

The Complete Guide to add or create problems for the Caribbean Online Judge

Yonny Mondelo Hernández
ACM-ICPC Caribbean Chief Judge
Caribbean Judges Committee President
Caribbean Online Judge Development Team
University of Informatics Sciences. Havana, Cuba. 2016, February 20.
ymondelo@uci.cu

I. ABSTRACT

The Caribbean Online Judge (COJ - <http://coj.uci.cu>) provides a space, sponsored by the Caribbean Community of the ACM-ICPC, where users of the world can exchange experiences and knowledge. They can test, improve and share skills on the problem-solving, the computer programming, and the teamwork. They can get training to participate in programming competitions (ACM-ICPC, IOI, and others).

This system which is certainly a vital link for collaborative work and training of the mentioned community is managed by the Productive Center of Technologies for Training of the University of Informatics Sciences. It's also conducted by the Movement of Competitive Programming "Tomás López Jiménez" of the mentioned university and other partners involved to ACM-ICPC, promoting the whole time the effort and the preparation of the contestants in their insatiable quest for knowledge, experience and outcomes in terms of competitive programming.

This complete guide is especially for those who want to create problems to be used in the Caribbean Online Judge, although most of the content in this document is not exclusive for this system because it can be applied to the work as Problem Setters, in general cases. Contrary to what many people think the work as Problem Setters is a hard task, mainly strengthened by the experience and knowledge in mostly of the cases.

This document is a compilation of topics we made seeking to increase quality and standardization of the problems created for the Caribbean Online Judge. It must serve as one step by step guide for Problem Setters working, including tips and most used practices related with the task of creating good problems statements, generating datasets, testing solutions and constructing special validators.

Keywords: Guide, Problem Setter, Competitive Programming, Collaborative Work, Caribbean Online Judge.

I. INTRODUCTION

The Caribbean Online Judge (COJ - <http://coj.uci.cu>) provides a space, sponsored by the Caribbean Community of the ACM-ICPC, where users of the world can exchange experiences and knowledge [1].

A young group of professors and students, most of them original belonging to the University of Informatics Sciences (UCI, by their meaning in Spanish), the Productive Center of Technologies for Training (FORTES, by their meaning in Spanish), and the Movement of Competitive Programming "Tomás López Jiménez" (MPC-TLJ, by their meaning in Spanish), are the vital component in the development and improvement of the current system [2].

The first version of the COJ who comes to Internet was published on June 05 of 2010; in October 2011 the base system was replaced by another that was reprogrammed from scratch (for almost one year) by two young students of the UCI [3][4]. Since then the working teams and methods have been improved to get a better system.

For giving an idea, currently in almost six years of availability on Internet, the Caribbean Online Judge has more than 23600 users (19000 more than in 2012, 9000 more than in 2014, and 2500 more than in 2015). They represents to a bit more than 1000 institutions that belongs to 175 countries. Between 5 and 15 users are newly registered every day. [9][13][14][15]



Fig. 1: Main Page of Caribbean Online Judge.

The Caribbean Online Judge is certainly a vital link for collaborative work and training of Caribbean Community of the ACM-ICPC, promoting the whole time the effort and the preparation of the contestants in their insatiable quest for knowledge, experience and outcomes in terms of competitive programming [2]. But nothing of this would be possible without the effort of Problem Setters in the process for creating problems to be used in the Caribbean Online Judge. Those problems without which, contests could not make properly.

This complete guide is especially for those who want to create problems to be used in the Caribbean Online Judge, although most of the content in this document is not exclusive for this system because it can be applied to the work as Problem Setters, in general cases. Contrary to what many people think the work as Problem Setters is a hard task, mainly strengthened by the experience and knowledge in mostly of the cases.

This document is a compilation of topics we made seeking to increase quality and standardization of the

problems created for the Caribbean Online Judge. It must serve as one step by step guide for Problem Setters working, including tips and most used practices related with the task of creating good problems statements, generating datasets, testing solutions and constructing special validators. The last version of this document could be always found as COJ Complete Guide for Problem Setters in the download section of the Caribbean Online Judge (<http://coj.uci.cu/general/downloads.xhtml>).

The 24 hours problem archive of the Caribbean Online Judge (<http://coj.uci.cu/24h/problems.xhtml>) is a collection which holds several problems; as you could suppose it is available 24 hours per days. Even when navigation is not possible for any reason there is a way to keep solving problems and to continue in contact with the COJ by email [5][6][12]. Those mentioned problems could be solved using the programming languages most used on competitive programming of ACM-ICPC, IOI and others [19]; reason for what is attracted the attention of the user community who grows fast and constantly.

More than 2400 problems conforms the 24 hours problem archive of the COJ; they cover several knowledge areas related with computing programming, although problems also could be closed related with Math, Physics, Chemistry, Biology, Society among others.

