



XXI Maratón Nacional de Programación

ACIS REDIS 2007

ACM ICPC

Problemas

(Este conjunto contiene 8 problemas; páginas numeradas de 1 a 17)



Regionals 2007
acm International Collegiate
Programming Contest



event
sponsor

Problem A

Alphametics

Source file name: `alpham.c`, `alpham.cpp` or `alpham.java`

Alphametics is a term coined by J.A.H. Hunter to design those puzzles where letters represent decimal digits that make true a certain mathematical relation. A well known example for this is the puzzle:

$$\begin{array}{r} \text{S E N D} \\ + \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

In this context, Alphametic Cryptarithm Masters (ACM) is a recently founded enterprise that is interested on applications of this kind of puzzles to cryptography. For that reason, they want to develop software to solve a reduced family of alphametics in an automated way and you are supposed to help them in this task.

Alphametic puzzles of interest to ACM satisfy the following constraints:

- Puzzles are stated by means of an arithmetic equation of the form described by the regular expression:

$$\langle word \rangle (\langle op \rangle \langle word \rangle)^* = \langle word \rangle (\langle op \rangle \langle word \rangle)^*$$

where $\langle word \rangle$ is a sequence of uppercase characters of the English alphabet, and $\langle op \rangle$ is any of the operators in the set $\{+, -\}$.

- No more than ten different characters occur in any puzzle.
- There will be at least one blank between words, operators and the equality symbol '='.
- Each letter in the puzzle statement stands for a different decimal digit $0, \dots, 9$.
- Each word represents a decimal number.
- + and - operators stand for the usual addition and subtraction operations.
- Numbers represented by words cannot begin by leading zeroes. We say that a word representing a number begins with a *leading zero* if it has at least two characters and its left-most character represents zero.

A solution for an alphametic puzzle is a value assignment for the letters in the words of the puzzle statement, such that the equation is satisfied.

Input

The problem input consists of several cases, each one defined by a line with the puzzle statement as described above. It is guaranteed that every problem statement is well formed. The end of the input corresponds to the end of the input file.

The input must be read from the file alpham.in.

Output

Output texts for each input case are presented in the same order that the input is read. For an input case in the puzzle statement, the output should be a ten character expression

$$\langle c_0 \rangle \langle c_1 \rangle \cdots \langle c_9 \rangle$$

where $\langle c_k \rangle$ is the letter of the statement whose value is the digit k . If the digit k is not assigned to any letter, $\langle c_k \rangle$ should be an asterisk, so the given assignment is a solution for the puzzle instance. If such a solution does not exist, a line with ten asterisks should be written.

The output must be written to standard output.

Sample Input	Example of output
SEND + MORE = MONEY CONTEST + ACM = ACIS + ACM + CONTEST VIOLIN + VIOLIN + VIOLA = TRIO + SONATA	OMY**ENDRS ***** AVTSLROIN*

Problem B

Antenna in the Cinoc Mountains

Source file name: cinoc.c, cinoc.cpp or cinoc.java

Cinoc is a mountain region that offers a spectacular natural environment. The area is sometimes called the last wilderness area, but in reality the Cinoc Mountains area is a cultural landscape. The mountains of Cinoc are also an important recreational area for people from around the world and have a particular characteristic: all of them have conic form (a right cone with circular base) and are placed on a completely flat land.

In order to attend any possible emergency, the administration of the Cinoc National Park requires to install two towers of communication, to connect two distant places in the park. The connection between the towers is only possible if there is a *line of sight* between the two towers, that is to say, the highest point of one tower can be seen from the highest point of the other tower. Also, to avoid interferences, the distance from any point in the line of sight to any point on the territory must be greater than zero.

Your task is to write a program that, given a map of the park and the description of the two towers, decides if the connection is possible or not.

