EK 90

levien Acmgock

Problem A : Woodworms

1991 ACM Scholastic Programming Contest
sponsored by AT&T Computer Systems

European Regional Finals
November 3, 1990

Source File: `WORMS.PAS`
Input File: `WORMS.INP`
Output File: `WORMS.OUT`

A carpenter has a little problem: his stock of wood is infected with woodworms. As he does not like to lose his whole stock at once, he decides to saw the largest possible plank containing no worms out of every infected board. He has still another problem: his sawing machine can saw only in directions parallel to the sides of the board. The carpenter has made a list of the coordinates of each of the wormholes in each board. Fortunately the worms go straight through the wood, so we can ignore the third dimension. It is your job to find the area of the plank with the largest area that can be recovered from each board. (If there are no holes then the whole board is recovered.)
The program should read a textfile and should give its output in a textfile. Both files are in 'strict format'.

Input: The input file consists of a number of blocks, each corresponding to one board. The different blocks are separated by a blank line. The data in each block consists of a number of lines containing two non-negative integer numbers, smaller than 10,000, separated by exactly one blank. The first number of the first line gives the x-dimension of the board in arbitrary "wood" units. The second number of the first line gives the y-dimension of the board in the wood units. The following lines each give the position of a hole. First number: x-coordinate, second number: y-coordinate. All holes are within the boundary of the board. The holes are given in random order. It is possible that the same hole occurs more than once in a block. The number of holes in a board is limited to 400. It may be a good idea to use the type `longint` for the area, as intermediate results can easily be larger than `MaxInt` .

Output: The output is a textfile containing one line for each block of the input file. Each line consists of one integer number. This number represents the area of the largest plank in square "wood" units.

Example:

WORMS.INP

```
45 60
1 1
1 1
44 59

70 40
```

WORMS.OUT

```
2610
2800
```

End of problem A

Acngoek

Problem B : BOGGLE II

1991 ACM Scholastic Programming Contest sponsored by AT&T Computer Systems

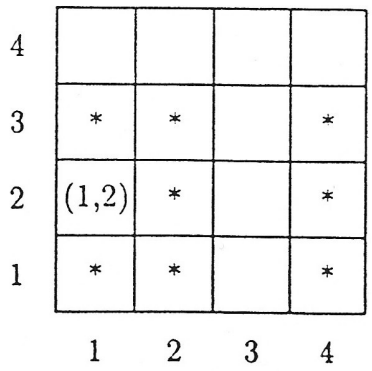
European Regional Contest
November 3, 1990

Source File:	`BOGGLE.PAS`
Input File:	`BOGGLE.INP`
Output File:	`BOGGLE.OUT`

In a Boggle game we have a board of $M \times N$ places where each place contains one upper case letter. A *word* is a sequence of letters. We say that word $c_1 c_2 .. c_k$ ($k \geq 0$) is *embedded* in the board if and only if $k = 0$ or for $k \geq 1$:

- there is a place (m_1, n_1) that contains letter c_1,
- for each letter c_{i+1} ($1 \leq i < k$) there is a place (m_{i+1}, n_{i+1}) that contains that letter and (m_i, n_i) and (m_{i+1}, n_{i+1}) are adjacent places on the board, and
- the word covers k places (that is, no places are used twice.)

Two (different) places (x_0, y_0) and (x_1, y_1) are said to be *adjacent* if the first coordinates satisfy $|x_0 - x_1| \leq 1$ or $|x_0 - x_1| = M - 1$, and the second coordinates satisfy $|y_0 - y_1| \leq 1$ or $|y_0 - y_1| = N - 1$. For example, on a square board of 16 places the set of places adjacent to $(1,2)$ equals $\{(1,1),(1,3),(2,1),(2,2),(2,3),(4,1),(4,2),(4,3)\}$, see Figure.



A *prefix* of a word is obtained by deleting zero or more letters at the end of the word. For example, the set of all prefixes of the word 'BOGGLE' equals { `'BOGGLE'`, `'BOGGL'`, `'BOGG'`, `'BOG'`, `'BO'`, `'B'`, `''` }, where `''` denotes the empty word.

Given a board in a Boggle game and a word, w, your program should determine the length of the longest prefix of w that can be embedded in the board.

Input: The input file (a textfile) contains zero or more problem descriptions. Each problem description consists of:

- One line containing two integers, M and N, that represent the dimensions of the board, $1 \leq M \leq 10$ and $1 \leq N \leq 10$; followed by
- M lines, each containing a string of length N
- One line containing an integer K, that denotes the number of words that your program has to process for this board, $K \geq 1$.
- K lines, each containing a string of at least 1 and at most 200 letters

The end of the input marked by standard end-of-file marker.

