Olympiads in Informatics, 2014, Vol. 8, 157–168

More Algorithms without Programming

Jakub RADOSZEWSKI
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
e-mail: jrad@mimuw.edu.pl

Abstract. It has been 4 years since the publication of "Algorithms without Programming" (*Olympiads in Informatics*, 4, 2010). In the past four years the set of algorithmic riddles proposed in that paper has been used on different occasions, including an olympic quiz and classes with high school and gymnasium students. It turned out that some of the problems fit better than the others, moreover the set has been extended with several more examples. We present here what we have learned in this period about teaching algorithmic and mathematical thinking without a computer.

Keywords: programming contests, non-programming tasks.

1 Introduction

In the first paper on algorithms without programming (Kubica and Radoszewski, 2010) we have presented several examples of pen-and-paper tasks, formulated using basic notions of combinatorics, that encapsulate a number of crucial concepts of algorithm design. The tasks could be solved using trial and error, however, such methods are quite painful to execute "by hand" (and such solutions are error-prone). This task set was designed for individual study for students primarily interested in mathematics and was originally published in a Polish popular monthly.

Since 2010 the task set has been used on several different occasions. First it formed the basis of an olympic quiz at an open event called the First Polish Informatics Camp held for high-school and gymnasium students during ACM International Collegiate Programming Contest Finals 2012 in Warsaw, Poland. The tasks were distributed on small pieces of paper and the solutions were checked automatically using a computer program. For this purpose the tasks had to be extended with several subtasks, so that the students would not exchange their answers too easily.

Next the task set was introduced to basic and intermediate level olympic programming camps, organized mainly for mathematically talented high-school and gymnasium students at the University of Warsaw, Poland, and to a series of popular introductory lectures to computer science. Problem solving sessions were a form of break from regular programming sessions taking place in computer rooms. They also turned out to be a networking and, at the same time, a competitive event for students. However, their main purpose was to show that computer science is not only about tackling technical is-

sues and to enhance students' problem-solving skills. The task set required a significant change to fit well for this application. Now we selected tasks in which the computations were kept at a strict minimum. The solution to each task required basically a single idea and at the same time could be obtained by students within just a few minutes.

Below we list other works, devoted mainly to specific competitions which contain related examples of non-programming tasks used in informatics training. It is worth noticing that, contrary to pen-and-paper competitions, for the classroom usage there is no need to keep inventing new tasks once every period of time. The ability to solve such tasks is just a bridge to solving regular programming tasks focused on algorithms, not an end in itself. In particular, whenever possible, we add a link to a programming task that is the base of the particular non-programming task, so that the students see a direct connection between the two types of tasks.

The "Beaver" contest consists of short tasks (each to be solved within 3 minutes) on informatics and computer literacy. The key features of the tasks: "attraction, invention, tricks, surprise", "thinking, not guessing answers", and "independence from any curriculum", see Dagienė (2006), Dagienė and Futschek (2008), are the same as in the task set that we propose. The apparent difference is that the tasks at the Beaver contest are to be solved on a computer, whereas we generally aim at pen-and-paper competition. We also aim only at a subset of the list of topics from the Beaver competition which generally fits within the following categories mentioned by Dagienė and Futschek (2008): *Structures, patterns and arrangements* (combinatorics and discrete structures) and *Puzzles* (logical puzzles).

There are several national informatics competitions and olympiads which consist of pen-and-paper tasks of similar flavour in at least one stage. This includes the South African Computer Olympiad, SACO (Merry *et al.*, 2008), Australian Informatics Competition, AIC (Burton, 2010) and Dutch Olympiad in Informatics (van der Vegt, 2012). The works of Burton (2010) and van der Vegt (2012) contain several examples of such tasks. The former provides key characteristics which apply also to our task set, namely "puzzle-based setting" and "no assumed knowledge". Again, the scope of AIC is actually wider than what we consider; our tasks fit in the *Algorithmic tasks* category mentioned by Burton (2010). The AIC contains also three-stage tasks, an idea similar to the idea of subtasks that we consider (with a slightly different motivation).

