FOREWORD

The materials in this booklet were used at IOI2002 held August 18-25, 2002 in Yong-In, Korea. The IOI venue was the central library building at Kyung Hee University. More than 270 contestants from 77 countries took part in the competition.

In Korea, the national programming contest for secondary school students was first held in 1984. Korea first participated as an Observer at the 3rd IOI held in Greece. This year we confidently hosted IOI2002 in Korea.

The Scientific Committee of IOI2002 consisted of three subcommittees: Task, Evaluation and Technical. Creating an excellent set of competition tasks is the key to the success of IOI while ensuring continuity. Computing environments remain almost the same as in IOI2001. A new grading system was designed and implemented to fit the IOI2002 competition rules.

A new evaluation policy for an output-only task was used. In this evaluation policy, the score of each contestant's solution depends on the best solution from among all contestants' submissions. This policy allows for the possibility of partial credit, and the output-only nature of the tasks also avoids the problem of imprecision and other errors in timing program execution.

We hope that this booklet helps in preparing future IOI competitions.

Kyung-Yong Chwa,
Chair of Scientific Committee of IOI2002

Yong-In city, August 18, 2002

TABLE OF CONTENTS

INTRODUCTION

First we will explain the procedure used by the ISC (International Scientific Committee) and the HSC (Host Scientific Committee) to prepare and select the competition tasks. Then we will give an overview of the tasks, including solutions for and the basic theory behind them.

After the "Call for Tasks" announcement our HSC received 20 problems from Korea and abroad. Of these, we excluded 4 problems which were not mature enough to be used in IOI-style competition tasks. The HSC then prepared all solutions and test data sets (20 test cases for each task) and discussed these sets with the ISC members in July. The meetings held concerning IOI2002 task preparation were follows:

Call for Tasks announcement (Oct. 2001)
1st HSC meeting at KAIST (Jan. 4)
- discussed computer specification
- discussed Competition Rules

2nd HSC meeting in Pusan (Feb. 4-6)
- discussed submitted tasks
- selected 17 candidate tasks

International Committee Meeting in Seoul (Feb. 17)
- reported the current state of the tasks to IC

3rd HSC meeting at KAIST (Mar. 29-30)
- reviewed the first draft of candidate tasks
- programmed solutions

4th HSC meeting at KAIST (Apr. 26-28)
- reviewed solution programs and test data
- tested solution programs for timing

5th HSC meeting at KAIST (May 4)
ISC meeting at KAIST (May 11-18)
- HSC and ISC reviewed and discussed the first draft of tasks
- Prof. Kunsoo Park explained our evaluation system and discussed with ISC
- ISC selected 7 competition tasks out of 16 tasks
- ISC improved the presentation style of 7 selected tasks.

6th HSC meeting at KAIST (Jul. 3-5)
7th HSC meeting at KAIST (Jul. 22-23)
- demonstrated task evaluation system
- loaded and tested solution programs and task library on the evaluation server

Evaluation system was tested on-line and off-line (Jul. 25)
Internet Practice Session Opened (Aug. 6)
8th HSC meeting at KAIST (Aug. 7)
- finalized tasks and solutions

TASK SUMMARY

TASK NAME	DECISION by ISC and HSC	TYPE	FINAL DECISION
BATCH	Accept	Typical	DAY2, TASK1
BRIDGE	Reject	Typical, Heuristics	
BUS	Accept	Typical	DAY2, TASK2
DIAMOND	Accept	Typical	Backup Task
FROG	Accept	Typical	DAY1, TASK1
GENE	Reject	Typical	
INTERNET	Reject	Typical	
NETWORK	Practice	Typical	
PICTURE	Reject	Reactive, Heuristics	
ROBOT	Reject	New kind	
RODS	Accept	Reactive	DAY2, TASK3
SEED	Reject	Reactive	
STRING	Practice	Reactive	Practice Task
UTOPIA	Accept	Typical	DAY1, TASK2
XOR	Accept	Output only	DAY1, TASK3

IOI2002 used a new evaluation method, namely relative scoring (initially called tournament scoring). In this scoring policy, the number of points awarded to each competitor's solution depends on the best solution submitted by any competitor: a competitor whose answer on a test case is as good as the best submission gets full marks, and other students get partial credit according to how close to that answer their submission was. XOR was chosen to be scored in this manner, and it was also decided that it be phrased as an "output-only" task. This means that all (10) test data input sets are given to contestants at the beginning of competition, and instead of submitting source code as in a typical IOI task, contestants to submit only the output files corresponding to the given data sets. Grading using relative scoring is performed in two steps: first, we evaluate the absolute performance of each solution, and second we compare the performance of contestant's solution to the performance of others. The final score given for each test case is the relative ratio of "Contestant's result" / "Best result among submitted solutions."

It is worth noting that the ISC and HSC weighed the fact that ROBOT is of a type of task new to the IOI. However, ROBOT was in the end rejected since IOI2002 did not seem to be the right moment to introduce it in view of the other changes which were considered more immediately workable. We include that task in this IOI2002 book with one solution, and we hope this innovative type of task will be accepted in future IOIs to broaden the IOI task repertoire even further.

One difficulty in preparing tasks and solutions is the difference between C/C++ and Pascal; much effort has been put into examining and understanding the differences and their impact on an IOI. Generally, we find C/C++ to be faster than Pascal, on one occasion up to 4 times faster for equivalent code. Of course, this depends on the programming style used in both languages. However, the relative performance also varies by task; we were surprised to find that in some of the backup tasks, our optimized Pascal programs ran faster than our optimized C/C++ programs.

We found little performance difference between Linux and Windows XP, although developing two different contest environments for the two operating systems while trying to keep them as identical as possible proved to be extremely difficult. We hope this two-OS problem will be eliminated in future IOIs.

We hope the material presented here will be helpful to the IOI2003 Scientific Committee and will be instructive to IOI2002 leaders and contestants.

DAY0 (Practice Session)

Practice Task 1: NUMBER

Hwan Gue Cho

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, X2, …, XN.

EXAMPLE INPUTS AND OUTPUTS

`Example 1: input` `output`

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

```
5
```

`Example 2: input` `output`

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

```
3
```

SCORING

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

Practice Task 2: STRING

Sung Kwon Kim

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 So= "`ATTGCGCGATCG`". Then "`ATTG`" and "`CGCG`", "`T`", "`AT`" are substrings of So. But neither of "`TGG`" or "`GCGATG`" is not a substring of So. When a string So is represented as So= (s[1], s[2], s[3], …, s[N]), where s[i] is the i-th character of So, then a substring of So is a consecutive subsequence represented as (s[i], s[i+1], s[i+2], …, 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 library 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 $1 \leq L \leq 255$. 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.

A. Solutions

(a) simple solution

A simple solution is to try each of 4^k strings from k=1, 2, …. Stop when there is no string of length k+1 with answer "yes." The string of length *k* with a "yes" is the hidden string. In case of English strings, replace 4 by 26.

(b) Suggested solution

4(N+2) oracle calls are sufficient. (26(N+2) in case of English strings) Assume that a string `s` is known to be a substring of the hidden string. Append each character to s to have four strings, `sA`, `sC`, `sG`, and `sT`. Call the oracle with each of the four strings. If any of the four calls gets an answer "yes," them we have succeeded in incrementing the length by one. We may repeat. If every one gets an answer "no," then we can say that s is a suffix of the hidden string. We prefix each character to `s`, having `As`, `Cs`, `Gs`, `Ts`. Again call the oracle with each of these four strings. If any one is answered "yes," the length of the string known to be a substring of the hidden string can be incremented by one. If none is answered "yes," then s itself is the hidden string. Initially, `s=A`, or `C`, or `G`, or `T`. Four calls to the oracle to determine the initial single character `s`.

Four calls to determine each character of the hidden string. Four calls to determine the end of the string, and another four to determine the start.

(c) A more complicated solution

A more sophisticate method with $3N+\Theta(\text{sqrt}(N))$ is possible. See the reference [1].

B. Test Data Information

Test data sequences can be obtained from a "real" E.coli or C.elegance whose whole DNA sequence (multi mega bytes) can be downloaded via several bioinformatics related web sites.

C. Background

This is an important problem in computational molecular biology, especially in genome sequencing. Genome sequencing is to read the DNA sequences of an organism, such as human, mouse, and worm. One of the methods used in sequencing a (unknown) DNA sequence is to use DNA chips. An experiment with DNA chips tell us which substrings in the chips belong to the (unknown) DNA sequence and which substrings do not. Constructing the whole sequence by collecting these information is a place where this problem might apply.

D. History

This problem is a variant of the so-called 'superstring' problem, in which, given a set of strings, we want to find a shortest string that contains the strings in the set as substrings. This problem is known to be NP-complete.

E. References

[1] S. Skiena, **Reconstructing strings from substrings**, *Journal of Computational Biology* 2 (1995) 333-353.

Practice Task 3: RED

Chong-Dae Park

Red Devil

PROBLEM

Red Devil, the Korean national soccer team supporters club will have a country-wide tour to *N* cities to celebrate the achievement in 2002 FIFA World Cup Korea/Japan.

Cities are represented by numbers 1, 2, …, *N*. The Red Devil's tour starts with city 1 and visits all *N* cities exactly once, after which it should return to city 1, the starting position. The travel distance between two cities *I* and *J*, denoted by $d(I,J)$ is known.

Note that the distance from city *I* to city *J* is symmetric to the distance from city *J* to city *I*, that is, $d(I,J) = d(J,I)$. For any three distinct cities *I*, *J*, and *K*, it holds that $d(I,K) \leq d(I,J) + d(J,K)$. Furthermore it holds that $d(I,I)=0$ for any city *I*.

Given the distance between cities, you are to find a Red Devil's tour with the shortest possible length. You are given the input files describing the distances. You must submit files describing the tours, not a program to find the tours.

INPUT

You are given 4 problem instances in the text files named `red1.in` to `red4.in`. Each input file is organized as follows. The first line contains one integer: the number of cities, *N*, $5 \le N \le 50$. The following *N* lines represent the distance $d(I,J)$, where for each $d(I,J)$ we have $0 \le d(I,J) \le 50$. These *N* lines are organized in such a way that the *K*th of these *N* lines contains *N* integers: the distance $d(K,1)$, $d(K,2)$, …, $d(K,N)$. This way, the input is organized in the following form:

N
$d(1,1)$ $d(1,2)$ … $d(1,N)$
$d(2,1)$ $d(2,2)$ … $d(2,N)$
…
$d(N,1)$ $d(N,2)$ … $\mathrm{d}(N,N)$

OUTPUT

You are to submit 4 output files corresponding to the given input files. You do not need to submit your solution program source.

The first line contains the text
`#FILE red I`
where integer I is the number of the respective input file. The second line contains *N*+1 integers, which represent the cities in the order in which they are visited in the tour of your solution.

EXAMPLE INPUTS AND OUTPUTS

`Example1: red0.in` `red0.out`

```
5
0 2 5 9 5
2 0 3 7 5
5 3 0 4 6
9 7 4 0 4
5 5 6 4 0
```

```
#FILE red 0
1 3 2 5 4 1
```

SCORING

If the output is not a valid tour, your score is zero. Otherwise, your score is 5 + 20 × DistanceInBestAnswer / DistanceInYourAnswer.

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

Suppose that you submit the tour "1→3→2→5→4→1". The length of your tour is 26. If the best of submitted solutions is a tour "1→2→3→4→5→1", whose length is 18, your score becomes 5+20×18/26(= 18.846…), which will be rounded off to 18.8.

A. Solution

This is a famous Traveling Sales Person (TSP) problem, which is one of typical NP-hard problem. There are lots of heuristics for this problem. It is easy to find useful references and books.

ftp://ftp.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsp/index.html gives traveling sales person problem instances and solutions. http://www.math.princeton.edu/tsp/ is one of TSP Home Page. According to this site, Mathematical problems related to the traveling salesman problem were treated in the 1800s by the Irish mathematician Sir William Rowan Hamilton and by the British mathematician Thomas Penyngton Kirkman. A nice discussion of the early work of Hamilton and Kirkman can be found in the book Graph Theory 1736-1936 by N. L. Biggs, E. K. Lloyd, and R. J. Wilson, Clarendon Press, Oxford, 1976. The general form of the TSP appears to be have been first studied by mathematicians starting in the 1930s by Karl Menger in Vienna and Harvard. A detailed treatment of the connection between Menger and Whitney, and the growth of the TSP as a topic of study can be found in Alexander Schrijver's paper "On the history of combinatorial optimization (till 1960)".

Most recent result on this problem is work done by D. Applegate, R. Bixby, V. Chvátal, and W. Cook who solved 15,112 cities instance D15112 (this is one of the larger TSP instances in TSPLIB. It contains 15,112 cities in Germany (D = Deutschland). The data set was contributed to TSPLIB by Andre Rohe.

See http://www.math.princeton.edu/tsp/d15112/d15112_info.html for D15112.

B. Relative Scoring

Other issue in RED is the *relative scoring policy*, which was first proposed and implemented in IOI 2002. Relative scoring gives scores according to the relative quality of the solution. This is an open-ended competition, which means better solution will receive more scores and any valid answer receives some points. It is suitable for NP-hard class or real world problems. Score will be given by a function of submitted solution (A) and the best solution (BEST) Scoring function: *base* + *proportional* × (BEST/A). For example, Task RED if the submitted tour length is 26 and the shortest submitted tour length is 18, score = 5 + 20 × (18/26) = 18.846…, rounded to 18.8. So best solution earns full score: 5 + 20 × (18/18) = 25. If the best solution is changed by an appeal, then grading will be performed again. This might affect the others score.

Evaluation Results of Internet Practice Session

We have announced three practice tasks to all contestants to help them understand the procedure of submission, evaluation system and task styles for IOI2002. On-line practice session was conducted by Internet in August 6. We received several solutions and output files (for RED) and evaluated those materials of contestants.

Task name	Number of submissions	Number of Full score solutions	Average score of contestants
NUMBER	91	82	94.07
STRING	69	56	86.38
RED	85	8	82.90

DAY1

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.

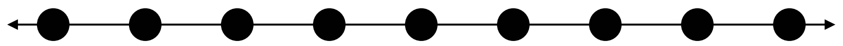
Task 1: FROG

Soo Hwan Kim, Greg Galperin

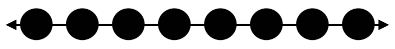
The Troublesome Frog

PROBLEM

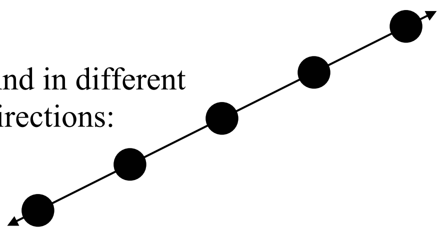
In Korea, the naughtiness of the *cheonggaeguri*, a small frog, is legendary. This is a well-deserved reputation, because the frogs jump through your rice paddy at night, flattening rice plants. In the morning, after noting which plants have been flattened, you want to identify the path of the frog which 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:

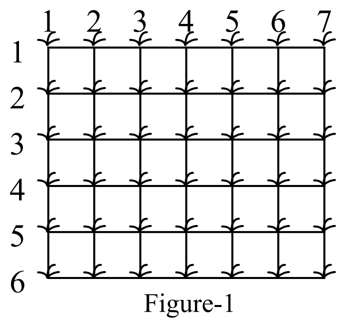


Figure-1

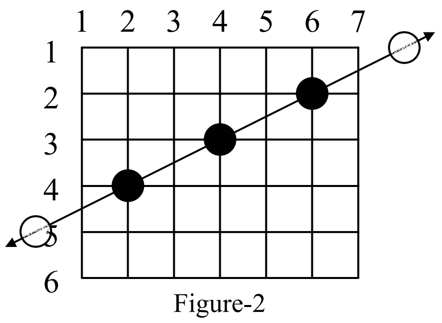


Figure-2

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:

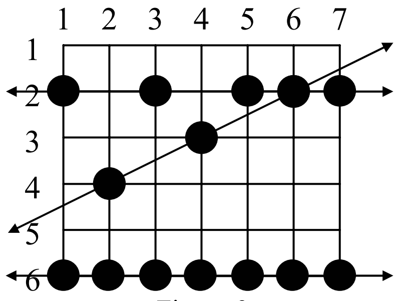


Figure-3

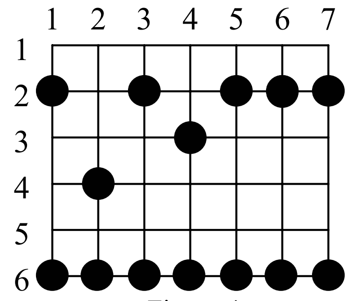


Figure-4

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 ($1 \le$ row number $\le R$) and the column number ($1 \le$ column number $\le C$) 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` (the example of Figure-4) `output`

```
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
```

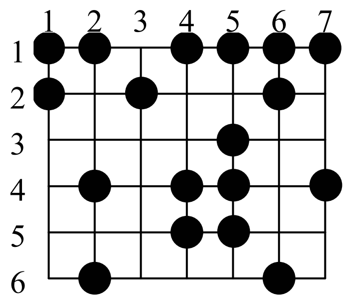


Figure-5

`output`

```
4
```

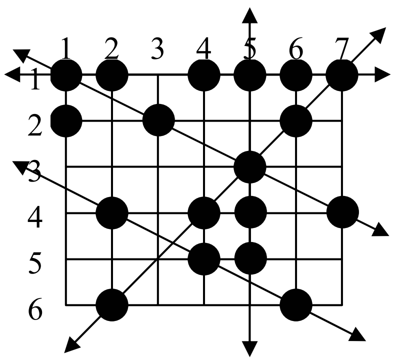


Figure-6: The maximum number of plants flattened by a frog is 4.

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.

A. Solution

A naïve O(N^3) time algorithm for the problem iterates through all O(N^2) line segments induced by the point set *S*, and determines how far each segment spacing can be extended to either direction within the point set (*O*(# landings) = *O*(*N*)).

An efficient $O(N^2)$ time algorithm for the problem is based on an algorithm for finding an equally-spaced collinear subset of a set. The algorithm works by "overlapping" all equally spaced triples in order to determine all maximal equally-spaced collinear subsets. The "overlapping" is performed by constructing an undirected graph where for each equally-spaced triple (p_A, p_B, p_C) we create nodes <A,B> and <B,C> and the edge (<A,B>, <B,C>); connected components in this graph correspond to maximal equally-spaced collinear subsets in the original set. Observe that a frog path is simply a linear chain of connected nodes (with at least one edge and two nodes, meaning at least 3 flattened plants) in this graph. Each node in this graph has degree at most two, so the edge set and vertex set both have size $O(N^2)$. Hence we can find all maximal equally-space collinear subsets in $O(N^2)$ time from the graph.

The only detail here is how to efficiently find the equally-spaced triples from which the graph is created. The obvious method of iterating over all triples of flattened plants would worsen the complexity to $O(N^3)$. If instead the field is stored as a two-dimensional array (every plant has an entry) giving the identity of the landing on that plant (e.g., if the 100th flattened plant were at (10, 12), then the array value at (10, 12) is 100), you can loop over pairs of flattened plants p_A and p_B, and then look up p_C from the array in constant time since you know what the location of p_C must be if it exists. This strategy takes $O(N^2)$ time but uses *O*(field size) memory – in particular, it needs 5000*5000 entries of a short integer each, or 50MB. Because the above graph also needs $O(N^2)$ space to store it, this strategy unfortunately would exceed the memory limit of 64MB. However, as this array is very sparse, it may be stored in memory as a hash table, which in the expected case does not affect the time complexity (but which in the worst case does). The third and best option is to construct the graph in linear time and constant memory by sorting the locations (e.g., row major, column minor) and keeping 3 pointers into the list (A, B, and C for pointing to p_A, p_B, and p_C, A<B<C) as follows; loop A over all values, and for each A march B and C down the list, moving either B or C forward at each step so as to try to maintain as close to equal spacing as possible; when exactly equal spacing is found, enter the nodes and edge into the graph.

There is also an $O(N^2)$ dynamic programming algorithm to solve this problem, which is plagued by the same memory problems as illustrated above. In addition to storing the identity matrix described above, store another $O(N^2)$ matrix containing whose rows are indexed by p_A; along the row are *N* entries, one per plant p_B giving the number of landings in a candidate frog path which goes through p_A and p_A but which only uses points which sort before p_A in the ordered list (i.e., pretend the field ends at p_A, and look for frog paths of any length in that smaller field – the idea is to find partial frog paths which violate none of the frog path conditions in the region of the field already examined). Assuming the table is filled up to row A, row A+1 is filled by considering all $O(N)$ flattened plants B before p_A, and if there is a flattened plant C such that A, B, and C are equally spaced, look up in the array the number of landings in the candidate frog path through B from C, increment by 1, and store as the Bth entry in the row for A. If C would be outside the field, then enter it as having 2 flattenings. At the same time check to see if the next flattened plant (D) would be outside the graph, and if so, you have a completed frog path. To efficiently determine C, the same 50MB array as above is needed; a hash table can again be used, with no increase in average-case time complexity, but an increase in worst-case time complexity.

B. Testing

Sungjoon Choi

The test data contains 25 test cases. Most of data are initially generated by random function, then they are modified by manual work.

Each test case has size N (the number of points) in the range between 10 and 5000. Among 25 test cases, 10 test cases have size $N \leq 1000$. The remaining 15 test cases have size $N \geq 2000$. The detail on the test data is summarized in the following table.

Each test case is worth 4 points. A program which implements a cubic time algorithm can solve the test cases within time limit such that their size $N \leq 1000$. An implementation of this algorithm should be able to get the first 10 test cases correct but will likely run out of time on all other cases (scoring 40% of the points).

Testing Data Description FROG

No.	*N*, (*R*C*)	Description	Solution
1	18, (6 * 7)	Sample data in the task description	4
2	10, (10 * 10)	Manually designed	5
3	25, (50 * 50)	Manually designed	13
4	50, (10 * 10)	Several Lines + random points	10
5	100, (20 * 20)	modified random point set	10
6	300, (30 * 30)	modified random point set	15
7	500, (55 * 55)	Several Lines + random points	28
8	500, (100 * 100)	Special case for no solution	0
9	1000, (100 * 100)	Several Lines + random points	34
10	1000, (1000 * 1000)	Several Lines + random points	250
11	2000, (50 * 50)	Random (uniform) points	25
12	2000, (100 * 200)	Several Lines + random points	33
13	2000, (1000 * 2000)	Several Lines + random points	333

14	3000, (60 * 60)	Uniformly random points	31
15	3000, (500 * 500)	X shapes and random points	500
16	3000, (5000 * 1)	Horizontal line	20
17	3000, (5 * 1000)	Several Lines + random points	17
18	4000, (100 * 100)	Random points (uniformly)	34
19	4000, (200 * 20)	Very dense points set	200
20	4000, (1000 * 1000)	Several Lines + random points	500
21	4000, (5000 * 5000)	Several Lines + random points	311
22	5000, (100 * 100)	Chess board style	100
23	5000, (1000 * 1000)	Several Lines + random points	334
24	5000, (3000 * 3000)	Irregular linear points	1000
25	5000, (5000 * 5000)	Modified random points	72

C. Background

The problem "*The Troublesome Frog*" is related to the problem for detecting spatial regularity in images. Spatial regularity detection is an important problem in a number of domains such as computer vision, scene analysis, and landmine detection from infrared terrain images. The AMESCS(All Maximum Equally-Spaced Collinear Subset) problem is defined as follows. Given a set $\boldsymbol{P}$ of N points in $\boldsymbol{E}^d$, find all maximal equally-spaced, collinear subset of points. Kahng and Robins[1] present an optimal quadratic time algorithm for solving the AMESCS problem.

D. REFERENCE

[1] A B. Kahng and G. Robins, **Optimal algorithms extracting spatial regularity in images**, *Pattern Recognition Letters*, 12, 757-764, 1991.

E. Source Code for FROG

Sungjoon Choi

```
/*
TASK: frog
LANG: C
*/

#include <stdio.h>
#define Max         5000

int p[Max][2], l[Max][Max];
int r, c, n, sol;

void ReadInput()
{
       int i;

       scanf("%d%d", &r, &c);
       scanf("%d", &n);
       for (i = 0; i < n; i++)
             scanf("%d%d", &p[i][0], &p[i][1]);
```

```
}

void Sort(int l, int r)
{
	int i, j, x0, x1, y;
	i = l; j = r; x0 = p[(l + r) / 2][0]; x1 = p[(l + r) / 2][1];
	do
	{
		while ((p[i][0] < x0) || ((p[i][0] == x0) && (p[i][1] < x1))) i++;
		while ((x0 < p[j][0]) || ((p[j][0] == x0) && (x1 < p[j][1]))) j--;
		if (i <= j)
		{
			y = p[i][0]; p[i][0] = p[j][0]; p[j][0] = y;
			y = p[i][1]; p[i][1] = p[j][1]; p[j][1] = y;
			i++;
			j--;
		}
	} while (i <= j);
	if (l < j) Sort(l, j);
	if (i < r) Sort(i, r);
}

int IsRegular(int a, int b, int c)
{
	if (p[b][0] - p[a][0] == p[c][0] - p[b][0])
	{
		if (p[b][1] - p[a][1] < p[c][1] - p[b][1]) return 1;
		else if (p[b][1] - p[a][1] == p[c][1] - p[b][1]) return 2;
		else return 3;
	}
	else
		if (p[b][0] - p[a][0] < p[c][0] - p[b][0]) return 1; else return 3;
}

void Run()
{
	int i, j, k, a, b, count, flag;

	for (i = 0; i < n - 2; i++)
	{
		flag = 0;
		j = i + 1;
		k = i + 1;
		while (flag || (k < n))
		{
			if (flag) j++; else k++;
			switch (IsRegular(i, j, k))
			{
				case 1: flag = 1;   break;
				case 2: l[i][j] = k;      break;
				case 3: flag = 0;   break;
			}
		}
	}

	sol = 0;
	for (i = 0; i < n - 1; i++)
		for (j = i + 1; j < n; j++)
		{
			if ((l[i][j] != 0) &&
			    ((p[i][0] * 2 - p[j][0] < 1) || (p[i][1] * 2 - p[j][1] <
1) || (p[i][0] * 2 - p[j][0] > r) || (p[i][1] * 2 - p[j][1] > c)))
			{
				a = i;
				b = j;
				count = 2;
				while (l[a][b] != 0)
				{
					count++;
					k = a;
					a = b;
					b = l[k][a];
					l[k][a] = 0;
```

```
                                }
                                if (((p[b][0] * 2 - p[a][0] < 1) || (p[b][1] * 2 -
p[a][1] < 1) || (p[b][0] * 2 - p[a][0] > r) || (p[b][1] * 2 - p[a][1] > c)) &&
(count > sol))
                                        sol = count;
                        }
                }
}

void WriteOutput()
{
        printf("%d\n", sol);
}

int main()
{
        ReadInput();
        Sort(0, n - 1);
        Run();
        WriteOutput();

        return 0;
}
```

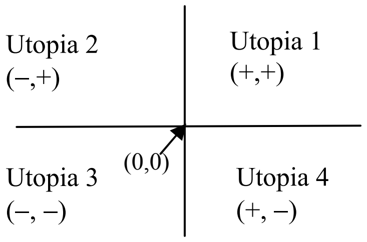
Task 2: UTOPIA

Sergey Melnik, Jung-Heum Park,
Chong-Dae Park, Kee Moon Song,
Ian Munro

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 $2N$ 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 $(x+\underline{u}, y-v)$. You have the $2N$ 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* ($1 \le N \le 10000$). The second line contains the 2*N* distinct integer code numbers ($1\le$ 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