Output: The output file of your program is a textfile containing the answers to the problems of the input file. The format of input and output file is strict. Answers for different problem descriptions are separated by an empty line. The answers to each problem description should be in the following format:

- K lines (one line for every word that has to be embedded) containing one integer that represents the length of the maximal prefix of the input word that can be embedded in the board.

Example:

BOGGLE.INP

```
4 4
ODET
XVRT
ZUES
WIPA
2
PIETER
DRUIF
2 3
ABC
DEF
3
AB
ABCDEFGHIJKLMNOPQRSTUVWXYZ
XYZ
```

BOGGLE.OUT

```
6
4

2
6
0
```

Hint: notice that in the second case of the second problem the input string is much longer than the number of places on the board. If prefix ABCDEF is embedded all places are covered, resulting in the answer 6.
Keep this in mind when writing your program.

End of problem B

Acmgoek

Problem C : Semi–Circles

1991 ACM Scholastic Programming Contest sponsored by AT&T Computer Systems

European Regional Contest
November 3, 1990

Source File: `CIRC.PAS`
Input File: `CIRC.INP`
Output File: `CIRC.OUT`

Given a number of points on a circle with center $(0,0)$, the question may arise whether or not there exists a line through $(0,0)$ such that all these points are on the same side of this line (and not on this line). If so, these points are said to satisfy the 'Semi–Circle' property. I don't know of any practical application of the Semi–Circle property, but your program has to answer the burning question of whether or not points satisfy this property.

Input: The input of your program resides in a textfile. Each line of input contains an even number of reals, with a maximum of 1000 per line, representing the coordinates of points on a circle with center $(0,0)$ and a positive radius. For example the inputline '`0.5 1.0 1.0 -0.5 0.5 1.0`' denotes the points $(0.5,1),(1,-0.5)$, and $(0.5,1)$ again, on the circle with center $(0,0)$ and radius $\sqrt{1.25}$. At the end of an input line no trailing spaces occur.

Output: The textfile must consist of one output line for each input line, every line containing exactly one character, being a '`Y`' if the points on the corresponding input line satisfy the Semi–Circle property and a '`N`' otherwise.

Remark: Accuracy of numerical computations is *not* the topic of this problem. Therefore the input points are chosen such that the restricted accuracy of ordinary real arithmetic in Turbo Pascal suffices to decide whether they satisfy the Semi–Circle property or not: for example the line '`0.5 1.0 1.0 -0.5 -0.5 -1.0`' will not occur in the input.

Example:

CIRC.INP
`0.5 1.0 1.0 -0.5 0.5 1.0`
`0.5 1.0 1.0 -0.5 -0.5 -1.0 -1.0 0.5`

CIRC.OUT
Y
N

End of problem C

Acmgoek

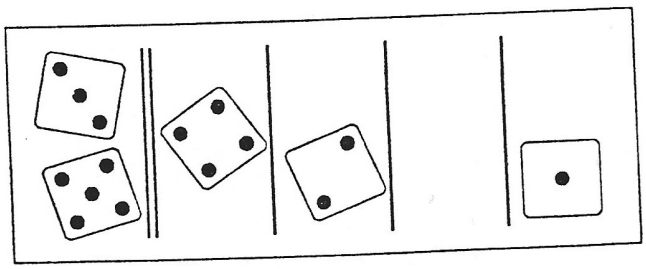
Problem D : Counting with Dice

1991 ACM Scholastic Programming Contest sponsored by AT&T Computer Systems

European Regional Contest
November 3, 1990

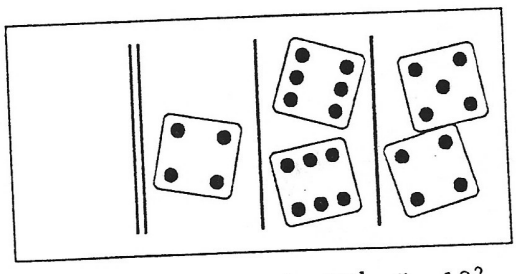
Source File: DICE.PAS
Input File: DICE.INP
Output File: DICE.OUT

During a recent excavation, remainders have been found of an ancient civilization. Some Eindhoven professor claims that it was a civilization of dice-cutters and gamblers, from cave paintings and other clues he even deduces that these people used to count with dice on a wooden 'dice tray'. Regretfully these trays have been lost over time. To illustrate this counting system, we show you two schematic representations of our number 1024 with five six-sided (ordinary) dice.