Tasks of an algorithmic flavour with purely mathematical statements can also be found in Ugāle team competition (Opmanis, 2009) and Project Euler (`projecteuler.net`). There is a substantial difference between the format of the two and the format of our task set: the former can be solved by the students with the aid of a computer. A more comprehensive description of tasks types which go beyond simple programming tasks can be found in Hakulinen (2011) and Forišek (2013). Yet another approach to introduction of elements of computer science in an attractive form can be found in the books of Bell *et al.* (1998), Vöcking *et al.* (2011), Forišek and Steinová (2013).

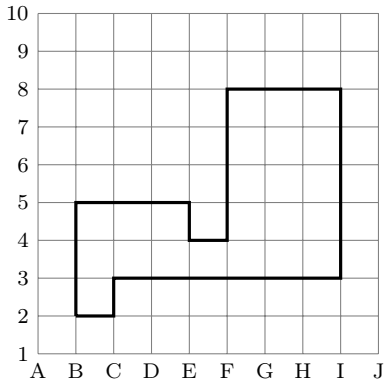
We present our task set with each section devoted to one task. Each task contains 5 subtasks that are normally distributed among students. The task descriptions are followed by a solution description and some methodological comments on the usual performance of students on the particular task. Two of the tasks are tasks from Kubica and

Radoszewski (2010) properly adapted to the new setting. Afterwards in a Conclusions section we list an updated list of conditions from Kubica and Radoszewski (2010) that a task from the set should satisfy.

2. Polygon

2.1. *Problem*

I own a parcel of a polygonal shape. It has 10 sides and its area equals 23 (that is, it contains 23 unit squares). The corners of the parcel are: B2, B5, E5, E4, F4, F8, I8, I3, C3, C2, see figure.



Can you draw a polygon with:

(a) 6 sides and area 6?
(b) 8 sides and area 8?
(c) 10 sides and area 10?
(d) 12 sides and area 12?
(e) 14 sides and area 14?

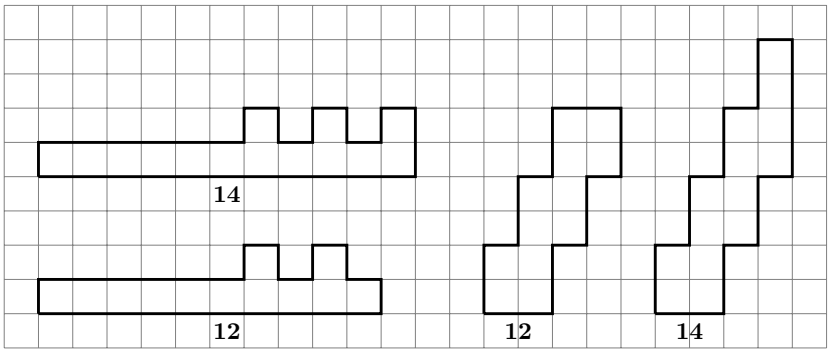
Or a polygon with 13 sides and area 13? All sides of the polygon must be contained in grid lines. Each two consecutive sides must be perpendicular.

This riddle is based on a task from Algorithmic Engagements 2011 (`http://main.edu.pl/en/archive/pa/2011/geo`).

2.2. *Solution*

For all subtasks the answer is positive and there are numerous different solutions.

Below we draw two different parcels for subtasks (d) 12 and (e) 14:



There is no polygon with 13 sides and area 13. Actually there is no polygon with an odd number of sides in which every two consecutive sides are perpendicular. This is because all even-numbered sides must be vertical and all odd-numbered sides must be horizontal (or *vice versa*).

2.3. *Methodological Comments*

At first the students usually solve the subtasks using trial and error. After all the subtasks have been solved, the students can be asked to solve a sixth subtask, with 20 sides and area 20, and then encouraged to provide a general structure of a polygon for any given even number of sides and unit squares. Construction of such scheme requires elements of algorithmic thinking.