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

output

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

Example 2: input

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

output

```
+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.

A. Solution

Jung-Heum Park, Ian Munro

The problem is two-dimensional, but is most easily solved as two one-dimensional problems solved independently. Let us first concentrate on the one-dimensional problem. The one dimensional problem is: Given *N* code numbers and a sequence of *N* region signs (each of which is + or -), produce a sequence of *N* signed code values $\{x_i\}$ so that the sign of $\Sigma_{i<k} x_i$ matches the i^{th} region sign.

The basic approach is quite intuitive; though seeing that it works requires some care. We start by sorting the *N* input code numbers into increasing order, and then assigning alternating signs to them so that $|x_i| > |x_{i+1}|$, though $x_i > 0$ iff $x_{i+1} < 0$. (The sign of x_1, and so all the others is yet to be determined.) We then start at an appropriate place in the "middle" of this sequence and move outward, using large numbers to change the sign of the sum from that of the current situation, and small values to keep it the same. A few definitions and lemmas make things much clearer:

Definition: A sequence of non-zero integers $X = (x_a, x_{a+1}, \ldots, x_b)$, $a \le b$ is an *alternating sequence* if

(i) $|x_a| < |x_{a+1}| < |x_{a+2}| < \cdots < |x_b|$, and

(ii) for all i, $a < i \le b$, the sign of x_i is different from the sign of x_{i-1}.

Here, $|x_a|$ is the absolute value of x_a.

Lemma 1. Let $X = (x_a, x_{a+1}, \ldots, x_b)$ be an alternating sequence. The sign of x_b is equal to the sign of $\sum_{a \le i \le b} x_i$, the total sum of elements in X.

Proof. We assume without loss of generality that x_b is a positive integer. When the number $b - a + 1$ of elements in X is even (resp. odd), the partial sums $x_a + x_{a+1}$, $x_{a+2} + x_{a+3}$, $\ldots$, $x_{b-1} + x_b$ (resp. x_a, $x_{a+1} + x_{a+2}$, $\ldots$, $x_{b-1} + x_b$) are positive, and thus the total sum $\sum_{a \le i \le b} x_i$ is positive. □

Example 1. The total sum of elements in an alternating sequence $X = (-1, +2, -5, +6)$ is $(-1 + 2) + (-5 + 6) = +2$, which is positive.

Example 2. For an alternating sequence $X = (+3, -4, +5, -6, +7)$, the total sum of elements in X is equal to $(+3) + (-4 + 5) + (-6 + 7) = +5$, which is also positive.

We let $S = (s_a, s_{a+1}, \ldots, s_b)$, $a \le b$ be a sequence of signs, plus and minus. A sequence $X' = (x_{i_a}, x_{i_{a+1}}, \ldots, x_{i_b})$, $a \le b$ is *valid* with respect to S if the sign of $\sum_{a \le j \le k} x_{i_j}$ is equal to s_k for each k, $a \le k \le b$.

Theorem 1. Let $X = (x_a, x_{a+1}, \dots, x_b)$, $a \le b$ be an alternating sequence, and let $S = (s_a, s_{a+1}, \dots, s_b)$, $a \le b$ be a sequence of signs. If the sign of x_b is equal to s_b, then there exists a sequence $X' = (x_{i_a}, x_{i_{a+1}}, \dots, x_{i_b})$ such that

(i) $\{x_{i_a}, x_{i_{a+1}}, \dots, x_{i_b}\} = \{x_a, x_{a+1}, \dots, x_b\}$, and

(ii) X' is valid with respect to S.

Proof. The proof is by induction on the number k of elements in X. When $k = 1$, it is easy to see that $X' = X$ is a valid sequence with respect to S. Now we assume that $k \ge 2$. We let $S_1 = S - s_b$, that is, $S_1 = (s_a, s_{a+1}, \dots, s_{b-1})$.

Case 1. The sign of x_b is equal to s_{b-1}.

Let $X_1 = X - x_a$, that is, $X_1 = (x_{a+1}, x_{a+2}, \dots, x_b)$, and let $t = x_a$.

Case 2. The sign of x_{b-1} is equal to s_{b-1}.

Let $X_1 = X - x_b$, that is, $X_1 = (x_a, x_{a+1}, \dots, x_{b-1})$, and let $t = x_b$.

Note that X_1 is an alternating sequence, and S_1 is a sequence of signs such that the sign of the last element in X_1 is equal to s_{b-1}, the last element in S_1. Thus, by induction hypothesis, there exits a valid sequence X_1' with respect to S_1. Therefore, $X' = (X_1', t)$ is a valid sequence with respect to S. □

Example 3. For an alternating sequence $X = (-4, +5, -7, +8)$, and a sequence of signs $S = (+, -, -, +)$, we have

$S_1 = (+, -, -)$, $X_1 = (-4, +5, -7)$,

$S_2 = (+, -)$, $X_2 = (+5, -7)$,

$S_3 = (+)$, $X_3 = (+5)$.

Thus,

$X_3' = (+5)$,

$X_2' = (+5, -7)$,

$X_1' = (+5, -7, -4)$,

$X' = (+5, -7, -4, +8)$.

Example 4. For $X = (-1, +2, -3)$ and $S = (-, -, -)$,

$S_1 = (-, -)$, $X_1 = (+2, -3)$,

$S_2 = (-)$, $X_2 = (-3)$.

Thus,

$X_2' = (-3)$,

$X_1' = (-3, +2)$,

$X' = (-3, +2, -1)$.

Now, we are ready to present an algorithm for the problem of Utopia Divided.

Algorithm Utopia_Divided

```
Step 1.  // read input
        1.1  read $N$;
        1.2  read $2N$ code numbers and partition them into $A$ and $B$ such that $|A|=|B|$;
        1.3  read a sequence of regions $R=(r_1,r_2,\ldots,r_N)$;

Step 2.  // find $x$-coordinates of code pairs
        2.1 find a sequence of signs $S=(s_1,s_2,\ldots,s_N)$ such that
              for all $j$, $1\le j\le N$, $s_j$ ='+' if $r_j=1,4$; otherwise $s_j$ ='−'.
        2.2 find an alternating sequence $X=(x_1,x_2,\ldots,x_N)$ from $A$ such that
             the sign of $x_N$ is equal to $s_N$.
        2.3 given $X$ and $S$, find a valid sequence $X'=(x_{i_1},x_{i_2},\ldots,x_{i_N})$ w.r.t. $S$
               according to the proof of Theorem 1.

Step 3.  // find $y$-coordinates of code pairs
        3.1 find a sequence of signs $S=(s_1,s_2,\ldots,s_N)$ such that
              for all $j$, $1\le j\le N$, $s_j$ ='+' if $r_j=1,2$; otherwise $s_j$ ='−'.
        3.2 find an alternating sequence $Y=(y_1,y_2,\ldots,y_N)$ from $B$ such that
             the sign of $y_N$ is equal to $s_N$.
        3.3 given $Y$ and $S$, find a valid sequence $Y'=(y_{i_1},y_{i_2},\ldots,y_{i_N})$ w.r.t. $S$
              according to the proof of Theorem 1.

Step 4.  // write output
        print $(x_{i_1},y_{i_1}),(x_{i_2},y_{i_2}),\ldots,(x_{i_N},y_{i_N})$.
```

Theorem 2. Algorithm Utopia_Divided is correct, and its running time is $O(N\log N)$.

Proof. The correctness of the algorithm is mainly due to Theorem 1. The complexity of each step except Step 2.2 and 3.2 is $O(N)$, where Step 2.2 and 3.2 require $O(N\log N)$ time for sorting. □

We do not know of a faster solution, nor do we have a proof that sorting is necessary for this problem.

Another reasonable approach to the problem is backtracking. The approach is effective on small inputs, fewer than 100 points or so.

B. Test Data Information

Kee Moon Song

The test data consists of 25 test cases, each generated mainly at random. Some region sequences (i) consist of at most two regions such as 1 and 2, or (ii) visit regions in a circular way. Some small inputs (9 test cases of size $N \leq 100$ and 1 test case of size $N = 500$) are included so that inefficient but correct solutions can score some points.

Timing Tests for Utopia(sec.)

No.	Optimal *O*(*N*) In C/C++	Optimal *O*(*N*) in Pascal	BackTracking1	BackTracking2
1	< 0.1	< 0.1	< 0.1	< 0.1
2	< 0.1	< 0.1	< 0.1	< 0.1
3	< 0.1	< 0.1	< 0.1	< 0.1
4	< 0.1	< 0.1	< 0.1	< 0.1
5	< 0.1	< 0.1	< 0.1	< 0.1
6	< 0.1	< 0.1	< 0.1	< 0.1
7	< 0.1	< 0.1	< 0.1	< 0.1
8	< 0.1	< 0.1	< 0.1	< 0.1
9	< 0.1	< 0.1	0.15	0.28
10	< 0.1	< 0.1	< 0.1	< 0.1
11	< 0.1	< 0.1	2.57	3.77
12	< 0.1	< 0.1	> 20	> 20
13	< 0.1	< 0.1	> 20	> 20
14	< 0.1	< 0.1	> 20	> 20
15	< 0.1	< 0.1	> 20	> 20
16	< 0.1	< 0.1	> 20	> 20
17	< 0.1	< 0.1	> 20	> 20
18	< 0.1	< 0.1	> 20	> 20
19	< 0.1	< 0.1	> 20	> 20
20	< 0.1	< 0.1	> 20	> 20
21	< 0.1	< 0.1	> 20	> 20
22	< 0.1	< 0.1	> 20	> 20
23	< 0.1	< 0.1	> 20	> 20
24	< 0.1	< 0.1	> 20	> 20
25	< 0.1	< 0.1	> 20	> 20

Testing Data Description for Utopia

No.	Size	Description
1	N = 4	Example 2
2	N = 10	Key value : 1~20, Circular plane sweep
3	N = 10	Key value : Starting from 20, step 1 or 2, scrambled
4	N = 30	Key value : Starting from 30, step 1 or 2, scrambled
5	N = 30	Key value : Starting from 200, step 1 to 3, scrambled
6	N = 50	Key value : Starting from 1, step 1 to 3, scrambled
7	N = 50	Key value : Starting from 150, step 1 to 3, scrambled
8	N = 50	Key value : Starting from 300, step 1 to 4, scrambled Visit plane 1 except last visit. At last, visit plane 3
9	N = 100	Starting from 1000, step 1 to 4, scrambled Circular plane visit
10	N = 500	Key value : 1, 3, 5, …, 1999, Circular plane visit
11	N = 700	Key value : Starting from 1, step 1 to 5, scrambled, Visit plane 1, 2 only
12	N = 1000	Key value : Starting from 4000, step 1 to 5, scrambled
13	N = 1500	Key value : Starting from 9000, step 1 to 5, scrambled
14	N = 2000	Key value : Starting from 15000, step 1 to 5, scrambled
15	N = 2500	Key value : Starting from 30000, step 1 to 5, scrambled
16	N = 3000	Key value : Starting from 1, step 1 to 7, scrambled, Visit plane 1, 4 only
17	N = 3500	Key value : Starting from 20000, step 1 to 7, scrambled
18	N = 4000	Key value : Starting from 30000, step 1 to 7, scrambled
19	N = 4500	Key value : Starting from 50000, step 1 to 7, scrambled
20	N = 5000	Key value : Starting from 60000, step 1 to 7, scrambled
21	N = 6000	Key value : Starting from 1, step 1 to 9, scrambled Visit plane 2, 4 only
22	N = 7000	Key value : Starting from 40000, step 1 to 7, scrambled
23	N = 8000	Key value : Starting from 50000, step 1 to 5, scrambled
24	N = 9000	Key value : Starting from 30000, step 1 to 6, scrambled
25	N = 10000	Key value : Starting from 80000~99999, scrambled

C. Grading

If a contestant's program outputs the correct answer for a test case in the time limit, then he/she receive 4 points for that test, and otherwise he/she gets 0 points for the test case.

D. Remark

The original one-dimensional problem entitled "*Sign Representation*" was proposed by Sergey Melnik. It was extended to the two-dimensional problem by the Host Scientific Committee.

E. Source Code for UTOPIA

Kee Moon Song

```
/*
TASK : Utopia
LANG : C

    Optimal solution - O(N)
*/

#include <stdio.h>
#include <stdlib.h>

#define MAX 10000

int num;                        // N
int data[2][MAX], dest[MAX];
int answer[MAX][2];

void getdata()
{
       int i;

       scanf ("%d", &num);
       for (i = 0; i < num; i++) scanf ("%d %d", &data[0][i], &data[1][i]);
       for (i = 0; i < num; i++) scanf ("%d", &dest[i]);
}

int compare(const void* a, const void* b) // subroutine for qsort
{
       return (*(int*)a > *(int*)b) ? 1:-1;
}

void preprocessing() // Sort input values
{
       qsort(data[0], num, sizeof(int), compare);
       qsort(data[1], num, sizeof(int), compare);
}

int check(int plane, int axis) // if axis=0, return x<0?
                               // if axis=1, return y<0?
{
       if (axis == 0)
              return plane >= 2 && plane <= 3;
       else
              return plane >= 3;
}

void solveproblem()
{
       int bound[2][2];
       int *target, add;
       int i, loop;

       for (loop = 0; loop < 2; loop++)
       {
              // maintain that
              // x_n + x_(n-2) + ... + x_k(0 or 1) > x_(n-1) + x_(n-3) + ... +
x_l(1 or 0)
              bound[0][0] = check(dest[num - 1], loop) ^ (num % 2);
              bound[0][1] = num - 2 + check(dest[num - 1], loop);
              bound[1][0] = 1 - bound[0][0];
              bound[1][1] = num * 2 - 3 - bound[0][1];

              for (i = bound[0][0]; i <= bound[0][1]; i+= 2)
                     data[loop][i] = -data[loop][i];

              for (i = num - 2; i >= 0; i--)
              {
                     if (check(dest[i], loop) ^ check(dest[i + 1], loop))
```

```
                    {
                            target = (bound[0][1] > bound[1][1]) ?
&bound[0][1]:&bound[1][1];
                            add = -2;
                    }
                    else
                    {
                            target = (bound[0][0] < bound[1][0]) ?
&bound[0][0]:&bound[1][0];
                            add = 2;
                    }

                    answer[i + 1][loop] = data[loop][*target];
                    *target += add;
            }

            answer[0][loop] = data[loop][(bound[0][0] > bound[0][1]) ?
bound[1][0]:bound[0][0]];
       }
}

void writeresult()
{
       int i;

       for (i = 0; i < num; i++)
            printf ("%+d %+d\n", answer[i][0], answer[i][1]);
}

int  main()
{
       getdata();
       preprocessing();
       solveproblem();
       writeresult();

       return 0;
}
```

Task 3: XOR

Hwan Gue Cho, Chong-Dae Park
Jyrki Nummenmaa

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.

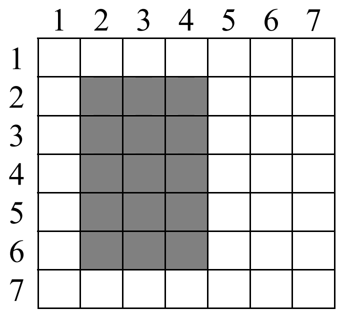


Figure-1

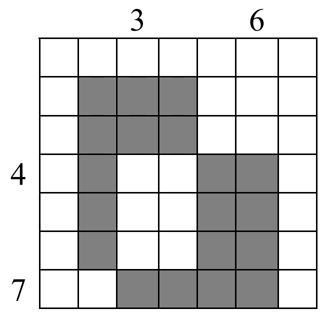


Figure-2

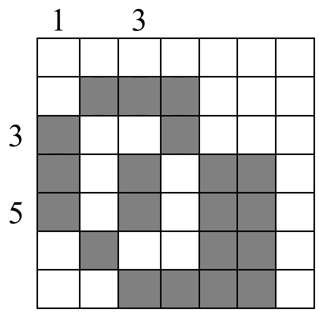


Figure-3

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 ≤ *N* ≤ 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` `xor0.out`

```
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
```

```
#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.

A. Solution

Jung Gun Lim, Chong-Dae Park

A TWO-OPTIMAL SOLUTION FOR XOR

We start by giving some definitions, which will help us to explain our solution.

Definition 1. A *grid point* is an intersection point of two grid lines.

Definition 2. An *imagecorner* is grid point adjacent to an odd number of black pixels. If a grid point *p* is an imagecorner, we say that the *ic-value* of *p* is 1 and we write *ic*(*p*)=1, and if *p* is not an imagecorner, we say that the ic-value of *p* is 0 and we write *ic*(*p*)=0.

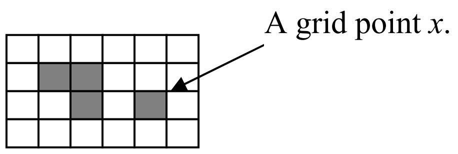


Figure-1. An example image.

For example, in Figure-1, the grid point *x* is adjacent to four pixels, of which 1 is black and 3 are white. This way, the grid point *x* is also an imagecorner. In total, there are 10 imagecorners in the image in Figure-1.

It is very easy to observe the following lemma.

Lemma 1. An image is all white if and only if there are no imagecorners.

We may now try to find a solution by working from the input image backward to an all white image by doing XOR operations.

The following lemma is the base of our solution.

Lemma 2. Suppose we perform XOR(L,R,T,B). Let Q be the rectangle affected by XOR(L,R,T,B), and let the grid point at left top corner of Q be *a*, at left bottom of Q be *b*, at right bottom corner of Q be *c*, and at right top corner of Q be *d*. Then XOR(L,R,T,B) will change the ic-value for *a*, *b*, *c*, and *d*, and the ic-values for all other grid points will remain unchanged.

Proof. Exactly one pixel adjacent to each of *a*, *b*, *c*, and *d* will change their color. It follows that the ic-values will change for these points.

Now consider the other grid points at the edge of Q. For all of these grid points, exactly two pixels adjacent to them (the ones inside Q) will change their color. Therefore, the number of odd pixels around those grid points does not change.

Finally, consider the grid points inside Q. For all of these grid points, all four pixels adjacent to them will change their color. Thus, the ic-values of these grid points will not change and this completes our proof. □

When trying to reduce the number of corners, we would like to reduce them maximally with each XOR call. However, it may not always be possible to choose an XOR call in such a way that it changes the ic-value of all of the corners of the area affected by the XOR call.

Lemma 3. If the image is not all white, it is always possible to choose such an XOR call that the number of imagecorners is reduced by either two or four.

Proof. First, observe that at least two of the imagecorners are on the same line. For this, examine the pixels in the topmost row which contains black pixels. There is an imagecorner at the right top corner of the rightmost of these pixels and at the left top corner of the leftmost of these pixels. These imagecorners can not be the same and they are on the same line.

Next, observe that the imagecorners can not all be on the same line. This can be done by first examining the topmost and bottommost imagecorners as above and finding out that the topmost imagecorners are higher up than the bottommost imagecorners. Then examine the leftmost and rightmost imagecorners and find out that they can not be on the same vertical line. It follows that not all imagecorners can be on the same line.

We can now choose such a rectangle that three of its corners are imagecorners as follows. We first choose the rightmost of the topmost imagecorners, and then some other imagecorner of the rightmost imagecorners. The third imagecorner can be found as follows. We start from the rightmost of the topmost imagecorners. Only the adjacent pixel left and down from it is black and the other adjacent pixels are white. We travel the gridpoints to the left one by one. At some point either the black pixels below end or black pixels above start. If both of these happen at the same time, we find no imagecorner yet, and the situation is reversed and we continue. When, as we go left, the pixel color changes only above or below (this must eventually happen as at least we must eventually come to a situation, where pixels both above and below are white), we encounter an imagecorner, which we choose. These three imagecorners specify a valid rectangle.

Therefore, it is always possible to choose the XOR call in such a way that either four or three of the grid points at the corners of the rectangle affected by the XOR are imagecorners. Now if all four of these corners are imagecorners, then by Lemma 2 we reduce the number of imagecorners by 4, and if exactly three of these corners are imagecorners, then by Lemma 2 we reduce the number of imagecorners by 2 (we remove three imagecorners and create one). This completes the proof. □

We now get the following 2-optimal method for solving XOR.

Algorithm 1. Two-optimal XOR

1. If there are black pixels in the image, find three imagecorners that can be used to specify a rectangle, and use the respective XOR call. Go to 1.
2. Halt.

Theorem 1. Algorithm 1 is correct and 2-optimal.

Proof. We will first show the correctness of Algorithm 1. From Lemma 3 it follows that as long as there are black pixels in the image, it is possible to find the required imagecorners in Step 1 of the algorithm. Since each of the executions of Step 1 reduces the number of imagecorners, Algorithm 1 will eventually halt and the image will be all white. It follows that Algorithm 1 works correctly. □

(a) 2-optimal solution (Solution 1)

```
For each row $i$
   For each column $j$
   If point $(i, j)$ is corner
   {
      Let $k$ be the nearest corner right to the point $(i, j)$ in row $i$
      Let $l$ be the nearest corner below to the point $(i, j)$ in column $j$
      XOR ($j$, $k$-1, $i$, $l$-1);
   }
```

(b) Randomized 2-optimal solution

```
Repeat
{
   Load initial data;
   While there are corner points
   {
      Choose a corner point (i,j) randomly
      Let $k$ be a random corner in row $i$
      Let $l$ be a random corner in column $j$
      XOR ($j$, $k$-1, $i$, $l$-1);
   }
} until there are no improvements
```

This randomized algorithm gives a chance of best solutions. If this algorithm runs several times, we can choose the best among the solutions.

(c) Simple greedy algorithm (Solution 2)

```
For each row $i$
   For each column $j$
      If pixel $(i, j)$ is black
      {
         Find the horizontally maximal black run from $i$.
         Assume pixels in $(i,j)$ to $(i,k$-1) are all black.
         Find the vertically and  maximal black run from $i$.
         Assume pixels in $(i,j)$ to $(j,l$-1) are all black.
         // so  rectangle region $[i,k-1] \times [j,l-1]$ is a black rectangle
         XOR ($j$, $k$-1, $i$, $l$-1);
      }
```

(d) Algorithm using Maximal Matching (Solution 3)

Solution 3 finds a set of "good" rectangles in a row. For a row there are even numbers of corner points C_1, C_2, C_3, …., C_k., where k=2N. Then we construct a complete weighted graph $G(V,E)$ from $\{C_i\}$. Let v_i of V be C_i. And edge $e(v_i, v_j)$ is given for every pair of v_i, v_j. The weight of each edge, $w(e)$, is given: $w(v_i, v_j) = 1$, if there is a rectangle(4 corners) with v_i, v_j. Otherwise $w(v_i, v_j) = 0$. In the following figure, circle denotes the corner point of given image. There are 6 corner points(vertices, namely v_1, v_2, v_3, v_4, v_5, v_6.) in the first row. Since there is a rectangle with four corner points with (v_1, v_4) edge, (four corners (1,1), (1,4), (3,1), (3,4) gives a rectangle), thus $w(v_1,v_4) = 1$. In a similar way we can set $w(v_1,v_3) = 0$, $w(v_1,v_6) = 1$, $w(v_1,v_7) = 0$, $w(v_3,v_7) = 0$.

O (1,1)		O (1,3)	O (1,4)		O (1,6)	O (1,7)	O
(1,2)							
O			O				
		O			O		
O					O		
						O	O

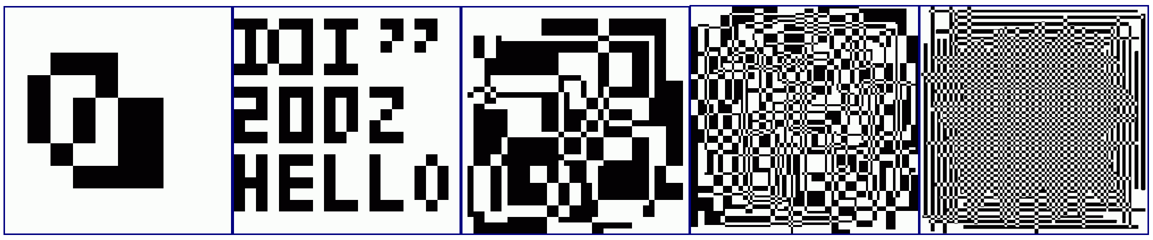
Then we compute the maximal matching of $G(V,E)$, and remove all rectangles corresponding to edges contained in maximal matching. This gives a better solution compared to the performance of Solution 1 and Solution 2.

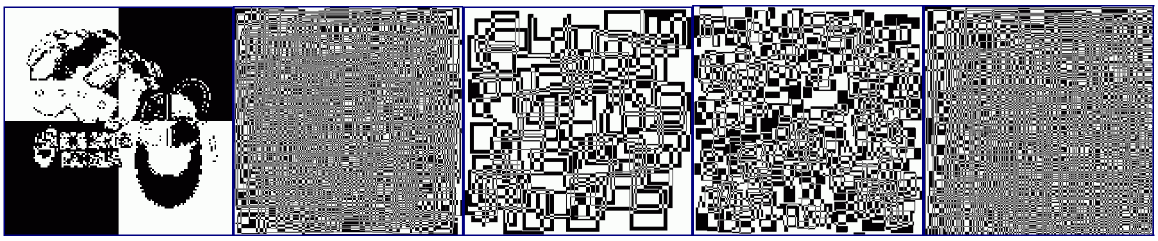
B. Test Data and 3 Solutions

Jung Gun Lim

Testing Data Description for XOR

No	Size (N)	Description	Solution 1 2-optimal	Solution 2 Greedy	Solution 3 Matching	Best of IOI 2002
1	10	Sample input (shifted)	**3**	7	**3**	**3**
2	20	"IOI 2002 HELLO"	38	50	**33**	34
3	40	25 random boxes	34	74	**27**	28
4	700	100 random boxes	153	678	**113**	122
5	900	66 wide rectangles	**74**	1893	**74**	**74**
6	128	Modified IOI emblem	598	681	**469**	481
7	500	500 random boxes	907	15765	**507**	566
8	600	200 pattern(□ shape)	222	1540	**200**	**200**
9	1500	Two kinds of 500 boxes	526	3732	**500**	**500**
10	2000	1000 special rectangles	841	71851	866	**810**





C. REMARK

After we proposed this task, we have found that some games are similar to this XOR task. The game is to ask a user to clear the given image(10 by 10) by only applying 'cross' shape XOR. It is interesting to see and play in the following web sites:
http://www.ida.net/users/housley/atag.htm
http://www.math.hkbu.edu.hk/~cstong/sci3510/xchess.html
http://www.math.hkbu.edu.hk/~cwyeung/xchess/