Input

The input consists of several test cases. For each test case: the first line contains an integer number k , $0 \leq k \leq 50$, and is followed by $k+2$ lines. In each case the number k is the number of mountains in Cinoc Mountains region. Each of the next k lines contains four positive, integer numbers xm, ym, hm and rm , $0 \leq xm, ym, hm, rm \leq 10000$, describing a conic mountain ((xm, ym, hm) denotes the coordinates of the top of the mountain and rm denotes the radius of the base). The last two lines contain the information associated with the towers: three numbers per line: xt, yt and ht , $0 \leq xt, yt, ht \leq 10000$, ((xt, yt, ht) denotes the location of the tower's top). The last case is followed by a single line containing -1 .

The input must be read from the file cinoc.in.

Output

For each test case, if there exists a valid line of sight between the corresponding top points of the towers, the following line must be printed:

Yes

In other case, the following line must be printed:

No

The answers for the different cases must preserve the order of the input.

The output must be written to standard output.

Sample Input	Output for the sample input
2	Yes
10 10 2 5	No
10 2 2 2	
10 2 3	
2 2 2	
1	
10 2 2 2	
5 2 1	
15 2 1	
-1	

Problem C

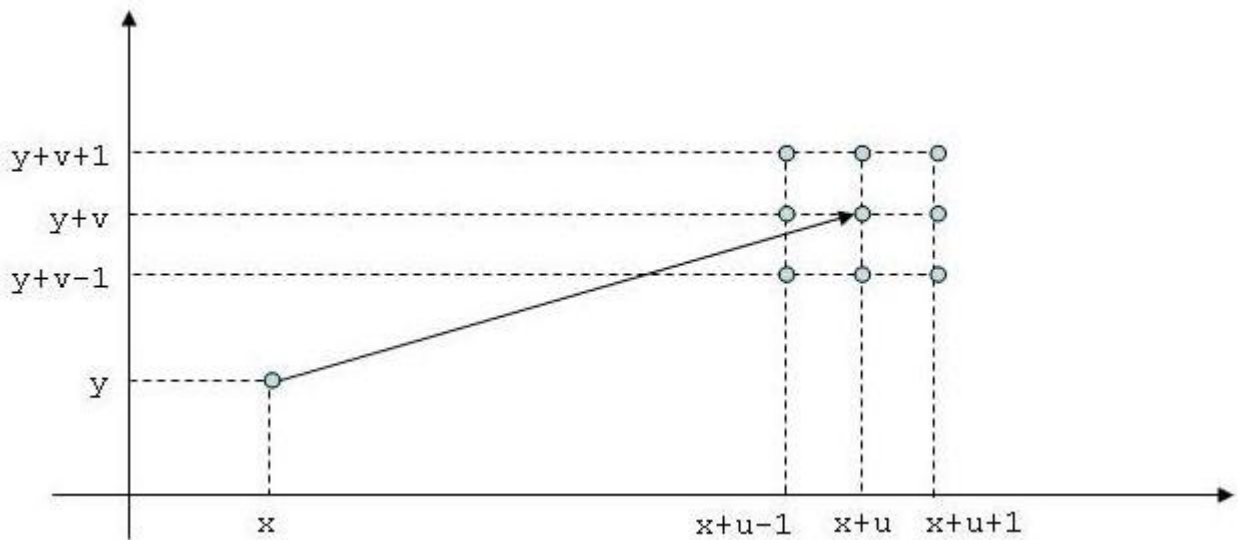
Discrete Pursuit

Source file name: `pursuit.c`, `pursuit.cpp` or `pursuit.java`

Robocops Inc. is a toy manufacturer company that develops a robot game in which a cop tries to catch a thief in a field that is simulated with a rectangular grid whose coordinates are denoted by pairs of integers. The resulting pursuit occurs in a discrete time scale: time is modeled with non-negative integers $0, 1, 2, \dots$

An object on the grid has a speed that determines how the object moves during the simulation. A *speed* is modeled with a pair of integers. The first one is the *horizontal speed* and the second one is the *vertical speed*. Additionally, horizontal and vertical speeds may vary in one unit (each one) to simulate slowing or accelerating.