On the other hand, the students usually put some effort in trying to draw a polygon with an odd number of sides and area. They are curious why such a polygon does not exist. Sometimes they attempt their own intuitive arguments. The main difficulty in providing such an argument is to note that the odd number of sides is the sole reason why no such polygon exists.

3. Palindromes

3.1. *Problem*

A palindrome is a word which is the same when read from left to right and from right to left. Examples of palindromes are `noon`, `radar`. Some words may be divided into **even** length palindromes (therefore, for example, the palindrome `noon` could be used in a division, while `radar` could not). For example, the word `aabbaaabbaaa` can be divided into 3 even length palindromes:

`aabbaaabbaaa` = `aabbaa|abba|aa`

What is the smallest number of even length palindromes in a division of the word:

(a) aabbaabaabaabbbb?
(b) aaaaabbaaabbabbabbbb?
(c) baabbbbaabaabaabbbbb?
(d) aabbaabaabaabbbbbbbb?
(e) aaaabbaaabbabbaabbbb?

This riddle is based on a task from 2nd Polish Olympiad in Informatics (http://main.edu.pl/en/archive/oi/2/pal).

3.2. *Solution*

Below are the optimal divisions of the words from all subtasks:

(a) aabbaabaabaabbbb = aa | bbaabaabaabb | bb
(b) aaaaabbaaabbabbabbbb = aa | aaabbaaa | bbabbabb | bb
(c) baabbbbaabaabaabbbbb = baab | bbbaabaabaabbb | bb
(d) aabbaabaabaabbbbbbbb = aa | bbaabaabaabb | bbbbbb
(e) aaaabbaaabbabbaabbbb = aa | aabbaa | abba | bbaabb | bb

There is something special about the words selected for these subtasks. They share the following property: if we try to select to the division the longest even palindrome which is a prefix of the word, it cannot be extended to an optimal solution. The same applies if we take the longest such suffix to the division.

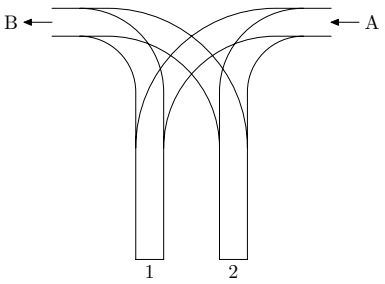
3.3. *Methodological Comments*

This task is to be solved using trial and error. Most commonly the students come up with suboptimal solutions at first. What they learn from this task is that greedy is not necessarily optimal. A correct algorithmic approach to this problem is via dynamic programming (which the students do not need to use here). Interestingly enough, if we were to divide the initial word into the *maximum* number of even length palindromes, a greedy approach would work!

4. Wagons

4.1. *Problem*

Let us consider a track siding: an incoming track A, an exit track B and two auxiliary tracks 1, 2. Track A contains 7 wagons numbered 1 through 7. The wagons arrive on track A in the following order:



(a) 6, 3, 2, 5, 1, 7, 4
(b) 5, 2, 4, 1, 6, 3, 7
(c) 6, 2, 5, 1, 3, 7, 4
(d) 7, 5, 2, 4, 1, 6, 3
(e) 6, 1, 3, 5, 2, 7, 4

The wagons are to exit via track B in increasing order of numbers 1, . . . , 7. Each wagon is to be moved from track A to one of the tracks 1, 2 and then to track B exactly once. There can be arbitrarily many wagons on each track at the same time.

Can this task be completed successfully? If so, to which auxiliary track should the subsequent wagons be moved?

(For example, if the initial order of wagons was 5, 2, 6, 4, 1, 3, 7 then the answer would be positive. The wagons could be moved using the auxiliary tracks 1, 1, 2, 2, 1, 1, 1 respectively.)

This riddle is based on a task from 17th Polish Olympiad in Informatics (`http://main.edu.pl/en/archive/oi/17/kol`).

