Practice Competition Instructions for Contestants

Practice competition is held for you to learn to operate your environment. The tasks are easy and printed solutions are provided at the competition.

First of all, you should study the programming environments you are planning to use so that you know how to use and customize them. In IOI 2002 you have a choice of several different program development tools. You should try to make a decision on which tools to use.

The primary choices available for Linux are:

- For C, C++, and Pascal: editor of your choice (for example vim, emacs), command line compilation, and gdb or ddd for debugging.
- For C and C++: RHIDE - an integrated development environment. RHIDE can also be used as just an editor for C, C++ or Pascal program development.
- For Pascal: fp - an integrated development environment that comes with freepascal. The fp is still in beta and may have some problems.

The primary choices available for Windows XP are:

- For C, C++, and Pascal: editor of your choice (for example Edit), command line compilation, and gdb for debugging.
- For C and C++: RHIDE - an integrated development environment.
- For Pascal: fp - an integrated development environment that comes with freepascal. The fp is known to have some problems.

Second, you need to learn to use the web service for IOI 2002: *submitting solutions, test executing your programs*, printing, and making backups. You should also try submitting solutions that do not compile or work correctly.

Third, you need to familiarize yourself with the task types and the materials that come with sample tasks. You may need to use a library in the actual competition. To be familiar with using libraries, you should try to link to your program the library provided for the task **string**, even though you do not really need it to solve the task.

2002 International Olympiad in Informatics

Informatics for Digital World

Version: 1.03

IOI 2002 Competition Rules

These Competition Rules include the Competition Procedures and Judging Procedures, which the host is obliged to send to invited countries four months prior to the competition. Minor changes to these rules will likely be made; the final version will be distributed in the first GA meeting of IOI 2002.

Competition Dates

IOI 2002 will take place from Sunday, August 18 (Arrival Day) to Sunday, August 25 (Departure Day). The First Competition Day is Tuesday, August 20, and the Second Competition Day is Thursday, August 22. On each competition day contestants will be given three tasks to complete in the five hours from 9:00 to 14:00.

There will also be a practice competition round on Monday, August 19. All contestants MUST take part in the practice competition round.

Competition Equipment

The specification is: a PC with a 1.7 GHz Pentium 4 processor, 256 MB RAM, a standard US keyboard, a mouse, and a color screen. If the model information is changed, this section will be updated, and announcements will be made on the web site and the IOI mailing list.

Blank paper and writing utensils will be provided. Contestants may NOT take any material such as computing equipment (including calculators, organizers, PDAs, computers, ...), books, manuals, written or printed materials, diskettes, CD-ROMs, or communication devices into the competition area. A contestant who is in possession of this type of material in the competition room may be disqualified.

Programming Environment

The computers have a dual-boot installation of Debian GNU/Linux 3.0 'woody' and Windows XP. In both the Linux and Windows environments, the programs installed for the competition are set up in such a way that they can be found in the users' path (i.e. no extra setup is needed to use the tools). Both the Linux and Windows platforms include:

- GCC compiler version 2.95.3, and
- Freepascal (fpc) compiler version 1.0.6.

These are the official compilers for IOI 2002.

Newer versions of software may be installed as necessary to resolve hardware problems and/or software compatibility/bug-patch issues. If so, the changes will be announced on the competition web site and the IOI mailing list. The contestant should be familiar with the programming package of his/her choice, including the use of libraries or units. The contestant should be able to execute programs, change the working directory and manage files, and use a web browser.

Similar installations will be used for the computers in the translation computer room. Windows installations include MS Word with some multi-lingual support and PowerPoint. In the Linux environment, TeX will be provided.

Additional tools may be available for assisting the contestants with the tasks. Documentation about the computing environment which does not reveal the nature of the tasks will be made available on the competition web site.

Competition Tasks

All of the tasks in IOI2002 are designed to be algorithmic in nature. There are two types of tasks: (1) tasks for which a solution comprises a single source file of a computer program, and (2) tasks for which a solution comprises a set of "output" data files.

Efficiency plays an important role in some tasks. Whenever efficiency of algorithmic computations is important, there will be at least one grading input for which some correct but inefficient programs can also score some points. It is important, therefore, for contestants to attempt a task even if the contestant does not know how to solve the hardest possible test cases.

(1) Tasks for which a program source file is requested as a solution:

When a program source file is required as a solution, the program source provided by the contestant must be contained in a single source file. The task documentation will specify:

- the input and output data format and value ranges,
- the resource limitations for the computations (e.g. cpu time, memory),
- any other constraints on the program behavior, and
- the comment tags required in the source code so that the grading system can identify the task and programming language.

The submitted source program must be smaller than 1 MB and the evaluation server computer must be able to compile it in less than 30 seconds. Submitted programs which do not meet these constraints will be rejected by the submission system and the contestant will be notified.

(2) Tasks for which output data files are requested as a solution:

There may be tasks for which input data is given to the contestant and the contestant is required to produce only the output data as an answer. If the contestant writes programs to help determine the output data, the programs should not be submitted with the solution. The input data will be provided in ASCII text files. For these tasks, the task documentation will specify:

- the structure of the input and output files, and
- the full set of official input files.

Input and output data:

In all tasks, input and output data consists of a sequence of items. An item is a string of printable non-white-space characters (ASCII code from 33 through 126). An item may represent an integer or a general string; the meaning of each item will be given in the task specification.

Spaces and end-of-line characters separate items. The format of the input data will be given in the task specification. The output data files should be formatted strictly according to the task-specific instructions. However, the grading system scores output files using C++ streams in such a way that extra white space (spaces and end-of-line characters) between or around items is ignored.

Directories:

In both Windows and Linux, the environment will be provided with a directory created for each task. Each directory will be named after its task and will contain any required task-related materials. As an example, consider a competition round with three tasks, named "number," "string," and "red." In Linux each contestant's home directory will have the three subdirectories `~/number/`, `~/string/`, and `~/red/`; and in Windows each computer will have the folders `C:\ioi\number\`, `C:\ioi\string\`, and `C:\ioi\red\`. All provided files relating to the "string" task will be contained in the `~/string/` subdirectory in Linux and in the `C:\ioi\string\` subdirectory in Windows.

Practicing

The competition computers will be available for practice during hours that will be announced at the competition. All contestants must take part in the practice competition round on Monday, August 19. Before each competition round, the computers will be assigned randomly to the contestants (with a different assignment each time).

Curfew

A curfew will be in effect beginning with the start of a GA meeting where tasks for a competition day are presented and approved, and ending on the following competition day after the start of the competition. During the curfew the contestants are not allowed to communicate by any means, direct or indirect, with any people who attend this meeting. The contestants and the GA meeting participants must obey any instructions which limit the area where they are allowed to be. The GA meeting participants are not allowed to communicate task-related information to anyone not at the meeting before the end of the curfew.

