

CZU: 372.8004

DOI: 10.36120/2587-3636.v18i4.73-85

DIDACTIC ASPECTS REGARDING TO CREATING TEST SETS FOR COMPETITION PROBLEMS

Angela GLOBA*, associate professor, PhD

Tiraspol State University (located in Chisinau)

Sergiu CORLAT** , university lecturer

Technical University of Moldova

*Deputy leader of the national team of Moldova at BOI, EJOI, JBOI, IATI (since 2015); member of the Scientific Council at the Balkan Computer Science Olympiad (2017); member of the Republican Olympic Council of Informatics.

**Team leader of the national team of Moldova at IOI (2014-2019) and other international competitions; coordinator of the national batch of Moldova in computer science (2014-2019); member of the Scientific Council at the Balkan Computer Science Olympiad (2017).

Abstract. A correct evaluation and a qualitative separation of the participants is a major objective for any programming contest. The article examines this aspect from the perspective of the correct elaboration of the tests for the competition problems. Examples of tests analyzed in the article show the added value brought in the elaboration and solving of the competition problems. The article is focused on the elucidation of the methodological aspects regarding the elaboration of tests for the competition problems in the categories of Combinatorics and computational Geometry.

Keywords: competition problem, test set, evaluation, programming contest, problem solving, algorithm, algorithm efficiency, computational geometry, Combinatorics, programming techniques.

ASPECTE DIDACTICE REFERITOR LA CREAREA SETURILOR DE TESTE PENTRU PROBLEME DE CONCURS

Rezumat. O evaluare corectă și o separare calitativă a participanților este un obiectiv major pentru orice concurs de programare. Articolul examinează acest aspect din perspectiva elaborării corecte a testelor pentru problemele de concurs. Exemplele de teste analizate în articol arată valoarea adăugată adusă în elaborarea și rezolvarea problemelor competitive. Articolul este axat pe elucidarea aspectelor metodologice privind elaborarea de teste pentru problemele de concurs în domeniile Combinatorică și geometrie computațională.

Cuvinte cheie: problemă de concurs, set de teste, evaluare, concurs de programare, rezolvare de probleme, algoritm, eficiență algoritm, geometrie computațională, combinatorică, tehnici de programare.

1. Introduction

At the current stage, the Informatics Olympiads and other Programming Contests are quite popular and diverse. They have a long history and a long life even after their development. The competition problems, however, represent the image of any contest, and the success of the competition depends on how successfully (correctly) they will be developed. Moreover, after the contest, the problems gain a new life: they spread rapidly among the future participants in the programming contests, are used in the subsequent preparations for competitions by teachers and mentors, and on their basis new problems are created for the Olympiads [1]. The elaboration of competition problems for any stage

(institutional, local, National, International) is a rather difficult task for authors, which is why it requires complex methodical approaches, which take into account: the theoretical and practical training of the participants; psychic aspect and stress situations; the complexity of the competition, etc.

A competition problem includes a minimum set of resources (fig. 1) [2].

The methodology for evaluating a competition problem is largely determined by its type and the form established for presenting its solution (the output data), and the test sets of the problem have a substantial importance.