More exactly: if at time k the object is at location (x, y) with speed (u, v) , then, at time $k + 1$, the object may appear at location $(x', y') = (x + u + \epsilon, y + v + \delta)$, for some $\epsilon, \delta \in \{-1, 0, 1\}$. The speed at time $k + 1$ is $(x' - x, y' - y)$. For an object that moves with constant speed the values ϵ and δ are always 0.



At time 0, the cop is located at the origin of the grid, and the thief is at coordinates $(a, 0)$. The thief is moving with a constant speed; on the other hand, the cop starts still, but may vary his speed according to the given rules. Clearly, the cop catches the thief at time k if at this time the positions of both coincide.

Your task is to develop a program to control the cop in order to catch the thief in an efficient way. Your algorithm should determine the minimum time in which the cop may catch the thief.

Input

The problem input consists of several cases, each one defined by a line with three integer values, separated by blanks, that stand for the initial position a of the thief in the X -axis ($0 \leq a \leq 1000$), the horizontal speed u ($0 \leq u \leq 10$) and the vertical speed v ($0 \leq v \leq 10$) at which he moves. The end of the input corresponds to the end of the input file.

The input must be read from the file pursuit.in.

Output

Output texts for each input case preserve the order in the input file.

For an input case, the output should be an integer that is the minimum time at which the cop may catch the thief.

The output must be written to standard output.

Sample Input	Output for the sample input
1 1 1	2
3 1 0	3

Problem D

DRM

Source file name: drm.c, drm.cpp or drm.java

DRM Inc. is a firm that produces digital road maps. A digital map is a set of places and a set of streets between places. Streets are not oriented, i.e., they are two-way streets.

A road between a place a and a place b is a sequence of places $\langle u_0, u_1, \dots, u_n \rangle$ such that $a = u_0$, $b = u_n$, and there is a street between u_i and u_{i+1} for $0 \leq i < n$.

The definition of a map is accomplished incrementally: a new version of a map is built adding details to an already defined one. The new map must be *consistent* with the old one, i.e., the new one must be *more detailed* than the old one, in the sense that

- the new map has at least the same places than the old one;
- for every street between places u and v in the old map, in the new one there is a road between u and v . Any intermediate place of this road must be a new place (not considered in the old map).

DRM building process includes a comparison step between consecutive map versions in order to assure consistence between them. You must help DRM to evaluate if a map is more detailed than another one.

Input

A map is represented with several input lines:

- the first line contains an identifier for the map
- the following lines, except the last one, contains identifiers of two places that define a street between them. Identifiers are separated with a blank character. It is guaranteed that a street is described only once, but places naming it could be given in any order. On the other hand, streets are named without any specific order.
- the last line contains the string "`* * *`" (star, blank, star, blank, star).

An identifier is a character string without blanks.

The problem input describes several cases, each one consisting of a pair of map representations. For each case you must evaluate if the second map of the given pair is a more detailed version of the first one.

The end of the input is specified by a line with the word `END`.

The input must be read from the file `drm.in`.

Output

Output texts for each input case are presented in the same order that input is read.

For each pair of maps named <id1> and <id2>, if the map named <id2> is more detailed than the map named <id1>, an output line of the form

YES: <id2> is a more detailed version of <id1>

must be written. In other case, the output must be of the form

NO: <id2> is not a more detailed version of <id1>

The output must be written to standard output.

Sample Input	Output for the sample input
COL1 Bogota Cali Bogota Barranquilla * * * COL2 Barranquilla Bogota Armenia Cali Barranquilla Armenia Bogota Cali Cali Barranquilla * * * COL1 Bogota Cali Bogota Barranquilla * * * COL3 Bogota Armenia Armenia Cali Cali Medellin Medellin Barranquilla * * * END	YES: COL2 is a more detailed version of COL1 NO: COL3 is not a more detailed version of COL1

Problem E

eXtreme RISC

Source file name: `xrisc.c`, `xrisc.cpp` or `xrisc.java`

XRISC stands for eXtreme Reduced Instruction Set Computer, a new computer architecture that has taken the RISC paradigm to the extreme: a computer with only one instruction!