4.2. *Solution*

The subtasks are constructed in such way that a solution exists (however, there exist permutations of wagons 1 , … , 7 which cannot be sorted using two auxiliary tracks).

Note that at all moments of time all wagons on each auxiliary track must be sorted from the smallest to the highest number (from the beginning of the track). There is a natural greedy approach to this problem: under the aforementioned order-condition, always put the next wagon on the auxiliary track where the first wagon has the smallest number. However, for all given subtasks this strategy *fails*. For example, in subtask (a) after processing the first 5 wagons we would obtain wagons 1, 2, 3, 6 on one auxiliary track and wagon 5 on the other auxiliary track. Next we move wagons 1, 2, 3 to track B in this order and there is no auxiliary track to move wagon 7 to.

Using trial and error, for example, the following solutions can be obtained:

(a) 1, 2, 2, 1, 1, 2, 1
(b) 1, 2, 1, 1, 2, 1, 1

(c) 1, 2, 1, 1, 1, 2, 1
(d) 1, 1, 2, 1, 1, 2, 1
(e) 1, 1, 2, 1, 1, 2, 1

4.3. *Methodological Comments*

The purpose of this task is similar to the task *Palindromes*. The students are not supposed to come up with any particular algorithm but to see that greedy does not work. The original task from the Polish Olympiad in Informatics was solved using a smart reduction to the problem of two-colouring of a particular graph (which seems too complex to be presented at this level).

5. Coins

5.1. *Problem*

Assume you were given coins with the values being powers of two:

$$1, 2, 4, 8, 16, 32, \ldots$$

and you had exactly one coin with each value. Then you could pay any (positive integer) amount of money using these coins (for example, $45 = 1 + 4 + 8 + 32$).

What is the smallest (positive integer) amount of money, which **cannot** be paid using the following set of coins:

(a) 6, 3, 2, 10, 21, 46, 1, 48?
(b) 12, 7, 3, 2, 31, 27, 28, 1?
(c) 27, 56, 1, 13, 60, 4, 7, 2?
(d) 44, 39, 5, 1, 9, 1, 18, 2?
(e) 62, 3, 26, 12, 53, 2, 1, 7?

Keep in mind that you have exactly one coin of each kind!

5.2. *Solution*

The same task in a traditional setting has already been presented in Kubica and Radoszewski (2010). We briefly recall the solution here.

The first step is to sort the sequence of coins in non-decreasing order; for the sequence in the first subtask we have:

(a) 1, 2, 3, 6, 10, 21, 46, 48

Using the first two coins we can pay any integer amount from [1, 3]. With the first three, we can pay any amount from [1, 6]. By including the coin 6, we can pay any amount from [1, 12], with the additional coin 10 we can pay any amount from [1, 22], and with the additional coin 21 we can pay any amount from [1, 43]. The next coin is 46, which concludes that the amount 44 cannot be paid. The answers to the remaining subtasks are as follows: (b) 26, (c) 55, (d) 37, (e) 52.

5.3. *Methodological Comments*

Students usually find this task quite hard. This is because trial and error requires quite a lot of effort to obtain the solution and often leads to mistakes in the answer. Eventually students manage to solve the majority of the subtasks.

This task encapsulates several algorithmic concepts: sorting, elements of dynamic programming approach and binary number system (in the task statement).

6. Triangle

6.1. *Problem*

We are given 11 line segments of the following lengths:

(a) 1, 49, 11, 3, 338, 128, 30, 78, 17, 208, 6
(b) 103, 1, 15, 8, 167, 271, 5, 3, 64, 25, 38
(c) 94, 154, 5, 8, 248, 35, 2, 1, 58, 23, 13
(d) 87, 3, 20, 12, 141, 4, 228, 52, 1, 33, 8
(e) 108, 25, 178, 15, 3, 42, 9, 68, 1, 4, 286

Is it possible to pick three different line segments from the set and use them to obtain a triangle with positive area? If so, which three segments to choose?