0
0

$4 * 10^0$ $2 * 10^1$ $1 * 10^3$



$4 * 10^0$ $6 * 10^1$ $5 * 10^2$
$6 * 10^1$ $4 * 10^2$

In general one K-sided die can be used to represent the number 0 or any number $k * 10^i$ for integers k and i satisfying $1 \leq k \leq K$ and $i \geq 0$. The number represented on a tray with dice is the sum of the numbers that are represented by the individual dice.

When discovering such a hitherto unknown system, the question arises how far one can count with a given number of dice. Having written a program for this problem, we may as well use it as a problem in this programming contest. For several cases of K and N, your program should produce the least natural number M which cannot be represented with N K-sided dice.

Input: The input of your program resides in a textfile with strict format. Each line in this file, except for the last, consists of a pair K and N (first K then a space, and then N) such that $0 < K \leq 10$ and $0 < N \leq 100$. The last line consists of the number 0 only.

Output: The output file is a text file and has one line for each pair K and N in the input file. This line gives the corresponding value of M, without leading spaces, without leading zeros (in our own familiar decimal representation). The file has strict format.

Example:

DICE.INP
6 2
3 2
0

DICE.OUT
17
7

End of problem D

ACMgeek

Problem E : Russian Shops

1991 ACM Scholastic Programming Contest sponsored by AT&T Computer Systems

European Regional Contest
November 3, 1990

Source File: `SHOP.PAS`
Input File: `SHOP.INP`
Output File: `SHOP.OUT`

At present, Russian shops are as good as empty, but the Russian people — disciplined as they are — still keep lining up in front of the shops. The Russians are even so disciplined that they arrive in the same order every day. As a consequence they always have to talk with the same people in the waiting queue.
A smart cashier therefore decides to use the random number generator on her Western pocket calculator — normally used to generate exchange rates for Western currencies — and gives each customer upon arrival a random number between zero and the number of customers that have arrived before him. This number tells the customer how many places he may move towards the front of the queue. In this fashion, a customer can meet, in the long run, any other customer in the queue.

Your program determines the final order in which the customers have to spend the rest of the day. Each line of the input file contains a sequence of which the i-th number is between 0 and i ($0 \leq i$). [To be more precise: if we denote the i-th number by a_i, then we have $0 \leq a_i \leq i$ for $0 \leq i$.] Consecutive numbers are separated by exactly one space, and there are no trailing spaces. The output of your program has the same structure. The i-th number on each output line is the number of the place in the queue where the i-th customer ends up when all customers have arrived ($0 \leq i$).

Notes: There are at most 1000 customers. Customers and places in the queue are numbered from 0 onwards.

Example:

`SHOP.INP`

```
0

0 0 1 2 1
0 1 2 3
0 0 0 0
```

`SHOP.OUT`

```
0

0 4 2 1 3
3 2 1 0
0 1 2 3
```

End of problem E

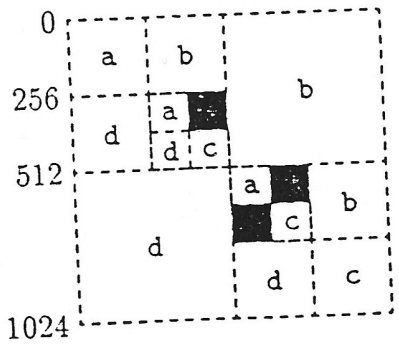
Problem F : Quad Monitor

1991 ACM Scholastic Programming Contest sponsored by AT&T Computer Systems

European Regional Contest
November 3, 1990

Source File: QMON.PAS
Input File: QMON.INP
Output File: QMON.OUT

Your computing department has bought a new monochrome monitor with a square screen of 1024x1024 pixels. On this monitor you wish to display a rectangle, but a quick glance through the manual reveals that you can only display a certain set of squares.
To display the three black squares in the picture below, you send a sequence of signals to the monitor (here represented as ASCII characters). First, you send a `@` to clear the screen. Then you select the first square you want to draw by sending a sequence of `a`'s, `b`'s, `c`'s, and `d`'s. Each letter sent selects a subsquare according to the scheme: `a` selects the upper-left square, `b` the upper-right, `c` selects the lower-right square, and the lower-left subsquare is selected by `d`. To select, for instance, the upper black square in the following example, you send `acb`. By the way, the input and output corresponding to this square is the second line in the example listed at the end of the problem description.



Finally, you send a `*` to fill this square. In order to display more squares, the `^` signal can be used to cancel the effect of the last *letter* sent. To put it in another way: `^` selects the next supersquare. A complete sequence to display the squares in the above picture is `@acb*^^^cad*^b*`. The sequence `@*` will blacken the entire screen.
The problem now is to produce a sequence of *minimal* length that displays a given rectangle. A rectangle is given by the xy-coordinates of its upper left corner and its lower right corner.