The instruction is:

SUBLEQ A B C

which means: *subtract the value in M(A) from M(B) and store it in M(B); if the result is non-positive jump to the instruction in position C.* $M(i)$ represents the value stored in memory position i . Since there is only one instruction, it is unnecessary to represent its opcode explicitly in memory. Therefore, an instruction is stored in main memory using three consecutive memory positions, which correspond to the three instruction parameters.

The computer has a memory of 9999 integer positions, numbered from 0 to 9998. In order to make sense to the instructions, it is required that $0 \leq A, B \leq 9998$ and $9 \leq C \leq 9996$. Values out of these ranges will cause errors, except for the case $C > 9996$, that indicates the end of the program. The memory is organized as follows:

Position	Content
0-8	input/output variables (M0 to M8)
9-9998	program memory (instructions+data)

The following pseudo-code shows a hypothetical XRISC computer simulator:

```
simulate(integer M[0..9998])
  integer pc,A,B,C
  pc = 9
  while (pc<9997)
    A = M[pc]; B = M[pc+1]; C = M[pc+2]
    M[B] = M[B] - M[A]
    if (M[B]>0)
      pc = pc + 3
    else
      pc = C
    end_if
  end_while
end_simulate
```

Each iteration of the above while instruction is called a *simulation cycle*. ACM Inc. (Awkward Computer Makers) has hired you to build a compiler able to generate XRISC machine code to evaluate arithmetic expressions.

Input

The input has several test cases, one test case per line. Each test case corresponds to an arithmetic expression in postfix notation. An expression may contain constants (integer values), input variables (M0 to M8) and arithmetic operators (+, -, *).

The input must be read from the file xrisc.in.

Output

For each test case, an XRISC program must be printed using the following format: the first line must show 'Expression i :', where i is the test case number starting at 1; the second line contains the original expression; the third line indicates m , the number of instructions of the program; and the following m lines contain the XRISC program, one instruction per line, where each instruction is represented by 3 integer values separated by one blank space.

To check the correctness of the output program, the following procedure is used:

- the output program is loaded (starting at memory position 9);
- arbitrary parameter values are stored on positions $M[0], \dots, M[8]$;
- the XRISC machine is simulated as above described;
- the simulation must terminate within 10^7 simulation cycles;
- the value in $M[0]$ must be the same as the result of evaluating the given input expression with the usual semantics for a postfix arithmetic expression (with the same parameter values that the program used).

The output must be written to standard output.

Sample Input	Example of output for the sample input
100 M1 M2 -	Expression 1: 100 4 0 0 12 18 0 15 19 19 10000 -100 0 0 Expression 2: M1 M2 - 4 0 0 12 1 2 15 2 0 18 21 21 10000

Problem F

Greatest Hits!

Source file name: ghits.c, ghits.cpp or ghits.java

Yesterday, Professor Calamaro read in the news that there were rumors about a possible comeback of Soda Stereo, a famous South American rock group. Although Soda Stereo was a well known group of the late '80s, Professor Calamaro had never before heard of them. He decided to go to the nearest music store to buy some of their greatest hits collections.

When he arrived to the music store, he found that there were many different greatest hits CDs. Because he didn't know any of their songs, he decided to spend a certain amount of money to buy a couple of them so that he could listen as many songs as possible. There were two main problems, however: (1) it could be that the money was not enough to buy all the CDs, and (2) many of the CDs have the same songs over and over again.

At that moment, he realized that it would be very useful for his purposes to have a program to decide which CDs to buy so that, altogether, he could maximize the number of non-repeated songs with the money budget that he has. If there were two choices that give the highest possible number of songs, he would prefer the cheapest one. And if two choices give the highest number of songs and both cost the same, the one that has one older CD (that the other does not have) should be preferred. Your task is to develop that program.