Almost all problems in the Caribbean Online Judge are associated with one or more tags. A tag is an element which includes both complexity and knowledge area of some problem. If you can't see the tags in the problems, you should probably go and edit the personal profile and mark the option "**Show tags**". The complexity is an

- Ad-hoc (AH)
- Arithmetic-Algebra (AA)
- Brute Force (BF)
- Combination (CO)
- Data Structures (DS)
- Dynamic Programming (DP)
- Game Theory (GA)

An important fact represents that more than 358 programming contests have been developed in the COJ, equivalent to one contest every six days [16]. Those numbers are only reached by main online judges oriented to competitive programming or holding contest ratings (Codeforces - <http://codeforces.com/>, TopCoder - <http://topcoder.com/>, CodeChef - <http://codechef.com/>, A2 - <http://ahmed-aly.com/>, etc.). Other important online judges like UVA - <http://uva.onlinejudge.org/>, SPOJ -

estimated integer number which go from 1 to 5: Very Easy (1), Easy (2), Medium (3), Hard (4), Very Hard (5). This is a relative value which is supposed to be established by their authors; but sometimes they are not available at all. Obviously, with the time the users could propose changes to adjust real complexity of some problem.

Although in general is really difficult to make a standard grouping/classification of the knowledge areas related with computing programming, the following proposal used by Caribbean Online Judge to group their problems in categories is pretty close to desired goal:

- General Geometry (GE)
- Graph Theory (GT)
- Greedy (GR)
- Set-Number Theory (NT)
- Sorting-Searching (SS)
- Strings (ST)

<http://www.spoj.com/>, ACM-ICPC Live Archive - <https://icpcarchive.ecs.baylor.edu/> and TIMUS - <http://acm.timus.ru/> have hundreds of problems on their archives but they are lesser characterized by making one standard flow of competitions. [18]

The following table has important statistics about the main competitions which are usual contests in the COJ in the whole year:

Table I

STATISTICAL SUMMARY ABOUT USUAL COMPETITIONS IN THE COJ (UNTIL FEBRUARY 20, 2016)

Type of Competition	Amount of Competitions Performed in COJ	Amount of added or newly created Problems for the Competitions in COJ	Amount of created Problems for the Competitions in COJ with Caribbean Authors	Amount of Coming Competitions (until December, 2016)
CTC	52	462	163 (35%)	3
MPC-TLJ	54	399	188 (47%)	5
ACM-ICPC	71	267	162 (60%)	10
INSTITUTIONALS	166	683	516 (76%)	8
PROGRESSIVES	10	216	156 (72%)	1
COJ ROUNDS	5	26	26 (100%)	5
TOTALS	358	2053	1211 (59%)	32

II. TIPS FOR CREATING PROBLEMS FOR COMPETITIONS

If you don't have too much experience for creating problems for competitions, then you should take into account the following general tips for writing a good problem (most of them are officially used by the Latin-American and the Caribbean Judges Committees of the ACM-ICPC independently):

- All problems are welcome no matter their difficulties. Good problem is not necessarily equivalent to hard problem.
- It should make clear which are the valid entries and the limits for all parameters (including the maximum number of cases in any single input dataset).
- There are many tags to problems (see above classifications). And it's okay to touch various topics in problems at the same time.
- Problems with single output for test case are preferable. If one had thought possible multiple output, there are various techniques that can be used to easily do a single output problem (lexicographically less, ask only the minimum / maximum and not a description of how to get to it, etc.).
- Try the possible outputs of your problem as having values (integer, string, etc.). Decision problems (e.g. with real numbers as output), may need extra validators for testing the solutions.
- Unless the idea of the problem is directly related to the input / output (e.g. parsing problems or drawing on screen), both input and output should be as simple as possible to read using the standard mechanisms (`scanf` / `printf`, `cin` / `cout`, `BufferedReader` / `System.out.println`, etc.).
- Avoid using huge datasets, this increases markedly the execution time difference between some programming languages, and it's not good.

Also if you are thinking to use the problems for a competition, then you should take into account the following tips too:

- When assessing a problem, think about how good it is for the contest, not how good it can be for other purposes (for instance, some problems are great for giving examples in class, but sometimes the statement is too tied to the solution or requires a lot of ad-hoc definitions that are reasonable only in the context of a solution, and that is not so great for a contest).
- Problems with great ideas but that present too much technical difficulty for the contestants (for instance, simple idea but 1000 lines of code) are also not great, because they end up testing something other than algorithmic/mathematical thinking (it is ok to test implementation skills, but a contest is not the best way to doing that).
- If a problem you find is repeated from another problem already in a contest or online judge (either being the exact problem or the main idea being the same, basically, if having solved the existing problem gives a decisive advantage to those who did) or it is too close to a well-known (for instance, the statement gives a graph and asks for the minimum path between two nodes. Notice that if the graph is not obvious, or its construction presents some difficulty, then the problem is not a well-known problem, however if the statement uses junctions and streets but these translate directly into nodes and edges, the problem is a well-known problem, even if it does not use the word graph) is recommendable not using that problem.
- Try to achieve a problem set with the widest coverage of knowledge areas, as well as having a fair distribution of complexity levels to promote the participation and good performance of new contestants, without missing the point on letting the problem set to spread as much as possible all teams along the ranking according to their actual skillsets.
- Please don't publish sensible information about the problem statement (previously to the contest). Do not discuss the problem with people directly related with contestants unless you are really sure that this don't implies a risk.

