.txt`.

It is also possible to estimate p_α using the context in which it occurs; for example, in English a "q" is very likely to be followed by a "u" (e.g., quick, quack, quit, quiz, but not qiviut, which happens to be the wool of a musk-ox).

Use this information, or any other information at your disposal, to build the best Compressor and Decompressor you are able.

Input

The input to `compress` will consist of n characters ($1 \leqslant n \leqslant 1000000$), as described above.

The input to `decompress` will consist of m 0s and 1s ($1 \leqslant m \leqslant 8n$).

Output

The output of `compress` will be a sequence of 0s and 1s, with no other characters (i.e., no newline characters should be outputted).

The output of `decompress` is a sequence of at most 1000000 uppercase letters, lowercase letters, spaces, newlines and punctuation symbols.

Sample Input (to compress)

```
To be or not to be?
```

Possible Output for Sample Input (`compress`)

```
01110010000101100001001101000011000100111100000111100100001011011111
```

Sample Input (to `decompress`)

```
01110010000101100001001101000011000100111100000111100100001011011111
```

Output for Sample Input (`decompress`)

```
To be or not to be?
```

Explanation

The sample compressor systematically uses the following codes for each of the input characters:

<space>	000
o	010
n	01100
r	01101
T	01110
t	011110
?	011111
b	10
e	11

The compressed output uses these codes, as shown below:

```
0111001000010110000100110100001100010011110000011110010000101101111
TTTTTooo  bbee  ooorrrrr  nnnnnooottttttt  ttttttooo  bbee??????
```

4.2. *Paint by Numbers (Sandra Graham, CCC 2006 Stage 2)*[1]

Way back before you were born, there was a really bad craft/hobby called *paint-by-numbers*: you were given a line drawing, with numbers in each enclosed region, and the number corresponded to a particular colour. An example is shown below:

The problem you have to solve is much more linear, in a way.

You will be given an n-by-m grid ($1 \leqslant n, m \leqslant 32$) which you will "colour" in with either a dot ('.') or a star ('*').

Of course, the grid will not be specified in the usual paint-by-numbers way, since this would be too easy.

Instead, you will have to infer which cells are blank and which contain a star. The only information you will be given is a collection of $n + m$ sequences of numbers, one sequence for each row and column. The sequence will indicate the size of each continuous block of stars. There must be at least one dot between two consecutive blocks of stars.

An example is shown below (which is supposed to look fish-like):

		1	1					
		1	1	2	2	4	4	2
2	2							
	5							
	5							
2	2							

(Unsolved Puzzle)

		1	1					
		1	1	2	2	4	4	2
2	2	*	*	.	.	*	*	.
	5	.	.	*	*	*	*	*
	5	.	.	*	*	*	*	*
2	2	*	*	.	.	*	*	.

(Solved Puzzle)

[1] It has been pointed out by one of our referees that this task was used at IOI 1992 as the problem "Islands in the Sea". The problem description can be found at http://olympiads.win.tue.nl/ioi/ioi92/tasks92.txt

You may notice that some paint-by-number patterns are not uniquely solvable. For example,

	1	1
1		
1		

has two solutions

	1	1
1	*	.
1	.	*

and

	1	1
1	.	*
1	*	.

For this problem, you may assume that *any* solution which satisfies the specification is correct.

You should note that 50% of the marks for this question will come from test cases where $1 <= n, m <= 6$.

Input

Input consists of a total of $n+m+2$ lines. The first line of input consists of an integer n $(1 \leqslant n \leqslant 32)$, the number of rows. The second line of input consists of an integer m $(1 \leqslant m \leqslant 32)$, the number of columns. On the next n lines, there will be sequences which describe each of the n rows (from top to bottom). Each line will contain some positive integers, with a space between adjacent integers, and the sequence will terminate with the integer 0. The next m lines describe the m columns (from left to right), the same format as the rows are specified.

Output

Output consists of n lines, each line composed of m characters, where each character is either a dot ('.') or a star ('*').