Input

The first line contains $N > 0$, the number of cases to analyze. The N cases come in the following lines. Each case is a block describing a possible scenario:

- One line with the Professor's Calamaro budget, an integer value B , $0 < B < 1000$.
- The CD descriptions (at least one description, but no more than 20 of them). Older CDs are listed first.
- Each CD with M ($0 < M < 50$) songs is described with $M + 2$ lines: one line with the name of the CD, M lines with the names of the songs (one line, one song) and one last line with an integer value c ($0 < c < 100$), the CD's cost.
- Names of CDs and song's titles are character strings of a non-zero length with no more than 80 ASCII characters. Money values are given as numerical strings preceded by a '\$' sign. There is not a song title, nor a CD name that may be understood as an amount of money. There are no two CDs with the same name.

The input must be read from the file ghits.in.

Output

The output for every scenario begins with a line containing “Scenario #*i*:”, where *i* is the number of the scenario (numbering starting at 1), followed by a blank and the total number of non-repeated songs he can afford. Then there is a line for each one of the CD names that Professor Calamaro should buy. The names of the CDs to buy are listed from the oldest one to the newest one. At the end of that there is a blank line.

The output must be written to standard output.

Sample Input	Output for the sample input
2	Scenario #1: 4
\$10	COMFORT Y MUSICA
COMFORT Y MUSICA	
En la ciudad	Scenario #2: 5
Un misil	TIT2
Pasos	TIT3
Entre canibales	
\$5	
EL ULTIMO CONCIERTO	
Disco eterno	
Planeador	
Pasos	
\$7	
\$60	
TIT1	
1	
2	
\$25	
TIT2	
2	
1	
3	
\$35	
TIT3	
5	
2	
4	
2	
\$20	

Problem G

Minefield

Source file name: mfield.c, mfield.cpp or mfield.java

There is a minefield in front of your backyard and you do not want to make it your graveyard whenever you try to get pass through it! Your task is to find out if one can cross the minefield, from one corner to the opposite one, without blowing up within certain constraint on the walking distance through the field. In addition to the walking distance constraint, the only available information is a set of points in the minefield that are safe, i.e. places where you can put your foot on without activating a landmine. You can move from one safe point to another safe point if their distance is at most 1.5 meters far from each other.

Input

The input consists of several test cases. Each test case starts with a line specifying the size of the mineland with two integers w and h separated by a blank corresponding, respectively, to the width and the height in meters of a rectangular landmine, $1 \leq w, h \leq 10000$. The next line specifies n , the number of safe points, with $0 \leq n \leq 10000$. The following n lines contain safe point coordinates x and y , separated by a blank, which are floating point numbers with $0 \leq x \leq w$ and $0 \leq y \leq h$. The $n + 1$ line contains a floating point number $d \geq 1.5$ corresponding to the constraint on the walking distance. The end of the test cases is indicated with a line with one * character. All floating point numbers are given with exactly two decimal figures.

The input must be read from the file mfield.in.

Output

One line per test case, preserving the input order. Each output line corresponds to

I am lucky!

if there is a safe path of distance at most d from $(0,0)$ to (w,h) , or to

Boom!

otherwise. You can always assume that the points $(0,0)$ and (w,h) are safe points.

The output must be written to standard output.

Sample Input	Sample output
5 4 7 1.00 1.00 1.00 2.00 2.00 1.00 2.00 2.00 3.00 3.00 4.00 2.00 5.00 3.00 8.50 5 4 8 1.00 1.00 1.00 2.00 2.00 1.00 2.00 2.00 3.00 3.00 4.00 2.00 5.00 3.00 5.00 4.00 7.90 *	I am lucky! Boom!