III. HOW TO SEND MY PROBLEM PROPOSAL FOR CARIBBEAN ONLINE JUDGE

If you want to send any problem proposal for the Caribbean Online Judge, then you must read those rules carefully:

- Where possible, all information must be in English. This is related to general information, e.g. problem description, input and output details, and hints. Except for problem specific information, like characters names and stuffs like that, any other information must be written in English. If you want to send versions of the problem in other languages for internationalization (e.g. Spanish or Portuguese), you must only send problem description, input and output details, and hints.
- All tests (filenames exactly as: 001.in, 001.out, 002.in, 002.out ... N.in, N.out) must be inside a unique RAR/ZIP archive (up to 5 MB of size). The archive's name could be "PROBLEMCODE-tests". If you need send more than one file, then use for example: "PROBLEMCODE-tests-1", "PROBLEMCODE-tests-2", and so on. Datasets should be preferably created or generated over UNIX operating and file systems to avoid problems with file codification when uploading datasets to the COJ servers (e.g. those related with end of lines for UNIX, Macs and Windows). Avoid using huge datasets, this is not good.
- All solutions must be inside a unique RAR/ZIP archive (up to 5 MB of size). The archive's name could be "PROBLEMCODE-solutions". If you need send more than one file, then use for example: "PROBLEMCODE-solutions-1", "PROBLEMCODE-solutions-2", and so on. Solutions should be preferably in C, C++ or Java. Please, don't send the projects for solutions, instead of that you should send a single file. Also, we strongly recommend that you validate the ranges of the problem, in each solution you create. You can use functions to validate (like assert and others), or typical codes in order to check ranges and format of the input datasets. Don't use functions to manipulate file system, memory or network; they will fall into security violation verdicts (typically called "invalid function").
- It should make clear which are the valid entries and the limits for all parameters (including the maximum number of cases in any single input dataset). Using terms like "several lines or cases" without specify the maximum amount of them is not recommended. Also you should clearly specify the exact input format (e.g. "three space-separated integer numbers", or "four lines each of them with one integer number") taking especially into account white spaces and blank lines.

- Each problem statement must have the following structure:

- General Info:

- Problem name: e.g. "Just Another Easy Problem"
- Proposed by: e.g. "Yonny Mondelo Hernández"
- Created by: e.g. "Yonny Mondelo Hernández"
- Test Time Limit (MS): e.g. "1000" (you can use the same value for all programming languages; you can specify different values for each programming language; or you can manage those values with dynamic multipliers specified by each programming languages)
- Total Time Limit (MS): e.g. "15000" (you can use the same value for all programming languages; you can specify different values for each programming language; or you can manage those values with dynamic multipliers specified by each programming languages)
- Memory Limit (Bytes): e.g. " $268435456 = 2^{24} = 256\text{MB}$ " (you can use the same value for all programming languages; you can specify different values for each programming language; or you can manage those values with dynamic multipliers specified by each programming languages)
- Source Limit (Bytes): e.g. " $16384 = 2^{14} = 16\text{KB}$ " (you can use the same value for all programming languages; you can specify different values for each programming language; or you can manage those values with dynamic multipliers specified by each programming languages)

- Languages: e.g. "C, C++, Java, and C#", "All except Perl and Bash", or only "All..."
 - Source: e.g. "Unpublished", "2015 ACM-ICPC Latin American Regional Contest", or <http://www.spoj.pl/problems/TEST/> (take into account that some sources have explicit rules about using their problems in another Online Judge or any online competition)
- Problem Description:
 - Problem Specifications
 - Input Specification + Output Specification (in correspondence with the Problem Specifications)
 - Sample Input + Sample Output (in correspondence with the Input and Output Specifications)
 - Hint(s): Some optional clue, or perhaps any additional samples
- Problem categorization, which areas the problem belongs to: e.g. "GE4" or "NT3, AA3" or "DP3, GT2" (problem can have multiple categorizations but please, avoid multiple complexities or unnecessary classifications):
 - Tags: Ad-hoc (AH), Arithmetic-Algebra (AA), Brute Force (BF), Combination (CO), Data Structures (DS), Dynamic Programming (DP), Game Theory (GA), General Geometry (GE), Graph Theory (GT), Greedy (GR), Set-Number Theory (NT), Sorting-Searching (SS), Strings (ST)
 - Difficulty: Very Easy (1), Easy (2), Medium (3), Hard (4), Very Hard (5)
- Send all data to the COJ Mail (coj@uci.cu), meeting with above specifications. Also you can send a copy to mail account provided on this document.