Input: The input for your program resides in a textfile. Each line contains four integers x_0, y_0, x_1, and y_1, say, satisfying $0 \leq x_0 < x_1 \leq 1024$ and $0 \leq y_0 < y_1 \leq 1024$, that represent a rectangle to be displayed. The input is terminated by the standard end-of-file marker.

Output: The output of your program is also a textfile. Each line contains the minimal length of a displaying sequence of the given rectangle. The format of the output should be 'strict'.

Example:

QMON.INP
0 0 1024 1024
384 256 512 384
0 0 512 1024
384 384 896 640

QMON.OUT
2
5
6
42

Remark: You may check that these rectangles could have been displayed by sending the following sequences to the monitor:

```
©*
©acb*
©a*^d*
©aaa*^^^bdd*^c*^^cd*^^^dbb*^^^caa*^b*^^ba*
```

End of problem F

Acm90 EK

Problem G : Tom and Jerry

1991 ACM Scholastic Programming Contest sponsored by AT&T Computer Systems

European Regional Contest
November 3, 1990

Source File: TOMJERRY.PAS
Input File: TOMJERRY.INP
Output File: TOMJERRY.OUT

As you all should know, the cartoon heroes Tom and Jerry are not the best housemates you can have. They are not very close friends; in fact, they are at a never ending war with each other. Every time when Tom is chasing Jerry, Jerry is fooling Tom. But, times are changing. Tom, though he is not the smartest one of the two, has bought a computer and made a plan. When Jerry is sleeping somewhere in the room he wants to take him by surprise. For doing this, he needs a fastest path to Jerry. A path is made up of straight pieces parallel with one of the walls of the room. A fastest path is a path containing a minimum number of bends. So, exploiting his computer, Tom desperately needs a computer program that calculates a fastest path.
Your task is to write Tom's program, given the following conditions:

- Tom and Jerry are both in the same room.
- Both our heroes are squares of size 1×1.
- The room is rectangular. The room is divided into squares of size 1×1 which may be free or occupied by an obstacle, Tom, or Jerry (see Figures 1, 2). Coordinates are given to the squares. Two walls are used as coordinate axes. The lower-left square gets coordinates $(0,0)$, the coordinates of the upper-right square are at most 75. The only legal places inside the room are the squares it is divided in.
- The obstacles in the room (possibly none) are of a rectangular shape and may differ in size. The obstacles are all fully inside the room and are placed parallel to the boundaries of the room. The coordinates of the obstacles are integers. Obstacles do not overlap.
- Jerry does not wake up .
- Tom and Jerry occupy different positions.
- Tom and Jerry are always outside the boundaries of the obstacles.

Input: The input for your program resides in a textfile in strict format. The file ends with the standard end-of-file marker. The input file contains a number of cases. The end of a case is given by a line consisting of a -1. The first line of a case gives the coordinates of

the upper–right square of the room. The second line gives first the coordinates of Tom and then those of Jerry. The next lines describe the obstacles. Each line gives the lower–left coordinates followed by the upper–right ones of an obstacle. Note, a square obstacle with size 1×1 has equal lower–left and upper–right coordinates.

Output: The output is also a textfile in strict format. For each case in the input, your program produces one output line with:

- if a fastest path exists, the number of bends in the path , or
- otherwise, the word `impossible`.

Example:

TOMJERRY.INP

```
4 4
1 1 3 3
0 2 4 2
-1
6 4
0 1 6 0
1 0 1 1
1 3 1 3
3 1 3 4
5 0 5 1
5 3 5 3
-1
```

TOMJERRY.OUT

```
impossible
6
```

Figure 1 gives a graphical representation of the first input case, Figure 2 of the second one.

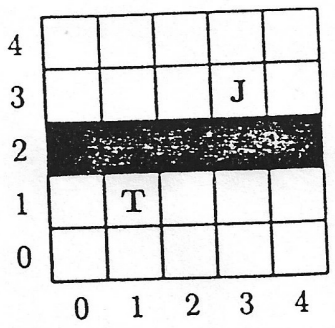


Figure 1.

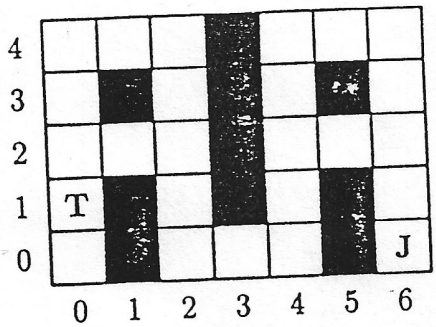


Figure 2.