Problem H

Sonnet Rhyme Verifier

Source file name: `sonnet.c`, `sonnet.cpp` or `sonnet.java`

According to Wikipedia

The term *sonnet* derives from the Provençal word *sonet* and the Italian word *sonetto*, both meaning “little song”. By the thirteenth century, it had come to signify a poem of fourteen lines that follows a strict rhyme scheme and logical structure. The conventions associated with the sonnet have evolved over its history. The writers of sonnets are known as *sonneteers*.

Written in Spanish, sonnets have a well defined structure over the rhymes. Moreover, these days sonneteers are scarce as good weather and the rules governing the rhymes are rather permissive. Let us consider the following poem:

```

ES ELLA
Locura, intensa y sin medida
andando errante te avisté,
una nota en tus manos encontré:
rumor de un amor y su partida.
Adelante, te diré de su vida
aunque en mi regazo ya no esté,
fueron sus labios con que tropecé
los que te llamaron despavorida.
Ojos negros, pestañas en un ramo,
rizos finos, humor, siempre bella;
¡es ella, amiga, a quien amo!
Zagales de secretos sus estrellas,
cómplices la lluvia y el álamo;
... es ella, nuestra musa, ¡es ella!
```

This poem must be considered as a sonnet because it meets the following rules:

1. It has 14 lines.
2. Its rhymes have the structure $ABBAABBACDCDCD$, where A , B , C and D correspond to the suffixes *ida*, *é*, *amo*, and *ella* of the given line. In general, we are to allow any rhyme structure as long as it follows one of the patterns $ABBAABBACDECDE$, $ABBAABBACDEDCE$ or $ABBAABBACDCDCD$.
3. It has hendecasyllable meter.

Today you are to help the local Spanish School of Sonneteers (SSS) writing a program that verifies if a given composition sticks to the first two rules of a Spanish sonnet. The last rule is

to be checked by the SSS's master sonneteer once the given poem passes the tests exercised by the verifier you are about to code.

For the purpose of your work, two lines rhyme if, after *cleaning* them, they have the same suffix. Given a line in the sonnet, its clean version is the result of deleting blanks, punctuation symbols (¡, !, ,, ., :, ;, ¿, ?, -) and the last s (if there is one) from the end of the original line.

Input

The input consists of several instances of the problem, each one occurring separated by a new line. Each instance consists of $n > 2$ lines: the first one consists of a list of s suffixes ($4 \leq s \leq 5$), separated by a blank character. The suffixes are to be called A , B , C , D and E according to the order in which they occur and the total number of them. The second line contains the title of the composition. The following $n - 2$ lines contain the actual lines of the composition. Each line in the sonnet has at most 80 characters. You can safely assume that, in a case, a given suffix is not suffix of another one.

The input must be read from the file sonnet.in.

Output

For each problem instance, and according to the order of the input, your program should print a sentence of the form

<NAME>: <STRUCTURE>

if the given composition adheres to the first two rules of a spanish sonnet or

<NAME>: Not a chance!

in other case, where <NAME> is to be replaced by the name of the composition and <STRUCTURE> corresponds to the structure of the rhyme, and is given by the order in which the suffixes occur.

The output must be written to standard output.

Sample Input	Output for the sample input
<p>ida é amo ella ES ELLA Locura, intensa y sin medida andando errante te avisté, una nota en tus manos encontré: rumor de un amor y su partida. Adelante, te diré de su vida aunque en mi regazo ya no esté, fueron sus labios con que tropecé los que te llamaron despavorida. Ojos negros, pestañas en un ramo, rizos finos, humor, siempre bella; ¡es ella, amiga, a quien amo! Zagales de secretos sus estrellas, cómplices la lluvia y el álamo; ... es ella, nuestra musa, ¡es ella!</p> <p>ura ima illa eto OTr0 Esta composición posiblemente rima pero nunca será un soneto con esta frase ya no sirve ni tampoco porque está incompleto.</p>	<p>ES ELLA: ABBAABBACDCDCD OTr0: Not a chance!</p>