IV. HOW TO ADD MY PROBLEMS IN THE CARIBBEAN ONLINE JUDGE

If you don't have ever added problems in the COJ, then this section should be like one "Step by Step" manual for doing that. Although information of this section could be interesting in general for several reasons, should be noted that normal users of the COJ are not allowed to add problems in the system. For adding problems in the system special privileges are needed; those privileges which need to be granted by admins of the COJ.

When everything is fine and you have assigned the mentioned privileges, a special link "**Administration**" must appears between available options specifically at the top. The administrative options available for some account depend directly of the privileges owned to it. Perhaps is needed to restart the session in the system to implement the privileges of the account, if they were recently added while account is logged. The image below shows the available options for common role of Problem Setters.



Fig. 2: Problem Setters administrative options.

Using the option "**Problems**" the section for adding and/or editing problems is opened. Once in this section a list of problems is shown including all problems which you are ever added and those problems which authors specifically marked to allow your access to them.

For adding a new problem the option "**Add New Problem**" (Manage Problems - <http://coj.uci.cu/admin/manageproblem.xhtml>) should be the next step to follow.

For each problem their status is shown (e.g. if it is enabled or not to be visible in the 24h archive, equivalent to be public or not to all users). If you don't want to publish one problem to all users of the COJ then it's really important being sure that column equivalent to "**Disabled 24h**" is marked as "**Yes**". Also are shown one option to edit the problem, another to internationalize the statement, and finally one option for normalizing files/datasets (for instance, convert dataset filenames exactly as: 001.in, 001.out, 002.in, 002.out ... N.in, N.out).

Add New Problem













	1	2	3	4	5	6	7	8	»»
id	title		Disabled 24h		enabled		edit		
1000	A+B Problem		NO		YES		  		
1001	Pipes		NO		YES		  		
1002	New House		NO		YES		  		
1003	General Election		NO		YES		  		
1004	Traversing Grid		NO		YES		  		
1005	Rent your Airplane and make Money		NO		YES		  		

Fig. 3: Problem Setters administrative options to manage problems.

When adding a new problem the following information should be provided on different sections which are explained below (**always use UTF-8 codification or text-plain to standardize new problems in the archive, these is really important in order to avoid font format problems, and gain on standardization of statements... never use copy/paste from PDFs, Web Pages or other source because will be also taken elements of format from the origin source**):

- Title/Name of the problem preferable in English and Capitalized.
- Description or problem specifications, general background.
- Input specification (in correspondence with the description or problem specifications).
- Output specification (in correspondence with the description or problem specifications).

- Sample input (in correspondence with the input and output specifications).
- Sample output (in correspondence with the input and output specifications).
- Hints with optional clues about the problem, or perhaps any additional samples.
- Source-Author, limits, input generator and model solution can be established when adding the

problem. The Source-Author must be added previously to the system in order to be selected on this step. Problems can have multiple datasets and the “**Case Time**” is measured for each of them independently. On other hand the “**Total Time**” is measured for the whole set. If a problem has only one dataset, then option “**Multidata**” should not be selected. The input generator and model solutions are optional files to be provided by Problem Setter for DataGen Tool [8].

Forum link

Author

2014 Caribbean Finals of the ACM-ICPC [Yonny Mondelo Hernández] ▼

Special judge

☐

Limits

Basic

All

Multipliers

Memory(B)		Case Execution Time(MS)		Total Execution Time(MS)		Source Code Length (B)	
268435456	Apply	1000	Apply	7000	Apply	16384	Apply

☒ Multipliers

*Apply All

Multidata

☒

Disable 24h

☐

Enable

☒

Input generator

Upload

Browse ...

Model solution

Upload

Browse ...

Fig. 4: Problem Setters administrative options to manage source-author, limits, input generator and model solution.

Remember, it’s really important being sure that option “**Disabled 24h**” is marked. Otherwise problem will be directly published and visible in the 24h archive for all users when the button “*update*” is pressed. The option “Special Judge” is for those problems which have special validators or checkers; later we will talk more about this

kind of problems. Finally, the option “Forum Link” is optional to link the corresponding thread created for the problem in the forum of the COJ (COJ-Forum - <https://coj-forum.uci.cu/>) [17]. We expect that in the near future, the integration between the COJ and the COJ-Forum will be completely automatic.

Limits

Basic All Multipliers

Memory (B)	Case Execution Time (MS)	Total Execution Time (MS)	Source Code Length (B)
268435456 <input type="button" value="Apply"/>	1000 <input type="button" value="Apply"/>	15000 <input type="button" value="Apply"/>	16384 <input type="button" value="Apply"/>

☒ Multipliers

Fig. 5: Problem Setters administrative options to manage limits of time, memory and source code length (Tab: Basic).

Through above functionality you can establish the same limit values for all programming languages; you can specify different values for each programming language; or you can manage those values with dynamic multipliers specified by each programming languages. In practice,

some values are provided by default in order to facilitate the process, but you can change them depending of your specific needs. For instance, bytes and milliseconds are default units of measure used, but you can select another unit perhaps more comfortable like megabytes or seconds.