Any contestant breaking the curfew may be disqualified. If some other person associated with a national delegation breaks this rule, then all contestants of that delegation may be disqualified.

Competition–time Procedures

Starting the competition:

Contestants will be taken to the competition hall before the competition starts. A randomly chosen computer is designated to each contestant (with a different assignment each time). The computer will be powered up and will display a menu from which the contestant may choose to boot either Linux or Windows. The competition envelope containing the task definitions and other necessary information will be in front of the computer. Contestants are not allowed to touch the keyboard or open the envelope until the start signal is given. At the starting whistle, contestants may open their envelopes and use their computers.

Logging in is not necessary for Windows. Under Linux, contestants should log in as:

```
username: ioi
password: ioi
```

Questions:

During the first hour of competition, contestants may submit written questions concerning any ambiguities or points needing clarification in the competition tasks. Questions must be submitted on the provided Clarification Request Forms, expressed either in the contestant's native language or in English. If required, delegation leaders will translate their contestants' questions to English after they are submitted before sending the questions to the Scientific Committee.

The Scientific Committee will answer every question submitted by the contestants. Since this may take some time, contestants should continue working while waiting for the answer to their questions. The only responses which will be given are "YES", "NO," and "NO COMMENT;" contestants should phrase their questions so that a yes/no answer will be meaningful. Contestants will not be involved in or exposed to discussion regarding their questions.

Assistance:

Contestants may ask the support staff for assistance at any time. The staff members will not answer questions about the competition tasks, but will deliver Clarification Request Forms and printouts, help locate toilets and refreshments, and assist with computer problems.

Printing:

Contestants will be able to print via a facility provided in the competition environment. The support staff will deliver the printouts to the contestants; there might be a small delay before printouts are delivered. Contestants should not leave their computer to find printouts.

Backups:

Contestants will be able to make and retrieve backups through a facility provided in the competition environment.

Test execution:

For tasks that require programs as solutions, a contestant will be able to submit a solution along with an input file for test execution. The test execution system will compile and execute the program under Linux, enforcing the resource limitations for the particular task. The program output, the execution time, and possibly error messages will be displayed. A contestant can have at most one test execution in progress at a time; until a test execution has completed further submissions will be blocked. The test execution facility will not be available during the last 5 minutes of the competition.

Submitting:

Contestants will be able to submit their solutions through a facility provided in the competition environment. For tasks which require output files as solutions, the submission facility will validate the format of the output file submitted, accepting the output file for grading if it passes. For tasks that require programs as solutions, the submission facility will verify that the program compiles and obeys the stated limits on source code size and compile time, and will run the program on a simple test case that is given in the task description, enforcing the relevant run-time resource constraints. If the submission produces the correct output, then the submission is accepted for grading.

Contestants may submit any number of times for each task; each accepted submission replaces any other submissions of that task by that contestant. The last accepted submission by a contestant for a task is officially graded in a separate process and contestants will not be informed about the results until after the competition.

Ending the competition round:

Warnings will be given with 15 minutes remaining in the round (3 short whistles and a verbal announcement "15 minutes"), 5 minutes remaining (2 short whistles and a verbal announcement "5 minutes") and 1 minute remaining (1 short whistle and a verbal announcement "1 minute"), and the end of the round will be announced (3 long whistles and a verbal announcement "end of competition round").

At the announcement ending the round, contestants must immediately stop working and put their keyboards on top of their terminals without switching off their computers. Contestants should then wait at their desks without operating their computers or touching anything on their desks; an additional announcement will be made instructing them to leave their tables and exit the competition hall. At this point, contestants may take with them the contents of their competition envelope.

Grading

The grading system evaluates the submitted tasks after the competition round. For tasks that require programs as solutions, the submitted source files will be re-compiled under Linux, enforcing the source file size and compilation time constraints. The compiler options for Pascal programs are "`-O2 -So -XS`" and the compiler options for C and C++ programs are "`-O2 -static -lm`".

The grading system will then execute the compiled program under Linux, enforcing the task-specific run-time resource constraints. Typically, there will be a CPU run-time limit and a limit on total memory use. Every limit applies independently for each test case; if any limit is exceeded, no points will be awarded for that test case. The actual limits will be specified in the task materials.

If the submission facility accepts a program, that only means that the compilation was successful and the program correctly solved the simple test case within the resource constraints, but no more. In particular, it does not mean that the program would obey the resource constraints when given different input.

The IOI 2002 schedule will specify the times when the grading results and evaluation data used for grading will be made available to the delegations, and when grading appeals are to be submitted to the Scientific Committee.

Other Information

A contestant

- trying to interfere with other contestants' activities,
- trying to break the installations or evaluation facilities,
- trying to harmfully interfere with the running of the competition in any way, or
- trying to communicate in any way during a competition round with anyone other than the competition staff

will be disqualified from the competition.

The competition computers are connected via a local area network for submitting solutions, running test executions, making backups, and printing. Contestants are not allowed to access the network for any other purpose or with any tools other than the tools provided by the organizers. Even sending a single 'ping' command is strictly prohibited. The competition staff should be contacted for help with any suspected network problems. Also, contestants are not allowed to make any material accessible to the network from their computers. The competition facilities are provided over secure connections. The network traffic is monitored and logged during the competition; a contestant breaking these rules will be disqualified.

Submitted programs

- are not allowed to access the network,
- are not allowed to fork,
- are not allowed to create files other than those required in the task definition,
- are not allowed to attack the system security or the grader,
- are not allowed to attempt to execute other programs,
- are not allowed to change file system permissions, and
- are not allowed to read file system information other than the input file given in the task description.

A contestant whose program attempts any of the above will be disqualified.

Version: 1.01

IOI 2002 Programming Environment

General

Please first check the general information about the competition programming environment from the Competition Rules.

The main environment for the contest is Linux. Linux is available as a programming environment (specifications below) and also the servers and evaluation (grading) runs on Linux. However, we provide the contestants with dual-boot computers where you can program either in Linux or in Windows environment.

The evaluation is based on source-code submission and the evaluation system compiles the submitted source code. As a consequence, also the programs written in the Windows environment are re-compiled for evaluation in Linux (using the same compiler). This is something that all contestants using Windows must be aware of. For example, uninitialized variables may cause undefined behavior when executing for the evaluation.

We favor fairly standard operating system installations. But we may modify the installations for hardware support and security fix.

The compilers used in the competition are GCC for C and C++ programs and Freepascal for Pascal programs.

Generally, the installations are designed for the following main alternatives:

1. Pascal as the programming language, Freepascal compiler, Freepascal IDE.
2. C/C++ as the programming language, GCC compiler, RHIDE IDE.
3. Editors(emacs, vim, ...), command-line compilation/debugging,
   a graphical front end "ddd" to debugging.