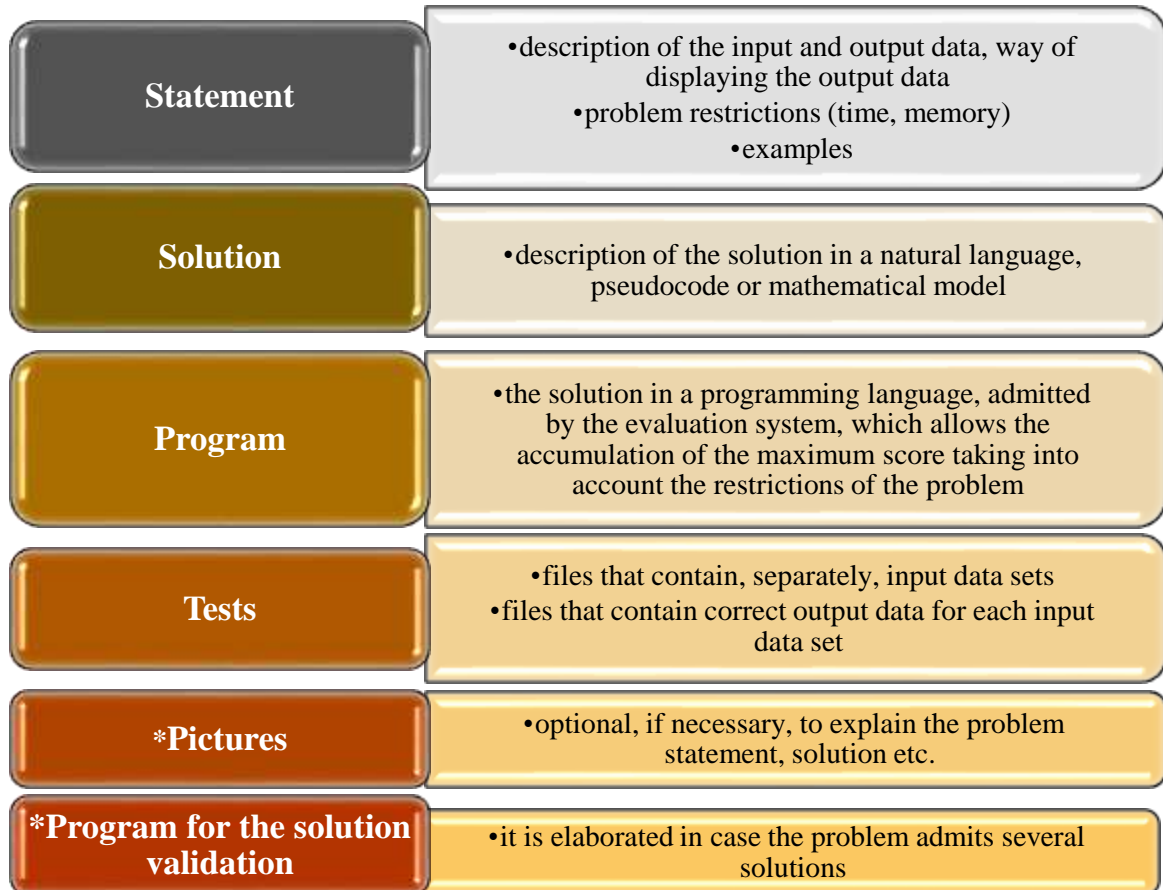


Figure 1. The minimum set of resources for a competition problem (* - optional)

2. Types of tests for a competition problem

In the process of developing the evaluation methodology, initially, it is necessary to establish the maximum score for a complete solution of the problem, then this score is distributed between the partial solutions of the problem or between the solutions for the subtasks highlighted in the statement of the problem. To determine the maximum number of points for a problem, usually two approaches are used. The first approach is based on the preliminary assessment of the complexity/difficulty of the problems selected for the competition. Thus, problems are evaluated with different scores, depending on the degree of difficulty. The second approach is that, each problem is evaluated in the same way, for example with 100 points, regardless of the complexity (degree of difficulty) estimated. The

second approach is used more often in all competitions: municipal, republican as well as in the international Informatics Olympiads and contests. This approach has as a solid foundation the following reasoning. It is difficult to answer the question *Which problem is the most complicated for the participants?* except when the competitors' level of training is known from the beginning. As the results of the participants in the contest are determined by several factors which are difficult to consider at the beginning of the competition, the introduction of the complexity coefficients of the problem loses its meaning. There are cases when, according to the jury, a simple problem can be quite complicated for all the participants and vice versa. There are cases when the first approach is applied, i.e. the problems are evaluated depending on the degree of difficulty, and a large part of the participants, uncertain of themselves, begin to solve problems evaluated with a minimum number of points, while another part of the participants, more confident in their own forces, do the opposite. As a result, both the first and the other spend a lot of time solving the first chosen problem and fail to solve other problems, not because they are complicated, but because they did not have enough time. There are situations when strong participants fail to solve the simplest problem, but such cases already relate to the psychological aspect of the participants, which has a role no less important than the level of preparation of the competitors for the competition.