Limits

Basic All Multipliers

Memory (B)	Case Execution Time (MS)	Total Execution Time (MS)	Source Code Length (B)
128 MB <input type="button" value="Apply"/>	<input type="button" value="Apply"/>	Max Total E <input type="button" value="Apply"/>	Max Source <input type="button" value="Apply"/>

☐ Multipliers

Multipliers dropdown: B, KB, MB, GB

Fig. 6: Problem Setters administrative options to manage limits of time, memory and source code length (Tab: Basic).

The whole time you can check the current values by clicking on the tab “All”, but take into account that you must click in the button “*update*” in order to save the last changes, otherwise they will be discarded. Optionally you can also use the dynamic multipliers by marking the option “*Multipliers*”; perhaps you need to press the buttons

“*Apply*” in order to effectively apply the multipliers to some value. Or you can just press the button “*Apply All*” in order to effectively apply the multipliers to all values. But in any case you should take care with those options because using them after change some specific value for any programming language could discard that step/change.

Limits

Basic

All

Multipliers

Memory (B)	Case Execution Time (MS)	Total Execution Time (MS)	Source Code Length (B)
C++			
268435456	1000	15000	16384
Java			
16106127360	3000	45000	16384
C#			
5368709120	2000	30000	16384

Fig. 7: Problem Setters administrative options to manage limits of time, memory and source code length (Tab: All).

Multipliers are a way to establish limits taking into account the known differences between programming languages. System provides default multipliers for limits of time, memory and source code length. But due to relativity of this issue, the Problem Setters have always the possibility to modify those multipliers or just change some specific values for any programming language.

Finally, it’s important to note that far to make more difficult the Problem Setter work, those functionalities are specifically created for making more flexible the process of adding problems. They are useful for several purposes and can be adjusted according to each personal need. But in any case, old methods can be used yet.

C#

20.02.02.01.0

PHP

5.06.06.01.0

C++11

1.01.01.01.0

Fig. 8: Problem Setters administrative options to manage limits of time, memory and source code length (Tab: Multipliers).

When adding datasets should be taken into account filenames rules (for instance, dataset filenames exactly as: 001.in, 001.out, 002.in, 002.out ... N.in, N.out). Existing datasets must be shown with their original names (see image below). When updating datasets, system will replace previous files with equal filenames. If datasets are added with incorrect names (or are wrong added) perhaps they did not appear into the list of existing datasets.

The datasets can be added to the problem one by one, or in some specific order or group. In the same way they can be deleted just clicking over the existing dataset (they can be deleted all at the same time too). Also if there are too much datasets to be added sequentially, all they can be compressed inside a unique ZIP file containing no folders and uploaded at the same time. They will be uncompressed with the rest of datasets; the ZIP file is deleted after that.

Fig. 9: Problem Setters administrative options to manage datasets.

The available languages for the problem must be selected. Typically are selected all available languages, because they can be restricted on each competition independently.

Finally should be selected the Problem Setters of the COJ for which the access will be allowed; those who can later view and/or edit the problem.

Fig. 10: Problem Setters administrative options to manage languages and access.

V. WHAT OTHER THINGS PROBLEM SETTERS SHOULD KNOW

Using the Problem Setter option “**Sources**” the section for adding and/or editing Source-Authors is opened. Once in this section a list of sources and authors is shown; each of them is a combination of sources and authors, or at least one of them. Before adding a new data to the list must be carefully reviewed if it was previously added or not.

Using the Problem Setter option “**Translations**” the section for accepting or rejecting translating proposals is opened. Once in this section a list of available translation proposals is shown; this list could be empty if there is no proposal to accept/reject. When selecting a proposal can be reviewed their content and when accepted the content of the problem is automatically updated.

The Problem Setter option “**Problem Classification**” is used to add one or more categorizations to problems,

which areas the problems belongs to. Initially a list with all uncategorized problems is shown allowing classifying them; this list don’t include disabled problems for reasons of security. For classifying a single problem it can be selected from the mentioned list or their ID can be written for directly classification (see image below).

To add one classification first should be adjusted their related complexity selecting the first circle for one and the last circle for five, and then dragged/moved the whole component from left panel to right panel, by clicking the rectangle and leaving them over the other side specifically inside the gray section. To remove one classification only should be dragged/moved the whole component from right panel to left panel by clicking the rectangle and leaving them over the other side specifically inside the gray section.