D. Source code for XOR (Solution3)

Jung Gun Lim

```
#include <cstdio>
#define MAXN 2000

typedef struct
{
    int rn;
    int cn;
} corner;

int n, count;
corner ct[MAXN+2][MAXN+2];  // data structure using linked list that contains
corners
int cor[MAXN+2], coc[MAXN+2]; // cor : corners on the row  coc : corners on the
column
int b[MAXN+2][MAXN+2];  // input
int sols[MAXN*MAXN][4]; // solution
int mcount, msols[MAXN*MAXN][4];  // best solution found

void flip_corner (int r, int c) // if (r,c) is a corner, make it not
{                               // otherwise, vice versa.
    int p;
    p=0;
    while (1)
    {
        if (ct[r][p].cn == c)
        {
            ct[r][p].cn = ct[r][c].cn;
            cor[r]--;
            break;
        }
        if (ct[r][p].cn > c)
        {
            ct[r][c].cn = ct[r][p].cn;
            ct[r][p].cn = c;
            cor[r]++;
            break;
        }
        p=ct[r][p].cn;
    }
    p=0;
    while (1)
    {
        if (ct[p][c].rn == r)
        {
            ct[p][c].rn = ct[r][c].rn;
            coc[c]--;
            break;
        }
        if (ct[p][c].rn > r)
        {
            ct[r][c].rn = ct[p][c].rn;
            ct[p][c].rn = r;
            coc[c]++;
            break;
        }
        p=ct[p][c].rn;
    }
}

void init_ct () // initializing corner table
{
    int i, j;
    for (i=1;i<=n+1;i++)
    {
        ct[0][i].rn = n+2;
        ct[0][i].cn = i+1;
        ct[i][0].rn = i+1;
        ct[i][0].cn = n+2;
        cor[i]=0;
        coc[i]=0;
    }
    for (i=1;i<=n+1;i++)
        for (j=1;j<=n+1;j++)
        {
            if ((b[i][j]+b[i-1][j]+b[i][j-1]+b[i-1][j-1])%2)
            {
                flip_corner (i, j);
            }
        }
    count=0;
}

void add_to_sol (int c1, int c2, int r1, int r2) // adding a rectangle to the
solution
{
    flip_corner (r1, c1);
    flip_corner (r2, c1);
    flip_corner (r1, c2);
    flip_corner (r2, c2);
    if (c1>c2) c1^=c2^=c1^=c2;
    if (r1>r2) r1^=r2^=r1^=r2;
    c2--;
    r2--;
    sols[count][0]=c1;
    sols[count][1]=c2;
    sols[count][2]=r1;
    sols[count][3]=r2;
    count++;
}

int links[MAXN+2], link[MAXN+2][MAXN+2]; // maximal matching algorithm
int match[MAXN+2], islinked[MAXN+2];

int traveled[MAXN+2];
int path[MAXN+2], pathlen;
int found;

void dfs (int k)
{
    int i, v;

    for (i=0;i<links[k];i++)
    {
        v=link[k][i];
        if (!traveled[v])
        {
            if (match[v]==-1)
            {
                path[pathlen]=v;
                pathlen++;
                found=1;
                return;
            }
            else
            {
                path[pathlen]=v;
                pathlen++;
                traveled[v]=1;
                path[pathlen]=match[v];
                pathlen++;
                dfs (match[v]);
                if (found) return;
                pathlen-=2;
            }
        }
    }
}
```

```
void matching (int elems)
{
    int i, flag, last;
    for (i=0;i<elems;i++)
    {
        match[i]=-1;
    }
    do
    {
        for (i=0;i<elems;i++)
        {
            traveled[i]=0;
        }
        flag=0;
        for (i=0;i<elems;i++)
        {
            if (match[i]==-1 && !traveled[i])
            {
                found=0;
                traveled[i]=1;
                path[0]=i;
                pathlen=1;
                dfs (i);
                if (found)
                {
                    for (i=0;i<pathlen;i++)
                    {
                        if (i%2==0)
                        {
                            match[path[i]]=path[i+1];
                        }
                        else
                        {
                            match[path[i]]=path[i-1];
                        }
                    }
                    flag=1;
                }
            }
        }
    } while (flag);
    last=-1;
    for (i=0;i<elems;i++)
    {
        if (match[i]!=-1)
        {
            islinked[i]=1;
        }
        else
        {
            islinked[i]=0;
            if (last==-1)
            {
                last=i;
            }
            else
            {
                match[last]=i;
                match[i]=last;
                last=-1;
            }

        }
    }
}

void destroy_row (int r) // removing all corners in the row
{
    int cols;
    int col[MAXN+2];
    int p, q, i, j;
    int min, minp;
    p=0;
    cols=0;
    while (ct[r][p].cn < n+2)
    {
        p=ct[r][p].cn;
        col[cols]=p;
        cols++;
    }
    for (i=0;i<cols;i++)
        links[i]=0;
    for (i=0;i<cols-1;i++)
// Could the pair of columns be destroyed with a rectangle of
                           // 4 corners?
    {
        for (j=i+1;j<cols;j++)
        {
            p=ct[0][col[i]].rn;
            q=ct[0][col[j]].rn;
            while (p<n+2 && q<n+2)
            {
                if (p>q)
                {
                    q=ct[q][col[j]].rn;
                }
                else if (p<q)
                {
                    p=ct[p][col[i]].rn;
                }
                else
                {
                    if (p!=r)
                    {
                        link[i][links[i]]=j;
                        links[i]++;
                        link[j][links[j]]=i;
                        links[j]++;
                        break;
                    }
                    p=ct[p][col[i]].rn;
                    q=ct[q][col[j]].rn;
                }
            }
        }
    }
    matching (cols); // Maximum matching
    for (i=0;i<cols;i++)
    {
        if (match[i]>i)
        {
            if (islinked[i]) // Destroying matched pairs
            {
                p=ct[0][col[i]].rn;
                q=ct[0][col[match[i]]].rn;
                min=n+2;
                while (p<n+2 && q<n+2)
                {
                    if (p>q)
                    {
                        q=ct[q][col[match[i]]].rn;
                    }
                    else if (p<q)
                    {
                        p=ct[p][col[i]].rn;
                    }
                    else
                    {
                        if (p!=r)
                        {
                            if (min>cor[p])
                            {
                                min=cor[p];
                                minp=p;
                            }
                         }
                        p=ct[p][col[i]].rn;
                        q=ct[q][col[match[i]]].rn;
                    } // prefers the row that contains least column
                }
                add_to_sol (col[i], col[match[i]], r, minp);
            }
        }
    }
    for (i=0;i<cols;i++)
    {
        if (match[i]>i)
        {
            if (!islinked[i]) // not matched pairs
            {
                min=n+1;
                p=ct[0][col[i]].rn;
                while (p<n+2)
                {
                    if (cor[p]<min && p!=r)
                    {
                        min=cor[p];
                        minp=p;
                    }
                    p=ct[p][col[i]].rn;
                }
                p=ct[0][col[match[i]]].rn;
                while (p<n+2)
                {
                    if (cor[p]<min && p!=r)
                    {
                        min=cor[p];
                        minp=p;
                    }
                    p=ct[p][col[match[i]]].rn;
                }
                add_to_sol (col[i], col[match[i]], r, minp);
            }
        }
    }
}

void destroy_col (int c) // destroying all corners in the corner; the same
structure with destroy_row's.
{
    int rows;
    int row[MAXN+2];
    int p, q, i, j;
    int min, minp;
    p=0;
    rows=0;
    while (ct[p][c].rn < n+2)
    {
        p=ct[p][c].rn;
        row[rows]=p;
        rows++;
    }
    for (i=0;i<rows;i++)
        links[i]=0;
    for (i=0;i<rows-1;i++)
    {
        for (j=i+1;j<rows;j++)
        {
            p=ct[row[i]][0].cn;
            q=ct[row[j]][0].cn;
            while (p<n+2 && q<n+2)
            {
                if (p>q)
                {
                    q=ct[row[j]][q].cn;
                }
                else if (p<q)
                {
                    p=ct[row[i]][p].cn;
                }
                else
                {
                    if (p!=c)
                    {
                        link[i][links[i]]=j;
                        links[i]++;
                        link[j][links[j]]=i;
                        links[j]++;
                        break;
                    }
                    p=ct[row[i]][p].cn;
                    q=ct[row[j]][q].cn;
                }
            }
        }
    }
    matching (rows);
    for (i=0;i<rows;i++)
    {
        if (match[i]>i)
        {
            if (islinked[i])
            {
                p=ct[row[i]][0].cn;
                q=ct[row[match[i]]][0].cn;
                min=n+2;
                while (p<n+2 && q<n+2)
                {
                    if (p>q)
                    {
                        q=ct[row[match[i]]][q].cn;
                    }
                    else if (p<q)
                    {
                        p=ct[row[i]][p].cn;
                    }
                    else
                    {
                        if (p!=c)
                        {
                            if (min>coc[p])
                            {
                                min=coc[p];
                                minp=p;
                            }
                         }
                        p=ct[row[i]][p].cn;
                        q=ct[row[match[i]]][q].cn;
                    }
                }
                add_to_sol (c, minp, row[i], row[match[i]]);
            }
        }
    }
    for (i=0;i<rows;i++)
    {
        if (match[i]>i)
        {
            if (!islinked[i])
            {
                min=n+1;
                p=ct[row[i]][0].cn;
                while (p<n+2)
                {
                    if (coc[p]<min && p!=c)
                    {
                        min=coc[p];
                        minp=p;
                    }
                    p=ct[row[i]][p].cn;
                }
                p=ct[row[match[i]]][0].cn;
                while (p<n+2)
                {
                    if (coc[p]<min && p!=c)
                    {
                        min=coc[p];
                        minp=p;
                    }
                    p=ct[row[match[i]]][p].cn;
                }
                add_to_sol (c, minp, row[i], row[match[i]]);
            }
        }
    }
}

void update_sol () // updating the best solution
{
    int i, j;
    if (mcount>count)
    {
        mcount=count;
        for (i=0;i<mcount;i++)
            for (j=0;j<4;j++)
                msols[i][j]=sols[i][j];
    }
}

void delete_min_row_or_col () // destroying the row or the column that contains
least corners
{
    int mins, minp, i, isrow;
    init_ct ();
    while (1)
    {
        mins=n+1;
        for (i=1;i<=n+1;i++)
        {
            if (mins>cor[i] && cor[i])
            {
                mins=cor[i];
                minp=i;
                isrow=1;
            }
            if (mins>coc[i] && coc[i])
            {
                mins=coc[i];
                minp=i;
                isrow=0;
            }
        }
        if (mins==n+1) break;
        if (isrow)
        {
            destroy_row (minp);
        }
        else
        {
            destroy_col (minp);
        }
    }
    update_sol ();
}

void delete_one_by_one ()
{
    int i;
    init_ct ();
    for (i=1;i<=n+1;i++) // destroying the top row first
    {
        if (cor[i]!=0) destroy_row (i);
    }
    update_sol ();
    init_ct ();
    for (i=1;i<=n+1;i++) // the bottom row
    {
        if (coc[i]!=0) destroy_col (i);
    }
    update_sol ();
    init_ct ();
    for (i=n+1;i>=1;i--) // the left column
    {
        if (cor[i]!=0) destroy_row (i);
    }
    update_sol ();
    init_ct ();
    for (i=n+1;i>=1;i--) // the right column
    {
        if (coc[i]!=0) destroy_col (i);
    }
    update_sol ();
}

void read_data()
{
    int i, j;
    scanf ("%d", &n);
    for (i=0;i<n;i++)
    {
        for (j=0;j<n;j++)
        {
            scanf ("%d", &b[i+1][j+1]);
        }
    }
    for (i=0;i<=n+1;i++)
    {
        b[i][0]=0;
        b[i][n+1]=0;
        b[0][i]=0;
        b[n+1][i]=0;
    }
    mcount=0x7fffffffl;
}

void print (char *ID)
{
    int i;
    printf ("#FILE xor %s\n", ID);
    printf ("%d\n", count);
    for (i=0;i<mcount;i++)
    {
        printf ("%d %d %d %d\n", msols[i][2], msols[i][3], msols[i][0],
msols[i][1]);
    }
    printf ("%d\n", mcount);
}

int main(int argc, char **argv)
{
    read_data ();
    delete_min_row_or_col ();
    delete_one_by_one ();
    print (argv[1]);
    return 0;
}
```

E. Source code for XOR (SensQ Award Winner)

The contestant Tiankai Liu gets 100 points(99.7 pts, exactly) in Task XOR. His solution is the best except the test case 3. (his solution uses 29 XORs. The best one uses 28 XORs.)

Tiankai LIU (USAC03)

```
#include <stdio.h>
#include <assert.h>

int N, K=0, Waas=0;
int color[2000][2000];
int needy[2001][2001] = {0};

FILE *fin, *fout;


void get_needy (int i, int j)
{
  if (i > 0) {
    if (j > 0)
      needy[i][j] += color[i-1][j-1];
    if (j < N)
      needy[i][j] += color[i-1][j];
  }
  if (i < N) {
    if (j > 0)
      needy[i][j] += color[i][j-1];
    if (j < N)
      needy[i][j] += color[i][j];
  }
  needy[i][j] &= 1;
}

void use_rect (int i, int j, int ii, int jj)
{
  fprintf (fout, "%d %d %d %d\n", j+1, jj, i+1, ii);
  // notice the tricky +1 and lack of +1
  K++;
  needy[i][j] ^= 1;
  needy[i][jj] ^= 1;
  needy[ii][j] ^= 1;
  needy[ii][jj] ^= 1;
}

int main (int argc, char *argv[])
{
  int i, j, ii, jj, r, c;
  int ninrow, nincol;
  int inrow[2000], incol[2000];
  bool found;

  // get input and figure out what is needy
  fin = fopen (argv[1], "r");
  assert (fin);
  fscanf (fin, "%d", &N);
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      fscanf (fin, "%d", &color[i][j]);
      get_needy (i, j);
    }
    get_needy (i, N);
  }
  for (j = 0; j <= N; j++)
    get_needy (N, j);
  fclose (fin);
  fout = fopen (argv[2], "w");
  assert (fout);

  // try to do stuff
  for (i = 0; i <= N; i++) {
    for (j = 0; j <= N; j++) {
      if (!needy[i][j])
       continue;

      assert (i < N);
      assert (j < N);

      ninrow = 0;
      for (jj = j+1; jj <= N; jj++)
       if (needy[i][jj])
         inrow[ninrow++] = jj;

      assert (ninrow & 1);  // should be odd

      nincol = 0;
      for (ii = i+1; ii <= N; ii++) {
       if (needy[ii][j])
         incol[nincol++] = ii;
      }

      assert (nincol & 1);

      found = 0;
      for (c = 0; !found && c < nincol; c++) {
       for (r = 0; !found && r < ninrow; r++) {
         if (needy[incol[c]][inrow[r]]) {
           // found the perfect rectangle
           found = 1;
           use_rect (i, j, incol[c], inrow[r]);
         }
       }
      }

      if (!found) {
       // no perfect rectangle, go for inferior stuff
       //     printf ("Waa. ");
       Waas++;
       use_rect (i, j, incol[0], inrow[0]);
      }
    }
  }

  // sanity check
#ifndef NDEBUG
  for (i = 0; i <= N; i++)
    for (j = 0; j <= N; j++)
      assert (needy[i][j] == 0);
#endif
  fclose (fout);

  // give some feedback
  printf ("Number of rectangles used: %d;   Number of Waas: %d;  Score
>= %g\n\n", K, Waas, 1 + 9 * (double) (K * 4 - Waas) / (K * 4));

  return 0;
}
```

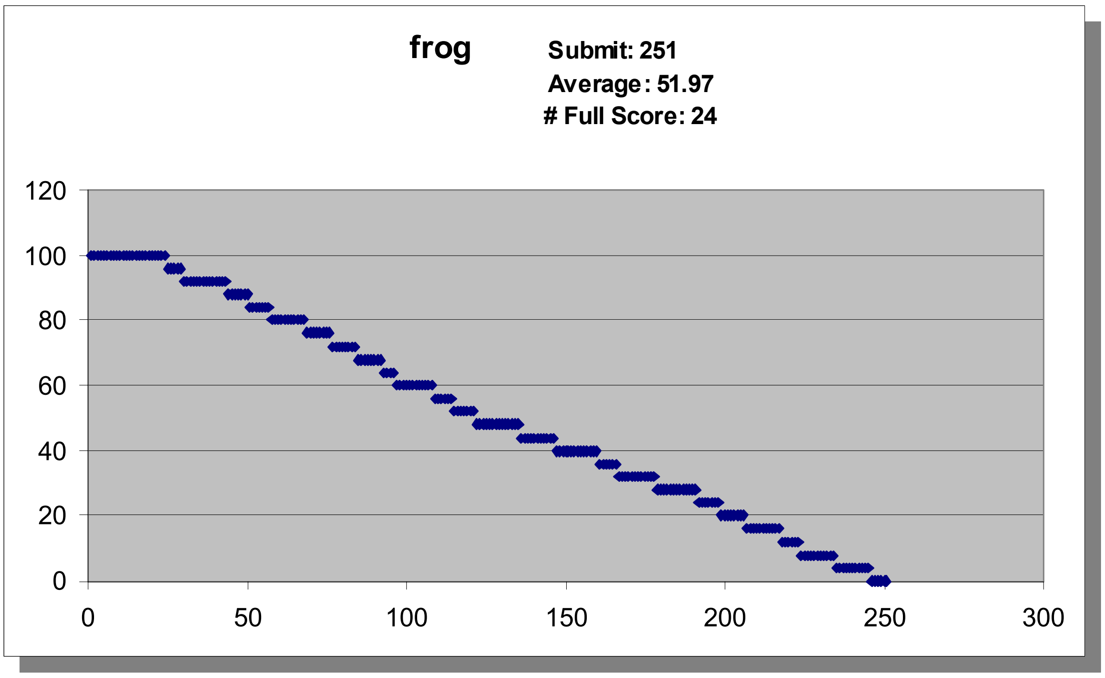
Result of Day1 Competition

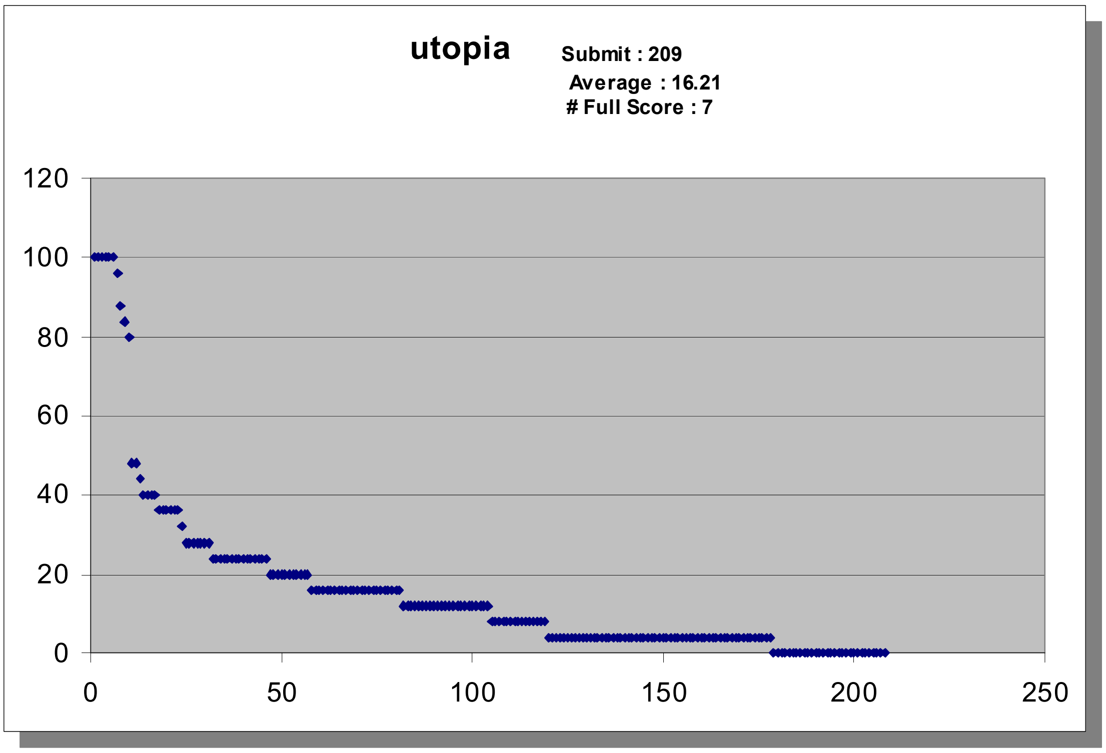
A. Summary

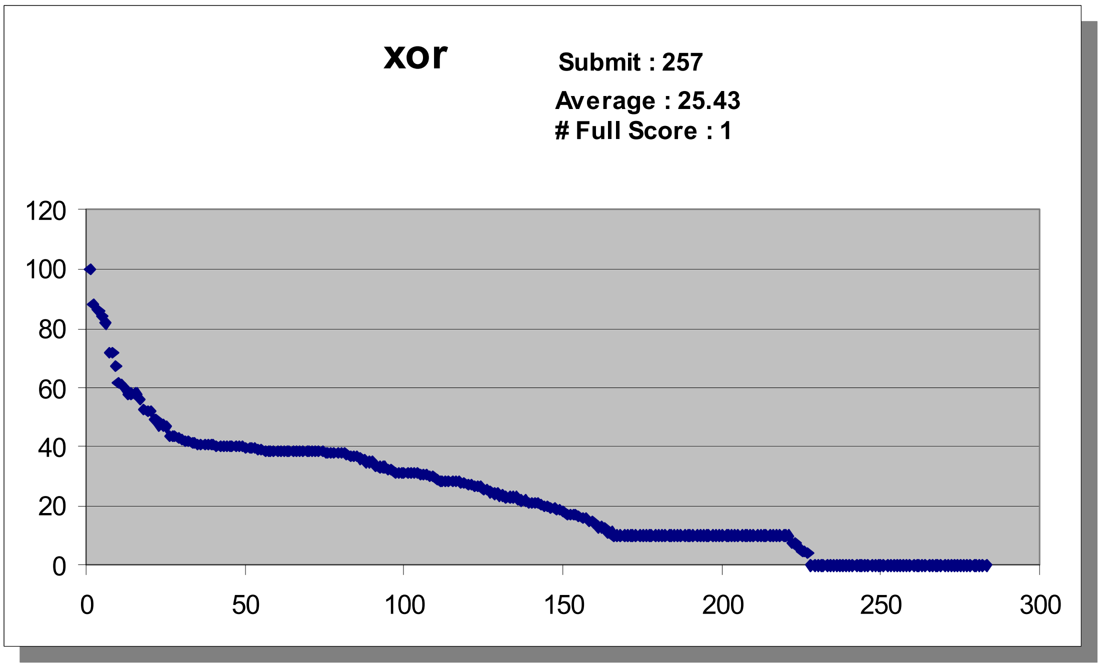
Task Name	Submission	# of full scores	Average	Standard deviation
FROG	251	24	51.97	30.97
UTOPIA	209	7	16.21	21.56
XOR	257	1	25.43	18.96

Note: The averages and standard deviations are calculated from submitted solutions only.

B. Contestants' Scores (sorted to X-axis)




utopia
Submit : 209
Average : 16.21
# Full Score : 7
120
100
80
60
40
20
0
0
50
100
150
200
250


xor
Submit : 257
Average : 25.43
# Full Score : 1
120
100
80
60
40
20
0
0
50
100
150
200
250
300

DAY2

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.

Task 1: BATCH

Hee Chul Kim, Jyrki Nummenmaa

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.

A. Solutions

This problem can be solved using dynamic programming. Let C_i be the minimum total cost of all partitionings of jobs $J_i, J_{i+1}, \dots, J_N$ into batches. Let $C_i(k)$ be the minimum total cost when the first batch is selected as $\{J_i, J_{i+1}, \dots, J_{k-1}\}$. That is, $C_i(k) = C_k + (S + T_i + T_{i+1} + \dots + T_{k-1}) * (F_i + F_{i+1} + \dots + F_N)$.

Then we have that
$C_i =$ min $\{ C_i(k) \mid k = i+1, \dots, N+1\}$ for $1 \le i \le N$,
and $C_{N+1} = 0$.

(a) $O(N^2)$ Time Algorithm

The time complexity of the above algorithm is $O(N^2)$.

(b) $O(N)$ Time Algorithm

Investigating the property of $C_i(k)$, P. Bucker[1] showed that this problem can be solved in $O(N)$ time as follows.

From $C_i(k) = C_k + (S + T_i + T_{i+1} + \dots + T_{k-1}) * (F_i + F_{i+1} + \dots + F_N)$, we have that for $i < k < l$,

$$C_i(k) \le C_i(l) \Leftrightarrow C_l - C_k + (T_k + T_{k+1} + \dots + T_{l-1}) * (F_i + F_{i+1} + \dots + F_N) \ge 0$$
$$\Leftrightarrow (C_k - C_l) / (T_k + T_{k+1} + \dots + T_{l-1}) \le (F_i + F_{i+1} + \dots + F_N)$$

Let $g(k,l) = (C_k - C_l) / (T_k + T_{k+1} + \dots + T_{l-1})$ and $f(i) = (F_i + F_{i+1} + \dots + O_N)$

Property 1: Assume that $g(k,l) \le f(i)$ for $1 \le i < k < l$. Then $C_i(k) \le C_i(l)$
Property 2: Assume $g(j,k) \le g(k,l)$ for some $1 \le j < k < l \le$ n. Then for each i with $1 \le i < j$, $C_i(j) \le C_i(k)$ or $C_i(l) \le C_i(k)$.

Property 2 implies that if $g(j,k) \le g(k,l)$ for $j < k < l$, C_k is not needed for computing F_i. Using this property, a linear time algorithm can be designed, which is given in the following.

Algorithm Batch

The algorithm calculates the values C_i for $i = N$ down to 1. It uses a queue-like list $Q = (i_r, i_{r-1}, \dots, i_2, i_1)$ with tail i_r and head i_1 satisfying the following properties:

$i_r < i_{r-1} < \dots < i_2 < i_1$ and
$g(i_r, i_{r-1}) > g(i_{r-1}, i_{r-2}) > \dots > g(i_2, i_1)$ -------- (1)

When C_i is calculated,

1. // Using $f(i)$, remove unnecessary element at head of Q.
   If $f(i) \ge g(i_2,i_1)$, delete i_1 from Q since for all $h \le i$, $f(h) \ge f(i) \ge g(i_2,i_1)$ and $C_h(i_2) \le C_h(i_1)$ by Property 1.
   Continue this procedure until for some $t \ge 1$, $g(i_r, i_{r-1}) > g(i_{r-1}, i_{r-2}) > \dots > g(i_{t+1}, i_t) > f(i)$.

   Then by Property 1, $C_i(i_{v+1}) > C_i(i_v)$ for $v = t, \dots, r-1$ or
   $r = t$ and Q contains only i_t.
   Therefore, $C_i(i_t)$ is equal to $\min\{C_i(k) \mid k = i+1, \dots, N+1\}$.

2. // When inserting i at the tail of Q, maintain Q for the condition (1) to be satisfied.
   If $g(i, i_r) \le g(i_r, i_{r-1})$, delete i_r from Q by Property 2.
   Continue this procedure until $g(i, i_v) > g(i_v, i_{v-1})$.
   Append i as a new tail of Q.

Analysis

Each i is inserted into Q and deleted from Q at most once. In each insertion and deletion, it takes a constant time. Therefore the time complexity is $O(N)$.

B. Test Data Information and Grading

Kee Moon Song

In total, 20 test cases are prepared and tested. Each test case is of 5 credits. Among them, 19 test cases are randomly generated so that negative integer by overflow does not occur during computing F_i. The remaining 1 test case is that setup time and all processing times and cost factors are 1.
The test case is mainly prepared to distinguish whether the competitors design an efficient algorithm or not. Among 20 test cases, algorithm by enumeration may solve for three ones within the given time limit, and an $O(N^2)$ time algorithm may solve for 14 test cases within the given time limit. If the competitors submit a correct $O(^N)$ time algorithm, they will get 100 credits.

Timing Test for BATCH

No.	Optimal in C/C++	Optimal in Pascal	SubOpt. in C/C++	SubOpt. in Pascal
	O(N)	O(N)	$O(N^2)$	$O(N^2)$
1	< 0.01	< 0.01	< 0.01	< 0.01
2	< 0.01	< 0.01	< 0.01	< 0.01
3	< 0.01	< 0.01	< 0.01	< 0.01
4	< 0.01	< 0.01	< 0.01	< 0.01
5	< 0.01	< 0.01	< 0.01	< 0.01
6	< 0.01	< 0.01	< 0.01	< 0.01
7	< 0.01	< 0.01	< 0.01	< 0.01
8	< 0.01	< 0.01	< 0.01	< 0.01
9	< 0.01	< 0.01	< 0.01	< 0.01
10	< 0.01	< 0.01	< 0.01	< 0.01
11	< 0.01	< 0.01	< 0.01	< 0.01
12	< 0.01	< 0.01	< 0.01	0.02
13	< 0.01	< 0.01	< 0.01	0.04
14	< 0.01	< 0.01	0.03	0.06
15	< 0.01	< 0.01	0.23	0.87
16	< 0.01	< 0.01	0.3	1.19
17	< 0.01	< 0.01	0.33	1.3
18	< 0.01	< 0.01	0.39	1.37
19	< 0.01	< 0.01	0.41	1.52
20	< 0.01	< 0.01	0.45	1.81

Testing Data Description for BATCH

No.	Size(N)	Description	Solution
1	N = 5	Example 2	153
2	N = 10	Randomly generated data	170820
3	N = 15	Randomly generated data	322305
4	N = 20	Randomly generated data	596614
5	N = 30	Randomly generated data	1414590
6	N = 50	Randomly generated data	3900980
7	N = 100	Randomly generated data	12636575
8	N = 200	Randomly generated data	50649757
9	N = 300	Randomly generated data	124220878
10	N = 500	Each time & priority = 1	135794
11	N = 700	Randomly generated data	635041453
12	N = 1000	Setup time = 0	331524426
13	N = 1500	Randomly generated data	744367663
14	N = 2000	Randomly generated data	863732491
15	N = 7000	Randomly generated data	757615479
16	N = 8000	Randomly generated data	1003361707
17	N = 8500	Randomly generated data	915744544
18	N = 9000	Randomly generated data	1042629359
19	N = 9500	Randomly generated data	925702728
20	N = 10000	Randomly generated data	1025371921

C. Variations

There are other several variations of the batch problem [2].

(1) If there is no restriction on the scheduled sequence of jobs, that is, a batch consists of arbitrary set of jobs, then the problem is NP-hard.

(2) If the cost factor of all jobs are all 1 and there is no restriction on the scheduled sequence of jobs, then the problem can be solved in O(N log N) time.

(3) If all jobs have the same processing time and there is no restriction on the scheduled sequence of jobs, then the problem can be solved in O(N log N) time.

D. References

[1] P. Brucker, **Efficient algorithm for some path problems**, *Discrete Applied Mathematics* 62, pp. 77-85, 1995.
[2] S. Albers and P. Brucker, **The complexity of one-machine batching problems**, *Discrete Applied Mathematics* 47, pp. 87-107, 1993.