Option 3 is targeted primarily for Linux, although it is possible to use Windows Edit and command-line compilation.

Hardware

The specification is: a PC with a 1.7 GHz Pentium 4 processor, 256 MB RAM, a standard US keyboard, a mouse, and a 19 inch CRT.

Linux

For Linux, we are using Debian release 3.0 `woody'. You can get more information from Debian's home pages at http://www.debian.org. The tasks are chosen by tasksel with the following choices:

- X window system
- desktop environment
- C and C++

And additional packages are chosen by dselect:

- ddd - The Data Display Debugger, a graphical debugger frontend.
- mc - Midnight Commander - A powerful file manager. - normal version
- mozilla - Mozilla Web Browser - dummy package
- vim - Vi IMproved - enhanced vi editor
- vim-gtk - Vi IMproved - GTK version
- exuberant-ctags - multi-language reimplementation of ctags
- emacs21 - The GNU Emacs editor.
- emacs21-el - GNU Emacs LISP (.el) files.
- joe - user friendly full screen text editor

GCC on Linux:

We use gcc-2.95 which is installed as a part of the Linux Debian woody.

You can learn about the availability of various GCC versions through http://gcc.gnu.org. If you install a Linux version and include development tools, then you are extremely likely to get a GCC version.

Pascal on Linux:

You can get the Freepascal software through http://www.freepascal.org, which shows a number of mirror sites. We have installed the binary version of freepascal 1.0.6. You can download fpc-1.0.6.ELF.tar (14.3 MB) file, which contains a standard tar archive, with an installation script. After untarring the archive, you can run the installation script in the created directory by issuing the command "sh install.sh".

RHIDE for Linux:

The debian woody doesn't contain the RHIDE package. You can download the tarball file from http://www.rhide.com.

Pascal IDE for Linux:

You can download the snapshot version of Linux IDE with debugging support. You should be able to download it at the development section from http://www.freepascal.org.

Linux and Cygwin:

You may want to learn about using Linux and do not want to install it. The GNU tools are in the core of the Linux facilities, and you can obtain a much larger collection of them from the DJGPP package (see Windows/gcc). A collection of GNU facilities can also be obtained from http://www.cygwin.com. This Cygwin package has even more of the feel of Linux, as they are being used through the bash shell, which is common in Linux systems.

Note that the Cygwin is not a part of the the competition environment.

Windows

We are using Windows XP. We expect support for the hardware to be available in Windows XP. You can get information about Windows from http://www.microsoft.com/windows/.

The windows environment includes vim and emacs as well as notepad.

GCC on Windows:

The GCC compiler version we are using in the windows environment is GCC 2.95.3.

WARNING: If you install Freepascal and GCC (e.g. as in DJGPP) in the same Windows installation, be sure to have DJGPP in your path before Freepascal, or GCC won't work. This seems to be because it finds cpp.exe from the pascal binaries and then thinks that the pascal binary directory is the place for its compiler binaries, which it subsequently fails to find.

For windows, we are using the DJGPP. You can find out about DJGPP and downloading it from http://www.delorie.com/djgpp/.

Our current installation includes the following packages:

- v2/copying.dj - DJGPP Copyright info
- v2/djdev203.zip - DJGPP Basic Development Kit
- v2/faq230b.zip - Frequently Asked Questions
- v2/readme.1st - Installation instructions
- v2gnu/bnu2121b.zip - Basic assembler, linker
- v2gnu/fil41b.zip - GNU fileutils
- v2gnu/gcc2953b.zip - Basic GCC compiler
- v2gnu/gdb511b.zip - GNU debugger
- v2gnu/gpp2953b.zip - C++ compiler
- v2gnu/grep24b.zip - GNU Grep
- v2gnu/lss374b.zip - GNU Less
- v2gnu/mak3791b.zip - Make (processes makefiles)
- v2gnu/txi41b.zip - Info file viewer
- rhide15b-20020625-prerelease.zip - RHIDE snapshot (from http://www.rhide.com)

Pascal on Windows:

We have installed Freepascal 1.0.6. See http://www.freepascal.org for obtaining a copy. If you install the full version dos106full.zip, you just first unzip the file and run install.exe.

You can use Freepascal with its own IDE.

TASK OVERVIEW SHEET / DAY-0

TASK		NUMBER	STRING	RED
Task material directory	Linux	`~/number`	`~/string`	`~/red`
	WinXP	`C:\IOI\number`	`C:\IOI\string`	`C:\IOI\red`
Time limit per test		1 secs	1 secs	-
Memory limit		32 MB	32 MB	-
Compiler options	C and C++	`-O2 -static -lm`	`-O2 -static -lm`	-
	Pascal	`-So -O2 -XS`	`-So -O2 -XS`	-
Number of tests		5	5	4
Maximum points per test		20	20	25
Maximum total points		100	100	100

Program header comment when using C		`/*`
		`TASK: number`
		`LANG: C`
		`*/`
			`/*`
			`TASK: string`
			`LANG: C`
			`*/`
				-

Program header comment when using C++		`/*`
		`TASK: number`
		`LANG: C++`
		`*/`
			`/*`
			`TASK: string`
			`LANG: C++`
			`*/`
				-

Program header comment when using Pascal		`{`
		`TASK: number`
		`LANG: PASCAL`
		`}`
			`{`
			`TASK: string`
			`LANG: PASCAL`
			`}`
				-

Submission is accepted, if;		Example 1 is solved.	Example 1 is solved.	The file format is correct.

TASK OVERVIEW SHEET / DAY-1

TASK		**FROG**	**UTOPIA**	**XOR**
Task material directory	**Linux**	`~/frog`	`~/utopia`	`~/xor`
	WinXP	`C:\IOI\frog`	`C:\IOI\utopia`	`C:\IOI\xor`
Time limit per test		2 secs	2 secs	-
Memory limit		64 MB	32 MB	-
Compiler options	**C and C++**	`-O2 -static -lm`	`-O2 -static -lm`	-
	Pascal	`-So -O2 -XS`	`-So -O2 -XS`	-
Number of tests		25	25	10
Maximum points per test		4	4	10
Maximum total points		100	100	100

Program header comment when using C		`/*`
		`TASK: frog`
		`LANG: C`
		`*/`
			`/*`
			`TASK: utopia`
			`LANG: C`
			`*/`
				-

Program header comment when using C++		`/*`
		`TASK: frog`
		`LANG: C++`
		`*/`
			`/*`
			`TASK: utopia`
			`LANG: C++`
			`*/`
				-

Program header comment when using Pascal		`{`
		`TASK: frog`
		`LANG: PASCAL`
		`}`
			`{`
			`TASK: utopia`
			`LANG: PASCAL`
			`}`
				-

Submission is accepted, if;		Example 1 is solved.	Example 1 is solved.	The file format is correct.