The distribution of the maximum number of points (for example, 100) for a problem between its partial solutions is generally based on the set of tests developed. If the solution of the proposed problem is obtained after running the program proposed by the participants, then the test set is elaborated in such a way as to allow the jury to correctly evaluate different algorithms proposed by the participants in order to obtain the maximum score according to their correctness and effectiveness.

The complete test set for a competition problem should include the following test groups:

- 1) tests of minimum size (trivial tests);
- 2) tests for particular cases, which allow to identify the specific characteristics of the implemented algorithms;
- 3) tests for the accuracy of calculations with real numbers, if the input data cause numerical instability of the algorithms;
- 4) tests that highlight the specific characteristics of the programming language or environment (for example, the difficult accomplishment of the operations with large numbers in the Pascal language, the inefficient implementation of the input-output flows and the sequential containers in C ++);
- 5) random tests of different dimensions (here we refer to the dimensions of the input data: from simple tests to complicated tests);
- 6) tests that check the presence of heuristic in algorithms (which does not analyze all the available information, a very fast procedure for solving a problem);

7) maximum size tests (allow to evaluate the effectiveness of the proposed algorithms at maximum dimensions of the input data specified in the problem restrictions).

The distribution of the maximum number of points for a problem is made initially between all the test groups, and then the score for each group is distributed between the tests in each group. This distribution is made in a table where each test and group of tests is assigned a certain number of points. For example, for the *Rectangles* problem (IOI 2019) it is proposed table 1.

Table 1. Distribution of the points number for each subtask (Rectangles problem, IOI 2019)

Subtasks	Points	Restrictions
1 11 tests	8	$n, m \leq 30$
2 8 tests	7	$n, m \leq 80$
3 8 tests	12	$n, m \leq 200$
4 16 tests	22	$n, m \leq 700$
5 11 tests	10	$n \leq 3$
6 8 tests	13	$0 \leq a[i, j] \leq 1$ (for all $0 \leq i \leq n - 1, 0 \leq j \leq m - 1$)
7 18 tests	28	No additional constraints
Total	100 points	

When distributing the score for a competition problem between all test groups, the following principle will be taken into account: the correct solution for all restrictions in the problem statement will accumulate the maximum score, while the correct solution for some subtasks, but inefficient for all restrictions, should accumulate between 30% and 70% of the maximum score offered for the given problem.

Since each test in the group is used to test the properties of the proposed algorithm for solving the formulated problem, the score within the group is distributed taking into account the importance of these properties for solving the problem as a whole. If the contestant gets correct answers to the tests from a group or to certain tests within that group, he is credited with the number of points set for this group or for the tests covered, otherwise points are not awarded.

Analyzing the information in table 1 it can be concluded that for subtasks 1, 2, 5 the accumulation of a point is possible only in case of passing/covering at least two tests from the group. For subtasks 3, 4, 6, 7 the distribution of the score per test is more complicated, but not less than one point per test.

If the competition problem is divided into several separate subtasks, then the evaluation of the proposed solution for a subtask can be carried out for the whole test group, i.e. the maximum score is offered only if the entire test group has been passed successfully,

otherwise no points are awarded. The evaluation can also be done for each separate test, but in this case, a prior distribution of the score is required for each test from the group. The first approach is simpler in terms of implementation, but more "painful" for the contestants. The second approach is more "friendly" with the participants in the competition, but requires a careful analysis of each separate test with the identification of the test subsets for accumulating a point or the test subsets for accumulating the maximum score per group, etc.

The total score, obtained by the contestant, for solving a problem represents the sum of the score accumulated in the tests from all the test groups elaborated for the proposed problem. Thus, the final score accumulated by the contestant in all the problems proposed within the contest is formed as the sum of the total scores received by him for each problem.

3. Methodological benchmarks regarding the creation of the test set for a competition problem

The problems proposed for solving to the participants in the Informatics competitions fall into the following categories: combinatorial problems; problems with computational graphics; problems focused on data structures; simulation problems etc.

2.1. The test set for combinatorial problems

To solve combinatorial problems (processing the elements of a set, generating the set/subset, searching for a subset or element of the set with certain properties, etc.) the contestants must implement a programming technique: the most elementary - Brute force, Backtracking, Greedy, Divide and Impera and, the most difficult - Dynamic Programming [3].