E. Source Code for BATCH

Kee Moon Song

```
/*
TASK : Batch
LANG : C

    Optimal solution - O(N) with Dynamic programming
*/


#include <stdio.h>

#define MAX 10000

int num, stime, answer;             // # of Jobs, Setup Time, answer
int data[MAX][2];                   // processing time &  priority
int queue[MAX + 1], table[MAX + 1]; // tables for DP

void getdata()
{
       int i;

       scanf ("%d %d", &num, &stime);

       for (i = 0; i < num; i++)
              scanf ("%d %d", &data[i][0], &data[i][1]);
}

void preprocessing()
{
       int i;

       for (i = 1; i < num; i++)
       {
              data[i][0] += data[i - 1][0];
              data[i][1] += data[i - 1][1];
       }
}

int func(int k)
{
       return data[num - 1][1] - (k ? data[k - 1][1] : 0);
}

int cost(int i, int j
{
       return func(i) * (stime + data[j - 1][0] - (i ? data[i - 1][0] : 0));
}

int delta(int i, int j)
{
       return (table[i] - table[j]) / (data[j - 1][0] - (i ? data[i - 1][0] :
0));
}

void solveproblem
{
       int head = 0, tail = 1;
       int i, j;

       queue[0] = num;
       table[num] = 0;

       for (j = num - 1; j >= 0; j
       {
              for (i = head; i < tail - 1; i++)
```

```
                    if (func(j) > delta(queue[i + 1], queue[i])) head++;
                    else break;

              table[j] = table[queue[head]] + cost(j, queue[head]);

              for (i = tail - 1; i > head; i--)
                    if (delta(j, queue[i]) <= delta(queue[i], queue[i - 1]))
                          tail--;
                    else break;

              queue[tail++] = j;
       }
       answer = table[0];
}

void outputs()
{
       printf ("%d\n", answer);
}

int main()
{
       getdata();
       preprocessing();
       solveproblem();
       outputs();

       return 0;
}
```

Task 2: BUS

Chan-Su Shin, Djura Paunic

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, $2 \leq N \leq 500$, 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

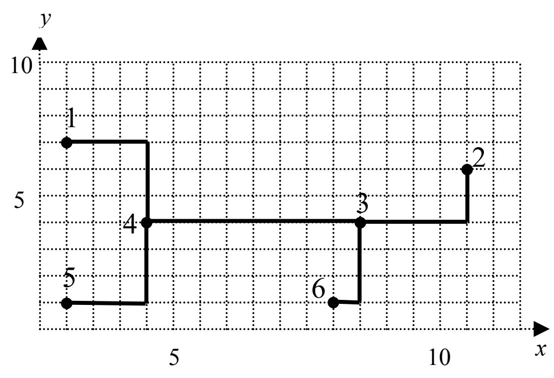
```
7
7 9
10 9
5 3
1 1
7 2
15 6
17 7
```

output

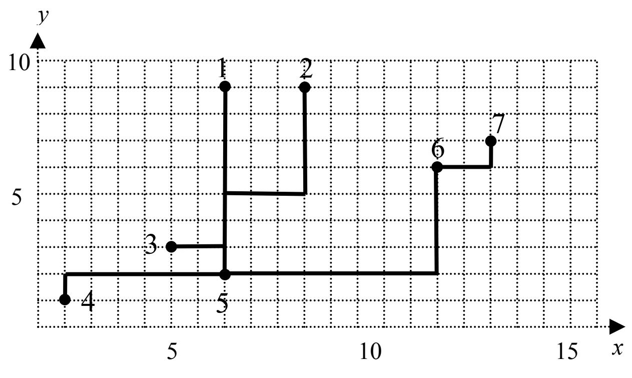
```
25
```

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.

A. Solution

(a) Algorithm Description

The solution is based on the algorithm, running in $O(N^3)$ time, presented in [1]. Recently, it is slightly improved in [2], but its implementation is too complicated to accept for the competition, so we use the algorithm in [1] as a solution.

The *diameter* of a bus network is the longest length of the route between any two bus stops in the bus network. Our goal is to find the minimum value of the diameters over all possible choices of the hubs and assignments of bus stops. As did in [1], we consider two cases separately. For it, we need some notations. Let D_1 be the minimum value of the longest length between two bus stops which are connected through only one hub over all possible choice of one hub, and let D_2 be the minimum value of the longest length between two bus stops which are connected through both two hubs over all possible choice of two hubs and the corresponding assignments of bus stops. We can now find the diameter in the following way presented in [1]. First, compute D_1 and D_2. Next, output the minimum of D_1 and D_2 as the minimum diameter of the entire network.

First we will explain the computation of D_1. If a point p will be served as the hub through which the longest route passes, the longest length is $d(p, q) + d(p, r)$, where the points q and r is the farthest and the second farthest ones from p, respectively. Then $D_1 = \min_p \{ d(p, q) + d(p, r) \}$ over all points p of the input. This can be obtained in $O(N^2)$ time because the farthest and second farthest bus stops for each point p are easily found in $O(N)$ time. Of course, we can reduce the time complexity to $O(N \log N)$ by using the second-order farthest Voronoi diagram. But we simply implement the $O(N^2)$-time algorithm because the complexity does not affect the total complexity of $O(N^3)$.

Second we will explain how to compute D_2 with a simple example. Note that in this case the longest route between two bus stops will pass both two hubs H_1 and H_2.

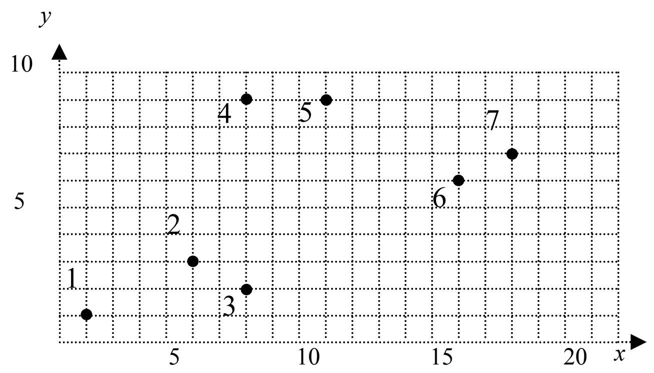


Figure 1

In Figure 1 it shows a distribution of 7 bus stops in Yong-In. We consider all pairs of bus stops of the input as possible two hubs H_1 and H_2, and select the pair of the bus stops that gives a minimum diameter. Let at the beginning D_2 be sufficiently large (e.g., `maxint`). Consider now fixed two hubs H_1 and H_2. Each of the remaining $N - 2$ points will be initially connected to one of two hubs, say H_1. Sort the remaining $N - 2$ points in the array

P in non-decreasing order according to the distance from the hub H_1(Figure 2).

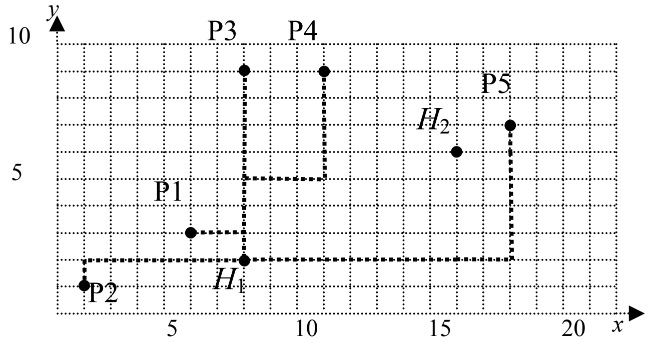


Figure 2

Denote by $r_1 = d(H_1, P[N\text{-}3])$, $r_2 = d(H_2, P[n\text{-}2])$ and $d_{12} = d(H_1, H_2)$. If $r_1 + d_{12} + r_2 < D_2$, then the point $P[N\text{-}2]$ is connected to the hub H_2 and set D_2 to the new value $D_2 = r_1 + d_{12} + r_2$. Figure 3 represents this step, $r_1 = d(H_1, P[N\text{-}3]) = d(H_1, P[4]) = 10$, $r_2 = d(H_2, P[N\text{-}2]) = (H_2, P[5]) = 3$, $d_{12} = d(H_1, H_2) = 12$, so $D_2 = r_1 + d_{12} + r_2 = 10 + 12 + 3 = 25$.

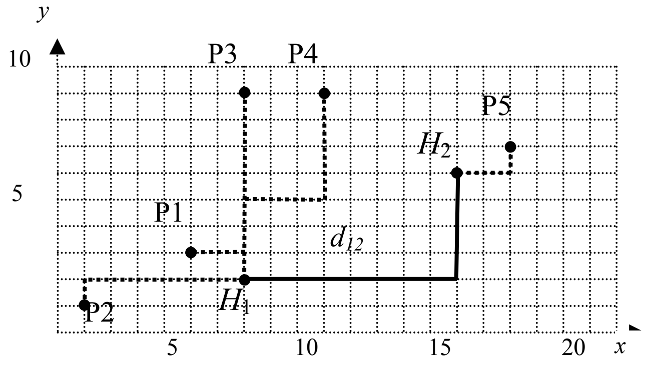


Figure 3

Now we repeat the same procedure with $r_1 = d(H_1, P[N\text{-}4])$, $r_2 = d(H_2, P[N\text{-}3])$, same $d_{12} = d(H_1, H_2)$, and get $r_1 + d_{12} + r_2 = d(H_1, P[3]) + d_{12} + d(H_2, P[4]) = 7 + 12 + 8 = 27$. Since we got the new distance which is greater than the previous diameter, the value D_2 remains unchanged, so D_2 still has value 25. (If 25 is turned out to be the minimum of D_2 at the end of the procedure, the point $P[4]$ shall be connected to H_1 although its distance to H_2 is smaller than the distance from H_1.) This situation is represented in Figure 4 where the point $P[4]$ is connected with a thin line to H_2 which is shorter than the distance from H_1 to $P[4]$.

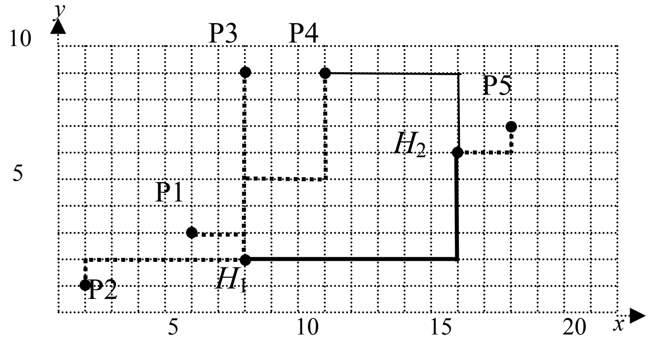


Figure 4

This procedure is repeated by decreasing the index of the array *P* one by one until the index 1 is reached. For the example, the minimum value of D_2 is 25 after the procedure and the corresponding network is shown in Figure5 below.

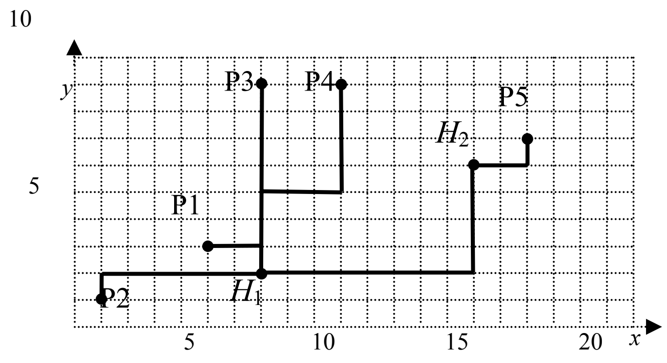
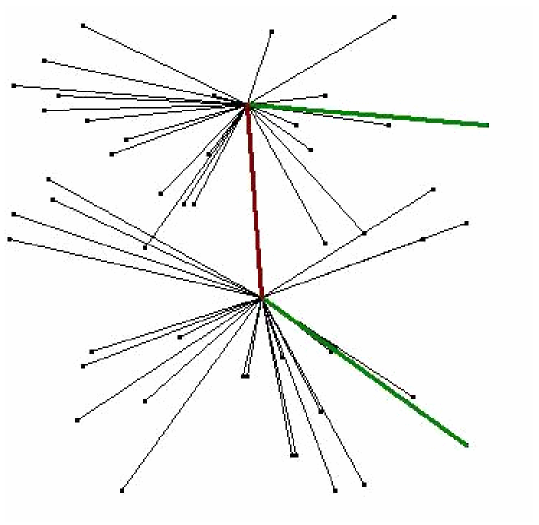


Figure 5

(b) Seemingly nice heuristic, but wrong approach

The correct and the nearly best algorithm by Ho et al. considers all $O(N^2)$ pairs of points as two bus hubs. Let $D(H_1, H_2)$ be the longest length between two bus stops for fixed two hubs H_1 and H_2. The main difficulty in this problem is how well contestants compute $D(H_1, H_2)$ for each pair (H_1, H_2). Many contestants will try to take a (seemingly natural and intuitive) heuristic approach to connect each of N - 2 points to the nearest one of two hubs. But this is wrong because there is a counter example shown in Figure 6. Of course, this approach can produce correct answers for some inputs.

The following two images are caught from the screen shots of the optimal network produced by the correct algorithm and the non-optimal network produced by the heuristic for the same input of 100 points. The left one has diameter 167 and the right one has diameter 168. The longest path defining the diameter is represented with thick lines. In the left image, some of pairs of edges are crossing, but this does not affect to the minimality.


(a) Optimal network by Ho et al's algorithm

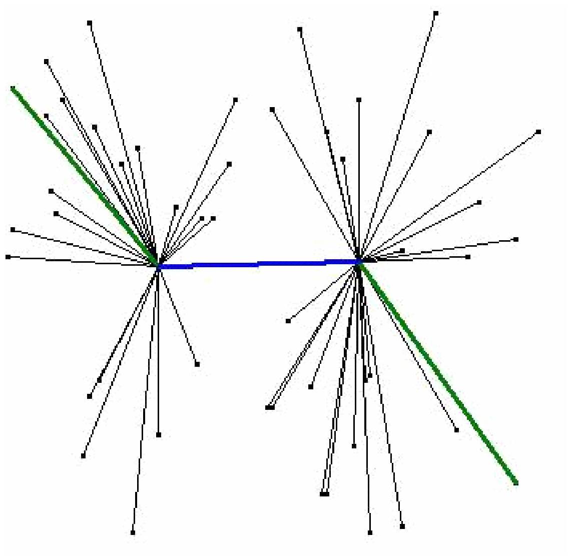

(b) Non-optimal network by heuristic

Figure 6

(c) Brute force approach

We can make a brute force algorithm running in $O(N^4)$ time. It considers all pairs of points as hubs H_1 and H_2, and computes $D(H_1, H_2)$ for each pair in $O(N^2)$ time.

B. Test Data Information

Kyung-Young Lim

In total, 20 data will be tested and each data is of 5 credits. All data are made with parameters n, the number of bus stops, in the range between 2 and 500, and the range of x-, y-coordinates between 10 and 5,000. Among them, 15 data are randomly generated. The remaining ones are designed to test the special cases. The data of test no. 1 consists of only two points, and the data of test no. 12 is 20 × 20 grid with a regular shape with easy solution. Three data, from test no. 9 to 11 have dumbbell-shape distributions – two points are lying on (-45)-degree, 45-degree, 0-degree line with fixed positions and the other points are randomly generated evenly around those two points.

The test data is mainly prepared to distinguish the heuristic and brute force programs with the correct one. Among 20 test data, only six ones have the same answers by both of the correct and heuristic programs. Hence, if competitors submit the heuristic program, they will get at most 30 credits. A brute force program running in $O(N^4)$ time will be successful only for the first six test data within the time limit. For the other test data, its running time will exceed the time limit, so it gets at most 30 credits.

The detail on the test data is summarized in the following table. The time limit is 4 second. (The largest running time is 3.343 second for the last test data.)

Testing Data Description for BUS

No.	Size (N)	Range of x,y	Description	Correct answer	Heuristic's answer
1	2	10	Extreme case	18	**18**
2	7	20	Example 2	25	26
3	10	30	Random	42	**42**
4	10	30	Random	52	53
5	50	100	Random	167	168
6	100	20	Random	35	36
7	170	1000	Random	1884	1896
8	180	1000	Random	1845	1849
9	300	650	Dumbbell, (-45)-degree	911	**911**
10	300	650	Dumbbell, 45-degree	995	**995**
11	400	675	Dumbbell 0-degree	689	**689**
12	400	20	20x20 grid	39	**39**
13	300	100	Random	186	188
14	300	1000	Random	1876	1882
15	350	150	Random	286	287
16	350	500	Random	945	946

17	350	2000	Random	3697	3709
18	400	1000	Random	1908	1912
19	400	5000	Random	9381	9405
20	500	500	Random	970	971

Timing Test for BUS (sec.)

No.	**Optimal in C/C++**	**Optimal in Pascal**	**Brute force in C/C++**	**Brute force in Pascal**
	N^3	N^3	N^4	N^4
1	0	0	0	0
2	0	0	0.01	0.01
3	0	0	0	0
4	0	0	0	0
5	0	0.01	0.07	0.07
6	0.01	0.04	0.97	0.98
7	0.07	0.13	4.54	8.02
8	0.08	0.16	5.67	10.03
9	0.37	0.9	36.66	65.38
10	0.39	0.91	> 50	> 50
11	0.83	1.95	> 50	> 50
12	0.79	1.85	> 50	> 50
13	0.33	0.76	> 50	> 50
14	0.33	0.75	> 50	> 50
15	0.51	1.2	> 50	> 50
16	0.52	1.2	> 50	> 50
17	0.52	1.18	> 50	> 50
18	0.8	1.8	> 50	> 50
19	0.77	1.75	> 50	> 50
20	1.5	3.343	> 50	> 50

C. References

[1] J.-M. Ho, D. T. Lee, C.-H. Chang, C. K Wong, **Minimum Diameter Spanning Trees and Related Problems**, *SIAM J. on Computing*, 20(5):987—997, 1991.

[2] T. Chan, **Semi-online maintenance of geometric optima and measures**, *13th ACM-SODA*, 474—483, 2002.

D. Source Code for BUS

Kyung-Young Lim

```
/*
TASK: BUS
LANG: C
Optimal Solution O(N^3)
*/

#include <stdio.h>
#include <math.h>

#define maxn 1001

int             n;
int             x[maxn], y[maxn];
int             p[maxn];
int             dis[maxn][maxn];
int             pivot;
int             min;

void input()
{
    int             i;
    scanf("%d", &n);
    for (i = 1; i <= n; i++) {
       scanf("%d", &x[i]);
       scanf("%d", &y[i]);
    }
}

void preprocess()
{
    int             i, j;
    for (i = 1; i <= n; i++)
       for (j = i; j <= n; j++) {
           if (i == j)
              dis[i][j] = 0;
           else
              dis[i][j] = abs(x[i] - x[j]) + abs(y[i] - y[j]);
           dis[j][i] = dis[i][j];
       }
}

void qsort(int s, int e)
{
    int             i, j, t;
    int             v;
    if (s < e) {
       v = dis[pivot][p[e]];
       i = s - 1;
       for (j = s; j <= e - 1; j++)
           if (dis[pivot][p[j]] <= v) {
              i++;
              t = p[i];
              p[i] = p[j];
              p[j] = t;
           }
       t = p[e];
       p[e] = p[i + 1];
       p[i + 1] = t;
       qsort(s, i);
       qsort(i + 2, e);
    }
}

void process()
{
    int             i, j, k;
    int             bp, bq;
    int             mm;
    for (i = 1; i <= n; i++)
       p[i] = i;
    min = 9999999;

    for (i = 1; i <= n - 1; i++) {
       pivot = i;
       qsort(1, n);
       for (j = i + 1; j <= n; j++) {

           if (min > dis[i][p[n]] + dis[i][p[n - 1]]) {
              min = dis[i][p[n]] + dis[i][p[n - 1]];
           }

           bp = p[n];
           bq = -99;

           for (k = n; k >= 2; k--) {


              if (dis[j][bp] < dis[j][p[k]]) {
                  bq = bp;
                  bp = p[k];
              }

              if ((bq == -99) || (dis[j][bq] < dis[j][p[k]]))
                  if (p[k] != bp)
                     bq = p[k];

              mm = dis[i][p[k - 1]] + dis[i][j] + dis[j][bp];
              if (k > 2)
                  if (mm < dis[i][p[k - 1]] + dis[i][p[k - 2]])
                     mm = dis[i][p[k - 1]] + dis[i][p[k - 2]];
              if (bq != -99)
                  if (mm < dis[j][bp] + dis[j][bq])
                     mm = dis[j][bp] + dis[j][bq];

              if (min > mm) {
                  min = mm;
              }
           }

           if (dis[j][bp] < dis[j][p[1]]) {
              bq = bp;
              bp = p[1];
           }
           if ((bq == -99) || (dis[j][bq] < dis[j][p[1]]))
              if (p[1] != bp)
                  bq = p[1];

           if (min > dis[j][bq] + dis[j][bp]) {
              min = dis[j][bq] + dis[j][bp];
           }
       }
    }
}

void output()
{
    printf("%d\n", min);
}

int main(void)
{
    input();
    preprocess();
    process();
    output();
    return 0;
}
```

Task 3: RODS

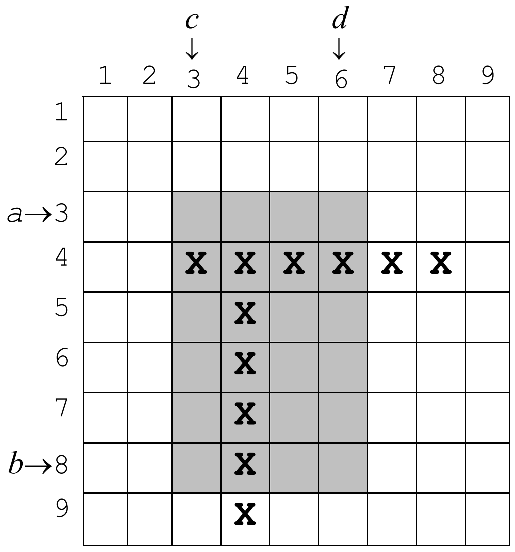
Hwan Gue Cho, Ian Munro

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*]×[*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*]×[*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.

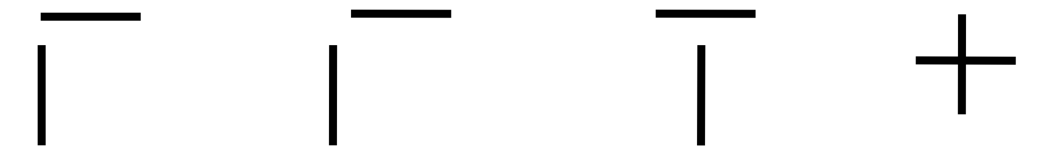
If your program's output is correct, then your score depends on the number of `rect` calls for each testing data. For each test case if the number of `rect` calls is at most 100, then you get 5 points. If your program calls `rect` 101 to 200 times, you get 3 points. If the number of `rect` calls is between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

A. Solution

Ian Munro

The fastest approach we know is to perform six binary searches.

- Using entire rows/columns as the query rectangle, the top and bottom rows, and leftmost and rightmost columns containing any portion of a rod can be found using 4 binary searches, or $4\lceil \lg N \rceil$ calls to `rect`.
- We now have the smallest rectangle containing all of both rods. By checking the corners of this rectangle (4 calls to `rect`, each with a 1 by 1 query rectangle), we can determine the general form of the structure, as in the examples of the figure and their rotations.
- Finally the solution can be found in one or two more binary searches depending on the case.


3 corners intersect rods
2 opposite corners intersect rods
2 adjacent corners intersect rods
no corners intersect rods

This leads to a $6\lceil \lg N \rceil + 4$ solution. The maximum value of $\lceil \lg N \rceil$ in the test data is 14, so we have a solution that takes at most 88 calls to rect. With care, this can be reduced to $6\lceil \lg N \rceil + 1$. Notice that the queries we ask of the data have only two possible answers. As there are $N^4(N\text{-}1)^2/4$ possible placements of the rods, $\lceil \lg (N^4(N\text{-}1)^2) \rceil \approx \lceil 6 \lg N \rceil - 2$ calls are necessary, on average, for any algorithm. We do not claim our testing is exhaustive, so we simply take this as a worst case lower bound. We implemented two versions of the general approach suggested.

There are a number of variations on this approach. For example, one could try to finding the bounding rectangle more quickly when it is large. Approaches of this type tend to double the number of calls to `rect` and will lead to full marks on the 4 small cases and 3 marks on each of the larger cases.

The most naïve approach involves scanning individual cells to find some portion of a rod, then looking around for the rest of the rod. This can clearly lead to an N^2 solution, or even slightly worse if one is careless. The approach receives full marks in the four cases of size 10. Another exhaustive approach involves taking entire rows (or columns) as query rectangles to find the bounding rectangle, then applying a similar approach to find the rods inside this rectangle. This will require $O(N)$ calls, though the constant will vary with details of the implementation. Depending on the details of the implementation, such an approach could also gain credit in several of the larger examples in which the rods are small and near a corner.

B. Test Data Information

Jung Gun Lim

Testing Data/Timing Description for RODS

No	Size(N)	Solution 1	Time(sec)	Solution 2	Time(sec)	6⌈lg N⌉+4
1	10	14	0.00	15	0.00	28
2	1000	51	0.00	51	0.00	64
3	5000	77	0.00	77	0.00	82
4	7000	75	0.00	76	0.00	82
5	10000	80	0.00	79	0.00	88
6	10	17	0.00	18	0.00	28
7	1000	50	0.00	51	0.00	64
8	5000	69	0.00	68	0.00	82
9	7000	77	0.00	76	0.00	82
10	10000	79	0.00	77	0.00	89
11	10	16	0.00	15	0.00	28
12	1000	62	0.00	63	0.00	64
13	5000	73	0.00	71	0.00	82
14	7000	79	0.00	76	0.00	82
15	10000	79	0.00	77	0.00	88
16	10	16	0.00	15	0.00	28
17	1000	52	0.00	51	0.00	64
18	5000	56	0.00	61	0.00	82
19	7000	65	0.00	64	0.00	82
20	10000	70	0.00	70	0.00	88

C. Source code for Library(LINUX)

Jung Gun Lim

```
#include "crectlib.h"
#define errmsgs 10
static const char *errmsg[errmsgs]={"Cannot open the input file.",
                                    "Invalid input file.",
                                    "N is out of range. (should be 5~10000)",
                                    "One of the coordinates is out of range.",
                                    "ROD1 is not valid.",
                                    "ROD2 is not valid.",
                                    "Cannot create the log file.",
                                    "Invalid library function call.",
                                    "Too many rect() calls.",
                                    "No rect() call."};


#define ERROPENINPUTFILE 0
#define ERRINVINPUTFILE 1
#define ERRNOUTOFRANGE 2
#define ERRCOORDINATEOUT 3
#define ERRFIRSTROD 4
#define ERRSECONDROD 5
#define ERRCREATELOGFILE 6
#define ERRINVCALL 7
#define ERRTOOMANYCALLS 8
```

```
#define ERRNORECTCALL 9

#define upperbound 400

#ifdef USERLIB

#define De(x) (x)
#define En(x) (x)

#define INFILE "rods.in"
#define OUTFILE "rods.out"
#define LOGFILE "rods.log"


static int rectlib_initialized=En(0);

static int sr1,
           sp2,
           sc1,
           sp1,
           sq2,
           sc2,
           sr2,
           sq1;

static FILE *lgfile, *oufile;
static int count, N;

#else

#define En(x) ((x) * 3 + 5)
#define De(x) (((x) - 5) / 3)

#define INFILE "rods.sin"
#define OUTFILE "rods.sout"
#define LOGFILE "rods.slog"

static int sr1,
           sp2,
           sc1;
static int rectlib_initialized=En(0);
static int count, N;
static int sp1,
           sr2,
           sq1,
           sq2,
           sc2;
static FILE *lgfile, *oufile;

#endif

static void erroroutput (int errno)
{
    FILE *f;
    f=fopen (OUTFILE, "w");
    printf ("%s\n", errmsg[errno]);
    fprintf (f, "%d\n%s\n", En(0), errmsg[errno]);
    fclose (f);
}


static void init()  // Read Data and initilization
{
    FILE *f;
    if (De(rectlib_initialized)) return;
    rectlib_initialized=En(1);

    count=En(0);

    f=fopen (INFILE, "rt");
    if (!f)
    {
        erroroutput (ERROPENINPUTFILE);
        exit (0);
    }

    if (fscanf (f, "%d", &N)!=1)
    {
        erroroutput (ERRINVINPUTFILE);
        exit (0);
    }

    if (De(N)<5 || De(N)>10000)
    {
        erroroutput (ERRNOUTOFRANGE);
        exit (0);
    }

    if (fscanf (f, "%d %d %d %d %d %d %d %d", &sr1, &sc1, &sr2, &sc2, &sp1,
&sq1, &sp2, &sq2)!=8)
    {
        erroroutput (ERRINVINPUTFILE);
        exit (0);
    }

    N=De(N);
    if (De(sr1)<1 || De(sc1)<1 || De(sr2)<1 || De(sc2)<1 || De(sp1)<1 ||
De(sq1)<1 || De(sp2)<1 || De(sq2)<1 ||
        De(sr1)>N || De(sc1)>N || De(sr2)>N || De(sc2)>N || De(sp1)>N ||
De(sq1)>N || De(sp2)>N || De(sq2)>N)
    {
        erroroutput (ERRCOORDINATEOUT);
        exit (0);
    }
    N=En(N);

    if (De(sc1)>=De(sc2) || De(sr1)!=De(sr2))
    {
        erroroutput (ERRFIRSTROD);
        exit (0);
    }
    if (De(sp1)>=De(sp2) || De(sq1)!=De(sq2))
    {
        erroroutput (ERRSECONDROD);
        exit (0);
    }
    fclose (f);

    lgfile = fopen(LOGFILE, "w");
    if (lgfile==0)
    {
        erroroutput (ERRCREATELOGFILE);
        exit (0);
    }
}

int gridsize ()
{
    if (!De(rectlib_initialized)) init ();
    fprintf (lgfile, "gridsize() = %d\n", De(N));
    return De(N);
}

int rect (int a, int b, int c, int d)
{
    int result;
    if (!De(rectlib_initialized)) init ();

    count=En(De(count)+1);

    fprintf (lgfile, "%d : rect (%d,%d,%d,%d) = ", De(count), a, b, c, d);
    if (a<1 || a>De(N) ||
        b<1 || b>De(N) ||
        c<1 || c>De(N) ||
        d<1 || d>De(N) ||
        a>b || c>d)
    {
        fprintf (lgfile, "%s\n", errmsg[ERRINVCALL]);
        fclose (lgfile);
        erroroutput (ERRINVCALL);
        exit (0);
    }

    if (De(count)>upperbound)
    {
        fprintf (lgfile, "%s\n", errmsg[ERRTOOMANYCALLS]);
        fclose (lgfile);
        erroroutput (ERRTOOMANYCALLS);
        exit (0);
    }

    if (De(sr2)>=a && De(sr1)<=b && De(sc2)>=c && De(sc1)<=d)
        result=1;
    else if (De(sp2)>=a && De(sp1)<=b && De(sq2)>=c && De(sq1)<=d)
        result=1;
    else
        result=0;

    fprintf (lgfile, "%d\n", result);
    return result;
}

void report (int r1, int c1, int r2, int c2, int p1, int q1, int p2, int q2)
{
    if (!De(rectlib_initialized)) init ();
    fprintf (lgfile, "report (%d,%d,%d,%d,%d,%d,%d,%d)\n", r1, c1, r2, c2, p1,
q1, p2, q2);
    fclose (lgfile);
    if (De(count)==0)
    {
        erroroutput (ERRNORECTCALL);
        exit (0);
    }
    oufile=fopen (OUTFILE, "w");
    fprintf (oufile, "%d\n%d %d %d %d\n%d %d %d %d\n",
             count, En(r1),En(c1),En(r2),En(c2),
                    En(p1),En(q1),En(p2),En(q2));
    fclose (oufile);
    exit (0);
}
```