Sample Input 1

```
4
7
2 2 0
5 0
5 0
2 2 0
1 1 0
1 1 0
2 0
2 0
4 0
4 0
2 0
```

Sample Output 1 for Sample Input 1

```
**..**.
..*****
..*****
**..**.
```

Sample Input 2

```
4
4
2  1  0
3  0
3  0
1  1  0
4  0
3  0
3  0
1  0
```

Sample Output for Sample Input 2

```
**.*
***.
***.
*.*.
```

5. Conclusion

5.1. *Effectiveness of the Particular Open-Ended Tasks*

Based on the solutions provided by the competitors, despite the fact that Codec required students to create two pieces of software (the encoder and decoder) to work together correctly, it was the case that 18 of the 22 students made a serious attempt at this problem. On the other task, however, only 7 out of 22 made a serious attempt at Paint-by-Numbers.

There could be a number of factors that may explain the difference in the attempted solutions. In a typical contest, many students leave a question blank, since there are only 3 hours to complete the competition and there are 3 difficult tasks. Students must make difficult time-management decisions in order to optimize the number of points attained. Thus, rarely all students complete all problems. In the problem statement of Codec, we described a simple base-line solution, whereas for Paint-by-Number there was no base-line solution given, and even the statement that significant marks may be attained by dealing with "small" cases was not enough to encourage more students to attempt this task. Students were given an outline for the baseline solution in Codec, which they are to implement, which models more closely their typical classroom experience, unlike Paint-by-Numbers which required students to devise their own solution from scratch.

However, we feel that both problems were successful, in the sense that students were creative and implemented different approaches. For instance, in the Paint-by-Numbers, we saw submissions that used

- recursive search,
- recursive search with heuristics (such as dealing with either completely full or complete empty rows or columns),
- simulated annealing, and
- random search.

For Codec, the majority of students implemented the baseline suggested solution, but others attempted a dictionary encoding using a variety of methods, including using a prefix tree and frequency counting.

5.2. *Future Considerations and Recommendations*

We have found that open-ended tasks are a useful tool to provide interesting problems for a wide diversity of students: students at both the highest level of ability and at the lower level of ability can both have positive achievement on these tasks.

We suggest that it is important to describe a baseline solution and promise that a certain number of marks will be given for implementing this baseline solution. Providing this assurance gives students from a wide diversity an opportunity to succeed.

We found that open-ended tasks encouraged discussion amongst the students, which both helped foster interest in the discipline of informatics and provided a sense of community to the competitors.

We will be exploring the use of collaborative versions of open-ended tasks in future Stage 2 competitions (perhaps as a task outside the typical individual tasks) and we hope that our experience can be transferred to other problems, and other competition organizers can use open-ended tasks to enhance their competitions.

References

Cormack, G., G. Kemkes, I. Munro and T. Vasiga (2006). Structure, scoring and purpose of computing competitions. *Informatics in Education*, **5**, 1–22.

Fisher, M., and A. Cox (2006). Gender and programming contests: mitigating exclusionary practices. *Informatics in Education*, **5**, 47–62.

Verhoeff, T. (2006). The IOI is (not) a science olympiad. *Informatics in Education*, **5**, 147–159.

G.V. Cormack is a professor in the David R. Cheriton School of Computer Science, University of Waterloo. Cormack has coached Waterloo's International Collegiate Programming Contest team, qualifying ten consecutive years for the ICPC World Championship, placing eight times in the top five, and winning once. He is a member of the Canadian Computing Competition problem selection committee. He is currently an elected member of the IOI Scientific Committee. Cormack's research interests include information storage and retrieval, and programming language design and implementation.

G. Kemkes has participated in computing contests as a contestant, coach, and organizer. After winning a bronze medal at the IOI and two gold medals at the ACM ICPC, he later led and coached the Canadian IOI team. He has served on the program committee for Canada's national informatics olympiad, the Canadian Computing Competition. Kemkes is currently writing his PhD thesis on random graphs in the Department of Combinatorics & Optimization, University of Waterloo.

I. Munro is professor of computer science and Canada Research Chair in algorithm design, at the University of Waterloo. His research has concentrated on the efficiency of algorithms and data structures. He has served the International Scientific Committee of the IOI as well as on the editorial boards of CACM, Inf & Comp, and B.I.T., and the program committees of most of the major conferences in his area. He was elected fellow of the Royal Society of Canada in 2003.

T. Vasiga is a lecturer in the David R. Cheriton School of Computer Science at the University of Waterloo. He is also the director of the Canadian Computing Competition, which is a competition for secondary students across Canada, and has been the delegation leader for the Canadian Team at the International Olympiad in Informatics.