acm International Collegiate Programming Contest 2002 IBM event sponsor

ACM International Collegiate Programming Contest 2002

South America

9th November 2002

(This problem set contains 8 problems; pages are numbered from 1 to 16)

Hosted by

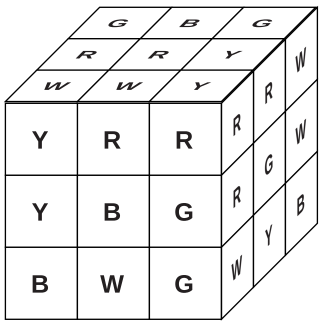
University of Buenos Aires, Buenos Aires, Argentina
University of São Paulo, São Paulo, Brazil
Federal University of Pernambuco, Recife, Brazil
Federal University of Rio Grande do Sul, Porto Alegre, Brazil
University of Atacama, Copiapó, Chile
University Nueva Esparta, Caracas, Venezuela
University Lisandro Alvarado, Barquisimeto, Venezuela

Problem A

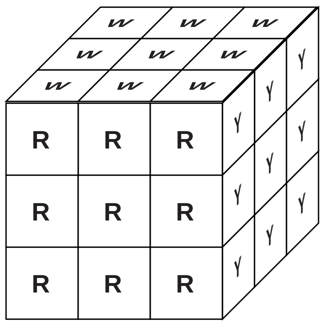
Grandpa's Rubik Cube

Input: cube.in

A very well-known toy/pastime, called Rubik's cube, consists of a cube as shown in Figure 1a, where letters stand for colors (e.g. B for blue, R for red,...). The goal of the game is to rotate the faces of the cube in such a way that at the end each face has a different color, as shown in Figure 1b. Notice that,



(a) Scrambled



(b) Winning position

Figure 1: Rubik Cube

when a face is rotated, the configuration of colors in all the adjacent faces changes. Figure 2 illustrates a rotation of one of the faces. Given a scrambled configuration, reaching the final position can be quite challenging, as you may know.

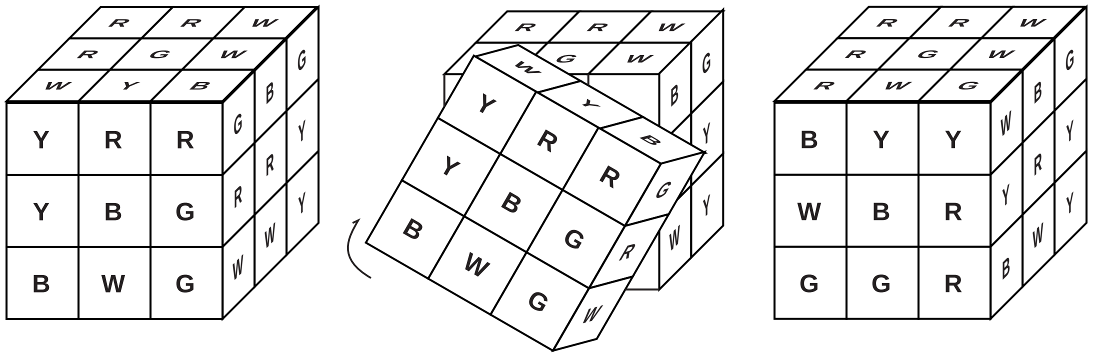


Figure 2: Rotation example

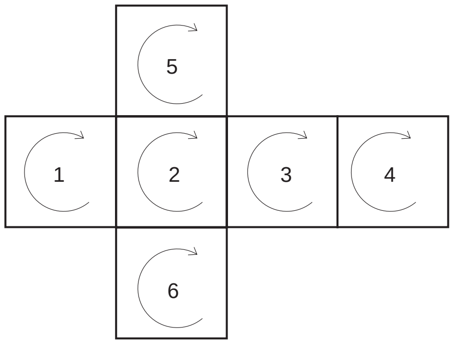
But your grandpa has many years of experience, and claims that, given any configuration of the Rubik cube, he can come up with a sequence of rotations leading to a winning configuration.