TASK OVERVIEW SHEET / DAY-2

TASK		BATCH	BUS	RODS
Task material directory	Linux	`~/batch`	`~/bus`	`~/rods`
	WinXP	`C:\IOI\batch`	`C:\IOI\bus`	`C:\IOI\rods`
Time limit per test		0.1 secs	4 secs	1 sec
Memory limit		32 MB	32 MB	32 MB
Compiler options	C and C++	`-O2 -static -lm`	`-O2 -static -lm`	`-O2 -static -lm`
	Pascal	`-So -O2 -XS`	`-So -O2 -XS`	`-So -O2 -XS`
Number of tests		20	20	20
Maximum points per test		5	5	5
Maximum total points		100	100	100

Program header comment when using C		`/*`
		`TASK: batch`
		`LANG: C`
		`*/`
			`/*`
			`TASK: bus`
			`LANG: C`
			`*/`
				`/*`
				`TASK: rods`
				`LANG: C`
				`*/`

Program header comment when using C++		`/*`
		`TASK: batch`
		`LANG: C++`
		`*/`
			`/*`
			`TASK: bus`
			`LANG: C++`
			`*/`
				`/*`
				`TASK: rods`
				`LANG: C++`
				`*/`

Program header comment when using Pascal		`{`
		`TASK: batch`
		`LANG: PASCAL`
		`}`
			`{`
			`TASK: bus`
			`LANG: PASCAL`
			`}`
				`{`
				`TASK: rods`
				`LANG: PASCAL`
				`}`

Submission is accepted, if;		Example 1 is solved.	Example 1 is solved.	Example is processed.

Number of Distinct Values

PROBLEM

You are to write a program, which, given N positive integer values X_1, X_2, …, X_N, compute the number of distinct values in those N integer values.

INPUT

Your program reads input from standard input. The first line contains one integer: the number N, $2 \leq N \leq 1000$. The following N lines represent the values X_1, X_2, …, X_N, and each of these lines contains one integer: a value X_i , $1 \leq X_i \leq 10000$.

OUTPUT

Your program writes output to standard output. The output contains one integer: the number of distinct values in X_1, X_2, …, X_N.

EXAMPLE INPUTS AND OUTPUTS

Example 1: input

```
10
1
2
3
4
5
4
3
2
1
2
```

output

```
5
```

Example 2: input

```
5
123
321
123
231
321
```

output

```
3
```

SCORING

If the output is correct, you get full score for the test case. Otherwise the score for the test case is 0.

String from Substrings

PROBLEM

Every DNA string consists of only 4 letters, `A`, `T`, `G` and `C`. You have an unknown DNA string and must determine the string. The only way you can access information about the hidden string is through an oracle. The oracle, when given a query string *S*, answers whether the hidden string contains *S* as a substring. For example, let the hidden string be S_o= "`ATTGCGCGATCG`". Then "`ATTG`" and "`CGCG`", "`T`", "`AT`" are substrings of S_o. But neither of "`TGG`" or "`GCGATG`" is not a substring of S_o. When a string S_o is represented as S_o= $(s[1], s[2], s[3], \ldots, s[N])$, where $s[i]$ is the *i*-th character of S_o, then a substring of S_o is a consecutive subsequence represented as $(s[i], s[i+1], s[i+2], \ldots, s[j])$, where $1 \le i \le j \le N \le 255$.

You are to write a program, which determines the hidden string using as few oracle queries as possible.

LIBRARY

You are given a library in the following.

GNU C/C++ Library: (`oracle.h, oracle.o`)

The C/C++library has the following three functions:

`void start_string()`: The call to start. It should be called only once at the beginning.

`int oracle_call(char *S)`: If `S` is a substring of the hidden string, this function returns 1. Otherwise, this function returns 0. The query string `S` should not be an empty string, and the length of `S` should be equal or less than 255.

`void answer_string(char *S)`: This function will terminate your program. Your program passes the string `S` as an answer. This should be called only once at the end of the program.

Instruction: To compile your string.c or string.cpp, use the include statement

```
 #include "oracle.h"
```

in the source code and compile it as:

```
 gcc -O2 -static string.c oracle.o -lm
 g++ -O2 -static string.cpp oracle.o -lm
```

`lib_test.c` shows how to use the GNU C/C++ library.

FreePascal Library: (`oracle.ppu`, `oracle.o`)

The corresponding pascal library functions are

```
procedure start_string;
function oracle_call(S: string): integer;
procedure answer_string(S: string);
```

Instruction: To compile your `string.pas`, include the import statement

```
 uses oracle;
```

in the source code and compile it as

```
 fpc -So -O2 -XS string.pas
```

`lib_test.pas` shows how to use the FreePascal library.

The libray generates a file named `string.out` automatically in a call to `answer_string`. The file `string.out` has two lines. The integer in the first line shows the number of calls to `oracle_call` made by your program and the second line contains the hidden string your program has given as a solution.

EXAMPLE INPUTS AND OUTPUTS

The length L of the hidden string satisfies $1 \le L \le 255$. You can experiment with the library by creating a text file `string.in` with a single line which contains the hidden DNA string. Note that the `string.out` in the following example is not necessarily optimal. It merely shows the file format of input and output.

`Example1:` `string.in`

```
ATTGCGCGATCG
```

`string.out`

```
41
ATTGCGCGATCG
```

SCORING

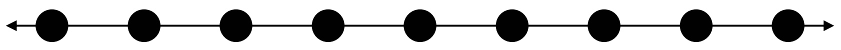
If your program violates one of constraints (e.g. calling too many function calls), then you get 0 point.

If your solution is not correct, then the score is 0. When the output solution is correct, then your score depends on the number of library function calls for each testing data. For each data if the number of function calls is less than a bound B (that is fixed independently for each data), then you get full score. Otherwise you will get 0 points.

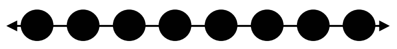
The Troublesome Frog

PROBLEM

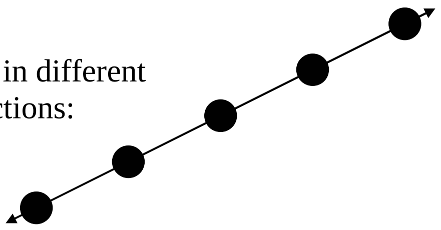
In Korea, the naughtiness of the *cheonggaeguri*, a small frog, is legendary. This is a well-deserved reputation, because the frogs jump through your rice paddy at night, flattening rice plants. In the morning, after noting which plants have been flattened, you want to identify the path of the frog which did the most damage. A frog always jumps through the paddy in a straight line, with every hop the same length:



Different frogs can jump with different hop lengths:



And in different directions:



Your rice paddy has plants arranged on the intersection points of a grid as shown in Figure-1, and the troublesome frogs hop completely through your paddy, starting outside the paddy on one side and ending outside the paddy on the other side as shown in Figure-2:

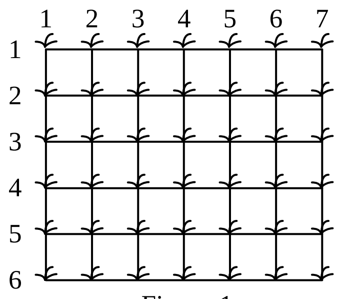


Figure-1

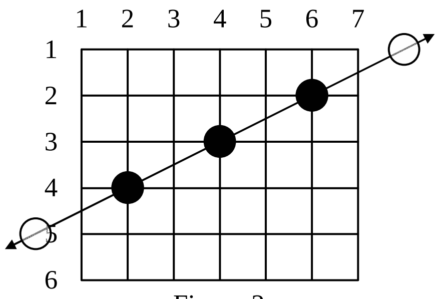


Figure-2

Many frogs can jump through the paddy, hopping from rice plant to rice plant. Every hop lands on a plant and flattens it, as in Figure-3. Note that some plants may be landed on by more than one frog during the night. Of course, you can not see the lines showing the paths of the frogs or any of their hops outside of your paddy – for the situation in Figure-3, what you can see is shown in Figure-4:

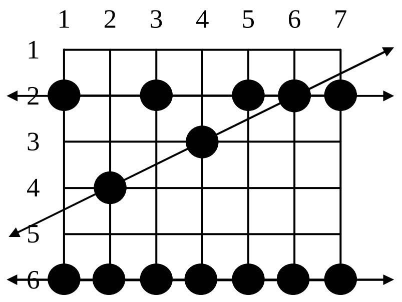


Figure-3

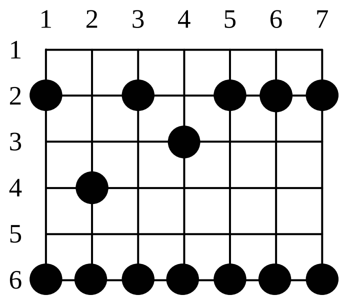


Figure-4

From Figure-4, you can reconstruct all the possible paths which the frogs may have followed across your paddy. You are only interested in frogs which have landed on at least 3 of your rice plants in their voyage through the paddy. Such a path is said to be a frog path. In this case, that means that the three paths shown in Figure-3 are frog paths (there are also other possible frog paths). The vertical path down column 1 might have been a frog path with hop length 4 except there are only 2 plants flattened so we are not interested; and the diagonal path including the plants on row 2 col. 3, row 3 col. 4, and row 6 col. 7 has three flat plants but there is no regular hop length which could have spaced the hops in this way while still landing on at least 3 plants, and hence it is not a frog path. Note also that along the line a frog path follows there may be additional flattened plants which do not need to be landed on by that path (see the plant at (2, 6) on the horizontal path across row 2 in Figure-4), and in fact some flattened plants may not be explained by any frog path at all.

Your task is to write a program to determine the maximum number of landings in any single frog path (where the maximum is taken over all possible frog paths). In Figure-4 the answer is 7, obtained from the frog path across row 6.

INPUT

Your program is to read from standard input. The first line contains two integers *R* and *C*, respectively the number of rows and columns in your rice paddy, $1 \le R,C \le 5000$. The second line contains the single integer *N*, the number of flattened rice plants, $3 \le N \le 5000$. Each of the remaining *N* lines contains two integers, the row number (1≤ *row number* ≤ *R*) and the column number (1≤ *column number* ≤ *C*) of a flattened rice plant, separated by one blank. Each flattened plant is only listed once.

OUTPUT

Your program is to write to standard output. The output contains one line with a single integer, the number of plants flattened along a frog path which did the most damage if there exists at least one frog path, otherwise, 0.

EXAMPLE INPUTS AND OUTPUTS

`Example 1:` `input` `output`

(the example of Figure-4)

```
6 7
14
2 1
6 6
4 2
2 5
2 6
2 7
3 4
6 1
6 2
2 3
6 3
6 4
6 5
6 7
```

```
7
```

`Example 2:` `input` (the example of Figure-5)

```
6 7
18
1 1
6 2
3 5
1 5
4 7
1 2
1 4
1 6
1 7
2 1
2 3
2 6
4 2
4 4
4 5
5 4
5 5
6 6
```

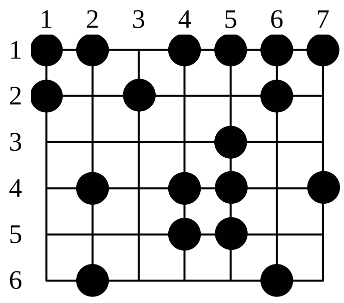


Figure-5

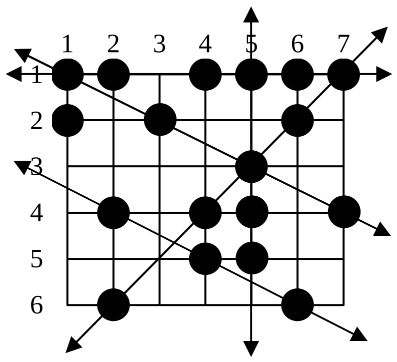


Figure-6: The maximum number of plants flattened by a frog is 4.

`output`

```
4
```

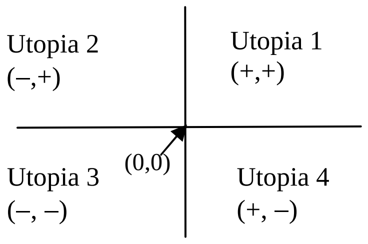
SCORING

If your program outputs the correct answer for a test case within the time limit, then you get full points for the test case, and otherwise you get 0 points.

Utopia Divided

PROBLEM

The beautiful land of Utopia was once ravaged by war. When the hostilities subsided the country was divided into four regions by a longitude (north-south line) and a latitude (east-west line). The intersection of these lines became known as the point (0,0). All four parts claimed the name Utopia, but as time went by they generally became known as Utopia 1 (northeast), 2 (northwest), 3 (southwest) and 4 (southeast). A point in any of the regions was identified by its distance east and its distance north of (0,0). These distances could be negative; hence a point in Utopia 2 was designated by a (negative, positive) pair, in Utopia 3 by a (negative, negative) pair, in Utopia 4 by (positive, negative) and in Utopia 1 by a pair of positive numbers.



A major problem was that citizens were not permitted to cross borders. Fortunately, some ingenious IOI contestants from Utopia developed a safe means of teleportation. The machine requires code numbers, each of which can only be used once. Now the challenge facing the team, and you, is to guide the teleporter from its initial position of (0,0) to the regions of Utopia in the order requested. You don't care where in a region you land, but you will have a sequence of *N* region numbers that specify the regions in which the teleporter is to land. You may be asked to land in the same region in two or more consecutive stops. After leaving the initial (0,0) point, you must never land on a border.