D. Source code for RODS (Solution1 in C)

Jung Gun Lim

```
/*
TASK:RODS
LANG:C
*/
#include <stdio.h>
#include "crectlib.h"

int n;

void swi (int *a, int *b)  // swapping two integers
{
    int t=*a;
    *a=*b;
    *b=t;
}

// correcting and submit output
void soutput (int r1, int c1, int r2, int c2, int p1, int q1, int p2, int q2)
{
    if (c1==c2) // the first rod should be horizontal
    {
       swi (&r1, &p1);
       swi (&r2, &p2);
       swi (&c1, &q1);
       swi (&c2, &q2);
    }
    if (c1>c2) // correcting the order of output
    {
       swi (&c1, &c2);
    }
    if (p1>p2)
    {
       swi (&p1, &p2);
    }
    report (r1, c1, r2, c2, p1, q1, p2, q2);
}

// getting result of the rect() function with virtual coordinates.
int pseudo_rect (int r1, int r2, int c1, int c2);

// finding white space in the area by binary searching top to bottom
int top_to_bottom_search (int r1, int r2, int c1, int c2)
{
    int center;
    if (r1>r2) return r2;
    r1--;
    while (r1!=r2) {
        center=(r1+r2+1)/2;
        if (pseudo_rect(r1+1, center, c1, c2) == 0)
            r1=center;
        else
            r2=center-1;
    }
    return r2;
}

// bottom to top
int bottom_to_top_search (int r1, int r2, int c1, int c2)
{
    int center;
    if (r1>r2) return r1;
    r2++;
    while (r1!=r2) {
        center=(r1+r2)/2;
        if (pseudo_rect(center, r2-1, c1, c2) == 0)
            r2=center;
        else
            r1=center+1;
    }
    return r1;
}

// left_to_right
int left_to_right_search (int r1, int r2, int c1, int c2)
{
    int center;
    if (c1>c2) return c2;
    c1--;
    while (c1!=c2) {
        center=(c1+c2+1)/2;
        if (pseudo_rect(r1, r2, c1+1, center) == 0)
            c1=center;
        else
            c2=center-1;
    }
    return c2;
}

// right_to_left
int right_to_left_search (int r1, int r2, int c1, int c2)
{
    int center;
    if (c1>c2) return c1;
    c2++;
    while (c1!=c2) {
        center=(c1+c2)/2;
        if (pseudo_rect(r1, r2, center, c2-1) == 0)
            c2=center;
        else
            c1=center+1;
    }
    return c1;
}

int br1, br2, bc1, bc2; // boundary of rods

void boundary_search ()
{
    n=gridsize();
    br1=top_to_bottom_search (1, n-2, 1, n);
    br2=bottom_to_top_search (br1+3, n, 1, n);
    bc1=left_to_right_search (1, n, 1, n-2);
    bc2=right_to_left_search (1, n, bc1+3, n);
    br1++;
    bc1++;
    br2--;
    bc2--;
}

// flipping coordinates horizontally, vertically, diagonally
int h_flip=0, v_flip=0, d_flip=0;

// boundary with virtual coordinates
int pr1, pc1, pr2, pc2;

// initializing the virtual boundary
void pseudo_init ()
{
    if (h_flip)
    {
        pc2=(n+1)-bc1;
        pc1=(n+1)-bc2;
    }
    else
    {
        pc1=bc1;
        pc2=bc2;
    }
    if (v_flip)
    {
        pr2=(n+1)-br1;
        pr1=(n+1)-br2;
    }
    else
    {
        pr1=br1;
        pr2=br2;
    }

    if (d_flip)
    {
       swi (&pr1, &pc1);
       swi (&pr2, &pc2);
    }
}

int pseudo_rect (int r1, int r2, int c1, int c2)
{
    if (d_flip)
    {
        swi (&r1, &c1);
        swi (&r2, &c2);
    }
    if (v_flip)
    {
        r1=(n+1)-r1;
        r2=(n+1)-r2;
        swi (&r1, &r2);
    }
    if (h_flip)
    {
        c1=(n+1)-c1;
        c2=(n+1)-c2;
        swi (&c1, &c2);
    }
    return rect (r1, r2, c1, c2);
}

// submitting result with virtual coordinates

void pseudo_output (int r1, int c1, int r2, int c2, int p1, int q1, int p2, int
q2)
{
    if (d_flip)
    {
       swi (&r1, &c1);
       swi (&r2, &c2);
       swi (&p1, &q1);
       swi (&p2, &q2);
    }
    if (v_flip)
    {
        r1=(n+1)-r1;
        r2=(n+1)-r2;
        p1=(n+1)-p1;
        p2=(n+1)-p2;
    }
    if (h_flip)
    {
        c1=(n+1)-c1;
        c2=(n+1)-c2;
        q1=(n+1)-q1;
        q2=(n+1)-q2;
    }
    soutput (r1, c1, r2, c2, p1, q1, p2, q2);
}

// finding two rods in the boundary found.

void find_shape ()
{
    int l1, l2, l3, l4, rk, ck, rl, cl;

    l1=pseudo_rect (br1, br1, bc1, bc1);
    l2=pseudo_rect (br2, br2, bc1, bc1);
    l3=pseudo_rect (br1, br1, bc2, bc2);
    // watching 3 corners.

    if (l1+l2+l3==0) // cross shape
    {
        rk=top_to_bottom_search (br1+1, br2-2, bc1, bc1);
        ck=left_to_right_search (br1, br1, bc1+1, bc2-2);
        soutput (rk+1, bc1, rk+1, bc2, br1, ck+1, br2, ck+1);
    }
    // 0 or 1 or 3 of 4 corners could be filled
    if (l1+l2+l3==1)
        l4=1;
    else if (l1+l2+l3==3)
        l4=0;
    else
        l4=pseudo_rect(br2, br2, bc2, bc2);

    if (l1+l2+l3+l4==3)
    {
        if ((br2-br1)==1 && (bc2-bc1)==1)   // in 2 by 2 square
        {
            if (l1==0)
            {
                soutput (br2, bc1, br2, bc2, br1, bc2, br2, bc2);
            }
            if (l2==0)
            {
                soutput (br1, bc1, br1, bc2, br1, bc2, br2, bc2);
            }
            if (l3==0)
            {
                soutput (br1, bc1, br2, bc1, br2, bc1, br2, bc2);
            }
            if (l4==0)
            {
                soutput (br1, bc1, br2, bc1, br1, bc1, br1, bc2);
            }
        }

        if (l2==0 || l4==0) v_flip=1;
        if (l3==0 || l4==0) h_flip=1;
        if ((br2-br1)==1) d_flip=1;
        pseudo_init ();
//          *
//          *
//          *
//          *
//
//   ********
//
        if ((pc2-pc1)==1)
        {
            rk=bottom_to_top_search (pr1+2, pr2-1, pc2, pc2)-1;
            if (rk == pr2-1) rk=pr2;
            pseudo_output (pr1, pc2, rk, pc2, pr2, pc1, pr2, pc2);
        }
        if (pseudo_rect (pr2-1, pr2-1, pc2, pc2) == 0)
        {
            rk=bottom_to_top_search (pr1+2, pr2-2, pc2, pc2)-1;
            pseudo_output (pr1, pc2, rk, pc2, pr2, pc1, pr2, pc2);
        }
        else
        {
            ck=right_to_left_search (pr2, pr2, pc1+2, pc2-1)-1;
            if (ck==pc2-1) ck=pc2;
            pseudo_output (pr1, pc2, pr2, pc2, pr2, pc1, pr2, ck);
        }
    }
    if (l1+l2+l3+l4==2)
    {
        if ((l1==1 && l2==1) ||
            (l2==1 && l4==1) ||
            (l1==1 && l3==1) ||
            (l3==1 && l4==1))
        {
//
//   **********
//
//        *
//        *
//        *
//


            if (l2==1 && l4==1)
            {
                v_flip=1;
              d_flip=1;
            }
            if (l1==1 && l3==1)
            {
                d_flip=1;
            }
            if (l3==1 && l4==1)
            {
                h_flip=1;
            }
            pseudo_init ();
            rk=top_to_bottom_search (pr1+1, pr2-2, pc2, pc2) + 1;
            if (pc2-pc1==1)
                pseudo_output (pr1, pc1, pr2, pc1, rk, pc1, rk, pc2);
            ck=left_to_right_search (rk, rk, pc1+1, pc2-2)+1;
            if (ck==pc1+1) ck=pc1;
            pseudo_output (pr1, pc1, pr2, pc1, rk, ck, rk, pc2);
        }
        else
        {
            if (l1==0)
            {
                v_flip=1;
            }
            pseudo_init ();

//         *****
//
//                *
//                *

            if (pseudo_rect (pr1, pr1, pc1+1, pc1+1)==0)
            {
                rk=bottom_to_top_search (pr1+2, pr2-1, pc1, pc1)-1;
                cl=left_to_right_search (pr2, pr2, pc1+1, pc2-1)+1;
                pseudo_output (pr1, pc1, rk, pc1, pr2, cl, pr2, pc2);
            }
            else
            {
                ck=right_to_left_search (pr1, pr1, pc1+2, pc2-1)-1;
                rl=top_to_bottom_search (pr1+1, pr2-2, pc2, pc2)+1;
                pseudo_output (rl, pc2, pr2, pc2, pr1, pc1, pr1, ck);
            }
        }
    }
}

int main()
{
    boundary_search ();
    find_shape ();
    return 0;
}
```

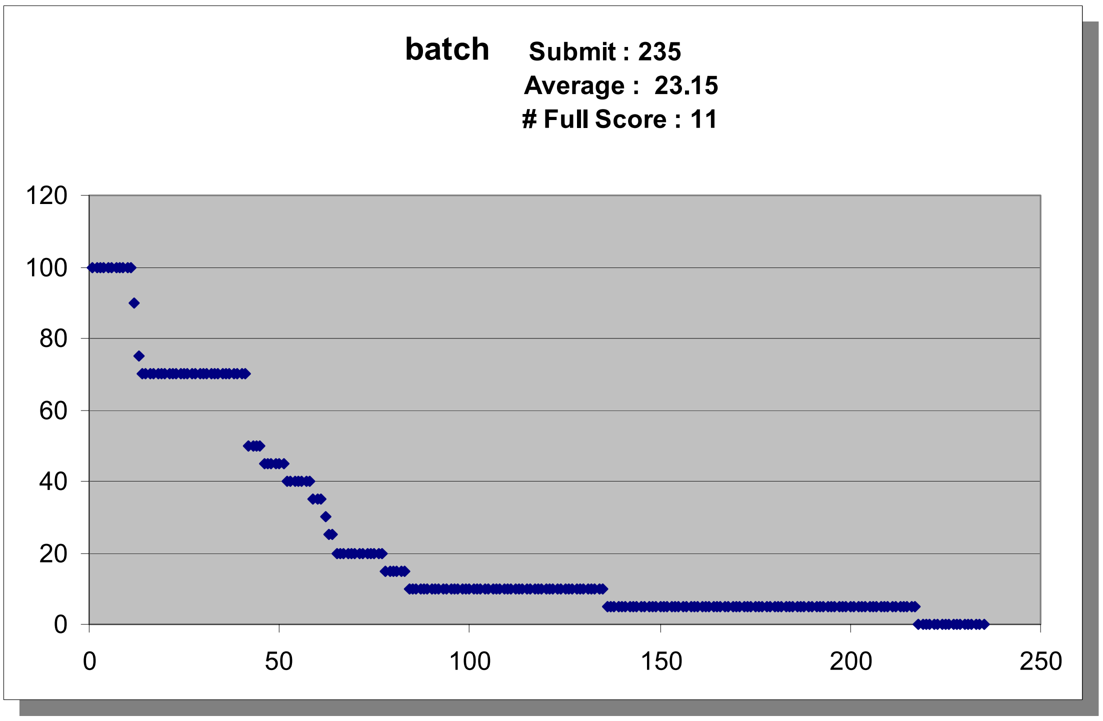
Result of Day2 Competition

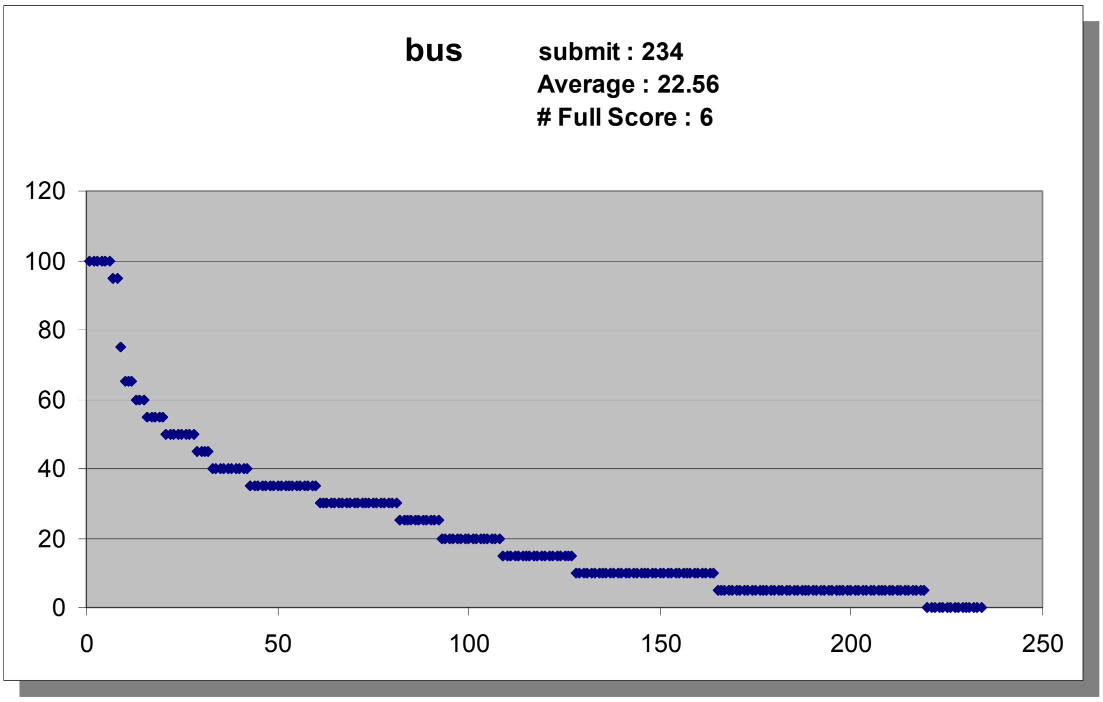
A. Summary

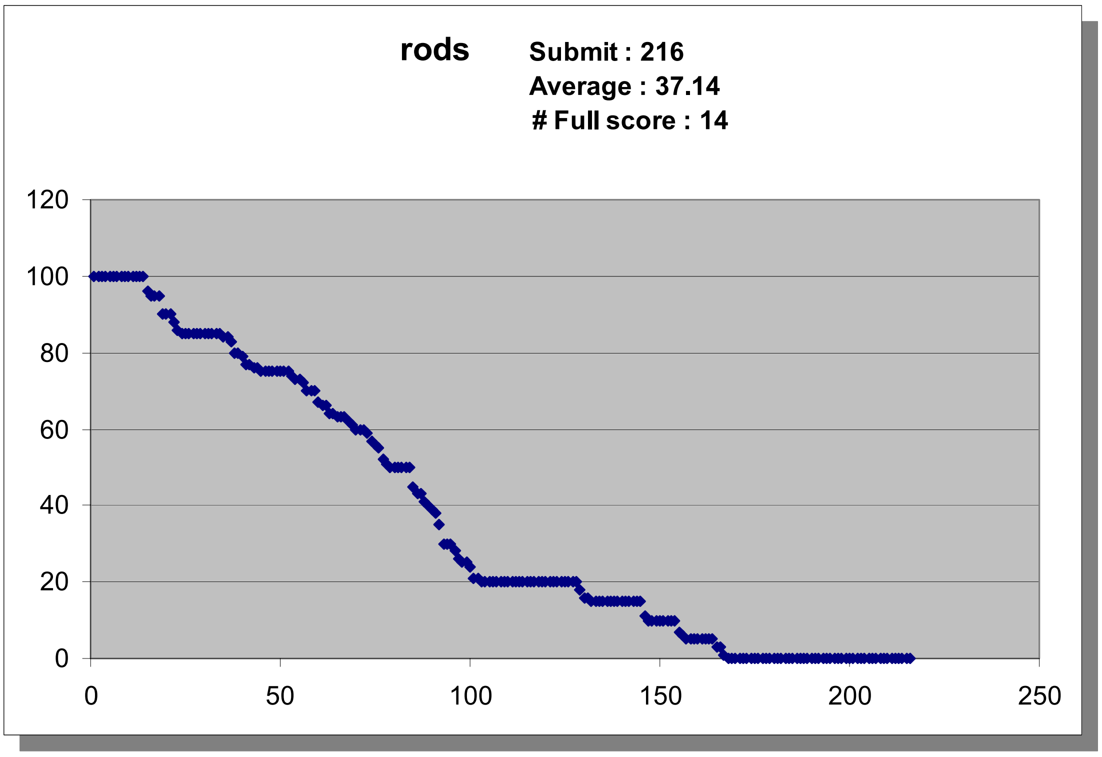
Task Name	Submission	# of full scores	Average	Standard deviation
BATCH	235	11	23.15	28.28
BUS	234	6	22.56	21.76
RODS	216	14	37.14	34.85

Note: The averages and standard deviations are calculated from submitted solutions only.

B. Contestants' Scores (sorted to X-axis)




bus
submit : 234
Average : 22.56
# Full Score : 6
120
100
80
60
40
20
0
0
50
100
150
200
250


rods
Submit : 216
Average : 37.14
# Full score : 14
120
100
80
60
40
20
0
0
50
100
150
200
250

BACKUP TASKS

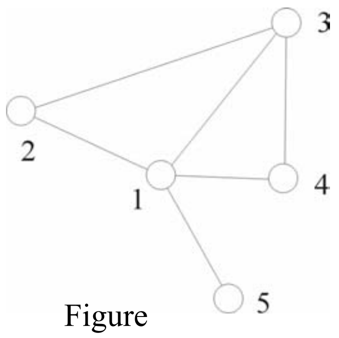
Back-Up Task 1: NETWORK

Jung-Heum Park

Robust Communication

PROBLEM
Your company has a head office in Seoul and a number of local branch offices located in other cities around Korea. Every office has a communication computer. The computer in each local branch is connected to the main computer in the head office via a bidirectional communication link. Furthermore, some pairs of computers in local branches are also connected to each other via direct communication links. However, there can only be a single communication link between any pair of computers. Messages sent from one computer to another can follow any continuous path between the computers, using any links and passing through any intermediate computers.

Figure 1 shows an example communication network of a company with one head office and four local branches. Circles represent communication computers and lines represent communication links. The number shown next to a circle is the index of that computer. The main computer always has index 1; the remaining computers are numbered sequentially starting at index 2. Observe that there are three different communication paths between computer 2 and computer 3:



Figure

2-3 (use the direct communication link between 2 and 3)
2-1-3 (use the link from 2 to the head office 1, plus the link from 1 to 3)
2-1-4-3 (from 2 to the head office 1, from 1 to 4, and from 4 to 3 over the direct link)

Unfortunately, computers or links can fail, knocking out communication paths that use them. In the example above, if the link from 1 to 3 were to fail, then path #2 would not be usable, but paths #1 and #3 would still work. If instead computer 4 were to fail, then path #3 would not be usable but paths #1 and #2 would still work.

Your company wants its communication network to be robust to a single failure, that is, designed so that neither a single computer failure nor a single communication link failure would interfere with the ability of each fault-free computer to communicate with all other fault-free computers. You may add bidirectional communication links between pairs of computers in local branches to achieve this goal, but each new link has an associated construction cost. Your task is to figure out the minimum total cost of any set of new links you could add to your communication network which would make the network robust to a single failure either of a computer or of a link (though not necessarily both).

INPUT

The first line contains the two integers n and m, where *N* is the number of computers in the network (3≤*N*≤1000) and m is the number of existing direct communication links between computers in local branches. The second line contains m pairs of integers (a_1,b_1), (a_2,b_2), …, (a_m,b_m) describing those direct links, where each pair (a_k,b_k) indicates that there is a bidirectional communication link between computer a_k and computer b_k in local branches. Recall that every branch office has a link to the head office; those links are not listed here. The following n lines contain the costs of building communication links: line *i*+2 in the input file contains *N* integers between 0 and 1000 inclusive which are the costs of building a link between computer *i* and each computer 1, 2, …, *N*, respectively. Note that since the communication links are all bidirectional, this table of costs will be symmetric (that is, $\text{cost}_{ij} = \text{cost}_{ji}$)).

OUTPUT

The output contains a single integer on one line: the minimum cost to make the communication network robust to a single failure.

EXAMPLE INPUT AND OUTPUT

Input

```
5 2
3 2 3 4
0 100 50 100 100
100 0 100 100 100
50 100 0 20 100
100 100 20 0 80
100 100 100 80 0
```

output

```
80
```

SCORING

For each test case, if the program outputs the minimum cost, full points are awarded; else no points are awarded for that test case.

A. Comment

The (Weighted) Biconnectivity Augmentation problem is NP-hard[1]. The computer network can be modeled by a graph G(V,E). We let H(V', E') be the graph obtained by removing the vertex representing the main computer and its associated edges from G. It holds true that G(V,E U X) is biconnected if and only if H(V', E' U X) is connected for any set X of pairs of computers in local branches. To find a minimum cost connected graph of H, employing a minimum cost spanning tree algorithm is sufficient

B. Test Data Information

The test data consists of 25 test cases, each generated mainly at random.

C. Grading

If a contestant's program outputs the correct answer for a test case in the time limit, then he/she will get 4 points for that test, and otherwise he/she get 0 points for the test case.

D. References

[1] M.R. Garey and D.S. Johnson, **Computers and Intractability: A Guide to the Theory of NP-Completeness**, Freeman, 1979.

Back-Up Task 2: DIAMOND

Tae Cheon Yang

Digital Diamond

PROBLEM

Given is an $m \times n$ grid of equal sized cells. Cells have row numbers counted from top to bottom and column numbers counted from left to right; top left corner has row number 1 and column number 1. In the grid, B cells are black and the rest is white. A digital diamond of size k ($k \geq 1$) is a square such that its diagonals are horizontal and vertical, and the sides of the square have k cells placed diagonally (see example).
.

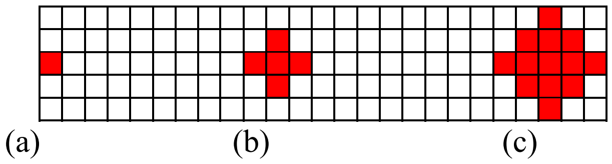


Figure 1. (a) A digital diamond of size 1
(b) A digital diamond of size 2
(c) A digital diamond of size 3

There is an $n \times m$ grid and some grids are filled with black as shown in Figure 2.

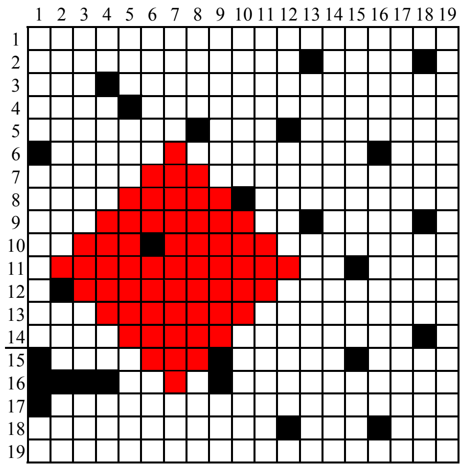


Figure 2 A digital diamond of size 6.

This problem is to find a largest digital diamond which contains at most one black grid.

INPUT

The first line contains two integers m and n ($1 \leq m, n \leq 100000$): first m the number of rows and then n the number of columns in the grid. The second line contains one integer B

($0 \leq B \leq 5000$) the number of black cells. The following B lines contain two positive each, the row number and the column number of the black cell separated by one blank.

OUTPUT

The output file contains the size of the digital diamond and the coordinate of the center of the digital diamond.

EXAMPLE INPUTS AND OUTPUTS

Example1: diamond.in

```
19 19
21
2 13
2 18
3 4
4 5
5 8
5 12
6 1
6 16
8 10
9 13
9 18
10 6
11 15
12 2
14 18
15 1
15 9
15 15
16 1
16 2
16 3
16 4
17 1
18 12
18 17
```

diamond.out

```
6
11 7
```

A. Comment

This problem can be reduced to the problem for the finding the largest empty circle in L_1-metric. So, the solution of this problem can be computed by constructing an L_1-metric Voronoi diagram.

We can consider a black grid as a point in Euclidean plane, then each black grid has integer coordinates. To construct an L_1-metric Voronoi Diagram of these points, we should compute the bisector between two points. Due to L_1-metric, the distance of every two points is an integer. To compute a bisector between two points, no floating point arithmetic is needed. Moreover, the slope of all bisector is either 1, -1, 0, or ∞.

Therefore we can easily compute an L_1-metric Voronoi diagram in $O(N \log N)$ by using divide-and-conquer technique. In this diagram, we can find the largest empty diamond which contains no black grid in $O(N)$. Let D be the size of this diamond and let D^* be the optimal size of this problem. Let D^*=D.

Using a Voronoi diagram of L_1-metric, we can solve our problem as follows. For every point *p*, we reconstruct the Voronoi diagram of L_1-metric partially without *p*, then update D*. This step can be done in O(*N*). Therefore, we can solve this problem in O(*N* log *N*). If we try to solve this problem directly in grid space, then at least $O(N^3)$ time algorithm may be needed.

B. Reference

[1] D.T. Lee and C. K. Wong, **Voronoi diagrams in L1 metrics with 2-dimensional storage applications**, *SIAM J. Computing*, 0:200-211, 1980

SUBMITTED TASKS

Submitted Task 1: ROBOT

Sam-Myo Kim

Robot

PROBLEM

There is a robot on a checkerboard (see Figure 1), which is divided into cells. The robot can read and write a symbol on the current cell (i.e., the cell where it is positioned), and move to its neighboring cell. Maneuvering this robot with a sequence of commands we want to draw a figure 8 of 8's as shown in Figure 2.

Each command should be expressed in one of the following three forms, where T and T′ can be either **R, L, U, D,** or **S,** respectively, denoting a move to the **right, left, up, down,** and **stop**.

(i) **<a, b, T> :** Reading symbol **a** on the current cell, rewrite it with **b** and move to the next neighboring cell in direction **T**. For example, **<B, 8, R>** denotes the command for "Reading **B** (for the blank symbol) on the current cell, rewrite it with **8**, and move to the right."

(ii) **{a, b, T}<c, d, T′> :** While reading **a**, keep moving to direction **T** after rewriting the symbol **a** with **b** until you read **c**. Then rewrite **c** with **d** and move to direction **T′. This command should satisfy the condition that a ≠ c.** For example, {8, 8, U}<B, 8, L> denotes the command for "While reading 8 on the current cell, keep moving up rewriting the 8 with 8 until you read B. Then rewrite the B with 8 and move on to the left cell."