In order to show all faces of the cube we shall represent the cube as in Figure 3a. The six colors are Yellow, Red, Blue, Green, White and Magenta (represented by their first letters).

You will be given an initial configuration and a list of rotations. A rotation will be represented by an integer number, indicating the face to be rotated and the direction of the rotation (a positive value means clockwise rotation, negative value means counter-clockwise rotation). Faces of the cube are numbered as shown in Figure 3b. You must write a program that checks whether the list of rotations will lead to a winning configuration.

```
      W R W
      W W M
      G W M
Y Y B R M G W G G R B R
Y Y M B M M Y G G W R R
M G R Y Y M B G M Y W B
      B B Y
      B B R
      G R W
```

(a) Colors



(b) Face identifiers for rotations

Figure 3: Representation of the cube

Input

The input contains several test cases. The first line of the input is an integer which indicates the number of tests. Each test description consists of ten lines of input. The first nine lines of a test will describe an initial configuration, in the format shown in Figure 3a. The next line will contain a list of rotations, ending with the value 0.

Output

For each test case your program should print one line. If your grandpa is correct, print “Yes, grandpa!”, otherwise print “No, you are wrong!”.

(See next page for example.)

Sample input	Output for the sample input

Sample input

```
3
      G Y Y
      G Y Y
      G Y Y
W W W Y R R M M M G G B
W W W Y R R M M M G G B
W W W Y R R M M M G G B
      R B B
      R B B
      R B B
-1 0
      G Y Y
      G Y Y
      G Y Y
W W W Y R R M M M G G B
W M W Y R R M W M G G B
W W W Y R R M M M G G B
      R B B
      R B B
      R B B
-1 0
      M W M
      W W G
      W W Y
G Y Y M M B M B G W R B
B Y Y M M B M G G W R R
Y M G W B B R R G R R W
      R Y Y
      G B Y
      R G B
+4 +6 -2 +3 -4 +2 -3 -6 0
```

Output for the sample input

```
Yes, grandpa!
No, you are wrong!
Yes, grandpa!
```

Problem B

This Sentence is False

Input file: sentence.in

The court of King Xeon 2.4 is plagued with intrigue and conspiracy. A document recently discovered by the King's Secret Service is thought to be part of some mischievous scheme. The document contains simply a set of sentences which state the truth or falsehood of each other. Sentences have the form "Sentence *X* is true/false" where *X* identifies one sentence in the set. The King's Secret Service suspects the sentences in fact refer to another, yet uncovered, document.

While they try to establish the origin and purpose of the document, the King ordered you to find whether the set of sentences it contains is consistent, that is, if there is a valid truth assignment for the sentences. If the set is consistent, the King wants you to determine the maximum number of sentences which can be made true in a valid truth assignment for the document.

Input

The input file contains several instances of documents. Each document starts with a line containing a single integer, *N*, which indicates the number of sentences in the document ($1 \le N \le 1000$). The following *N* lines contain each a sentence. Sentences are numbered sequentially, in the order they appear in the input (the first is sentence 1, the second is sentence 2, and so on). Each sentence has the form "Sentence *X* is true." or "Sentence *X* is false.", where $1 \le X \le N$. The value $N = 0$ indicates the end of input.

Output

For each document in the input your program should output one line. If the document is consistent, your program should print the maximum number of sentences in a valid truth assignment for the document. Otherwise your program should print the word 'Inconsistent'.

Sample input	**Output for the sample input**
1 Sentence 1 is false. 1 Sentence 1 is true. 5 Sentence 2 is false. Sentence 1 is false. Sentence 3 is true. Sentence 3 is true. Sentence 4 is false. 0	Inconsistent 1 3

Problem C

Will Indiana Jones Get There?