You will receive as input a sequence of 2*N* code numbers and are to write them as a sequence of *N* code pairs, placing a plus or a minus sign before each number. If you are currently at the point (*x*,*y*) and use the code pair (+*u*,–*v*), you will be teleported to the point (*x*+*u*, *y*–*v*). You have the 2*N* numbers, and you can use them in any order you like, each with a plus or a minus sign.

Suppose you have code numbers 7, 5, 6, 1, 3, 2, 4, 8 and are to guide the teleporter according to the sequence of region numbers 4, 1, 2 ,1. The sequence of code pairs (+7,–1), (–5,+2), (–4,+3), (+8,+6) achieves this as it teleports you from (0,0) to the locations (7,–1), (2,1), (–2,4) and (6,10) in that order. These points are located in Utopia 4, Utopia 1, Utopia 2, and Utopia 1, respectively.

TASK

You are given 2*N* distinct code numbers and a sequence of *N* region numbers indicating where the teleporter is to land. Construct a sequence of code pairs from the given numbers that guide the teleporter to go through the given region sequence.

INPUT

Your program is to read from standard input. The first line contains a positive integer N ($1 \leq N \leq 10000$). The second line contains the $2N$ distinct integer code numbers ($1\leq$ *code number* ≤ 100000) separated by single spaces. The last line contains a sequence of N region numbers, each of which is 1, 2, 3 or 4.

OUTPUT

Your program is to write to standard output. The output consists of N lines, each containing a pair of code numbers each preceded by a sign character. These are codes pairs that will direct the teleporter to the given region sequence. Note that there must be no blank following a sign, but there must be a single space after the first code number.

If there are several solutions your program can output any one of them. If there are no solutions your program should output the single integer 0.

EXAMPLE INPUTS AND OUTPUTS

`Example 1: input` `output`

```
4
7 5 6 1 3 2 4 8
4 1 2 1
```

```
+7 -1
-5 +2
-4 +3
+8 +6
```

`Example 2: input` `output`

```
4
2 5 4 1 7 8 6 3
4 2 2 1
```

```
+3 -2
-4 +5
-6 +1
+8 +7
```

SCORING

If your program outputs a correct answer for a test case within the time limit, then you get full points for that test, and otherwise you get 0 points for the test case.

XOR

PROBLEM

You are implementing an application for a mobile phone, which has a black-and-white screen. The *x*-coordinates of the screen start from the left and the *y*-coordinates from the top, as shown in the figures. For the application, you need various images, which are not all of the same size. Instead of storing the images, you want to create the images using the phone's graphics library. You may assume that at the start of drawing an image, all pixels of the screen are white. The only graphics operation in the phone's library is `XOR(L,R,T,B)`, which will reverse the pixel values in the rectangle with top left coordinate (`L`,`T`) and bottom right coordinate (`R`,`B`), where `L` stands for the left, `T` for the top, `R` for the right and `B` for the bottom coordinate. Note that in some other graphics libraries the order of the arguments is different.

As an example, consider the image in Figure-3. Applying `XOR(2,4,2,6)` to an all white image gives the image in Figure-1. Applying `XOR(3,6,4,7)` to the image of Figure-1 gives the image in Figure-2, and applying `XOR(1,3,3,5)` to the image in Figure-2 finally gives the image in Figure-3.

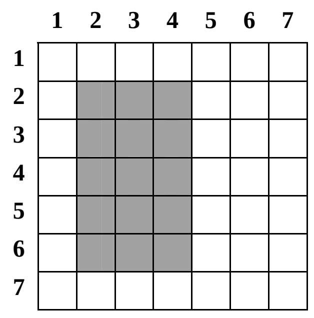


Figure-1

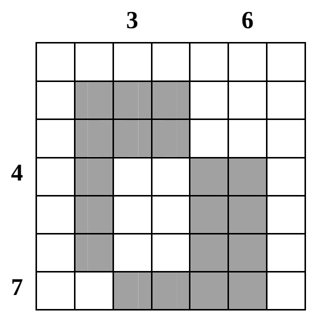


Figure-2

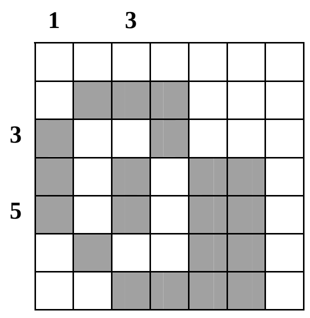


Figure-3

Given a set of black-and-white pictures, your task is to generate each picture from an initially white screen using as few `XOR` calls as you can. You are given the input files describing the images, and you are to submit files including the required `XOR` call parameters, not a program to create these files.

INPUT

You are given 10 problem instances in the text files named `xor1.in` to `xor10.in`. Each input file is organized as follows. The first line of an input file contains one integer *N*, $5 \le N \le 2000$, meaning that there are *N* rows and *N* columns in the image. The remaining lines represent the rows of the image from top to bottom. Each line contains *N* integers: the pixel values in the row from left to right. Each of these integers is either a 0 or a 1, where 0 represents a white pixel and 1 represents a black pixel.

OUTPUT

You are to submit 10 output files corresponding to the given input files.

The first line contains the text
`#FILE xor I`
where integer I is the number of the respective input file. The second line contains an integer *K*: the number of `XOR` calls specified in the file. The following *K* lines represent these calls from the first call to the last call to be executed. Each of these *K* lines contains four integers: the `XOR` call parameters `L`, `R`, `T`, `B` in that order.

EXAMPLE INPUT AND OUTPUT

Example: xor0.in

```
7
0 0 0 0 0 0 0
0 1 1 1 0 0 0
1 0 0 1 0 0 0
1 0 1 0 1 1 0
1 0 1 0 1 1 0
0 1 0 0 1 1 0
0 0 1 1 1 1 0
```

xor0.out

```
#FILE xor 0
3
2 4 2 6
3 6 4 7
1 3 3 5
```

SCORING

If
- the `XOR` calls specified in your output file do not generate the required image, or
- the number of `XOR` calls specified in your output file is not *K*, or
- in your output file, K > 40000, or
- your output file contains such an `XOR` call that L>R or T>B, or
- your output file contains an `XOR` call which does not have positive coordinates, or
- your output file contains an `XOR` call with a coordinate value exceeding *N*,

then your score is zero. Otherwise, your score is
1+9×CallsInBestAnswerOfAllContestants/CallsInYourAnswer.

The score is rounded to the first decimal place for each case. The total score is rounded off to the nearest integer.

Suppose that you submit a solution with 121 `XOR` calls. If that is the best submission of all contestants, your score is 10. If the best of the submitted solution of all contestants uses 98 `XOR` calls, your score becomes 1+9×98/121(=8.289…), which will be rounded to 8.3.

