ICPC Latin American Regional Contests – 2025

Testing Environment and Submission System

1 Information on the Testing Environment

1.1 Environment

The submission correction system will run on the `Ubuntu GNU/Linux 24.04 LTS amd64` distribution, with the following compilers and interpreters configured:

```
C: gcc version 13.3.0 (Ubuntu 13.3.0-6ubuntu2~24.04)
C++20: g++ version 13.3.0 (Ubuntu 13.3.0-6ubuntu2~24.04)
Python 3: PyPy3 version 7.3.15 with GCC 13.2.0 providing Python 3.9.18
Java: openjdk 21.0.8
Kotlin: kotlin version 2.1.10 (JRE 21.0.8+9-Ubuntu-0ubuntu124.04.1)
```

1.2 Memory limits

All submissions in all languages are limited to a total of 1GB of RAM and max stack size of 128MB.

Note: The 1GB memory limit specified includes the memory used by the execution environment, so you have to take that into account. For example, for Java, Python and Kotlin you have to consider the memory limit includes the memory used by the runtime environment, which can be several megabytes. For C and C++ the memory used by the C runtime is relatively small, but should still be considered.

1.3 Time limits

Before the contest, the judges will have solved all problems in languages from at least two of the three distinct language groups (C/C++, Java/Kotlin, and Python3). Time limits for each problem will be calculated based on the runtime of those solutions.

A link to the document containing time limits for each problem will be available on the `Problems` tab of Boca's web interface.

1.4 Other limits

```
Source file size: 100KB
Compiled output size: 32MB
Compilation time: 10 seconds
```

1.5 Compilation and execution

It is strongly recommended that you compile and run your solutions using the commands specified in the language-specific sections below. While these commands do not enforce memory or time limits as the judge does, they match the environment the judges will use to compile and execute submissions.

1.5.1 C/C++20

- The executable file generated by the compiler will be executed to generate the output of the submission.
- Your program must return a zero, executing, as the last command, `return 0` or `exit(0)`.
- It is known that for problems with large inputs, `iostream` objects can be slow as, by default, they use a buffer synchronized with the stdio library. If you want to use `cin` and `cout`, disabling such synchronization is advised. This can be achieved by calling `std::ios::sync_with_stdio(false);` at the beginning of your main function. Note that, in this case, using `scanf` and `printf` in the same program should be avoided, since, with separate buffers, misbehavior might occur.
- Compile C solutions with:
  `gcc -x c -g -O2 -std=gnu11 -static {submitted_file} -lm`
- Compile C++ solutions with
  `g++ -x c++ -g -O2 -std=gnu++20 -static {submitted_file}`

1.5.2 Java

- The compiled main class will be executed to generate the output of the submission.
- DO NOT declare a `package` in your Java program; if you declare a package the program will not execute in Boca.
- The convention for the solution file name must be obeyed, which means that your public class name must be a capital letter (A, B, C, ...).
- It is known that for problems with large inputs, `Scanner` can be too slow. Using buffered I/O (for example, `BufferedReader` and `PrintWriter`) is advised.
- Compile Java solutions with:
  `javac -sourcepath . -d . {submitted_file}`
- Run Java solutions with:
  `java -XX:+UseSerialGC -Xms1024m -Xmx1024m -Xss128m {problem_code}`

1.5.3 Python

- The main source file will be executed by the PyPy3 Python3 interpreter to generate the output of the submission.
- Python programs will be "syntax checked" when submitted; programs which fail the syntax check will receive a "Compilation Error" verdict.
- Note that, Python submissions will be executed with the **PyPy3** interpreter, instead of the traditional CPython. This generally results in better performance, but be aware of any subtle differences in behavior.
- "Compile" (syntax-check) Python solutions with:
  `pypy3 -m py_compile {submitted_file}`
- Run Python solutions with:
  `pypy3 {submitted_file}`

1.5.4 Kotlin

- The compiled main class will be executed to generate the output of the submission.
- DO NOT declare a `package` in your Kotlin program; if you declare a package the program will not execute in Boca.
- The convention for the solution file name must be obeyed, which means that your source file must be named (A.kt, B.kt, C.kt, ...).
- Compile Kotlin solutions with:
  `kotlinc -d . {submitted_file}`
- Run Kotlin solutions with:
  `kotlin -J-XX:+UseSerialGC -J-Xms1024m -J-Xmx1024m -J-Xss128m {problem_code}Kt`

2 Instructions for the Usage of the Boca Submission System

2.1 Submission of Solutions

To submit a solution, you must use the Boca's web interface:

- Open your browser.
- Login as a team (username and password assigned to your team).
- Access the tab `Runs`. Choose the appropriate problem, the language used and upload the source file.

The verdicts you may receive from the judges are:

- 1 - YES
- 2 - NO - Compilation error
- 3 - NO - Runtime error
- 4 - NO - Time limit exceeded
- 5 - NO - Wrong answer
- 6 - NO - Contact staff

The meanings of 1, 2, 3, 4 and 5 are self-explanatory. 6 is used for unforeseeable circumstances. In this case, use the "Clarifications" menu and provide the "run" number for further clarification.

Your program may be executed on multiple input files. Note that this means that if your program has more than one error (say, "Time Limit Exceeded" and "Wrong Answer"), you may receive any of these errors as the verdict.