Input file: get.in

Indiana Jones is in a deserted city, annihilated during a war. Roofs of all houses have been destroyed and only portions of walls are still standing. The ground is so full of mines that the only safe way to move around the city is walking over the remaining walls. The mission of our hero is to save a person who is trapped in the city. In order to move between two walls which are not connected Indiana Jones thought of taking with him a wooden board which he could place between the two walls and then cross from one to the other.

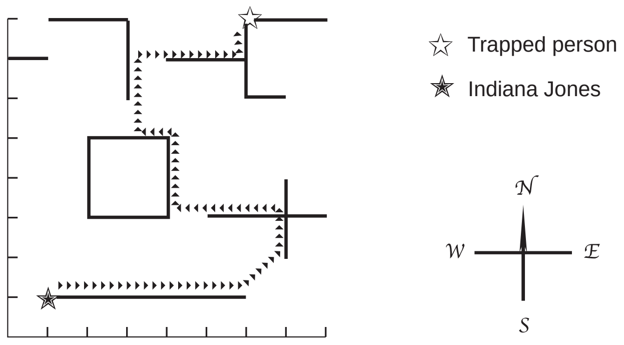


Fig. 1: City map with route used by Indiana Jones

Initial positions of Indiana Jones and the trapped person are both on some section of the walls. Besides, walls are either in the direction South-North or West-East.

You will be given a map of the city remains. Your mission is to determine the minimum length of the wooden board Indiana Jones needs to carry in order to get to the trapped person.

Input

Your program should process several test cases. Each test case starts with an integer N indicating the number of wall sections remaining in the city ($2 \le N \le 1000$). Each of the next N lines describes a wall section. The first wall section to appear is the section where Indiana Jones stands at the beginning. The second section to appear is the section where the trapped person stands. Each wall section description consists of three integers X, Y and L ($-10000 \le X, Y, L \le 10000$), where X an Y define either the southernmost point of a wall section (for South-North sections) or the westernmost point (for West-East wall sections). The value of L determines the length and direction of the wall: if $L \ge 0$, the section is West-East, with length L; if $L < 0$, the section is North-South, with length $| L |$. The end of input is indicated by $N = 0$.

Output

For each test case in the input your program should produce one line of output, containing a real value representing the length of the wooden board Indiana Jones must carry. The length must be printed as a real number with two-digit precision, and the last decimal digit must be rounded. The input will not contain test cases where differences in rounding are significant.

Sample input

```
14
1 1 5
6 8 2
7 2 -2
5 3 3
2 5 2
2 3 2
2 3 -2
4 3 -2
0 7 1
1 8 2
3 6 -2
4 7 2
6 6 1
6 6 -2
3
-10 0 20
-5 1 10
50 50 100
0
```

Output for the sample input

```
1.41
1.00
```

Problem D

Duty Free Shop

Input file: shop.in

Pedro travelled to Europe to take part in the International Olympiad in Informatics and is coming back home. Since all his friends asked him to bring them some gift, he bought two big bags of chocolates (one of Mindt and one of Lilka). Each of these two bags contains a certain number of small chocolates. Buying those two bags was much less expensive than buying smaller, individual boxes of chocolates. At home, Pedro has some empty chocolate boxes that he kept from other trips. Pedro intends to distribute the chocolates he just bought into these smaller boxes, to give them to his friends.

As soon as Pedro begins filling the small boxes, he realizes he has a big problem: since he has two different brands of chocolates, if he mixes chocolates of different brands into one small box, the friend who receives this small box will discover Pedro's trick to save money, and will not be pleased with him.

You must help poor Pedro distribute the chocolates into the small boxes in such a way that every small box is completely full, and contains only one brand of chocolates. A number of chocolates may however be left unassigned to any box (Pedro will keep these chocolates to himself).