This riddle is based on a task from 2nd Polish Olympiad in Informatics (`http://main.edu.pl/en/archive/oi/2/tro`).

6.2. *Solution*

Recall that a triangle with positive area can be built using segments of length a, b, c if the longest one (say c) is strictly shorter than the total length of the smaller two (that is, $a + b > c$).

The answers to all subtasks are positive. In each case there exist unique three segments that can be used to form a triangle:

(a) 30, 49, 78

(b) 15, 25, 38

(c) 13, 23, 35

(d) 20, 33, 52

(e) 42, 68, 108

It is easy to see that if a solution exists then there is also a solution formed by three consecutive segments in ascending order of lengths. For example, in the first subtask we have the following order:

(a) 1, 3, 6, 11, 17, 30, 49, 78, . . .

It suffices to stop at 78, since we have already discovered the requested triad of segments: 30 + 49 > 78.

6.3. *Methodological Comments*

I usually present this task just after the task *Coins*. Thanks to that several students discover fast that having the line segments in ascending order of lengths simplifies the solution considerably.

At the conclusion of the task one can try to argue why exactly it suffices to choose three consecutive segments in the sorted order. There is a simple greedy argument: we start with any three segments in the sorted order and show that by increasing the lengths of the smaller two (without exceeding the length of the longest one) we only raise the odds of obtaining a solution.

Young students usually do not have the habit of proving the correctness of their ideas. In this task set we try to create this habit threefold. First, by showing simple formal proof ideas like the one above. Second, by stimulating the students' curiosity ("Why does it work?"). And third, by showing that the most straightforward intuition (usually the greedy one) need not always work (as in the tasks *Palindromes* and *Wagons*).

7. Anti-binary Sets

7.1. *Problem*

An anti-binary set is a set of integers that does not contain two numbers of the form m and $2m$. For example, the set:

$$A = \{2, 3, 5, 8, 11, 13\}$$

is anti-binary whereas the set:

$$B = \{2, 3, 5, 6, 8, 11, 13\}$$

is not, since both 3 and 6 are its elements.

What is the largest size of an anti-binary set that is a subset of

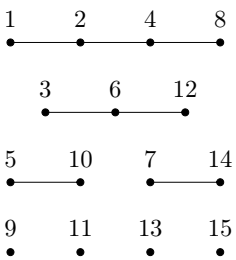
(a) $\{1, \ldots, 11\}$?
(b) $\{1, \ldots, 12\}$?
(c) $\{1, \ldots, 13\}$?
(d) $\{1, \ldots, 14\}$?
(e) $\{1, \ldots, 15\}$?

How many different anti-binary subsets of the same size exist?

This riddle is a simplified version of a task from Algorithmic Engagements 2007 (`http://main.edu.pl/en/archive/pa/2007/pod`).

7.2. *Solution*

The same task in a traditional setting has been presented in Kubica and Radoszewski (2010). The most reliable way to obtain the solution is to draw a graph with vertices numbered 1 through n ($n = 11, \ldots, 15$ depending on the subtask) and edges connecting every two nodes with numbers m and $2m$. Note that such graph is always a collection of disjoint paths. The graph for $n = 15$ is presented below.



Now the task boils down to finding the largest independent set in this graph and counting all largest independent sets, which are both extremely simple. We obtain the following answers to subtasks:

(a) 12 anti-binary sets of size 7
(b) 6 anti-binary sets of size 8
(c) 6 anti-binary sets of size 9
(d) 12 anti-binary sets of size 9
(e) 12 anti-binary sets of size 10

7.3. *Methodological Comments*

This task introduces basic notions of graph theory. A good idea is to collect the solutions of both parts of the task separately. Students usually manage to solve the first part of the task (that is, to find the largest anti-binary set for a given n) quickly without any graphical illustration. However, they will need to perform the graph construction (either explicitly or implicitly) to count the number of largest anti-binary subsets.