(iii) **[*, i, j] :** Repeat previous i-th command through up to j-th command. For example, [*, 4, 9] is the command "Repeat previous fourth command through ninth command." **Using this command, we have the following restriction;** the first entry of the next command, if any, to this one and the first entry of **j**+1st command should be different. In other words, suppose that **<a, b, c>** is the next command to this repeat command and **<d, e, f>** is **j**+1st command. Then it must be **a ≠ d**.

It is possible to have **a = b** (or **c = d)** in the above commands, which means that the robot does not change the symbol. The robot executes one command at a time.

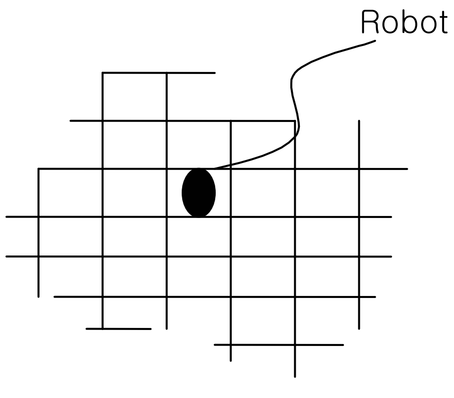


Figure 1

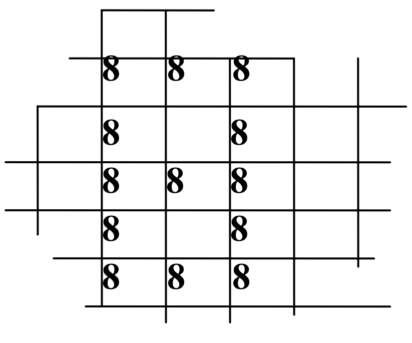


Figure 2

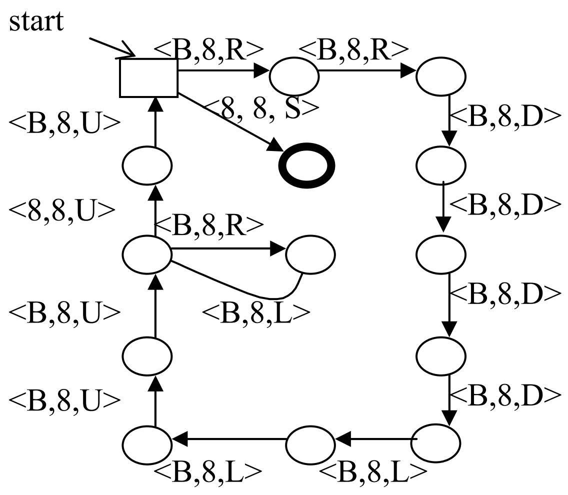
Assume that all the board cells are initially written with blanks, and the robot can draw the figure positioned at any location on the board. Now, we have the following questions:

Question (a): Under the restriction that the robot can only read and write two symbols, B (for blank) and 8, what sequence of commands (i.e., program) will you send to the robot to draw the figure? **The robot should stop when it is done with the work.** Your answer will be evaluated according to the correctness and brevity (i.e., in terms of the number) of your program.

Question (b): Now, suppose that the robot is allowed to read and write an extra symbol, say #, in addition to B and 8. How can you reduce the length of your commands? Again, your answer will be evaluated according to the correctness and brevity of your program.

A. Solutions and Remarks:

- We can find a naïve answer by drawing a graph having edges correspond to the commands given to the robot as shown in Figure (3) below. Obviously, the sequence of 15 commands (i.e., edges) is as show in Figure (4). Actually, the graph represents a finite state automaton that draws the figure. So, the problem can be interpreted as a robot motion planning as well as designing a finite state automaton. (Assuming that the participating students do not understand the automata theory, I wrote the problem in terms of robot motion planning, which can be dealt with their "bright" common sense.) Notice that the number of states does not necessarily match to the length of the program.



1. <B, 8, R>
2. <B, 8, R>
3. <B, 8, D>
4. <B, 8, D>
5. <B, 8, D>
6. <B, 8, D>
7. <B, 8, L>
8. <B, 8, L>
9. <B, 8, U>
10. <B, 8, U>
11. <B, 8, R>
12. <B, 8, L>
13 <8, 8, U>
14. <B, 8, U>
15. <8, 8, S>

Figure 3. A naïve motion planning. **Figure 4.** A naïve answer for part (a)

- Notice that the following "cleaver" answer utilizes symmetric property of the figure. (See **Figure 7** on the following page for the robot motion profile.)

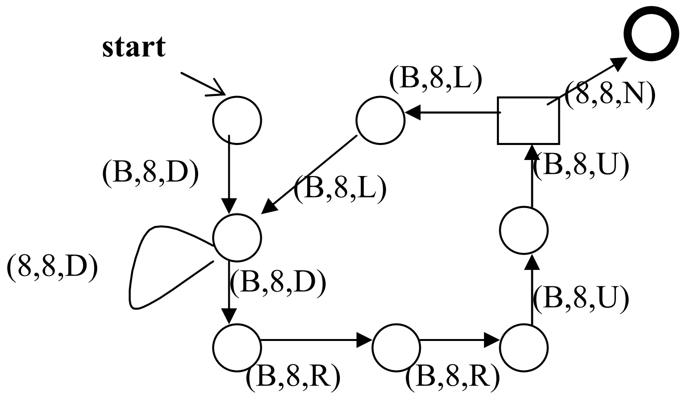


Figure 5. An intelligent motion planning for question (a).

1. <B, 8, D>
2. <B, 8, D>
3. <B, 8, R>
4. <B, 8, R>
5. <B, 8, U>
6. <B, 8, U>
7. <B, 8, L>
8. <B, 8, L>
9. {8, 8, D}<B,8,D>
10. [*, 3, 6]
11. <8, 8, S>

Figure 6. An intelligent answer for question (a).

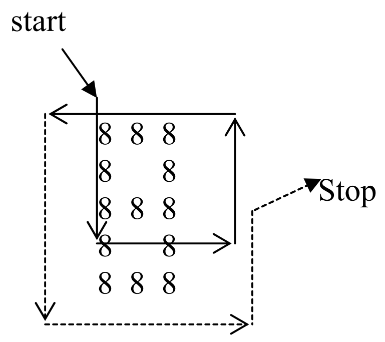


Figure 7. A Cleaver motion planning.

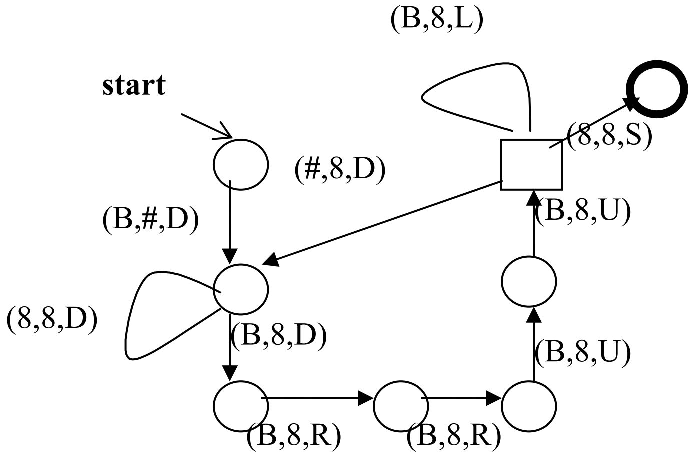


Figure 8. Motion planning using one additional symbol (i.e., flag #) for question (b).

1. <B, #, D>
2. <B, 8, D>
3. <B, 8, R>
4. <B, 8, R>
5. <B, 8, U>
6. <B, 8, U>
7. {B, 8, L}<#,8,D>
8. {8, 8, D}<B,8,D>
9. [*, 3, 6]
10. <8, 8, S>

Figure 9. An answer for question (b): Robot program with 10 commands using one additional symbol #.

- **Figures 10** and **11** show another motion plan and an answer for part (b). I designed this problem out of one of homework assignments given to my automata class. The original problem is to design a 2-D automaton with smallest possible number of states. There are more solutions than the ones shown here. However, as I commented before, with the set of 3 commands given in this problem, small automaton does not necessarily give small program.

- It is easy to see that with command (ii) **{a, b, T}<c, d, T'>** we can implement command (i), because if current symbol is **c**, which is not equal to **a**, then the iteration part **{a, b, T}** would not be executed. I kept the command (i) for convenience.

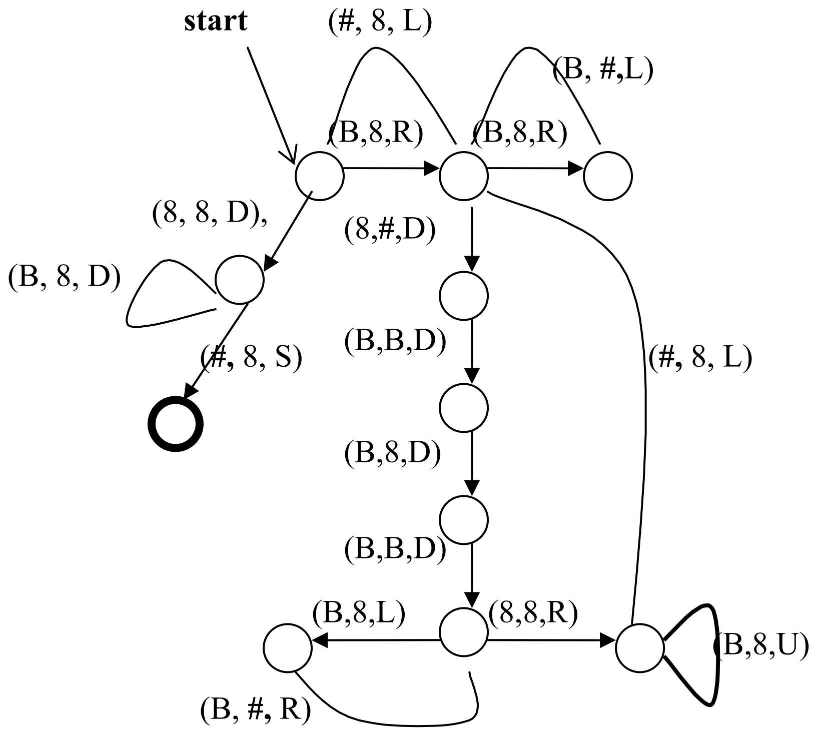


Figure 10. Another motion planning with flag #.

1. <B, 8, R>
2. <B, 8, R>
3. <B, #, L>
4. <8, #, D>
5. <B, B, D>
6. <B, 8, D>
7. <B, B, D>
8. <B, 8, L>
9. <B, #, R>
10. <8, 8, R>
11. {B, 8, U}<#, 8, L>
12. <#, 8, L>

Figure 11. Another answer for part (b) with 14 commands.

Submitted Task 2: PICTURE

Kunsoo Park

Picture

PROBLEM

You are given a small picture. You can carefully look at the given picture and remember every detail of the picture. Then you are supposed to go into a dark room. A large picture is hung on a wall of the dark room. Inside the large picture, there is at least one occurrence of the small picture given to you. You are to find (any) one occurrence of the small picture in the large picture. The problem is that you cannot see the large picture because you are in a dark room. You know only the size of the large picture. The only way you can access the large picture is by asking queries. You can ask a query to test whether or not a pixel of the small picture is the same as a pixel of the large picture. The answer will be either yes or no.

You are to write a program that, given a small picture and the size of a large picture, finds an occurrence of the small picture in the large one. If the output of your program is wrong, you will get 0. If your output is correct, your score depends on the number of queries. The query is implemented as a library. We assume that the pictures are squares.

LIBRARY

You are given a library with the following single operation:

`query`(x1,y1,x2,y2): (x1,y1) is the position of a pixel in the small picture; (x2,y2) is the position of a pixel in the large picture; if the two pixels are the same, `query`(x1,y1,x2,y2) returns 1. Otherwise, it returns 0.

INPUT

The input file name is `PICTURE.IN`. The first line of the input file contains two integers *m* and *n*. The size of the small picture is *m***m* and that of the large picture is *n***n*. The following m lines of the input file contain rows of the small picture.

OUTPUT

The output file name is `PICTURE.OUT`. The output file contains one line that consists of two integers *x* and *y*. The position (*x*,*y*) in the large picture should be an occurrence of the small picture.

EXAMPLE INPUT AND OUTPUT

If the large picture is the following, the answer should be (1,2) or (2,3).

```
Example 1: picture.in              large picture

           2 5                     aabcd
           ab                      aaaba
           aa                      baaab
                                   aaaab
                                   ababa
```

A. Solution and testing data

There can be three levels of solutions

(a) Very hard to get this solution

Consider the small picture as a set of its rows. Build a trie that represents the set of rows. Ask queries to see if some specified rows of the large picture contain a row of the small picture. Every m-th row of the large picture is a specified row (thus there are n/m specified rows in the large picture). If a specified row of the large picture contains a row of the small picture, then check whether there is an occurrence of the small picture. This gives an optimal expected time algorithm. See Karkkainen and Ukkonen.

(b) Pretty good solution

Determine a fingerprint of the small picture. A simple fingerprint can be the first row of the small picture. Search the large picture for fingerprints (here one needs to use a fast algorithm such as Horspool). For each fingerprint, check whether it is an occurrence of the small picture.

(c) Naïve algorithm.

For each position of the large picture, check whether it is an occurrence of the small picture.

B. Problem Type

Reactive problem

C. References

[1] Karkkainen and Ukkonen, **Two and higher dimensional pattern matching in optimal expected time**, *SIAM J. Comput*. 29, 2 (1999), 571-589.

[2] Horspool, **Practical fast searching in strings**, *Software-Practice and Experience* 10 (1980), 501-506.

Submitted Task 3: BRIDGE

Chong-Dae Park

Bridge Construction

PROBLEM

A long time ago, in an ocean far far away, there was a small kingdom with 101 islands. The king of the kingdom wanted to build bridges to connect the islands. A large project is initiated to connect these islands by 100 bridges. As a chief engineer of the construction post, you had to make an arrangement plan. There were some restrictions for the arrangement. A bridge had to be straight and not to cross one another. Moreover, the direction of a bridge should be orthogonal, i.e., it should lie from south to north or from east to west. You would get the awards if your attempt was a success, but you might be punished if you were failed. Unfortunately it is not possible to obtain such an arrangement in any case. I wish you a good luck.

INPUT

The input file name is `bridge.in`. The first line contains one integer: the number of islands in the kingdom, M, $2 \le M \le 200$. The following M lines contain information about the positions (x, y) of M islands ($1 \le x, y \le 1000$).

OUTPUT

The output file name is `bridge.out`. The output file contains M-1 lines. Each of these lines contains two integers: the pair of islands to connect. The islands are numbered from 1 to M. If such an arrangement is not possible, the output should be just a single line that contains “0”.

EXAMPLE INPUTS AND OUTPUTS

`Example1: bridge.in` `bridge.out`

```
9
3 3
6 3
5 5
8 5
6 6
8 6
5 8
3 9
6 9
```

```
1 2
3 4
5 6
8 9
1 8
7 3
4 6
9 5
```

```
Example 2: bridge.in                        bridge.out
```

```
5
1 2
1 1
2 8
5 2
2 1
```

```
0
```

A. Comment

This problem belongs to NP-hard.

Given a set P of n grid points in the plane, deciding whether P possesses a crossingfree spanning tree with only axis-parallel edges is NP-complete. [1]

And given a topological layout(i.e., a drawing in the plane) of a graph, the deciding the existence of a noncrossing path connecting path connecting two given vertices is NP-complete, even if the graph is 3-regular. [2]

B. Testing data generation plan

Although this problem belongs to NP-hard, many cases could be solved easily. The test should not contain many impossible cases, since the lazy program says just “0” may get higher scores.

C. References

[1] K. Jansen, and G. J. Woeginger, **The complexity of detecting crossingfree configurations in the plane**, *BIT,* 33(4):580-595, 1993.
[2] J. Kratochvil, A. Lubiw, and J. Nesetril, **Noncrossing subgraphs in topological layout**, *SIAM J. Discrete Mathematics*, 4:223-244, 1991.

APPENDIX I: 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.

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.

APPENDIX II: 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 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.

APPENDIX III: User Manual for IOI 2002

GENERAL

Contestant materials

Contestants will get the competition materials in the competition envelope. The envelope will contain a sheet for a user id and password for the web services. You need them to access the services.

Selecting operating system

You can select Linux or Windows XP at booting. Competition computers are configured for dual-booting. Use the cursor key to highlight your choice and type Enter key.

Login

You do not need to login to Windows XP. You can login to Linux with

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

Restarting your computer

In Windows XP, click 'Start' and 'Turn off computer'. From the menu that appears choose 'Restart'.

In Linux, you may press Ctrl-Alt-F1 to change console mode. (You can return to X-Windows mode by pressing Ctrl-Alt-F7.) Then you may press Ctrl-Alt-Del to restart.

If you need help

If you need any assistance (system trouble, go to toilet, whatever), please just raise your hand and wait for help.

PROGRAMMING

Comment tags for grading

Your program or output file (for output-only tasks) must have certain tags in comments for the grading system to identify the task and the programming languages (for source code file). For the source code submission, the syntax will be

```
TASK: taskname
LANG: LANGUAGE
```

where `LANGUAGE` is one of `C`, `C++`, or `PASCAL`. For the output only tasks, the syntax will be '`#FILE taskname data`'. See the task overviews and task descriptions for detail.

Range of integer variable

The ordinary type '`integer`' in freepascal has the range of -32768 to +32767. In some tasks, this may not be enough. Use 32bit variant '`longint`' type which has the range of -2147483648 to +2147483647.

Exit value

For C and C++ programmers: make sure that your program terminates with exit(0) or "return 0" in main(). Exiting with any other value is considered as an incorrect termination.

A normally terminated Pascal program returns a 0 when it terminates.

Differences between Linux and Windows XP

A program submitted for grading or testing will be compiled using the options given in the overview sheet. The compiled program will be executed on Linux.

Running programs on a Linux machine differs slightly from running programs on a Windows machine.

- If you access a pointer variable that points the memory outside your allocation, your program may stop abnormally.
- If you access outside the boundary of an array, your program may stop abnormally.
- Linux does not initialize local variables to any predictable value.

These differences mean that your program might work fine on Windows XP and fail on Linux. Be careful when using pointers, arrays and uninitialized variables to avoid these problems

TROUBLES

There are some inconveniences for programming in the competition environment. You can avoid these problems.

RHIDE and debugging problem in Windows XP
Trouble: If you set breakpoints and the program terminates without reaching any of the breakpoints, the DOS box will be terminated without even a message.
Solution: Set any breakpoints on the first or last line to reach in any case.

RHIDE and black screen in Windows XP
Trouble: If you launch RHIDE in full screen mode and you choose DOS shell, or execute the program, or exit RHIDE, you may see only a black screen.
Solution: Change to the window mode by pressing Alt+Enter. You may avoid this problem by launching RHIDE with '-S' option.

Freepascal IDE in Linux
Trouble: The function of viewing user screen in FP IDE has some trouble. It breaks the IDE editing screen.
Solution: Returning of editing mode makes the screen clean again. You may avoid this problem by using console compilation and debugging.

FORBIDDEN

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.

Submitted programs

- are not allowed to access the network,
- are not allowed to fork,
- are not allowed to read or create files directly,
- 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.

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

User Manual - Addendum

Timing Library will be provided

To measure the CPU usage of the program, we will provide a function called ‘`exectime`’. See **Contest System Users’ Manual Appendix B** for detail.

Online Help

C/C++ (including STL manual) and Pascal help file will be available at the competition web page.

Linux and Web Browser

Trouble: A large amount of output causes `mozilla` to freeze for several minutes.
Solution: When this happens, run the web browser “`opera`” and perform an action which replaces the contents of the box with the large amount of output (e.g., submit a different test program which produces less output).
Note: The contest system has not been tested with `opera`; use `mozilla` when possible.

Linux and FreePascal

Trouble: Running the Freepascal IDE from X-Windows, the terminal does not display correctly.
Solution: Switch to console mode via Ctrl-Alt-F1 before running the FreePascal IDE. (You should log in again). Ctrl-Alt-F7 switches back to X-Windows.

Linux

Trouble: The RHIDE and FreePascal IDEs do not recognize mouse actions in console mode.
Solution: Use the Alt-<KEY> hot keys to access the menus, or run RHIDE from X-Windows.

APPENDIX IV: IOI 2002 Contest System Users' Manual

Version 1.03

INTRODUCTION

The IOI 2002 Contest System is a group of server applications and modules designed to support International Olympiad in Informatics 2002. Its main functions are to support the contest by providing submit, test, print, backup/restore facilities during the contest and to support automated grading of participants' submissions after the contest. The participants of the contest are given web-interfaced contest supporting facilities during the contest. This manual is to provide contest participants with information about how to use the IOI 2002 Contest System during the contest. This manual does not contain information to setup and maintain the IOI 2002 Contest System for administering the contest.

USER INTERFACE

The user interface of the IOI2002 Contest System is web-based. A user can connect to the system with a web browser (Microsoft Internet Explorer or Mozilla) using the URL provided.

The IOI 2002 Contest System's user interface consists of three Web pages: login page, main page and restore page.

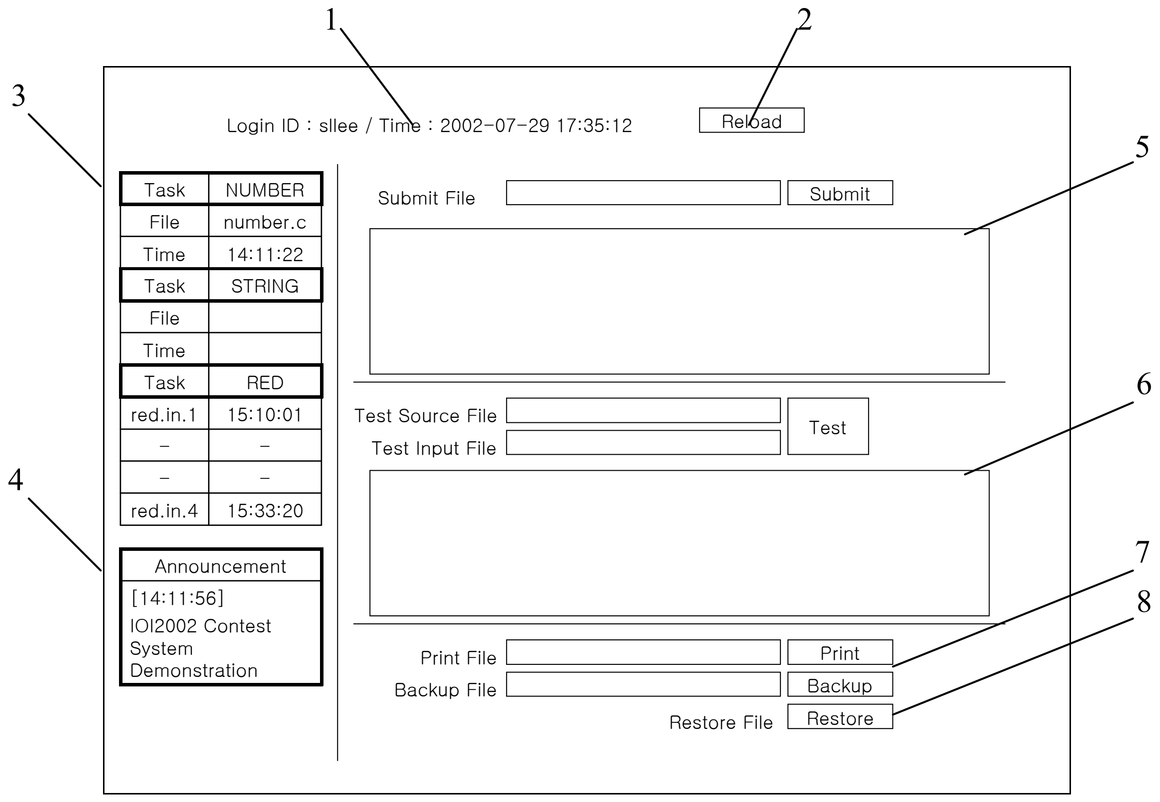
Login Page

When a user first connects to the IOI 2002 Contest System, a login screen with input boxes for *Login ID* and *Password* will be shown.
Type in login id and password and press *Login* button to open a new session with the IOI 2002 Contest System. Then the main screen will be displayed.

Contest is not running

When a contest is not running, which means that it is before or after the contest period, *Contest is not running* is displayed on the main screen and users cannot use submit or test facilities. Print and backup/restore facilities, however, are available even when a contest is not running. Viewing and downloading submitted files is also available when a contest is not running.

Main Page



Task	NUMBER
File	number.c
Time	14:11:22
Task	STRING
File	
Time	
Task	RED
red.in.1	15:10:01
-	-
-	-
red.in.4	15:33:20

Announcement
[14:11:56]
IOI2002 Contest System Demonstration

1 – Login and time information : The login ID of the user and the current time are shown. The current time is obtained from the system clock of the hardware where the IOI 2002 Contest System is running. Therefore the time shown is uniform among all participants regardless of their local machines' system clocks. **Note that the main page should be reloaded manually to get the current time display updated.**

2 – Reload button : Reload the main page from the web server and get the page updated. Users should manually reload the main page to see their submit or test operations in progress, to see the reports on submit or test operations when they are finished, or to update the current time and announcement window displayed. Pressing F5 button on the keyboard is equivalent to pressing the *Reload* button.

3 – Accepted submission table : The accepted submission table shows the task name, a filename, and the time the file is submitted for each of all the tasks of the currently running contest. Initially, only the task names are shown for the tasks of the day, and filenames and submission times are left blank. Filenames and submission times are filled only after a successful submission. A user can follow the HTML link at the filename to display its content or download the file. A new successful submission will overwrite the previous submission. There is no way for a user to recover an overwritten submission, except for starting a new submission process with the old file kept in his computer. **Note that when a user initiates a submission process that will turn out to be successful and the user never updates his main screen to see the updated submission table, the previous submission is still overwritten.**

There are two kinds of tasks. One accepts a single file (program) and the other (output-only task) accepts multiple files (output files). In the figure above, 'NUMBER' and

‘STRING’ are examples of single-file tasks, and ‘RED’ is an example of an output-only task which accepts 4 output files.

4 – Announcement window : The announcement window displays announcement messages from the administrator of the system. The time of announcement is also displayed. Every user sees the same message. **When the administrator puts up a new announcement message, users must reload the main screen manually to display the new message.** Thus it is advised that the announcement message does not contain any critical information.

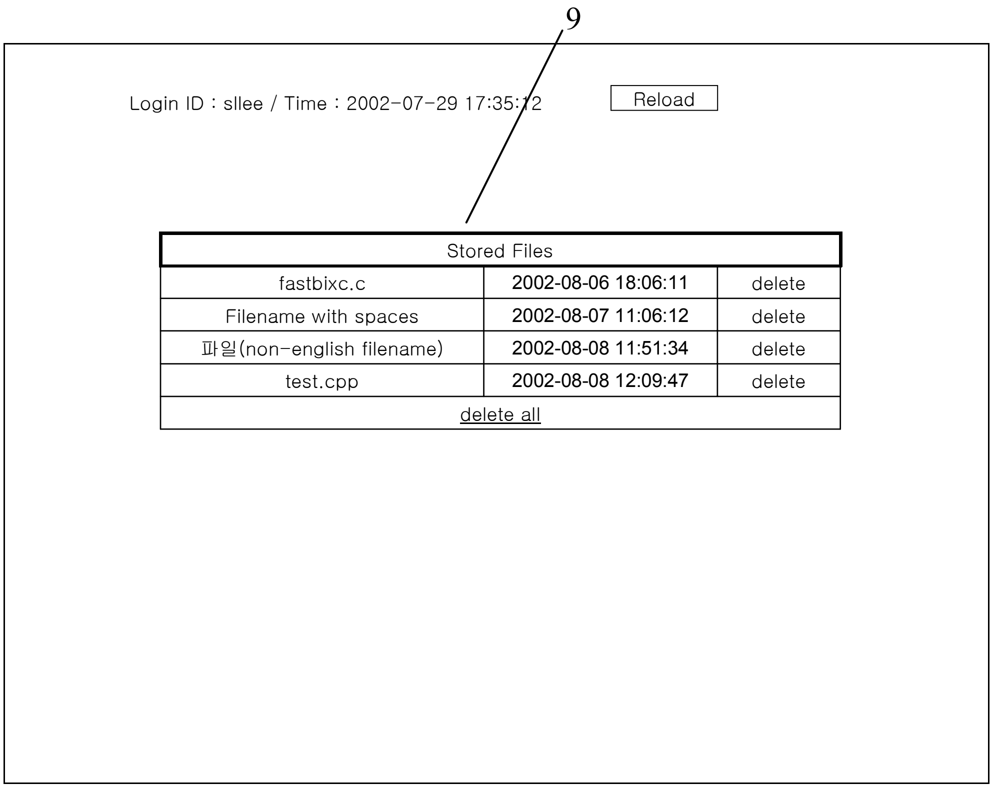
5 – Submit facility : It is used to initiate a submission process. Select a file to be submitted into *Submit File* box and press *Submit* button. During the submission process, the submit window is replaced by *Submission Progress… 0%* message. A user should reload the main screen manually to check the submission process progress and get submission results displayed.