Input

The input file contains several instances of the problem. Each instance consists of three lines. The first line contains two integers M and L that indicate respectively the number of chocolates Mindt and Lilka Pedro bought ($0 \leq M, L \leq 1000$). The next line contains an integer N representing the number of small boxes Pedro has ($N \leq M+L$). The third line contains N integers indicating the capacity $C_i > 0$ of box number i (that is, the number of chocolates needed to fill that box). The end of input is indicated by $M = L = 0$.

Output

For each instance of the input your program must produce one line of output. If it is possible to distribute the chocolates as defined in the problem statement, print the number of boxes to be filled with Mindt chocolate, followed by a space, followed by the list of box numbers, in ascending order. Each box number in the list should be followed by a space. If it is impossible to distribute the chocolates, print “Impossible to distribute”. If more than one solution exists, print any one.

Sample input

```
12 9
4
5 2 8 5
100 120
5
21 32 110 54 3
0 0
```

Output for the sample input

```
3 1 2 4
Impossible to distribute
```

Problem E

Not Too Convex Hull

Input file: convex.in

Nails and Rubber Bands. That is the suggestive name of a game played by a group of children (all of them offspring of geometry teachers). The children fix a number of nails on a plank of wood, randomly placed. Then they choose one of the nails to be the Origin, and a number *B* of rubber bands. The challenge is to use the *B* rubber bands to wrap the nails so that (i) each rubber band wraps a subset of the nails; (ii) all nails are inside some wrapping; (iii) wrappings do not overlap each other except at the Origin nail, which is touched by all rubber bands; (iv) rubber bands must form wrappings which are convex polygons with at least three corners; and (v) the total area inside the wrappings is the smallest among all possible ways of wrapping the nails. An instance of the game is shown in Figure 1.

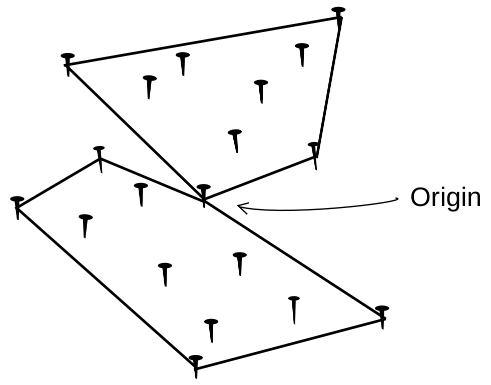


Figure 1: A game with 19 nails and 2 rubber bands

Input

Your program should solve several instances of the game. Each game description starts with a line containing two integers *B* and *N*, indicating respectively the number of rubber bands and the number of nails ($2 \leq B \leq 50$ and $2B+1 \leq N \leq 101$). The following *N* lines describe the position of the nails, each line containing two integers *X* and *Y* ($-10000 \leq X, Y \leq 10000$). The origin is the first nail in the input. The end of input is indicated by $B = N = 0$.

In all instances in the input:

- no two nails are in the same point;
- no three nails are in the same line;
- the origin nail does not belong to the convex hull of all nails (that is, if you use one rubber band to wrap all nails, it does not touch the origin nail).

Output

For each game in the input your program should output one line, describing the smallest total area inside the wrappings. The area must be printed as a real number with two-digit precision, and the last decimal digit must be rounded. The input will not contain test cases where differences in rounding are significant.

Sample input

```
2 5
0 0
9 4
-8 8
-10 -2
4 -8
2 6
0 0
3 6
-5 7
-4 -6
10 -10
3 5
0 0
```

Output for the sample input

```
92.00
74.00
```

Problem F

I hate SPAM, but some people love it

Input file: spam.in

Nowadays, unfortunately, SPAM messages are becoming more and more common. Some of them may have a multiplicative effect since they ask you to forward them to all your friends. Some SPAM messages wish good luck, others promise you will become rich, and others just remind you how important it is to tell your friends that you care for their friendship. Here is an example of a SPAM:

```
From: Alice
To: Bob, Mary, Julia, Paul

Hi, this is a good luck email. I wish you become a millionaire, but
that is up to you. If you
* send this email to 10 or more people you will be a millionaire
* send this email to 5 or more people you will be rich
* send this email to less than 5 people you will be poor
As I said, it is up to you. Write your email and be rich! :-)

Alice
```

People usually react in two different ways when they receive a SPAM:

- They discard the message immediately without even reading it (they hate SPAM);
- They forward the message to everyone they know (they love SPAM).