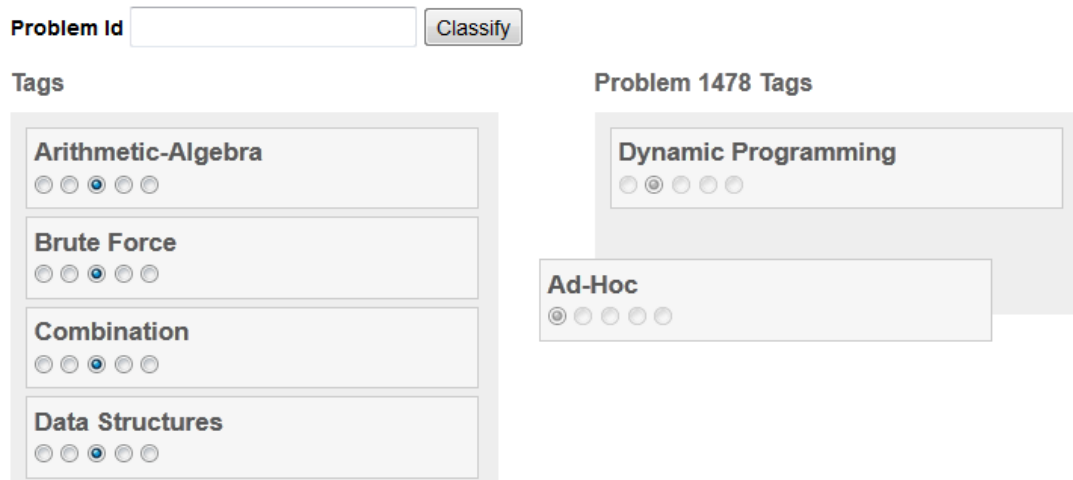


Fig. 11: Problem Setters administrative options to manage categorizations (e.g. problem 1478 of COJ).

Changing a bit of topic, for those Problem Setters and users with more experience and knowledge the module for “**Problems Recommendation**” could be an interesting tool for training [10]. Basically when a problem description is open, taking into account user profile and problem classifications, the system provide a list of similar problems carefully selected at the moment [11]. Lesser experienced users could use it as a special kind of tutor/coach to know which problem they should solve in any moment according to their personal behavior and characteristics.

Also, mainly designed for competitions purposes, there is a module in the Caribbean Online Judge for “**Plagiarism Detection**” [7]. This module allows comparing, using different algorithms and standards, a pair of codes or a group of them. In any case is possible to apply several filters by programming languages, verdicts, ranges, users, problems, and others. The use of this module is very stressful for the system and it’s not really needed by normal users in general cases; for those reasons the module is only allowed for administrators.

VI. WHAT VERDICT THE CARIBBEAN ONLINE JUDGE USE FOR JUDGMENT

When the Caribbean Online Judge has judged your submission, you will get a judgment that tells you what the verdict of your solution is. You can view the verdict on the judgments page.

The following judgments are possible:

- **Judging (JDG):** The judge is busy and it can't judge your solution at the moment or it's currently waiting for an answer from the judgment engine. Usually you just need to wait for a few seconds.
- **Unqualified (UQD):** The UQD verdict means that the program sent by the user couldn't be run in that moment (engine stopped, datasets no available, among other reasons).
- **Invalid Function (IVF):** The IVF judgment means that the program sent by the user tried to do something which isn't allowed on the COJ. This includes the manipulation of file descriptors, opening files, running functions as `fork()` or `exec()`, creating threads, sending signals, or basically anything needless to solve the problem. All IVF judgments are examined to detect any attempt to exploit the system, in such case the involved user will be analyzed and banned from the COJ. Please, contact us if you think that an IVF judgment is not correct or "fair".
- **Internal Error (SIE):** Internal Error is an error in judgments, or engine error, that is supposed which should not happen. Some errors are not internal, because they originate from the source code and they are expected, such as compilation errors or run-time errors; those are commonly reported to users as normal verdicts in the system. But there are errors related with inconsistent or missing datasets, or because the server does not have enough space, or even due errors in system programming; then an unexpected error are originated. That is an Internal Error, and must be closely reviewed by COJ Development Team (CDEVT).
- **Runtime Error (RTE):** The RTE judgment means that the program sent by the user has crashed during the execution with the judge's secret tests (e.g.: the process was signaled or exited with a status other than 0). Usually in C and C++, the value 0 must be returned from `main()`. In Java doing something to ensure that the exit status is 0 isn't necessary. Some of the common reasons and their meanings are:
 - **ACCESS_VIOLATION** - The program tried to read from or write to an address for which it doesn't have the appropriate access.
 - **ARRAY_BOUNDS_EXCEEDED** - The program tried to access an array element that is out of bounds and the underlying hardware supports bounds checking.
 - **FLOAT_DENORMAL_OPERAND** - One of the operands in a floating-point operation is denormal. A denormal value is one that is too small to represent as a standard floating-point value.
 - **FLOAT_DIVIDE_BY_ZERO** - The thread tried to divide a floating-point value by zero.
 - **FLOAT_OVERFLOW** - The exponent of a floating-point operation is greater than the magnitude allowed by the corresponding type.
 - **FLOAT_UNDERFLOW** - The exponent of a floating-point operation is less than the magnitude allowed by the corresponding type.
 - **INTEGER_DIVIDE_BY_ZERO** - The program tried to divide an integer value by zero.
 - **INTEGER_OVERFLOW** - The result of an integer operation caused a carry out of the most significant bit of the result.
 - **STACK_OVERFLOW** - The program used up its stack.
- **Time Limit Exceeded (TLE):** The TLE judgment means that the program sent by the user ran for too much time. When the time limit is exceeded, the program is terminated. The output produced isn't inspected in this case, so getting TLE doesn't mean that the program produced the correct output, it means only that it didn't finish in time.
- **Output Limit Exceeded (OLE):** The program sent by the user tried to write too much information. This usually occurs if it goes into an infinite loop. Currently the output limit is 64 MB.
- **Size Limit Exceeded (SLE):** The source code sent by the user exceeds the size limit. Currently

the maximum size limit is 100 KB, but usually this value is modified by Problem Setters.