The original task from Kubica and Radoszewski (2010) involved counting *all* antibinary subsets of a set $\{1, \ldots, n\}$. While this question is nice to be solved at home with a computer (or calculator) at hand, it involves too much computation to be solved in a class. Let us note that counting *all largest* antibinary subsets solely enforces the students to use some smarter method than trial and error.

8. Conclusions

We presented several algorithmic non-programming tasks to be solved during an exercise session with a group of high-school or pre-high-school students who are just starting to program (or have never programmed before). A number of desired features of an algorithmic non-programming task were already listed in Kubica and Radoszewski (2010). These conditions also apply for the tasks presented in this paper: we aim at small instances of regular programming tasks which admit a technically simple solution that avoids using advanced techniques or classical algorithms and which do not admit a trivial suboptimal or heuristic solution that behaves better on the particular data. However, due to the specific environment considered in this paper, more restrictive guidelines apply.

Obviously we need to produce several interesting instances (subtasks) for each task. This can often be done automatically using implementations of the solutions. Task statement should be based on simple concepts, possibly include a figure to increase its attractiveness, and have a short formulation. A student should be able to come up with a solution within 5 minutes. The solution should consist of a single idea and minimum computations (errors in computations aren't fun). Note that some computations are equally easy for a computer but their complexity differs dramatically in the pen-and-paper scenario, e.g. adding 10 numbers from $\{1, \ldots, 1000\}$ vs finding their maximum. Last but not least, the solution should include some of the crucial concepts of computer science and algorithms in particular, while the task statement should avoid using any concepts of programming.

We have also introduced a new type of tasks on which the most intuitive though incorrect solution fails. In such tasks we do not expect the students to invent the model solution of the corresponding programming task. The purpose of such tasks is to encourage students to perform verification of correctness of their solutions.

References

Bell, T., Fellows, M.R., Witten, I. (1998). *Computer Science Unplugged ... Off-Line Activities and Games for all Ages*. http://csunplugged.org/

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

Dagienė, V. (2006). Information technology contests – introduction to computer science in an attractive way. *Informatics in Education*, 5(1), 37–46.

Dagienė, V., Futschek, G. (2008). Bebras international contest on informatics and computer literacy: criteria for good tasks. In: *3rd International Conference on Informatics in Secondary Schools – Evolution and Perspectives, ISSEP 2008*. 19–30.

Forišek, M. (2013). Pushing the boundary of programming contests. *Olympiads in Informatics*, 7, 23–35.

Forišek, M., Steinová, M. (2013). *Explaining Algorithms Using Metaphors*. Springer.

Hakulinen, L. (2011). Survey on informatics competitions: developing tasks. *Olympiads in Informatics*, 5, 12–25.

Kubica, M., Radoszewski, J. (2010). Algorithms without programming. *Olympiads in Informatics*, 4, 52–66.

Merry, B., Gallotta, M., Hultquist, C. (2008). Challenges in running a computer olympiad in South Africa. *Olympiads in Informatics*, 2, 105–114.

Opmanis, M. (2009). Team competition in mathematics and informatics "Ugāle" – finding new task types. *Olympiads in Informatics*, 3, 80–100.

van der Vegt, W. (2012). Theoretical tasks on algorithms; two small examples. *Olympiads in Informatics*, 6, 212–217.

Vöcking, B., Alt, H., Dietzfelbinger, M., Reischuk, R., Scheideler, C., Vollmer, H., Wagner, D. (Eds.). (2011). *Algorithms Unplugged*.

J. Radoszewski (1984), assistant professor at Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, chair of the jury of Polish Olympiad in Informatics, co-chair of the Scientific Committee of CEOI'2011 in Gdynia, former member of Host Scientific Committees of IOI'2005 in Nowy Sącz, CEOI'2004 in Rzeszów, and BOI'2008 in Gdynia, Poland. His research interests focus on text algorithms and combinatorics.