6 – Test facility : It is used to initiate a test process. Select a source file to be tested into *Test Source File* box and its input file into *Test Input File* box, and press *Test* button. During the test process, the test window is replaced by *Test Progress… 0%* message. A user should reload the main screen manually to check the test process progress and get test results displayed.

7 – Print facility : A user may use *Print File* box and *Print* button to upload a text file to be printed.

8 – Backup/restore facility : *Backup File* box and *Backup* button can be used to store user’s file on the IOI 2002 Contest System server. The filenames of backup files are advised to be in English and follow UNIX filename conventions (alphanumeric, no white spaces). Press *Restore* button to display the restore page.

Restore Page



Login ID : sllee / Time : 2002-07-29 17:35:12

Stored Files		
fastbixc.c	2002-08-06 18:06:11	delete
Filename with spaces	2002-08-07 11:06:12	delete
파일(non-english filename)	2002-08-08 11:51:34	delete
test.cpp	2002-08-08 12:09:47	delete
delete all		

9 – Stored files table: The files stored in the IOI 2002 Contest System server using the backup facility in the main screen are listed in the stored files table. The files in the table are sorted by the backup time. The files can be displayed or downloaded by following the HTML links at the filenames in the first column of the table. Click *delete* on the right of the filename to delete the file from the server. The file will be permanently deleted. Clicking *delete all* at the bottom of the table will delete all the files stored in the server. A confirmation window will pop up for the delete options.

FEATURES

The IOI 2002 Contest System provides user facilities to help them through International Olympiad in Informatics. Users can submit solutions to tasks, test solutions, print files, and backup files. Readers are expected to have a good understanding of IOI contest rules. Readers who do not have sufficient knowledge are advised to read 'IOI 2002 Competition Rules' first.

Submit

A user may submit a source file or an output file using the submit facility on the main page. **Note that the main screen is not automatically updated as the submission process progresses. The user must manually reload the main screen (Either by pressing Reload button or F5 button on the keyboard) to get submission results**. However, once the user starts a submission, the submission process is internally processed

regardless of user's updating his or her main screen output. Thus even if the user does not check submission results (i.e. he/she never updates the main page or he/she immediately closes the browser after initiating a submission process or the user's computer is powered off immediately after initiating a submission process) the submission is processed unhindered and if the submission is accepted, it will replace the last submission. Also, reloading the main page frequently will not speed up the submission process.
The number displayed during a submission process as *Submission Progress… 0%* indicates the percentage of progression. When the number hits 90%, the submission process is actually started on the server side.
A user can have only one submission being processed at a time. A user should wait for his or her current submission process to finish before attempting to initiate another submission process. Note that logging out or turning off a user's machine will not cancel an ongoing submission process.
A submission is accepted only if it satisfies all the requirements for the task. The maximum size of a source file or an output file that can be submitted is 1M bytes (1048576 bytes). If the size exceeds, an error message will be immediately displayed and the submission process will not start at all. Other requirements for the task will be examined in the submission process. Failing to meet all the requirements will result in a submission failure. The table below contains common requirements for a submission to be accepted. **Note that a user program must return exit code of 0.** In C or C++, the user program should do "`return 0;`" or "`exit(0);`" at the end of the execution. In Pascal, the default return code is 0 unless specified otherwise in the source code.

Requirements for user programs	
Program exit code	0
Maximum source file size	1M bytes
Maximum output file size (output-only tasks)	1M bytes
Header format	Specified per task
Stdout size limit	200k bytes
Stderr size limit	200k bytes
Compile time limit	30 seconds
Execution time limit	Specified per task
Memory usage limit	Specified per task
Stack size limit	Default (8M bytes)

Test

The test facility is similar to the submit facility. The main difference is that a user must provide his or her own input file. The test facility is not available for an output-only task. The input file is fed to the user program as stdin and the user program is expected to use stdout and stderr for outputs. The user program's stdout and stderr will be shown in the *Test Output* window along with other information. All requirements for the task are examined as in the submission facility.
A test process might take more time than a submission process as submission processes are given much higher priority by the server. Only one test process is possible at a time. However, it is possible to proceed with one submission process and one test process at the same time.

Print

Select a file to be printed into *Print File* box and press *Print* button. *Print Successful* will be displayed if printing is successful and the printing job is fed to the printer spool (which means that it can still take some time before actual printing). *Print Failed* means the printing subsystem of the IOI 2002 Contest System is not available at the time and the user should try again some other time or consult with an assisting personnel. It may take some time before the main screen is displayed with either *Print Successful* or *Print Failed* when the file is big or there are many simultaneous printing requests from the users. (This is due to a system design to suppress excessive printing requests) The maximum size of the file to be printed is 1M bytes.

Backup

The backup/restore facility can be used to store user's files on the server. The files can be as large as 1M bytes and each user is guaranteed to use space for at least 10 files. Select a file to backup into *Backup File* box and press *Backup* button. The restore page will be displayed with the list of backup files on the server on a successful backup. An error message will be displayed on the main page on a failure. Press *Restore* button to display the restore page. A user may display or download backup files or delete one or all backup files. Press *Reload* button on the restore page or backspace key on the keyboard to go back to the main page.

SECURITY MEASURES

According to 'IOI2002 Competition Rules', submitted programs

- are not allowed to access the network,
- are not allowed to fork,
- are not allowed to read or create files directly,
- 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.

The IOI 2002 Contest System has features to enforce above rules. The competition rules can be enforced at the time a user tries to break the rules or afterwards by examining log files. The IOI 2002 Contest System supports both ways of enforcing the competition rules.

Resource limitations

A program that a user submitted is run under resource limitations. Process, memory, and output file size limitations are enforced by the resource limitations set on the user's executable file.

File accessibility

The running environment of a user program is set up so that the user program cannot access other files. Moreover, the hardware on which a user program is run is a separate system from the rest of the hardware including the server and it contains only a sample input file and a checker for the sample input.

Network security

The hardware where a user program is run is set up in a private network where it can only access the central system server. Any attempt to connect to the central system server is monitored from the server side. The private network itself is also packet-monitored.

Logging

All user programs submitted or tested to the server are kept with their output and activity log. The activity log of the system is kept in three different parts of the system and the three recordings are cross-checked after the contest.

CONTACT INFORMATION

The IOI 2002 Contest System is developed and maintained by:

Computer Theory & Cryptography Lab.,
School of Computer Science and Engineering,
Seoul National University, Seoul, Korea.

Email to sllee@snu.ac.kr for technical information.

APPENDIX A. ERROR MESSAGES

Web server messages

These messages are displayed in the main page.

Error Message	Explanation
Submission failed: Already processing	The user attempted to start a new submit process before the previous submit process is finished.
Submission failed: No file selected	No file is selected into the *Submit File* box. The file path specified is inadequate: retry after copying the file to some other place.
Submission failed: Contest not running	The contest is over and the user is not allowed to start a new submission process.
Test failed: Already processing	The user attempted to start a new test process

	before the previous test process is finished.
Test failed: Select two files	Not both files are present in the *Test File* box and *Test Input* box. The file path specified is inadequate: retry after copying the file to some other place.
Test failed: Contest not running	The contest will be over in less than 5 minutes and the user is not allowed to start a new test process.
Source code display failed: Please retry or consult administrator	Fail to follow the HTML link in the accepted submissions table. Check if the filename of the file submitted follows UNIX filename conventions and retry.
File upload interrupted: Please retry	The user canceled a file upload (possibly by pressing cancel button on the web browser) or a network error occurred.
Print Successful	A printing job is successfully spooled.
Print Failed	The printing subsystem is not in operation. Try again after some time. If the problem persists, consult administrator.
Unknown error	Try pressing reload button.

Submit, test output window messages

These messages are displayed inside the submit output window or the test output window. Other task specific messages are also displayed in the submit output window or the test output window. See the task description for each task for information on task specific messages.

Error Message	**Explanation**
! The system failed to process the job. Please consult administrator	Check your program and consult administrator.
[HEADER CHECK - OK]	
[HEADER CHECK - ERROR]	
[COMPILE – OK]	
[SAMPLE DATA TEST – OK]	
[SAMPLE DATA TEST – ERROR]	
output-only task need no testing	Test facility is not provided for the output-only tasks. Use the submit facility for format checking of output-only tasks.
header format error	Check header format.
task name is invalid	Check task name field in the header.
execution time limit exceeded!	User's program did not terminate within the execution time limit per task.
output size limit exceeded!	Output size limitation of 200k bytes for each of stdout and stderr is exceeded and execution of the program is stopped.
wrong answer	Failed to produce correct output with sample

	test data. This error message is shown in the submit output window only.
exit code is non-zero	All programs should have return code of 0. Put return 0; of exit(0); at the end of your code in case it is written in C or C++. For Pascal, the return code is always 0 unless you specified in the code differently.
Execution error (invalid memory reference)	Segmentation fault (possibly due to resource limitations)
...Submission Accepted!	
...Submission Failed!	The submitted file could not pass all submission tests and was not accepted. The submission is not added to accepted submission list of the user. In case there was a pervious accepted submission for the task, the previous submission is not overwritten and is still valid accepted submission.

APPENDIX B. MEASURING THE CPU USAGE OF A PROGRAM

Measuring the intermediate CPU usage of a program - Pascal users

The grading system will observe your program's execution time from outside. If you want to check intermediate execution times during test execution or submission execution, you may include this line in your code:
`{$i extime.inc}`
to include the execution function called "`exectime`". This function has no parameters and looks just like a scalar. Its value is the number of milliseconds of execution used so far. Here's a sample program to demonstrate its use:

```
program timetest;
{$i extime.inc}
var i, j, k:longint;
begin
k := 0;
    for i := 1 to 100 do
    begin
        for j:=1 to 1000000 do
        begin
            k := i + j + k;
        end;
    end;
    writeln(exectime);
end.
```

This facility is only available for test executions and submission executions.

Measuring the intermediate CPU usage of a program - C/C++ users

The grading system will observe your program's execution time from outside. If you want to check intermediate execution times during test execution or submission execution, you may use the ‘`exectime`’ function, which returns the number of milliseconds your program has used so far. Here is a sample program to demonstrate its use:

```
main()
{
    long i, j, k;
    k = 0;
    for (i = 0; i < 100; i++)
     for (j=0; j < 1000000; j++)
         k = i + j + k;
    printf("%d ms\n", exectime());
}
```

This facility is only available for test executions and submission executions.

APPENDIX V: List of Contestants

Gold Medalists (23)

CNTR	ID	Name	Day1	Day2	Score
KOR	KORC03	Wanyeong JUNG	260	250	510
POL	POLC01	Pawel PARYS	263	195	458
BGR	BGRC01	Velin TZANOV	213	235	448
USA	USAC03	Tiankai LIU	220	195	415
ROM	ROMC01	Radu BERINDE	158	240	398
KOR	KORC02	Kyung Yoon OH	200	190	390
RUS	RUSC01	Petr MITRITCHEV	130	255	385
TPE	TPEC01	Yin WANG	168	210	378
ROM	ROMC02	Daniel Octavian DUMITRAN	152	220	372
LVA	LVAC01	Aleksandrs BELOVS	186	175	361
RUS	RUSC02	Petr KALININ	156	200	356
CHN	CHNC01	Yifei ZHANG	180	170	350
EST	ESTC01	Martin PETTAI	92	255	347
CHN	CHNC02	Qiming HOU	111	235	346
CHN	CHNC03	Wei YU	140	200	340
SVK	SVKC01	Peter BELLA	139	200	339
IRN	IRNC03	Mohammadhossein BATENI	172	160	332
LVA	LVAC03	Andrejs IVANOVS	144	185	329
KOR	KORC01	Hyung-Sul KIM	170	140	310
CZE	CZEC01	Josef CIBULKA	82	225	307
ISR	ISRC02	Yair CHUCHEM	176	121	297
SWE	SWEC02	Daniel ANDERSSON	131	165	296
VNM	VNMC01	Khai TRAN QUANG	126	170	296

Silver Medalists (47)

CNTR	ID	Name	Day1	Day2	Score
USA	USAC01	Jacob BURNIM	102	191	293
RUS	RUSC04	Dmitri PAVLOV	91	200	291
IDN	IDNC01	Widagdo SETIAWAN	156	134	290
UKR	UKRC02	Petro LUFERENKO	143	146	289
FRA	FRAC04	Benjamin GAILLARD	248	40	288
SVK	SVKC03	Tomas DZETKULIC	195	91	286
CAN	CANC04	David ZHANG	68	215	283
VNM	VNMC02	Hieu NGUYEN VAN	108	172	280
HUN	HUNC02	Peter PALLOS	103	175	278
ROM	ROMC04	Marius Victor COSTAN	123	155	278
TUR	TURC01	Sedat GOKALP	192	80	272
USA	USAC02	Adam D'ANGELO	72	200	272
CUB	CUBC03	Ronny LÓPEZ TRUJILLO	163	105	268
YUG	YUGC01	Dejan KOLUNDZIJA	78	190	268

BGR	BGRC04	Nikolay NIKOLOV	105	160	265
HRV	HRVC03	Luka KALINOVCIC	148	115	263
POL	POLC02	Bartosz WALCZAK	167	95	262
YUG	YUGC03	Aleksandar ZLATESKI	68	193	261
AUS	AUSC04	David GREENAWAY	140	120	260
GEO	GEOC01	Nicholas JIMSHELEISHVILI	93	165	258
CAN	CANC03	Matei ZAHARIA	84	173	257
NLD	NLDC03	Tijmen TIELEMAN	142	115	257
BGR	BGRC03	Georgi TSANKOV	167	86	253
NOR	NORC03	Geir ENGDAHL	123	130	253
POL	POLC03	Karol CWALINA	49	204	253
TPE	TPEC02	Cheng-Yu LEE	111	141	252
RUS	RUSC03	Pavel MAVRIN	122	129	251
USA	USAC04	Alex SCHWENDNER	80	170	250
HUN	HUNC03	Gabor PELLADI	170	79	249
IRN	IRNC04	Mohammad MOHARRAMI	133	115	248
LTU	LTUC01	Viktor MEDVEDEV	148	100	248
CHN	CHNC04	Decheng DAI	90	157	247
EST	ESTC04	Andres LUUK	226	20	246
TUR	TURC04	Semsi Cihan YUCEL	144	100	244
IRN	IRNC01	Hamed AHMADI NEJAD	62	177	239
TPE	TPEC03	Shu-Chun WENG	128	110	238
DNK	DNKC01	Bjarke ROUNE	104	132	236
SGP	SGPC02	Jiquan NGIAM	131	104	235
EST	ESTC03	Mihkel KREE	109	125	234
SGP	SGPC01	Heng Ping Christopher MOH	152	80	232
SVK	SVKC02	Jozef TVAROZEK	96	135	231
COL	COLC03	Oscar RODRIGUEZ	145	85	230
HRV	HRVC01	Ivan SIKIRIC	100	130	230
FIN	FINC03	Markus OJALA	98	131	229
GBR	GBRC01	Paul JEFFERYS	103	125	228
AUT	AUTC02	Lukas STADLER	127	100	227
BLR	BLRC01	Maksim OSIPAU	132	94	226

Bronze Medalists (68)

CNTR	ID	Name	Day1	Day2	Score
SVK	SVKC04	Radovan BAUER	79	145	224
SWE	SWEC01	Erik BERNHARDSSON	98	124	222
POL	POLC04	Marcin MICHALSKI	110	110	220
THA	THAC04	Wittawat TANTISIRIROJ	145	75	220
DEU	DEUC01	Benjamin DITTES	135	84	219
ZAF	ZAFC01	David Jacques CONRADIE	113	106	219
HRV	HRVC02	Lovro PUZAR	100	116	216
KOR	KORC04	Heon JEONG	169	45	214
UKR	UKRC04	Andriy STASYUK	71	143	214
FIN	FINC01	Veli PELTOLA	68	140	208
FIN	FINC04	Olli-Pekka KAHILAKOSKI	145	60	205
THA	THAC01	Piyawat LAMSAM	88	117	205

MDA	MDAC03	Dumitru CIUBATII	118	86	204
NLD	NLDC02	Bram KUIJVENHOVEN	124	78	202
GBR	GBRC04	Nicholas KREMPEL	90	110	200
BGR	BGRC02	Veselin RAYCHEV	118	81	199
BRA	BRAC03	Rafael TEIXEIRA PAULINO	159	40	199
DEU	DEUC03	Alexander HULLMANN	182	15	197
LTU	LTUC02	Vilius NAUDZIUNAS	124	73	197
ARG	ARGC01	Pablo DAL LAGO	91	105	196
CAN	CANC01	Marcin MIKA	122	70	192
ITA	ITAC04	Stefano MAGGIOLO	112	80	192
VNM	VNMC03	Nhat LAM XUAN	107	85	192
EST	ESTC02	Hendrik NIGUL	177	10	187
ESP	ESPC01	Tomas LLORET	106	80	186
HUN	HUNC04	Gabor SIMKO	176	10	186
LKA	LKAC03	Chethiya ABEYSINGHE	114	70	184
SVN	SVNC03	Matej JAN	153	30	183
CUB	CUBC01	José David RODRÍGUEZ VELAZCO	88	93	181
BLR	BLRC03	Sarge ROGATCH	74	105	179
IRL	IRLC04	Martin ORR	110	68	178
FIN	FINC02	Olli-Pentti SAIRA	151	26	177
IRN	IRNC02	Siavosh BENABBAS	91	86	177
CHE	CHEC03	Ruben ANDRIST	138	35	173
TPE	TPEC04	Tsung-Chieh CHANG	118	55	173
UKR	UKRC03	Volodymyr TKACHUK	72	98	170
ARG	ARGC03	Alejandro DEYMONNAZ	124	45	169
BRA	BRAC02	Daniel BUENO DONADON	98	70	168
HKG	HKGC01	Man-Hon CHAN	108	59	167
LUX	LUXC02	Thierry STEINBERG	126	40	166
THA	THAC03	Vasan CHIENMANEETAWEESIN	126	40	166
HKG	HKGC03	Siu-On CHAN	95	70	165
ARM	ARMC02	Davit HAYKAZYAN	98	66	164
BLR	BLRC02	Aliaksei SIKORSKI	124	40	164
ARG	ARGC02	Diego Alejandro GAVINOWICH	88	75	163
SWE	SWEC04	Dan NILSSON	142	20	162
GBR	GBRC03	Adam BULL	121	40	161
TUR	TURC03	Mustafa Onur KILAVUZ	141	20	161
CZE	CZEC02	Pavel CIZEK	85	75	160
HRV	HRVC04	Marko ZIVKOVIC	70	90	160
LUX	LUXC01	Michel CONRAD	130	30	160
ZAF	ZAFC02	Heinrich DU TOIT	87	73	160
LKA	LKAC04	Nayana P.SOMARATNA	137	20	157
HKG	HKGC04	Koon-Ho WONG	126	30	156
NLD	NLDC04	Marijn KRUISSELBRINK	115	40	155
LTU	LTUC04	Martynas KRIAUCIUNAS	79	75	154
YUG	YUGC02	Nikola TODOROVIC	102	50	152
YUG	YUGC04	Aleksandar ILIC	60	92	152
IDN	IDNC02	Randy SUGIANTO	111	40	151
ISR	ISRC03	Noam RAFHAEL	61	90	151
GEO	GEOC04	Alexander TARKHNISHVILI	102	46	148
MEX	MEXC01	Jorge DEL RIO	90	55	145

ISR	ISRC01	Ariel GUTMAN	34	110	144
TUR	TURC02	Osman CELEP	58	85	143
MDA	MDAC02	Constantin JUCOVSCHI	60	81	141
MDA	MDAC01	Dumitru CODREANU	74	66	140
PRT	PRTC01	David RODRIGUES	104	35	139
BLR	BLRC04	Raman DZVINKOUSKI	115	20	135

Other Participants (137, sorted by ID)

CNTR	ID	Name
ALB	ALBC01	Ariel APOSTOLI
ARG	ARGC04	Martin VALDES DE LEON
AUS	AUSC01	David ANANIAN-COOPER
AUS	AUSC02	Clarence DANG
AUS	AUSC03	Alex FLINT
AUT	AUTC01	Roland SCHATZ
AUT	AUTC03	Christian WIRTH
AUT	AUTC04	Thomas WUERTHINGER
AZE	AZEC01	Isa ALIYEV
AZE	AZEC02	Teymur GULIYEV
AZE	AZEC03	Tofig HASANOV
AZE	AZEC04	Farid AHMADOV
BIH	BIHC01	Emir KUMALIC
BIH	BIHC02	Mirza SEJTO
BIH	BIHC03	Damir ZEKIC
BIH	BIHC04	Senad UKA
BRA	BRAC01	Lucas FURUKAWA GADANI
BRA	BRAC04	Cesario BARROS MARTINS
CAN	CANC02	Jason THEAN
CHE	CHEC01	Denis ROSSET
CHE	CHEC02	Urban SUPPIGER
CHE	CHEC04	David FREY
COL	COLC01	Jamer CUEVAS
COL	COLC02	Edwin NIÑO
COL	COLC04	Juan SARRIA
CUB	CUBC02	Alberto Eliseo PACHECO ALLENDE
CUB	CUBC04	Ledesma YOGERLAN ALMANZA
CYP	CYPC01	Avgoustinos KADIS
CYP	CYPC02	George PAPADOPOULOS
CYP	CYPC03	Kyriakos MATSIKARIS
CYP	CYPC04	Alexis CHRISTOFORIDES
CZE	CZEC03	Milan STRAKA
CZE	CZEC04	Jiri STEPANEK
DEU	DEUC02	Urs GANSE
DEU	DEUC04	Melanie SCHMIDT
DNK	DNKC02	Michael RASMUSSEN
DNK	DNKC03	Ulrik BUCHHOLTZ
DNK	DNKC04	Jens BOLDSEN
EGY	EGYC01	Dia SAMI

EGY	EGYC02	Mohamed ABDEL GAWAD
EGY	EGYC03	Omar FAWZY
EGY	EGYC04	Abd El Kader EL-HAIDARY
ESP	ESPC02	Xavier ROCA
ESP	ESPC03	Raul VINYES
ESP	ESPC04	Antoni-Italo DE MORAGAS
FRA	FRAC01	Guillaume RYDER
FRA	FRAC02	Chargueraud ARTHUR
FRA	FRAC03	Clément GENZMER
GBR	GBRC02	Adam LANGLEY
GEO	GEOC02	Giorgi BOCHORISHVILI
GEO	GEOC03	Zviad METREVELI
GRC	GRCC01	Theocharis ATHANASAKIS
GRC	GRCC02	Milos TZIOTAS
GRC	GRCC03	Xaralampos TSIMPOURIS
GRC	GRCC04	Christos SAVVOPOULOS
HKG	HKGC02	Siu-Man CHAN
HUN	HUNC01	Jozsef MARTON
IDN	IDNC03	Ilham Winata KURNIA
IDN	IDNC04	Felix HALIM
IND	INDC01	Rahil ALI
IND	INDC02	Anubhav GUPTA
IND	INDC03	Vivek KAPOOR
IND	INDC04	Adnan RAJA
IRL	IRLC01	Eamon PHELAN
IRL	IRLC02	Robert CUNNINGHAM
IRL	IRLC03	Daniel IRVINE
ISR	ISRC04	Michael KARASIK
ITA	ITAC01	Paolo CODENOTTI
ITA	ITAC02	Alessio ORLANDI
ITA	ITAC03	Maurizio SAMBATI
KAZ	KAZC01	Nurzhan BAKIBAYEV
KAZ	KAZC02	Khairosh YERZHAN
KAZ	KAZC03	Anton NIKOLAYEV
KAZ	KAZC04	Zhanibek DATBAYEV
KGZ	KGZC01	Kirill BARYSHNIKOV
KGZ	KGZC03	Stanislav VASHUK
KGZ	KGZC04	Abdimital PAZYLOV
KWT	KWTC01	Ali MUBARAK
KWT	KWTC02	Ibrahim ALMAYAS
KWT	KWTC03	Saleh ALSAFFAR
LKA	LKAC01	Harshana DANTANARAYANA
LKA	LKAC02	Amila DE SILVA
LTU	LTUC03	Dainius KUNIGAUSKAS
LUX	LUXC03	Jean-Marc HENGEN
LUX	LUXC04	Sidhath MYSORE
LVA	LVAC02	Karlis GANGIS
LVA	LVAC04	Sergejs KOZLOVICS
MAC	MACC01	Chi Hou HO
MAC	MACC02	Chon Kit WONG

MAC	MACC03	Cheng Chun NG
MAC	MACC04	Iat Man IEONG
MDA	MDAC04	Eugen SORBALO
MEX	MEXC02	Eduardo LOPEZ
MEX	MEXC03	Jesus PUENTE
MEX	MEXC04	Manuel TORRES
MKD	MKDC01	Aleksovski DARKO
MKD	MKDC02	Stojanovska VESNA
MKD	MKDC03	Atanasov VASIL
MKD	MKDC04	Mihajlovski NIKOLCE
MLT	MLTC01	Joel AZZOPARDI
MLT	MLTC02	Christian COLOMBO
MNG	MNGC01	Tulga TUMENDALAI
MNG	MNGC02	Gansukh BATKHUYAG
MNG	MNGC03	Dorjnamjil CHANDMANI
MNG	MNGC04	Otgontugs MIIMAA
MOZ	MOZC01	Sonia RAITHATHA
MOZ	MOZC02	Decio MACAMO
MOZ	MOZC03	Bachir BACHIR
MUS	MUSC01	Vishwaduthsingh GUNGA
MUS	MUSC02	Mohammad Zyaad JAUNNOO
MUS	MUSC03	Dominique Francois ADOLPHE
MUS	MUSC04	Rowan Rishi JUGERNAUTH
NLD	NLDC01	Cynthia KOP
NOR	NORC01	Dag SELJEBOTN
NOR	NORC02	Tormod LANDET
NOR	NORC04	Daniel LOKSHTANOV
PRT	PRTC02	Nuno PEREIRA
PRT	PRTC03	Pedro PEREIRA
PRT	PRTC04	Andre COIMBRA
ROM	ROMC03	Cosmin RAIANU
SGP	SGPC03	Junjie LIANG
SGP	SGPC04	Kee Tee Lawrence TAN
SVN	SVNC01	Ruben SIPOS
SVN	SVNC02	Ivo LIST
SVN	SVNC04	Tomaz GREGOREC
SWE	SWEC03	Carl NETTELBLAD
THA	THAC02	Pitchaya SITTHI-AMORN
TKM	TKMC02	Gurbannazar REJEPOV
TKM	TKMC03	Nagmat NAZAROV
TKM	TKMC04	Serdar MUHAMMETBERDIYEV
UKR	UKRC01	Oleksiy HLUKHOVSKIY
VEN	VENC01	Alfredo Enrique ROMAN REYES
VEN	VENC02	Alicson RUBIO
VEN	VENC03	Johan VIVAS
VNM	VNMC04	Son NGO THANH
ZAF	ZAFC03	Graham Leslie POULTER
ZAF	ZAFC04	Harry Zondberg WIGGINS

APPENDIX VI: Contestant Questionnaire

Questions for Contestants about IOI 2002 Competition

Which Operation System(s) did you use?

Day1:	❑ Linux	❑ Windows XP
Day2:	❑ Linux	❑ Windows XP

Which programming language(s) did you use?

Day1:	❑ Pascal	❑ C	❑ C++
Day2:	❑ Pascal	❑ C	❑ C++

Which programming editor(s) did you use?

❑ FreePascal IDE ❑ RHIDE ❑ Emacs
❑ vi/vim ❑ notepad(win) ❑ joe(Linux)
❑ Other: ____________________

Which debugger(s) did you use?

❑ FreePascal IDE ❑ RHIDE ❑ using extra printf/writeln statements
❑ gdb ❑ ddd ❑ Other: _________

What other tools did you use during the competition days?