Batch Scheduling

PROBLEM

There is a sequence of N jobs to be processed on one machine. The jobs are numbered from 1 to N, so that the sequence is 1,2, …, N. The sequence of jobs must be partitioned into one or more batches, where each batch consists of consecutive jobs in the sequence. The processing starts at time 0. The batches are handled one by one starting from the first batch as follows. If a batch b contains jobs with smaller numbers than batch c, then batch b is handled before batch c. The jobs in a batch are processed successively on the machine. Immediately after all the jobs in a batch are processed, the machine outputs the results of all the jobs in that batch. The output time of a job j is the time when the batch containing j finishes.

A setup time S is needed to set up the machine for each batch. For each job i, we know its cost factor F_i and the time T_i required to process it. If a batch contains the jobs x, x+1, … , x+k, and starts at time t, then the output time of every job in that batch is $t + S + (T_x + T_{x+1} + \ldots + T_{x+k})$. Note that the machine outputs the results of all jobs in a batch at the same time. If the output time of job i is O_i, its cost is $O_i \times F_i$. For example, assume that there are 5 jobs, the setup time $S = 1$, $(T_1, T_2, T_3, T_4, T_5) = (1, 3, 4, 2, 1)$, and $(F_1, F_2, F_3, F_4, F_5) = (3, 2, 3, 3, 4)$. If the jobs are partitioned into three batches {1, 2}, {3}, {4, 5}, then the output times $(O_1, O_2, O_3, O_4, O_5) = (5, 5, 10, 14, 14)$ and the costs of the jobs are (15, 10, 30, 42, 56), respectively. The total cost for a partitioning is the sum of the costs of all jobs. The total cost for the example partitioning above is 153.

You are to write a program which, given the batch setup time and a sequence of jobs with their processing times and cost factors, computes the minimum possible total cost.

INPUT

Your program reads from standard input. The first line contains the number of jobs N, $1 \leq N \leq 10000$. The second line contains the batch setup time S which is an integer, $0 \leq S \leq 50$. The following N lines contain information about the jobs 1, 2, …, N in that order as follows. First on each of these lines is an integer T_i, $1 \leq T_i \leq 100$, the processing time of the job. Following that, there is an integer F_i, $1 \leq F_i \leq 100$, the cost factor of the job.

OUTPUT

Your program writes to standard output. The output contains one line, which contains one integer: the minimum possible total cost.

EXAMPLE INPUTS AND OUTPUTS

`Example 1: input` `output`

```
2
50
100 100
100 100
```

```
45000
```

`Example 2: input` `output`

```
5
1
1 3
3 2
4 3
2 3
1 4
```

```
153
```

Example 2 is the example in the text.

REMARK

For each test case, the total cost for any partitioning does not exceed $2^{31} – 1$.

SCORING

If your program outputs the correct answer for a test case within the time limit, then you get full points for the test case, and otherwise you get 0 points.

Bus Terminals

PROBLEM

Yong-In city plans to build a bus network with N bus stops. Each bus stop is at a street corner. Yong-In is a modern city, so its map is a grid of square blocks of equal size. Two of these bus stops are to be selected as hubs H_1 and H_2. The hubs will be connected to each other by a direct bus line and each of the remaining $N-2$ bus stops will be connected directly to either H_1 or H_2 (but not to both), but not to any other bus stop.

The distance between any two bus stops is the length of the shortest possible route following the streets. That is, if a bus stop is represented as (x, y) with x-coordinate x and y-coordinate y, then the distance between two bus stops (x_1, y_1) and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$. If bus stops A and B are connected to the same hub H_1, then the length of the route from A to B is the sum of the distances from A to H_1 and from H_1 to B. If bus stops A and B are connected to different hubs, e.g., A to H_1 and B to H_2, then the length of the route from A to B is the sum of the distances from A to H_1, from H_1 to H_2, and from H_2 to B.

The planning authority of Yong-In city would like to make sure that every citizen can reach every point within the city as quickly as possible. Therefore, city planners want to choose two bus stops to be hubs in such a way that in the resulting bus network the length of the longest route between any two bus stops is as short as possible.

One choice P of two hubs and assignments of bus stops to those hubs is better than another choice Q if the length of the longest bus route is shorter in P than in Q. Your task is to write a program to compute the length of this longest route for the best choice P.

INPUT

Your program is to read from standard input. The first line contains one positive integer N, $2 \le N \le 500$, the number of bus stops. Each of the remaining N lines contains the x-coordinate followed by the y-coordinate of a bus stop. The x- and y-coordinates are positive integers ≤ 5000. No two bus stops are at the same location.

OUTPUT

Your program is to write to standard output. The output contains one line with a single positive integer, the minimum length of the longest bus route for the input.

EXAMPLE INPUTS AND OUTPUTS

Example 1: input

```
6
1 7
16 6
12 4
4 4
1 1
11 1
```

output

```
20
```

Example 2: input

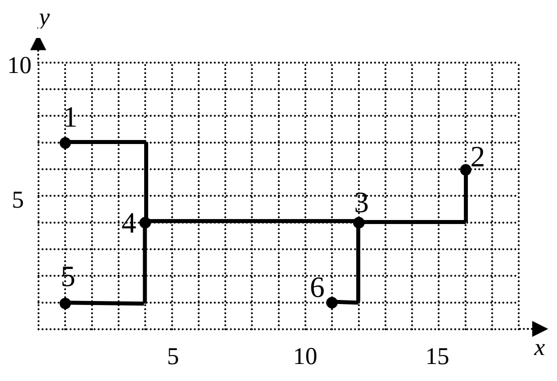
```
7
7 9
10 9
5 3
1 1
7 2
15 6
17 7
```

output

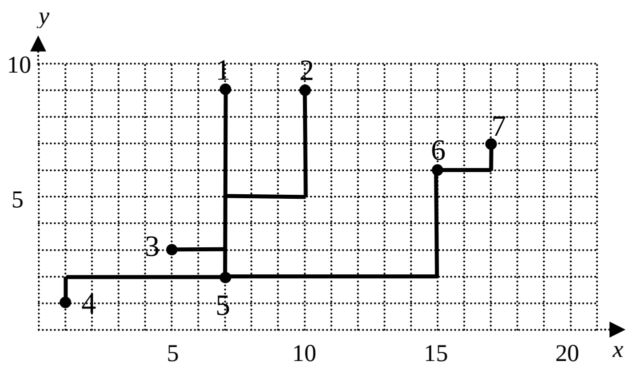
```
25
```

The following figures show the bus networks for the inputs given above. If in Example 1 bus stops 3 and 4 are selected as hubs then the longest route is either between bus stops 2 and 5 or between bus stops 2 and 1. There is no better choice for the hubs, and the answer is 20.