Obviously, when generating tests for such problems, the author will take into account the programming technique adopted. For example, the **Books** problem (Republican Informatics Olympiad, 2015), which, in fact, is a classic problem:

Description: The great tycoon BitMan has in one of its stores n boxes, numbered from 1 to n , full of books. There are no boxes with the same number of books. The i box contains x_i books ($x_i \neq x_j$, $1 \leq i, j \leq n$). BitMan expects to go to the orphanages and distribute books equally, taking only k boxes.

Task. Make up a program that would determine the maximum number of books that BitMan could donate to each orphanage.

Input. The standard input contains, on the first line, a natural number n - the number of boxes in the stores. The second line contains the natural numbers x_1, x_2, \dots, x_n , - the number of books from boxes 1, 2, ..., n , separated by a space.

Output. The standard output will contain a natural number - the amount of books each orphanage will receive.

Constraints: $1 \leq n \leq 255$; $1 \leq x_1, x_2, \dots, x_n \leq 1000$. The execution time will not exceed 1 second. The program will use up to 32 megabytes of operational memory.

Examples.

book.in	book.out	Explanation
5 12 9 14 17 11	8	BitMan can take boxes 1, 4 and 5 or boxes 2, 3 and 4, each orphanage receiving 8 books. Any other set of boxes will not allow BitMan to share books equally, distributing more than 8 books to each orphanage.

The task of the problem does not include the display of the boxes number that BitMan has to take, as it admits several solutions (for example, (1,4,5), (2,3,4)), but the maximum number of books that each orphanage will receive is unique. Obviously, all the solutions of the problem can be displayed, but for this it will be necessary to develop a validation program of the problem solution.

The solution of the problem is reduced to the processing of the elements of an A set of n elements, more precisely, the identification of a subset of the A set for which the sum of the elements of the respective subset is divided by n . The formulated problem always admits a solution.

Tests production. The tests set was produced according to programming technique adopted.

Programming technique	Description	Points
Brute force	It is known that the number of subsets of a set with n elements equals to 2^n . The brute force is reduced to: generating all the subsets of the given set; keeping subgroups generated in a vector; calculating the sum of the elements of each subset with checking the condition of divisibility of the sum calculated with n ; choosing the maximum amount divisible by n . The temporal complexity of this algorithm is $(O(2^n))$. $1 \leq n \leq 10$; $1 \leq x_1, x_2, \dots, x_n \leq 1000$	10
Backtracking	This technique is faster than Brute force, but it also admits a time complexity equal to $(O(2^n))$. In this case, only those subsets having the sum of the elements divisible by n are generated. In this case, it is necessary to save in a variable, at the generation stage of the i subset, the maximum sum divisible by n . $1 \leq n \leq 20$; $1 \leq x_1, x_2, \dots, x_n \leq 1000$	30
Dynamic Programming	The implementation of the Dynamic Programming technique admits a temporal complexity equal to $(O(n^2))$. $1 \leq n \leq 255$; $1 \leq x_1, x_2, \dots, x_n \leq 1000$	60

It is recommended that the tests, which allow to obtain a score by implementing the Brute force technique, admit a score of not more than 10 points out of 100 (depending on the level of the programming competition: sectoral, republican, international, etc.), with a purpose, more, to stimulate the participants in the contest than to evaluate them.

The use of the Backtracking technique to obtain the solution of the problem speaks of elementary knowledge of the programming techniques, by the participants in the contest, a fact which deserves to be stimulated and appreciated by the jury. It is also recommended

that the tests that support the application of the Backtracking technique for writing the solution cover a score of no more than 20-30 points out of the total score of 100 points.

A solution, which would cover all the tests, is provided by the Dynamic Programming technique - a difficult technique from several points of view. The contestant will have to find that principle of optimality, which would give him the solution of the problem in a useful computation/calculation time, but, he will need knowledge in the field of Mathematics, such as: theorem of division with remainder, Dirichlet's principle etc., and the correctness of the algorithm can be demonstrated using the mathematical induction. Greater attention will be given to the data structures used to store partial solutions.