End of problem G

ACM90EK

Problem H : Musical Repe(ti)on

1991 ACM Scholastic Programming Contest sponsored by AT&T Computer Systems

European Regional Contest
November 3, 1990

Source File:	`REPE.PAS`
Input File:	`REPE.INP`
Output File:	`REPE.OUT`

In order to reduce the length of the staffs, composers use a very simple way to denote repetition. Your program has to perform a similar, but simplified, kind of data compression. The problem is translated from staffs into ASCII text and you have to consider the effect of one repetition only.

First some definitions:

S is the set of finite–length strings (including the empty string) that contain characters of 'a'..'z' and spaces only.

R is the set of finite–length strings that contain exactly one repetition, which is either 'direct' or 'delayed': where

- direct repetition occurs in a string $v(w)y$, for v, w, and y elements of S. This string encodes the string $vwwy$ of S.
- delayed repetition occurs in a string $v(w|x)y$, for v, w, x, and y elements of S, which encodes the string $vwxwy$ of S.

Your program encodes strings of S into strings of R such that the encoding has a minimal length (the length of a string is its number of characters including the characters '(', ')', and '|').

Input: The input of your program resides in a textfile , one string of S per line. The length of each string is at most 400.

Output: The output is a textfile. It should consist of one line per input line, each line containing a string of R which is an encoding with minimal length of the corresponding input string.

Remark: On your keyboard the character '|' has a little gap in the middle, you will probably find it on the same key as the backslash ('\'). In Turbo Pascal it is also known as `Chr(124)`.

Example:

REPE.INP

```
this is an example string
it is cute is it not
```

REPE.OUT

```
th(is )an example string
it( is |cute)it not
```

End of problem H

1991 ACM Scholastic Programming Contest
sponsored by AT&T Computer Systems

European Regional Contest
Final Standings

#	Team Name	A	B	C	D	E	F	G	H	Time	Solved	Rank
2	Amsterdam Free U	6	1*	2*	4*	3*	1*	2*	4	1231	6	1
4	RU Groningen	.	.	.	2*	3*	1*	3*	2*	1256	5	2
7	RU Ghent	4	3*	4*	3*	1*	.	1*	3	1284	5	3
3	RU Leiden	2*	4*	1*	1	3*	2	6	.	698	4	4
21	Manchester U	.	2	1*	.	1*	2*	.	.	634	3	5
12	Chemnitz TU	2	2	5	2*	2*	.	1*	.	692	3	6
30	Sofia U	.	.	5	1*	1*	3	3*	.	726	3	7
13	U Twente	2	1	1*	1	1*	.	1	.	352	2	8
14	KU Nijmegen	.	1*	3	1	1*	1	.	.	437	2	9
1	Eindhoven UT	2	2	.	1	2*	2*	2	2	511	2	10
18	Eotvos Lorand U	.	2*	1	.	1*	.	3	.	514	2	11
5	RU Utrecht	2	4	3*	.	3*	4	.	.	550	2	12
27	Warsaw U	.	5*	1	2	5*	.	3	4	595	2	13
28	U of Amsterdam	2	1	1*	2	3*	.	.	1	612	2	14
29	U of Helsinki	.	4*	.	.	3*	.	.	.	669	2	15
26	Ecole Polytechnique	1	3	4	1*	2*	.	.	1	713	2	16
20	Comenius U	.	2	7	2	1*	2	1	3	73	1	17
22	Masaryk U	.	3	3	1	1*	.	2	.	106	1	18
33	U of Bucharest	1	.	.	.	1*	.	2	3	112	1	19
10	Ecole des Mines	.	.	4	.	2*	.	.	1	178	1	20
25	U of Hull	.	.	1	.	4*	.	.	.	256	1	21
16	TU Budapest	2	2	3	.	2*	.	3	2	265	1	22
19	U of Oldenburg	.	3	2	.	3*	.	.	.	343	1	23
8	U Coll Swansea	.	.	2	.	.	3	.	2		.	
23	Jagiellonian U	.	.	.	.	1	1	2	.		.	
32	Imperial College	.	.	1	1	1	.	.	.		.	
17	Catalonia PTU	.	2	.	.	.	.	1	.		.	
24	U of York	.	.	.	.	2	.	.	.		.	
	Total judged runs	25	48	54	24	53	22	36	27	total: 289		
	Total solutions	1	7	7	6	23	4	5	1	total: 54		

Time (in minutes) is used to discriminate between ties.

"*" indicates an accepted solution.