For the bus network in Example 2, if bus stops 5 and 6 are selected as hubs then the longest route is obtained between bus stops 2 and 7. There is no better choice for the hubs, and the answer is 25.



Bus network for Example 1



Bus network for Example 2

SCORING

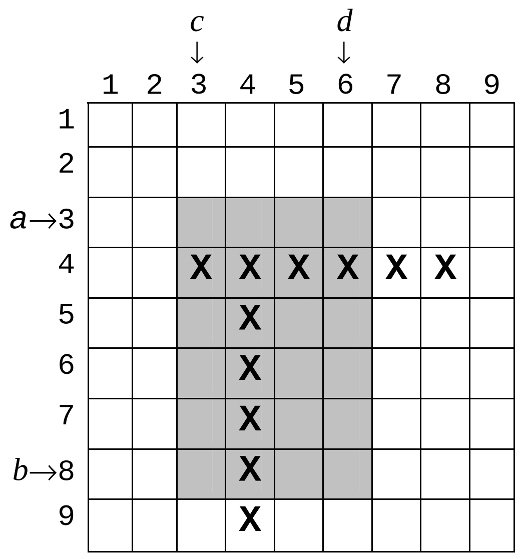
If your program outputs the correct answer for a test case within the time limit, then you get full points for that test case, and otherwise you get 0 points for that case.

Two Rods

PROBLEM

A rod is either a horizontal or a vertical sequence of at least 2 consecutive grid cells. Two rods, one horizontal and the other vertical, are placed on an N by N grid. In Figure-1, the two rods are shown by `X`'s. The rods may or may not be the same length; furthermore, they may share a cell. If, from a diagram such as Figure-1, it is possible to interpret a cell, e.g. (4,4), as being in just one rod or in both rods, we make the interpretation that the cell is in both. Hence, the top cell of the vertical rod is (4,4) rather than (5,4).

Figure-1



Initially we do not know where the two rods are, and so your task is to write a program to determine their locations. We call the horizontal rod ROD1, and the vertical rod ROD2. Each grid cell is represented by a row/column pair (r,c), and the top left corner of the grid is taken to be location (1,1). Each rod is represented as two cells, <(r_1, c_1), (r_2, c_2)>. In Figure-1 ROD1 is <(4,3), (4,8)> and ROD2 is <(4,4), (9,4)>.

This task involves the use of library functions for input, for determining the solution, and for output. The length of a side of the square grid is given by the library function `gridsize`, which your program is to call at the beginning of each test case. To locate the rods, you can only use the library function `rect(a, b, c, d)`, which examines the rectangular region $[a,b]\times[c,d]$ (shaded region in Figure-1), where $a \le b$ and $c \le d$. [Note carefully the order of these parameters.] If at least one grid cell of either rod falls inside the query rectangle $[a,b]\times[c,d]$, `rect` returns 1; otherwise it returns 0. So in the example, `rect(3,8,3,6)`returns `1`. Your task is to write a program to discover the exact location of the rods using a limited number of `rect` calls.

You produce output by calling another library function `report(`r_1, c_1, r_2, c_2, p_1, q_1, p_2, q_2`)` where ROD1 is <(r_1, c_1),(r_2, c_2)> and ROD2 is <(p_1, q_1),(p_2, q_2)>. Calling `report` terminates your program. Recall that ROD1 is horizontal and ROD2 is vertical, and (r_1,

c_1) is the left end cell of the horizontal rod ROD1. Cell (p_1, q_1) is the top end cell of ROD2. Hence $r_1 = r_2$, $c_1 < c_2$, $p_1 < p_2$, and $q_1 = q_2$. If your `report` parameters do not meet these constraints, then you will get error messages on standard output.

CONSTRAINTS

- You can access input only by using the library functions `gridsize` and `rect`.
- N, the maximum row (column) size of input, satisfies $5 \le N \le 10000$.
- The number of `rect` calls should be at most 400 for every test case. If your program calls `rect` more than 400 times, this will terminate your program.
- Your program must call `rect` more than once and call `report` exactly once.
- If a `rect` call is not valid (e.g., the query range exceeds the grid space), it will terminate your program.
- Your program must not read or write any files and must not use any standard input/output.

LIBRARY

You are given a library in the following:

FreePascal Library (`prectlib.ppu`, `prectlib.o`)

```
function gridsize: LongInt;
function rect(a,b,c,d : LongInt) : LongInt;
procedure report(r1, c1, r2, c2, p1, q1, p2, q2 : LongInt);
```

Instructions: To compile your `rods.pas`, include the import statement
```
 uses prectlib;
```
in the source code and compile it as
```
 fpc -So -O2 -XS rods.pas
```
The program `prodstool.pas` gives an example of using this FreePascal library.

GNU C/C++ Library (`crectlib.h, crectlib.o`)

```
int gridsize();
int rect(int a, int b, int c, int d);
void report(int r1, int c1, int r2, int c2, int p1, int q1,
             int p2, int q2);
```

Instructions: To compile your rods.c, use
```
 #include "crectlib.h"
```
in the source code and compile it as:
```
 gcc -O2 -static rods.c crectlib.o -lm
 g++ -O2 -static rods.cpp crectlib.o -lm
```
The program `crodstool.c` gives an example of using this GNU C/C++ library.

For C/C++ in the RHIDE environment

Be sure that you set the Option->Linker configuration to `crectlib.o` .

EXPERIMENTATION

To experiment with the library, you must create a text file `rods.in.` The file must contain three lines. The first line contains one integer: *N*, the size of the grid. The second line contains the coordinates of ROD1, r_1 c_1 r_2 c_2; where (r_1, c_1) is the left end cell of ROD1. The third line contains the coordinates of ROD2, p_1 q_1 p_2 q_2, where (p_1, q_1) is the top end cell of ROD2.

After running your program which calls `report`, you will get the output file `rods.out`. This file contains the number of `rect` function calls and the coordinates of the ends of the rods you submitted in your call to `report`. If there are any errors or violations of the requirements during library calls, then `rods.out` will contain the corresponding error messages.

The dialogue between your program and the library is recorded in the file `rods.log`. This log file `rods.log` shows the sequence of function calls your program made in the form of "`k : rect(a,b,c,d) = ans`", which means *k*-th function call `rect(a,b,c,d)` returns *ans*.

EXAMPLE INPUT AND OUTPUT

Example: rods.in

```
9
4 3 4 8
4 4 9 4
```

rods.out

```
20
4 3 4 8
4 4 9 4
```

SCORING

If your program violates any of the constraints (e.g., more than 400 `rect` calls), or if your program's output (the locations of the rods) is not correct, the score is 0.

If your program's output is correct, then your score depends on the number of `rect` calls for each testing data. For each test case if the number of `rect` calls is at most 100, then you get 5 points. If your program calls `rect` 101 to 200 times, you get 3 points. If the number of `rect` calls is between 201 and 400, then you get 1 point.