2.2. The test set for computational Geometry problems

The description of the test creation methodology for computational Geometry problems was made based on an example - the Tale problem (Balkan Olympiad in Informatics, 2017, [4]).

Description: Every year, in the middle of summer, a short-stature king, called Statu Palmă Barbă Cot, gives a feast in his castle at which all the knights from the Balkan countries are invited. Făt Frumos is one of the guests, and, like a true knight, he rides to the feast on his fairy horse, named Flămânzilă. Horses are forbidden inside the castle, therefore Făt Frumos intends to tie up Flămânzilă to one of the trees nearby, outside the castle. Flămânzilă is very quiet as long as it has something to eat (its favourite grass grows everywhere), but, after it grazes all the reachable grass, it becomes nervous and begins to blow fire, like a dragon. At such a point, Făt Frumos has to leave the feast in order to calm his quadruped companion. In order to prevent Flămânzilă from causing a fire, Făt Frumos needs to know the area of the figure within which it can move. The figure is shaped by:

- the tree to which the horse is tied up (a point of integer coordinates X_c, Y_c);
- the length of the rope L (a positive integer);
- the wall of the castle that the horse cannot jump over (the wall forms a convex polygon with N edges).

Task: Write a program to calculate the area of the figure within which Flămânzilă can move.

Input. The standard input contains, on the first line, three integers X, Y , and L , separated by a space - the coordinates of the tree to which the horse will be tied up and the length of the rope. The second line contains the positive integer N - the number of vertexes in the polygon. N lines follow, each of them containing two integers X_i, Y_i ($i = 1, \dots, N$), separated by a space - the vertexes of the polygon, given clockwise.

Output. The standard output will contain a real number with five decimals - the area of the figure within which Flămânzilă can move.

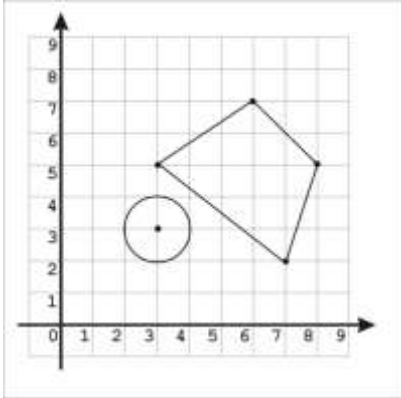
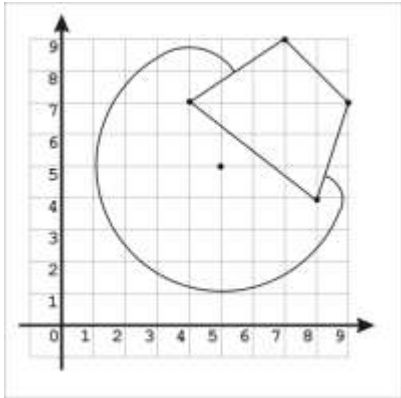
Constraints: $-10000 \leq X, Y \leq 10000$; $3 \leq N \leq 1000$; $1 \leq L \leq 10^9$.

Subtasks:

Subtask	Description	Points
1	The length of the rope does not exceed the distance from the tree to the wall	10
2	The length of the rope does not exceed the half-perimeter of the wall	60
3	The length of the rope allows reach all the points on the polygon edges, making a combined cover in two directions: clockwise and anticlockwise. However, each of the invisible vertices of the polygon (a vertex is invisible if the segment joining it with the tree intersects the polygon in an interior point) can be reached from no more than one direction.	30

Note: Results will be evaluated with a precision = 1.0.

Examples:

tale.in	tale.out	
3 3 1 4 3 5 6 7 8 5 7 2	3.14159	
5 5 4 4 4 7 7 9 9 7 8 4	36.71737	

Tests production. The tests set was produced according to subtasks description [5, 6].

First subtask was the simplest one: because the circle does not intersect the polygon, calculations has to be done using standard formula for circle area. (see example 1) The test production was also relatively simple, but a visual verification tool, programmed by task authors was used to exclude some overlaps between the circle and the polygon.