- **Memory Limit Exceeded (MLE):** The program sent by the user tried to use more memory than allowed.
- **Compilation Error (CE):** The CE judgment means that the source code sent by the user couldn't be compiled. Extra information (which can help you to debug the error) can be found on the judgments page. Any compilation time over one minute will cause a CE.
- **Wrong Answer (WA):** The WA judgment means that the program sent by the user finished within the time limit, but the output produced was

incorrect. It is important to check the output specification of the problem to avoid this judgment due to, for example, simple white-space errors.

- **Presentation Error (PE):** The output format of the program sent by the user isn't exactly the same as the judge's output, although the answer to the problem is correct. Check the program's output against the output specification of the problem to detect blank lines, spaces, and others.
- **Accepted (AC):** The AC judgment means that the program sent by the user has successfully passed all tests and produced the correct output. Congratulations!

VII. HOW TO CREATE SPECIAL VALIDATOR (ALSO CALLED AS CHECKERS)

One special validator or checker code is like any other language code. The difference is that initially reads three files that are passed as parameters to the executable checker (see comments in the sample). All validation is done on the information that is collected from those files. And sometimes the output files does not have to contain all solution information, but only what is needed the validator to check.

Whenever a wrong verdict "Wrong Answer" is detected an error message with an arbitrary description of the verdict could be printed, and the checker must return

the code 201 (instead of the usual return 0). If accepted, the error message must be exactly "Accepted" and the checker must return the code 200. If the format is validated and it is detected that is incorrect, a verdict of Presentation Error can be returned. In this case the error message must be exactly "Presentation Error" and the checker must return the code 202. The error description of the verdict "Wrong Answer" is the only which can be personalized. Other cases error descriptions are strict.

Below there is one special validator or checker code in C++ [20], for problem 2988 of COJ:

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  char cad[1001];
4  int n, s, d, cs, arr[105], posx[105], posy[105];
5
6  int GetDIST(int p, int q)
7  {
8      return (posx[p]-posx[q])*(posx[p]-posx[q])
9          + (posy[p]-posy[q])*(posy[p]-posy[q]);
10 }
11 bool FinalCheck()
12 {
13     for(int p = 1; p <= s; p++)
14         for(int q = p + 1; q <= s; q++)
15             if(GetDIST(arr[p], arr[q]) > d)
16                 return false;
17     return true;
18 }
19
20 int main(int argc, char *argv[])
21 {
22     FILE *in=fopen(argv[1], "r"); //Reading server input file...
23     FILE *out=fopen(argv[2], "r"); //Reading user output file...
24     FILE *cout=fopen(argv[3], "r"); // Reading server output file...
25     fscanf(in, "%d%d", &n, &d); d = d*d; // d ^ 2, Reading initial positions...
26     for(int p = 1; p <= n; p++) fscanf(in, "%d%d", &posx[p], &posy[p]);
27
28     fscanf(cout, "%d", &cs);
29     if(1 != fscanf(out, "%d", &s))
30     {
31         fprintf(stderr, "Input Missing");
32         return 201;
33     }
34     if(cs != s)
35     {
36         fprintf(stderr, "Wrong Answer");
37         return 201; //Invalid Solution...
38     }
```