Keep the following judging notes in mind:

- If your solution takes too long to compile or produces an executable/bytecode that is too large, you will receive a "Compilation Error" verdict.
- There is no such thing as "Presentation Error" or "Format Error". If you misspell the word "impossible", for example, and the problem requires that word as output, then your submission will be judged as "Wrong Answer".
- Unless a problem specifically indicates that uppercase or lowercase letters are important, then either will be accepted. For example, "Yes" or "yes" would be treated the same, but "yse" would be judged as "Wrong Answer".
- Output formatting should follow the sample output in the problem statement, although extra whitespace within reason is acceptable. For example, if you print out thousands of blank spaces within the time limit of the problem, that would be judged as a Wrong Answer; however, an extra blank at the end of a line or between tokens or an extra blank line is acceptable.
- For problems with floating-point output, the judges accept any answer that satisfies the constraints described in the statement. Unless otherwise stated, those constraints are given as an absolute and/or relative tolerance, so your answer does not need to match the sample digits exactly as long as it stays within the tolerance. If the correct value is x and you output y, the judge will accept it whenever the condition from the statement is met (for example, $|x - y| \leq \varepsilon$ for an absolute tolerance ε, or $\frac{|x-y|}{|x|} \leq \varepsilon$ for a relative tolerance). For instance, if the tolerance is 10^{-9} and the correct answer is 1, outputs such as 1, 1.000000, or 0.999999999 are all acceptable.
- When the statement specifies that multiple outputs are acceptable (for instance, any shortest path or any ordering that satisfies given constraints), the judges will check that your output matches the requested format and requirements; they will not expect it to match the sample output exactly.
- If your program generates excessive output, your submission will receive a "Runtime Error" verdict. Note that any content written to the standard error output (*stderr*) also counts towards this limit. Therefore, an excess of debugging messages can cause runtime errors. If you encounter an unexpected runtime error, consider reducing the logs in *stderr*.

Penalties

Each submission to a problem that receives a "`NO`" verdict before the first "`YES`" verdict for the same problem will incur a time penalty of 20 minutes that is added to the total time of the team. There is no penalty time for unsolved problems.
Exceptions for this rule are "`NO - Compilation error`" and "`NO - Contact staff`" that have no time penalty associated with them.

Response Times

Note that the time required for coming up with a verdict varies depending on the problem, the verdict, and the point at which the contest is in when the submission is received. As solutions are judged simultaneously, this means that you might receive the verdict of your runs out of order. Also, in some cases, manual verification is required from the judges. Depending on what needs to be verified and the number of submissions that require manual verification, even delays on the order of minutes are expected.

2.2 Clarifications

All communication with the judges is done through clarification messages.
To request a clarification concerning a problem statement, you must use Boca's web interface:

- Open your browser.
- Login as a team (username and password assigned to your team).
- Access the tab `Clarifications`. Choose the appropriate problem and type your question.

Both clarification replies from the judges and requests sent by you are displayed there. There will be an alert once your clarification is replied (or the judges issue a global clarification).
Note that it's quite uncommon to have clarifications issued and most of the time the answer to the questions received is already on the problem statement. Please read the problem statement and examine the sample test cases carefully before submitting any request for clarifications.

2.3 Score

To visualize the scoreboard showing the teams ranking, you must use Boca's web interface:

- Open your browser.
- Login as a team (username and password assigned to your team).
- Access the tab `Score`. You will have access to the local scoreboard.

2.4 Tasks

The tab `Tasks` allows the team to send files for printing, as well as ask for help from a staff member.

- To print a file, just select it from the disk and click on `Send`.
- To ask for help click on the `S.O.S.` button. Note: the help provided by the staff has to be only related to issues with the computers or other physical problem.

3 Interactive Problems

It is possible that the problem set contains one or more interactive problems.

In many respects, interactive problems are like any other: your program must read from standard input and write to standard output. The difference is that, in this case, your program's input and output are connected to another program (the interactor), with which it must communicate continuously, sending data and receiving responses.

For this, it is crucial that your program "flushes" the output immediately after each write, which can be done as follows:

- In C (or C++ using `cstdio`), you can use `fflush(stdout)`.
- A C++ output stream is flushed automatically each time you write the `endl` manipulator. When using other means or if you want to be sure, call `cout.flush()`.
- In Java and Kotlin, explicitly flush the stream after every message (for example, `System.out.flush()` or a `PrintWriter` created with autoFlush enabled).
- In Python, you can call `sys.stdout.flush()` or use `print(..., flush=True)`.

If you fail to do this, the communication may not work correctly, resulting in verdicts like "Wrong Answer" or "Time Limit Exceeded".

Additionally, pay attention to the following points:

- The problem statement defines the **interaction protocol**: who starts, the format of each message, and how the interaction should proceed. Your program must **follow this protocol exactly**, respecting spaces, line breaks, and formats.
- If your program fails to read the interactor's response, or tries to read when there is nothing available (e.g. because you forgot to flush your previous output), then the program will stall indefinitely and your submission will get "Time Limit Exceeded", even if the algorithm is correct.
- Malformed outputs or outputs in the incorrect order can lead the interactor to terminate the execution with a "Wrong Answer" verdict.
- The interactor may behave in an adversarial manner, adapting the inputs to your responses to try to expose errors in the algorithm; each problem statement will specify whether its interactor behaves adversarially or deterministically.
- You must still finish within the stated time limit; you can assume the limit already accounts for the interactor's own work, but if your solution runs longer than that you will receive "Time Limit Exceeded". If the statement requires the answer within a fixed number of steps or messages (for example, at most X queries), you must perform those interactions within that same time limit.