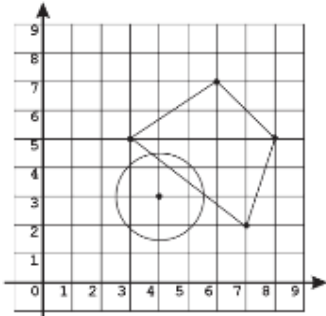
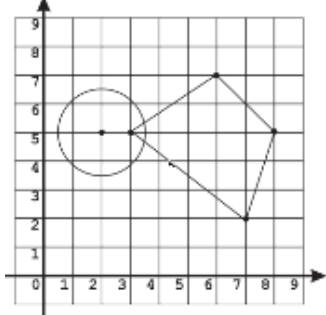
The second subtask is the biggest one. There are more situations to solve inside this subtask. The first case – circle is not transformed, it just intersects a side or couple of sides, no touch extreme viewpoints of the polygon. (see images a - c below) These case tests are easy-solved using simple calculations for circle sector area. Tests for this case were produced in two modes: “by hand”, in developed visual tool (for small test cases: $N < 20$)

and randomly, for medium and large sets of input data. The process of random test production was divided into three steps:

- generate a convex polygon (the problem solved by generating a set of points in R^2 , then finding a convex hull of this set);
- place randomly a “tree” point outside the polygon;
- finding extreme “viewpoints” in the polygon. Extreme “viewpoints” are the polygon vertexes, which, together with “tree” point forms an angle, that contains the whole polygon inside;
- set randomly rope length to be less then distance to nearest “viewpoint” and longer then the distance to polygon.

The second case takes in consideration circle deformation after touching a vertex (extreme viewpoint). The series of deformations continue with each next side, touched by the rope (image d). Some ”pie slices” will be added to the calculated area.

There were two types of generated tests: produced ”by hand” in a visual environment and random generated. The production follows the procedure, like described above. The only difference was that rope length restrictions were longer that the distance from “tree” point to nearest extreme “viewpoint” and less than a half of polygon perimeter.

 <p>a)</p>	<p>Subtask 2, case 1a intersection with one side</p>
 <p>b)</p>	<p>Subtask 2, case 1b intersection with two sides. No “full” sides inside the circle</p>

<p style="text-align: right;">c)</p>	<p>Subtask 2, case 1c intersection with two sides. some “full” sides inside the circle, but extreme viewpoints are not touched.</p>
<p style="text-align: right;">d)</p>	<p>Subtask 2, case 2 intersection with two sides. some “full” sides inside the circle, one or two extreme viewpoints are touched.</p>

Third subtask. The most complicated – rope overlap is possible. The overlap is allowed but will not exceed singular side zone. Like in previous cases small size tests were produced directly in visual environment, and large – using random parts, in a mode similar to previous. An additional step was the singularity check for the side that can be reached from both directions: by left and right polygon crossing. After production all tests were additionally tested by visual checker and solver. (see images below)

<p>Test case visualization result: one common side, test ok.</p>	<p>Test case visualization and solving: one common side, test ok. The calculated area and some intermediary info – upper left corner.</p>

4. The impact of the tests, created by the participants during the contest, on the solution of the problem

From experience, the success at the Informatics Contest is ensured not only by the participants' knowledge, but also by the strategy of approaching the competition problems.

It often happens that a contestant will find a good solution to a problem, which will not ensure the maximum score, giving up the search for the most efficient solution, because the time used is not directly proportional to the accumulated score. This time can be used to solve other problems. In this case, this is a correctly chosen competition strategy.

There is also a substantial impact on the success achieved by the contestant, both internal and external psychological factors: the temperament; the individual experience; the lack of sleep before the competition; the psychological climate established between the team members, including with the deputy and team leader; the psychological climate during the competition; the complexity/level of the competition, etc.

Usually, the first participation in an Informatics Olympiad provides the contestant with a maximum average score, which is due to the specificity of Informatics Olympiad. The experience accumulated by the participants matters a lot, as, historically, it is shown that the true results begin to appear starting with the second participation, sometimes even later.

To solve a competition problem, the contestant must go through several stages (fig. 2) [7].