```

39
40     for(int p = 1; p <= s; p++)
41     if(1 != fscanf(out, "%d", &arr[p]))
42     {
43         fprintf(stderr, "Input Missing");
44         return 201; //Data is needed to continue...
45     }
46
47     for(int p = 1; p <= s; p++)
48     if(arr[p] < 1 || arr[p] > n)
49     {
50         fprintf(stderr, "Out of Range");
51         return 201; //Values Out of Range
52     }
53     for(int p = 1; p <= s; p++)
54     for(int q = p + 1; q <= s; q++)
55     if(arr[p] == arr[q])
56     {
57         fprintf(stderr, "Repeated");
58         return 201; //Repeated Values
59     }
60     if(1 == fscanf(out, "%s", &cad))
61     {
62         fprintf(stderr, "Trash in Output");
63         return 201; //Trash at the End of Output...
64     }
65
66     //Do Final Check...
67     if(!FinalCheck())
68     {
69         fprintf(stderr, "Wrong Answer");
70         return 201; //Invalid Solution...
71     }
72     fprintf(stderr, "Accepted");
73     return 200; //Everything OK...
74 }
75

```


VIII. CONCLUSIONS

This complete guide is especially for those who want to create problems to be used in the Caribbean Online Judge, although most of the content in this document is not exclusive for this system because it can be applied to the work as Problem Setters, in general cases. Contrary to what many people think the work as Problem Setters is a hard task, mainly strengthened by the experience and knowledge in mostly of the cases.

This document is a compilation of topics we made seeking to increase quality and standardization of the problems created for the Caribbean Online Judge. It must serve as one step by step guide for Problem Setters working, including tips and most used practices related with the task of creating good problems statements, generating datasets, testing solutions and constructing special validators.

REFERENCES

- [1] “Caribbean Online Judge”, SAPCO Project, University of Informatics Sciences, **2010-2015** [Online] [Available: <http://coj.uci.cu/>].
- [2] **Mondelo Hernández, Yonny. Ripoll Méndez, Dovier A. Junco Vázquez, Tomás O.** “Caribbean Online Judge as tool of Collaborative Work in terms of Competitive Programming”, 9th Colombian Congress of Computation, Pereira, Colombia, September **2014**.
- [3] **Soria Ramírez, Jorge A. Altuna Castillo, Enrique J.** “Analysis, design and development of an Online Judge”, Thesis for Grade of Informatics Engineering, UCI, Havana, Cuba, June **2008**.
- [4] **Roque Álvarez, Jorge L. Lobaina Guzmán, Juan C.** "Development of Caribbean Online Judge v2.0", Thesis for Grade of Informatics Engineering, UCI, Havana, Cuba, June **2012**.
- [5] **Acosta Labrada, Nersa D.** “Development of module COJMail for Caribbean Online Judge”, Thesis for Grade of Informatics Engineering, UCI, Havana, Cuba, June **2012**.
- [6] **Acosta Labrada, Nersa D. Altuna Castillo, Enrique J. Martínez García, Jorge. Hechevarría Rojas, Arisbel.** “Development of an Online Judge over email”, International Conference UCIENCIA, UCI, Havana, Cuba, **2012**.
- [7] **Quiñones Guerrero, Yisel. González Vallejo, Leandro.** “Development of module Plagiarism Detection for Caribbean Online Judge”, Thesis for Grade of Informatics Engineering, UCI, Havana, Cuba, June **2012**.
- [8] **Yurell Villafranca, F.** “Development of module DataGen for Caribbean Online Judge”, Thesis for Grade of Informatics Engineering, UCI, Havana, Cuba, June **2013**.
- [9] **González Rodríguez, Susana. Plana Botello, Alain.** “Development of module Statistics for Caribbean Online Judge”, Thesis for Grade of Informatics Engineering, UCI, Havana, Cuba, June **2014**.
- [10] **González Fernández, José C. Doria Martínez, Junet.** “Development of module of problems recommendation for Caribbean Online Judge”, Thesis for Grade of Informatics Engineering, UCI, Havana, Cuba, June **2014**.
- [11] **Viltres Sala, Hubert.** “Method for problems recommendation in the Caribbean Online Judge”, Thesis for Grade of Master in Applied Informatics, UCI, Havana, Cuba, **2014**.
- [12] **Pérez Pérez, Yobanni.** "Update and support of module COJMail for Caribbean Online Judge", Thesis for Grade of Informatics Engineering, UCI, Havana, Cuba, June **2015**.
- [13] “Caribbean Online Judge - Rank of Users”, SAPCO Project, University of Informatics Sciences, 2010-2015 [Online] [Available: <http://coj.uci.cu/24h/usersrank.xhtml>].
- [14] “Caribbean Online Judge - Rank of Institutions”, SAPCO Project, University of Informatics Sciences, 2010-2015 [Online] [Available: <http://coj.uci.cu/24h/institutionsrank.xhtml>].
- [15] “Caribbean Online Judge - Rank of Countries”, SAPCO Project, University of Informatics Sciences, 2010-2015 [Online] [Available: <http://coj.uci.cu/24h/countriesrank.xhtml>].
- [16] “Caribbean Online Judge - Past Competitions”, SAPCO Project, University of Informatics Sciences, 2010-2015 [Online] [Available: <http://coj.uci.cu/contest/past.xhtml>].
- [17] “Caribbean Online Judge - Forum”, SAPCO Project, University of Informatics Sciences, 2010-2015 [Online] [Available: <https://coj-forum.uci.cu/>].
- [18] “Caribbean Online Judge - Links”, SAPCO Project, University of Informatics Sciences, 2010-2015 [Online] [Available: <http://coj.uci.cu/contest/links.xhtml>].
- [19] “Factsheet of ACM International Collegiate Programming Contest (ACM-ICPC); Edition of January, 2016.” [Online] [Available: <http://icpc.baylor.edu/download/worldfinals/pdf/Factsheet.pdf>].
- [20] **Mondelo Hernández, Yonny.** “How to write a checker or special validator in C++” [Online] [Available: <http://coj.uci.cu/downloads/files/Checker2988.cpp>].