____

Please grade the Grading System and the Web services:

	Usability	Response Time	Do You Like It
	Easy ⇔ Hard	Slow ⇔ Fast	No ⇔ Much
Submit	□ - □ - □ - □ - □	□ - □ - □ - □ - □	□ - □ - □ - □ - □
Testing	□ - □ - □ - □ - □	□ - □ - □ - □ - □	□ - □ - □ - □ - □
Print	□ - □ - □ - □ - □	□ - □ - □ - □ - □	□ - □ - □ - □ - □
Backup	□ - □ - □ - □ - □	□ - □ - □ - □ - □	□ - □ - □ - □ - □

Please rank the six tasks:

	Understand It	Find Algorithm	Write Program
	Easy ⇔ Hard	Easy ⇔ Hard	Easy ⇔ Hard
FROG	□ - □ - □ - □ - □	□ - □ - □ - □ - □	□ - □ - □ - □ - □
UTOPIA	□ - □ - □ - □ - □	□ - □ - □ - □ - □	□ - □ - □ - □ - □
XOR	□ - □ - □ - □ - □	□ - □ - □ - □ - □	□ - □ - □ - □ - □
BATCH	□ - □ - □ - □ - □	□ - □ - □ - □ - □	□ - □ - □ - □ - □
BUS	□ - □ - □ - □ - □	□ - □ - □ - □ - □	□ - □ - □ - □ - □
RODS	□ - □ - □ - □ - □	□ - □ - □ - □ - □	□ - □ - □ - □ - □

Which task did you like BEST of all? __________
Which task did you like LEAST of all? ___________

Was it useful to receive the contents of your home directory after the contest?

❑ Yes ❑ No

Are you subscribed to the IOI-LIST (mailing list)?
❑ Yes ❑ No

Suggestions for improving the Linux competition environment?
Suggestions for improving the Windows competition environment?
Suggestions for improving the Grading System competition environment?
Any other comments?

Brief Summary

There were 276 contestants in IOI 2002. By end of IOI 2002, we had received 103 completed forms. (about 37%).

- About 3/5 of the respondents use **Windows XP** and 2/5 of them use **Linux**.
  - About 4/5 of Pascal users use Windows XP. About 5/9 of C/C++ users use Linux.

- About 2/5 of the respondents use **Pascal**, 1/3 use **C++**, and 1/5 use **C**.
  - For Windows XP users, about 3/5 use Pascal. 2/7 use C++, and 1/6 use C.
  - For Linux users, about 3/7 use C++, 1/3 use C, and 1/5 use Pascal.

- Very few use both or switch the OS and the compiler between competition days.

- The **FreePascal IDE** and **RHIDE** are used by more than 1/3 of respondents.
- **Emacs** and **vim** are used by about 1/7. **Notepad** is about 1/9.

- About 2/3 use **debuggers integrated with IDE**.
- Almost 1/2 use **extra printf/writeln statements** in the program code.
- About 1/8 use **external debuggers**, such as gdb.

- Calculators and editors are mentioned as other tools used.

- The majority think the **grading system** of IOI 2002 easy and fast to use.

- BATCH is somewhat hard to **understand**. FROG and RODS are easier to understand.

- About 1/2 of respondents rate UTOPIA and BUS as very hard to **find algorithm**.
- XOR and BATCH are considered slightly harder; FROG and RODS slightly easier.

- RODS is hardest to **write program**. (though it is one of easier tasks to find algorithm.) BUS is considered slightly harder.
- FROG is slightly easier. Other tasks are around the middle.

- FROG is **likable** most by 1/3. RODS by 2/9. Other tasks are about 1/10.

- UTOPIA, XOR and BUS are **dislikable** most by 2/9.
- More than 3/4 think that it is useful to receive the **contents of the home directory**.
- Less than 1/4 of respondents were subscribed to the **IOI-LIST**.

OS

	Linux	Windows XP	Both
Day 1	40 (38.8%)	**60** **(58.3%)**	3 (2.9%)
Day 2	40 (38.8%)	**62** **(60.2%)**	1 (1.0%)

Language

	Pascal	C	C++	Pascal & C	C & C++
Day 1	**44** **(42.7%)**	23 (22.3%)	34 (33.0%)	1 (1.0%)	1 (1.0%)
Day 2	**44** **(42.7%)**	24 (23.3%)	34 (33.0%)	0 (0%)	1 (1.0%)

OS & Language

	Pascal	C	C++
Linux	9	**13**	**18**
Windows XP	**35**	10	17

Editor (multiple selections)

FreePascal IDE	**39** **(37.9%)**
RHIDE	37 (35.9%)
Emacs	13 (12.6%)
Vi/vim	16 (15.5%)
Notepad (win)	12 (11.7%)
Joe (Linux)	1 (1.0%)
Other*	5 (4.9%)

- other editors: edit(2), wordpad, kate, mcedit

Debugger (multiple selections)

FreePascal IDE	30 (29.1%)
RHIDE	36 (35.0%)
Extra printf/writeln statements	**47 (45.6%)**
gdb	11 (10.7%)
ddd	3 (2.9%)

Other Tools Used

Linux users	bash script (4), bc (10), cat (2), diff, gnome calculator (4), gprof, grep, joe, kate, kedit, less, make (2), mc (7), nano, perl (2), ps, time, xcalc
Windows users	debug.com, edit.com (2), notepad (8), windows calculator (20)

Grading System Usability

	No answer	1 (Easy)	2	3	4	5 (Hard)	Average
Submit	3 (2.9%)	**77 (74.8%)**	16 (15.5%)	4 (3.9%)	2 (1.9%)	1 (1.0%)	1.34
Testing	6 (5.8%)	**63 (61.1%)**	21 (20.4%)	8 (7.8%)	3 (2.9%)	2 (1.0%)	1.56
Printing	15 (14.6%)	**69 (67.0%)**	7 (6.8%)	7 (6.8%)	2 (1.9%)	2 (1.9%)	1.39
Backup	12 (11.7%)	**74 (71.8%)**	10 (9.7%)	4 (3.9%)	1 (1.0%)	2 (1.9%)	1.32

Grading System Response Time

	No answer	1 (Slow)	2	3	4	5 (Fast)	Average
Submit	3 (2.9%)	3 (2.9%)	7 (6.8%)	6 (5.8%)	18 (17.5%)	**66 (64.1%)**	4.37
Testing	8 (7.8%)	4 (3.9%)	6 (5.8%)	9 (8.7%)	19 (18.5%)	**57 (55.3%)**	4.25
Printing	22 (21.4%)	3 (2.9%)	2 (1.9%)	11 (10.7%)	16 (15.5%)	**49 (47.6%)**	4.31

Backup	13 (12.6%)	4 (3.9%)	1 (1.0%)	6 (5.8%)	12 (11.7%)	**67 (65.0%)**	4.52

Do you like the Grading System?

	No answer	1 (No)	2	3	4	5 (Much)	Average
Submit	3 (2.9%)	3 (2.9%)	3 (2.9%)	13 (12.6%)	27 (26.2%)	**54 (52.5%)**	4.26
Testing	8 (7.8%)	5 (4.8%)	7 (6.8%)	11 (10.7%)	28 (27.2%)	**44 (42.7%)**	4.04
Printing	21 (20.4%)	3 (2.9%)	3 (2.9%)	14 (13.6%)	17 (16.5%)	**45 (43.7%)**	4.20
Backup	14 (13.6%)	3 (2.9%)	7 (6.8%)	16 (15.5%)	15 (14.6%)	**48 (46.6%)**	4.10

Task: Understanding

	No answer	1 (Easy)	2	3	4	5 (Hard)	Average
FROG	2 (1.9%)	**45 (43.7%)**	31 (30.1%)	17 (16.5%)	5 (4.9%)	3 (2.9%)	1.91
UTOPIA	2 (1.9%)	**31 (30.1%)**	**32 (31.1%)**	16 (15.5%)	12 (11.7%)	10 (9.7%)	2.39
XOR	3 (2.9%)	**46 (44.7%)**	29 (28.1%)	11 (11.7%)	5 (4.9%)	9 (8.7%)	2.02
BATCH	2 (1.9%)	14 (13.6%)	19 (18.4%)	**29 (28.2%)**	26 (25.2%)	13 (12.6%)	3.05
BUS	2 (1.9%)	27 (26.2%)	**33 (32.0%)**	23 (22.3%)	9 (8.7%)	9 (8.7%)	2.41
RODS	2 (1.9%)	**50 (48.6%)**	27 (26.2%)	18 (17.5%)	4 (3.9%)	2 (1.9%)	1.82

Task: Finding Algorithm

	No answer	1 (Easy)	2	3	4	5 (Hard)	Average
FROG	2 (1.9%)	24 (23.3%)	**33 (32.0%)**	22 (21.4%)	15 (14.6%)	7 (6.8%)	2.49
UTOPIA	2 (1.9%)	1 (1.0%)	3 (2.9%)	15 (14.6%)	31 (30.1%)	**51 (49.5%)**	4.27
XOR	1 (1.0%)	4 (3.9%)	9 (8.7%)	19 (18.4%)	29 (28.2%)	**41 (39.8%)**	3.92
BATCH	2	2	5	22	**39**	33	3.95

	(1.9%)	(1.9%)	(4.9%)	(21.4%)	**(37.9%)**	(32.0%)	
BUS	1 (1.0%)	1 (1.0%)	7 (6.8%)	15 (14.6%)	33 (32.0%)	**46 (44.6%)**	4.14
RODS	2 (1.9%)	22 (21.4%)	**31 (30.1%)**	21 (20.4%)	16 (15.5%)	11 (10.7%)	2.63

Task: Writing Program

	No answer	1 (Easy)	2	3	4	5 (Hard)	Average
FROG	2 (1.9%)	24 (23.3%)	**27 (26.2%)**	23 (22.4%)	20 (19.4%)	7 (6.8%)	2.59
UTOPIA	4 (3.9%)	20 (19.4%)	21 (20.4%)	20 (19.4%)	15 (14.6%)	**23 (22.3%)**	3.00
XOR	2 (2.0%)	10 (9.7%)	25 (24.3%)	**28 (27.2%)**	19 (18.4%)	19 (18.4%)	3.12
BATCH	3 (2.9%)	16 (15.5%)	**27 (26.2%)**	23 (22.4%)	16 (15.5%)	18 (17.5%)	2.93
BUS	9 (8.7%)	8 (7.8%)	10 (9.7%)	**30 (29.1%)**	27 (26.2%)	19 (18.5%)	3.41
RODS	2 (1.9%)	3 (2.9%)	14 (13.6%)	28 (27.2%)	18 (17.5%)	**38 (36.9%)**	3.73

Preferring Tasks

	No mark	FROG	UTOPIA	XOR	BATCH	BUS	RODS
Best	3 (2.9%)	**34 (33.0%)**	13 (12.6%)	8 (7.8%)	10 (9.7%)	12 (11.7%)	23 (22.3%)
Least	5 (4.9%)	4 (3.9%)	**23 (22.3%)**	**23 (22.3%)**	17 (16.5%)	**22 (21.4%)**	9 (8.7%)

Receiving Contents of Students' Home Directories

No answer	Useful	Not useful
6 (5.8%)	**79 (76.7%)**	18 (17.5%)

IOI-LIST

	YES	NO
Are you subscribed to the IOI-LIST?	23 (22.3%)	**80 (77.7%)**

Written Comments

Linux Environment

1: Install RHIDE's help on Linux. Whenever I accidentally pressed the right mouse button I got a segmentation fault. Fix the mouse on the Linux shell.
8: ZSH was not installed under Linux.
14: Four Manager is very good shell, and it is free.
15: text mode 80x50.
18: Configure RHIDE problem.
23: Create shortcuts on desktop.
35: KDM maybe ? Need FTE.
42: The compiler options for g++ specified on the task summary sheet worked under Windows, but under Linux only a ".out" file was generated, no ".exe", so I could not make use of Linux because I had never used g++ before.
43: FTE editor. Nvidia drivers, 1600X1200 resoltion.
45: GPERF was missing - for generating perfect hash functions.
46: Privileged.
72: User screen feature of RHIDE. RHIDE Help files. RHIDE doesn't work in X. Mouse in console mode. RHIDE had a few crashes.
82: RHIDE 1.5 is unstable.
85: The tools used on Linux are a little bit unstable. The best example is RHIDE. Having the newest version is not the same as having the best version. Also, RHIDE would have worked much better if you gave us root access, which I think is not impossible.
87: Linux was not properly configured. Improve configuration.
99: More usable version of RHIDE
101: Use VESA VGA console mode for better console resolutions.

Windows Environment

1: Choose a windows version on which RHIDE works.
10: I don't like RHIDE very mush. The amount of code I can see on screen is small, and the program is unstable. It would be nice to have a Windows (not Dos) based IDE, though I don't if it's possible.
14: Borland C++ is very good thing.
15: File Manager (Any).
16: The FP crashes if memory limits exceeds. Can this be fixed ?
19: The RHIDE didn't work with FPC compiler. Although it was said before competition that it wound. That's not fair. There should have been file manager provided (for example windows commander) for Win XP users.
21: There should be windows commander available.
29: Use Windows 98se instead of WinXP, because RHIDE halts very often.
31: Make them stable.
33: I missed some file manager like Norton Commander.
34: RHIDE is very unstable with Win XP. Use Win 98.
37: RHIDE is not stable enough and it's quite old and looks very stupid, so new IDE would be nice.
38: File manager (like FAR, WC, …). Shortcuts on Desktop.

42: Native code editor would have been very useful. RHIDE is unstable in 50 line mode and responds slowly with the “-S” option, so a state of the art editor with a resizable window and syntax highlighting indentation would have been more ergonomic for coding.
60: RHIDE was very non stable.
63: More editors.
64: Better editors, e.g. Editplus or Writesource.
72: XP crashes consistently. Solution : use ’98. Windows was very unstable, if you offer an environment, make sure it works correctly, so as not to leave some people with disadvantages. Don’t let people down.
73: RHIDE not crashing; Visual C++.
74: Microsoft VC++ would be nice good compiler and debugger although it’s not gcc compatible. But it’s the only really useful Windows programming environment. Windows 2000 would be more useful.
75: Make it crash safe.
76: RHIDE doesn’t work.
79: Please use RHIDE with Win 98 and not with Win XP which accidentally crashes.
80: There are some problems with Freepascal IDE.
81: Include turbo Pascal as well.
84: Better IDE than RHIDE.
97: Add FAR(or NC or sth like it), BP.
98: Include FAR Manager (http://www.Farmanager.com) or any similar program. In Linux we have midnight commander, but no corresponding software in Windows XP.
99: More usable version of RHIDE that does not crash.

Grading System

8: You might want to disallow multiple logins form the same account in the web services for security reasons. The test facility checks for program headers. Thus, to test a simple test program (say, Windows vs. Linux differences), one has to include a valid program header (and be limited by that task’s memory and the limits). There could be an “accept submission regardless of whether it passes the sample input” checkbox on the submission page. Grading on a curve needs to be revised. It is unfair in its current form.
10: If a program (e.g. FROG) uses too much memory (over 64Mb), the error message was something like “Bad memory reference.” I think it should have been something more informative. I think the XOR format checker should have been stricter. The current checker didn’t notice trivial format errors, such as missing number of rectangles on second line. Also, for the sake of similarity, the submit program could have tested that the given rectangles really are a solution, at least for the first input.
29: Make the results show times and answers on each test case, as on IOI 2001.
43: Make it so you don’t have to press reload.
44: Use something that works in lynx.
50: The grading system should be available from command line. (Using X web browser and mouse is very uncomfortable).
72: The header format was too strict. It should allow extra spaces.
83: It must work no crashes. Should work with more than I serves.
85: The response time of your grader was far from being good. It took quite some time for my requests to be processed.
92: “Test” in grading system is not useful. Some script or batch file would be more useful for testing. Maybe the refresh could be automatically.
97: Make not need to “Reload”.

99: Quite ok, but using "Reload" is uncomfortable.
101: Pick up errors in output format for output only tasks.

Other Comments

4: Test data for "open test data" problems should be uploaded to the server for contestants to download in case the input files are erased accidentally.
8: The browser's homepage could have been set to the IP address for the web services. The web page should have included task related materials such as inputs for "XOR" and the library for "RODS." Why limit the size of stderr if you can just redirect it to /dev/null? It seems unreasonable for a contestant to lose points just because (s)he forgot to remove debugging information from his/her code.
9: we were forbidden to bring food with us, and we weren't given any food, except those cookies. So I was hungry.
13: There is no file manager on IOI. It makes work more hard.
16: Although XOR was a hard task, the idea of having a task with relative scoring is nice and makes the competition more interesting. Continue with the hard work!
31: More food and water, less cookies, less strict rules day before the competition.
43: Warn about extra output for submissions.
45: Internet competition.
65: The system should be able to auto refresh.
68: Install games and mp3s on the computers.
69: Provide contestants with free music and headphones.
70: Would be preferable if knowledge of specialized algorithms and especially important insights (e.g. O(n) for BATCH, solution for BUS) not be required as this unfairly benefits certain algorithms over others. It is better if questions asked for tricky applications of general algorithms (as in IOI 2002, CEOI 2002) rather than direct applications of more obscure algorithms which competitors can hardly expect to derive and prove correctness within 5 hours e.g. minimum diameter spanning tree.
72: More question time(an hour is too few). Time limit for tasks more permissive.
75: Make less mathematical and more algorithmic tasks, and at least one easy task per day.
81: The problems were so hard. Make it easy next time.
82: I should be able to see the sources I have submitted. I should also be able to test the sources I submitted (without having to select the file on my hard disk which might be different from the one submitted) I think a 'just submitted program' option should be added.
85: Scores were reported pretty late. I think that, due to the nature of the graders, you could have shown us our scores on the screens of our Woojungwon stations, might after the end of the competition.

Appendix VII: Delegation Questionnaire

Questions about IOI 2002 Competition

What do you think about the task?

	Suitability	To Understand It	To Translate It
	No ⇔ Much	Easy ⇔ Hard	Easy ⇔ Hard
FROG	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐
UTOPIA	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐
XOR	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐
BATCH	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐
BUS	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐
RODS	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐

Which task did you like BEST of all? __________
Which task did you like LEAST of all? __________

Do you like the idea of output-only tasks?
❑ Yes ❑ No

Do you like the idea of the relative scoring?
❑ Yes ❑ No

Please grade the Grading System and the Web services:

	Usability	Response Time	Do You Like It
	Easy ⇔ Hard	Slow ⇔ Fast	No ⇔ Much
Submit	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐
Testing	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐
Print	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐
Backup	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐	☐ - ☐ - ☐ - ☐ - ☐

Do you find it useful to receive the contents of the students' home directories after the contest?
❑ Yes ❑ No

What is your opinion of the 1-page solution explanation handout?
❑ Useful ❑ Not useful
❑ Too detailed ❑ Just right ❑ Not detailed enough

Are you subscribed to the IOI-LIST (mailing list)?
❑ Yes ❑ No

Suggestions for improving the Linux competition environment?
Suggestions for improving the Windows competition environment?
Suggestions for improving the Grading System competition environment?
Any other comments?

Brief Summary

There were 78 delegations at IOI 2002. By the end of IOI 2002, we had received 44 completed forms (almost 56%).

- All tasks are considered to **suitable** for the IOI competition. FROG is considered most suitable and UTOPIA is least.

- BATCH is considered hardest to **understand**, followed by UTOPIA. XOR and FROG are easy to understand.

- BATCH and RODS are rated harder to **translate**. XOR is easiest.

- RODS and FROGS are **likable** most by 2/9.
- XOR and UTOPIA are **dislikable** most by 2/9.
- The preferences are spitted evenly.

- Over 3/4 of respondents supports the idea of **output-only tasks**.
- Over 5/6 of respondents supports the idea of the **relative scoring**.

- The majority think the **grading system** of IOI 2002 easy and fast.

- The respondents definitely think that it is useful to receive the contents of students' **home directories**.

- The respondents think that **1-page solution handout** is useful and just right. Some people think that it is not detailed enough.

- More than 1/4 of respondents were not subscribed to the **IOI-LIST**.

Task Suitability

	No answer	1 (Less)	2	3	4	5 (Much)	Average
FROG	2 (4.5%)	0	1 (2.3%)	7 (15.9%)	9 (20.5%)	**25 (56.8%)**	4.38
UTOPIA	2 (4.5%)	6 (13.6%)	4 (9.1%)	5 (11.4%)	**14 (31.8%)**	13 (29.6%)	3.57
XOR	3 (6.8%)	1 (2.3%)	6 (13.6%)	11 (25.0%)	7 (15.9%)	**16 (36.4%)**	3.76
BATCH	2 (4.5%)	3 (6.8%)	7 (15.9%)	3 (6.8%)	13 (29.6%)	**16 (36.4%)**	3.76
BUS	2 (4.5%)	2 (4.5%)	2 (4.5%)	7 (16.0%)	11 (25.0%)	**20 (45.5%)**	4.07
RODS	2 (4.5%)	0 (0%)	5 (11.4%)	2 (4.5%)	15 (34.1%)	**20 (45.5%)**	4.19

Task Understandability

	No answer	1 (Easy)	2	3	4	5 (Hard)	Average
FROG	2 (4.5%)	**15 (34.1%)**	**15 (34.1%)**	10 (22.8%)	2 (4.5%)	0 (0%)	1.98
UTOPIA	2 (4.5%)	**15 (34.1%)**	12 (27.3%)	7 (15.9%)	7 (15.9%)	1 (2.3%)	2.21
XOR	2 (4.5%)	**20 (45.5%)**	15 (34.1%)	4 (9.1%)	0 (0%)	3 (6.8%)	1.83
BATCH	2 (4.6%)	10 (22.7%)	**12 (27.3%)**	6 (13.6%)	10 (22.7%)	4 (9.1%)	2.67
BUS	3 (6.8%)	**17 (38.6%)**	14 (31.8%)	5 (11.4%)	2 (4.6%)	3 (6.8%)	2.02
RODS	3 (6.8%)	**17 (38.6%)**	12 (27.3%)	5 (11.4%)	6 (13.6%)	1 (2.3%)	2.07

Task Translatability

	No answer	1 (Easy)	2	3	4	5 (Hard)	Average
FROG	6 (13.7%)	**15 (34.1%)**	9 (20.4%)	9 (20.4%)	4 (9.1%)	1 (2.3%)	2.13
UTOPIA	6 (13.6%)	12 (27.3%)	**13 (29.5%)**	9 (20.5%)	4 (9.1%)	0 (0%)	2.13
XOR	6 (13.6%)	**17 (38.6%)**	9 (20.5%)	8 (18.2%)	4 (9.1%)	0 (0%)	1.97
BATCH	6 (13.6%)	10 (22.7%)	**13 (29.6%)**	7 (15.9%)	6 (13.6%)	2 (4.6%)	2.39
BUS	6 (13.6%)	**13 (29.6%)**	10 (22.7%)	12 (27.3%)	1 (2.3%)	2 (4.5%)	2.18
RODS	6 (13.6%)	**16 (36.4%)**	6 (13.6%)	4 (9.1%)	9 (20.5%)	3 (6.8%)	2.39

Preferring Tasks

	No mark	FROG	UTOPIA	XOR	BATCH	BUS	RODS
Best	2 (4.5%)	**9.5 (21.6%)**	7 (15.9%)	5.5 (12.5%)	5 (11.4%)	5 (11.4%)	**10 (22.7%)**
Least	6 (13.6%)	2 (4.5%)	9 (20.5%)	**10 (22.7%)**	8 (18.2%)	4 (9.1%)	5 (11.4%)

Note:
- A delegation marked both of FROG and XOR for "BEST." (We count it as 0.5, each)
- A delegation marked "All are OK" for "LEAST." (We count it as "No mark")

The Idea of Output-Only and Relative Scoring

	YES	NO
Do you like the idea of output-only Tasks?	**34.5 (78.4%)**	9.5 (21.6%)
Do you like the idea of the relative scoring?	**37 (84.1%)**	7 (15.9%)

Note:

- A delegation was divided in opinion on output-only tasks. The leader claims "NO" and the deputy leader claims "YES." (We count it as 0.5, each)

Grading System Usability

	No answer	1 (Easy)	2	3	4	5 (Hard)	Average
Submit	10 (22.7%)	**21 (47.7%)**	9 (20.4%)	2 (4.6%)	0 (0%)	2 (4.6%)	1.62
Testing	10 (22.7%)	**18 (40.9%)**	11 (25.0%)	4 (9.1%)	1 (2.3%)	0 (0%)	1.65
Printing	9 (20.4%)	**25 (56.8%)**	5 (11.4%)	5 (11.4%)	0	0	1.43
Backup	9 (20.5%)	**21 (47.7%)**	8 (18.2%)	6 (13.6%)	0	0	1.57

Grading System Response Time

	No answer	1 (Slow)	2	3	4	5 (Fast)	Average
Submit	12 (27.3%)	0 (0%)	1 (2.3%)	11 (25.0%)	3 (6.8%)	**17 (38.6%)**	4.13
Testing	12 (27.3%)	0 (0%)	2 (4.5%)	10 (22.7%)	4 (9.1%)	**16 (36.4%)**	4.06
Printing	13 (29.5%)	0 (0%)	0 (0%)	10 (22.8%)	2 (4.5%)	**19 (43.2%)**	4.29
Backup	13 (29.5%)	0 (0%)	1 (2.3%)	9 (20.4%)	5 (11.4%)	**16 (36.4%)**	4.16

Do you like the Grading System?

	No answer	1 (No)	2	3	4	5 (Much)	Average
Submit	8	2	1	3	11	**19**	4.22

	(18.2%)	(4.5%)	(2.3%)	(6.8%)	(25.0%)	**(43.2%)**	
Testing	8 (18.2%)	0 (0%)	0 (0%)	5 (11.4%)	11 (25.0%)	**20 (45.4%)**	4.42
Printing	8 (18.2%)	0 (0%)	0 (0%)	4 (9.1%)	7 (15.9%)	**25 (56.8%)**	4.58
Backup	8 (18.2%)	0 (0%)	0 (0%)	7 (15.9%)	6 (13.6%)	**23 (52.3%)**	4.44

Receiving Contents of Students' Home Directories

Useful	Not useful
42 (95.5%)	2 (4.5%)

Opinion on the 1-page solution handout (multiple selections)

Useful	**31 (70.5%)**
Not useful	1 (2.3%)
Too detailed	0 (0%)
Just right	**17 (38.6%)**
Not detailed enough	11 (25.0%)

- 13 delegations marked both of "Useful" and "Just right."
- 3 delegations marked both of "Useful" and "Not detailed enough."

IOI-LIST

	YES	NO
Are you subscribed to the IOI-LIST?	**32 (72.7%)**	12 (27.3%)

Written Comments

Linux Environment

2: RHIDE IDE still has lot of bugs, we really need it
10: Try to document all the RHIDE bugs, and make a manual of them
14: For Linux, it is better to provide the environment under X window, and the tools with better GUI

28: Greater variety of useful editors we suggest FTE
31: To delete bugs in IDE and RHIDE
35: Stability of competition environment (mainly IDEs) still needs to be improved. Alternative or additional IDEs might be made available.
36: We need a better Pascal compiler.

Windows Environment

2: RHIDE IDE still has lot of bugs, we really need it
6: We need the better IDE
10: Try to document all the RHIDE bugs, and make a manual of them
30: Commercial compilers / tools are desirable
31: To delete bugs in IDE and RHIDE
35: Stability of competition environment (mainly IDEs) still needs to be improved. Alternative or additional IDEs might be made available.
36: We need a better Pascal compiler

Grading System

2: The way to grade "relative score" is good to identify who the best. However, it is not good for average contestant. The suggestion is to improve the formula of the grading system
14: It is better to make the interface more easy and the response time short for the Grading system
20: The grading system should give more response. The checker programs find out one reason to reject the result if it is not correct, you should write that reason to the checker response file. Also, when a program in a test case does not meet the time limit, it would be give to know when the program execution is cut. Naturally, it would be very useful to have the running time in cases, when the program solves the case within the time limit.
22: The idea of relative scoring seems to be natural for output only tasks. We suggest to add source file management tool to the default installation
25: Start to have problem in area of parallel algorithms
28: Need to constantly refresh web page to get status of test run is annoying (though not critical). Suggest an applet would improve this situation.
20: Whole system should be more stable.
37: Format checkers for all batch, library and output only tasks are an excellent idea and should be continued

Other Comments

5: Pascal and C-compiler also installed on the translator's machines. It would be nice to assume that people are fair and let them use internet while translating tasks.
More time for discussion on GA meetings. Less security restrictions, like forcing students to stay in their rooms on the night before competition. 'extime' library for measure execution times should be provided to students as well
6: I'd like we could be used Delphi second IDE for Pascal programmer in windows
7: Tasks made in windows checked up in Linux. This caused mistakes. All optimization problems (such as BATCH, BUS, RODS) must be made with relative scoring (But a formula may vary). I suggest to include continuous (non-discrete) problems, on

optimization, approximate solving of equations etc. Relative scoring makes such problems to be correct
9: In fact we need only the student's programs and input-output files of tasks
10: I felt that this Olympiad was too easy to score a lot of points with a bad solution. One of my students made a really inefficient N^3 solution for FROG, which took more than 2 minutes to run for the last case, and still he got 44 points. I fell that it's unfair for other contestants, that such a bad solution earns so many points. Its ok to give points to all solutions. Just don't give so many.
31: To improve the security of all systems.