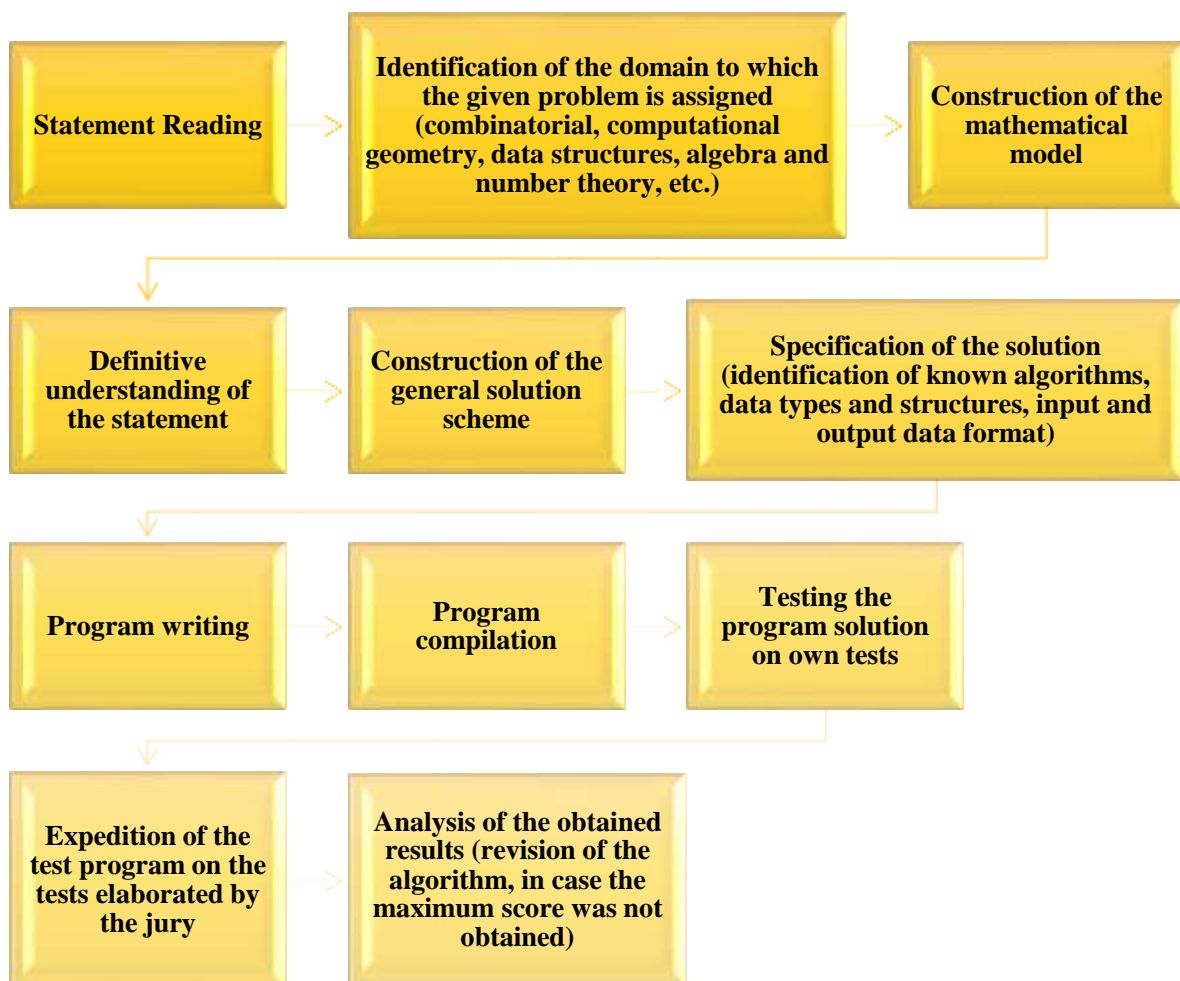


Figure 2. Stages of solving competition problems

If the compilation of the program was successful, then one can go to the test stage of the solution on his own tests, as, the contestant does not have access, that is, he cannot see the tests elaborated by the jury (the input files). It is recommended to go through the following steps:

- ✓ checking the solution on the tests from the example of the problem statement (for example, here errors can be identified regarding incorrect reading of the input data, display of the output data, etc.); the size of the input file can essentially change the difficulty of the problem (this is the computation time, which usually does not exceed a few seconds);
- ✓ checking the solution of the problem on a medium difficulty test, but which can be easily solved manually (if the test has been passed, the contestant can pass to other types of tests, minimum 4 tests and maximum 7-8);
- ✓ checking the solution on a set of small tests (For example, the program processes squares of side $0 \leq a \leq 10000$. Errors may occur for $a = 0$);
- ✓ checking the solution on a set of maximum size tests (If the program processes squares of side $0 \leq a \leq 10000$, *what happens if $a = 10000$?*); here you can create tests of a particular structure, so that it is easy to calculate the manual result (if it is complicated to manually create such a file, you can write a program for generating it);
- ✓ checking the solution on tests containing particular cases of the proposed problem (it may happen that these cases were not fully included in the program developed by the contestant);
- ✓ writing a program for generating random tests (this step allows you to check if the proposed solution does not block when allocating large areas of memory or if it does not exceed the time limit).

When testing the solution, it is better to consider the following recommendations:

1. If for some tests the program fails, it is necessary to run the program again, step by step; evaluate the results (variable values) after each step, thus you can precisely locate the subprogram, then the line that produced the error; run the program again on this test.
2. In case of program troubleshooting step by step, other errors may be highlighted; correct the errors and run the program on all previously tested tests.
3. Keep a copy of the unmodified program to restore the original form of the program, if the new version is not a better one;
4. If the problem statement does not specify the data, for example, the integer type, it is advisable to use the long long int type; increased attention should be paid to the accuracy of real numbers;
5. Pay attention to the calculation of the computation time.

Conclusions

In order to separate participants by their programming skills level the tests set will be generated in different sizes: small tests to check basic problem solution and, may be, some particular situations, easy to be identified; small tests for special cases, depending of problem domain; medium size tests for general solution based on standard techniques and/or data structures; medium size tests for combinations of special cases; large tests to test optimality of the algorithm and used data structures, for arbitrary generated data; extremal size tests for “sharpen” evaluation of best solutions.

The solutions of Computational Geometry problems usually are real numbers. The evaluation of floating-point results can be done in several modes: results are accepted by special checker program which calculates the difference between official solution and contestant’s solution or the task asks for approximation to a given range or even to an integer value.

The correctness of the test sets elaborated for the evaluation of the competition problems solution is essential in the organization and qualitative development of an Informatics contest. The validation of the created tests is quite difficult for several reasons, but necessary to avoid some confusion that may arise during the evaluation phase of the solutions. The test set created for a problem in the Computational Geometry category can be verified using an application such as GeoGebra, Mathematica, Maple etc. Obviously, checking the correctness of the algorithm created is essential.

Bibliography

1. Chiriac L., Globa A., Bobeică N. Procedee metodice privind pregătirea și desfășurarea olimpiadelor de informatică. Mathematics & Information Technologies: Research and Education (MITRE 2011) dedicated to the 65th anniversary of the Moldova State University. Chișinău, August 22-25, 2011. p.168-169.
2. Pregatire pentru concursuri informatice: <https://www.slideshare.net/scorlat/pregatire-pentru-concursuri-informatic>
3. Globa A. Abordări metodice privind implementarea unor tehnici de programare prin prisma complexității algoritmilor. În: Acta et Commentationes. Științe ale Educației. Revistă științifică. 2014, nr.1(4). Chișinău: Universitatea de Stat din Tiraspol, 2014. p.41-49. ISSN 1857-0623.
4. BOI 2017: <http://boi2017.utm.md/wp-content/uploads/2017/01/TaleEng.pdf>
5. Corlat S., Gremalschi A. Metodologia rezolvării problemelor de geometrie computațională. Universitatea de Stat din Tiraspol, 2015.
6. Skiena S., Revilla M. Programming challenges. The Programming contest Training Manual. Springer, 2002.
7. Оршанский С. А. О решении олимпиадных задач по программированию формата ACM ICPC. В: Журнал „Мир ПК”, Nr. 9, 2005.