For this problem, we will assume everyone loves SPAM, but one never forwards the same message twice. Each SPAM message has a different effect based on the number of friends you forward the message to. For example: a SPAM message could tell that you will be poor if you send the message to 5 friends, but you will be the rich if you send to 10, and you will be the richest man in the world if you send it to 20 friends, and so on.

We will consider only SPAM messages similar to the example above. More specifically, a SPAM message will define two threshold values T_1 and T_2 and three attributes A_1, A_2 and A_3. A person acquires one of the three attributes depending on the number of messages forwarded for that specific SPAM. If a person forwards T messages and $T < T_1$ then her/his attribute is A_1, if $T_1 \leq T < T_2$ then her/his attribute is A_2, otherwise her/his attribute is A_3.

You will be given the names of a group of people, and for each person in the group, the set of friends she/he knows the email address. You will also be given a set of distinct SPAM messages, and for each SPAM message its threshold values and attributes, and the information about which person started it. You have to write a program that determines, for each person in the given group, which attributes she/he acquired, based on all the SPAM they forward.

You may assume that the SPAM originator will have at least one friend (in other words, she/he will send at least one message), and a person will not send messages to herself.

Input

Your program should process several test cases. The first line of a test case contains an integer N indicating the number of persons in the group ($2 \le N \le 20$). In the input, a person is identified by an integer from 1 to N. The following N lines contain each a list of friends of each person (the i-th line contains the list of friends of person number i). The list of friends of person i describes the friends person i knows the email address, and consists of a list of integers F_i ($1 \le F_i \le N$, $F_i \ne i$) terminated by the value 0 (zero). Following the list of friends comes the description of the SPAM messages (there will be at most 100 messages). Each description appears in a different line. The description consists of an integer P identifying the person who is the SPAM originator ($2 \le P \le N$); two integers T_1 and T_2 representing the threshold values; and the three attributes A_1, A_2 and A_3 (each attribute is a word of no more than 20 letters). The SPAM list ends with a line containing only the value 0 (zero). The following N lines contain each a name, which is single word with no more than 20 letters. The name in the i-th line is the name of person number i. The end of input is indicated by $N = 0$.

Output

For each test case your program should output a list of names followed by the attributes they acquired. Your program should write the persons names in the order they appear in the input, followed by ':' and by a space, followed by their attributes according to the SPAM they sent. Attributes should be written in the order they appear in the input; each attribute should be followed by a space.

(See next page for example.)

Sample input

```
5
2 3 0
1 3 5 4 0
5 0
0
4 1 0
1 2 4 poor rich millionaire
5 3 10 sad normal happy
0
Bob
Paul
Mary
Alice
Julia
6
2 0
1 3 0
1 2 4 0
1 2 3 5 0
1 2 3 4 0
1 3 4 0
1 2 4 red green blue
1 2 4 dumb normal smart
6 3 5 ugly bad good
0
Peter
Paul
Victoria
John
Julia
Anne
0 0
```

Output for the sample input

```
Bob: rich sad
Paul: millionaire normal
Mary: poor sad
Alice: poor sad
Julia: rich sad
Peter: red dumb ugly
Paul: green normal ugly
Victoria: green normal bad
John: blue smart bad
Julia: blue smart bad
Anne: red dumb bad
```

Problem G

Noise Effect

Input file: noise.in

Cheap small industrial scanners can only acquire images on gray scale, which are images where the pixels have intensity values in the integer range [0..255]. A company that builds automatic vending machines wants to use these small scanners to validate the tokens used in its machines. Tokens are small square chips of metal with holes strategically pierced. Tokens with different holes are used for different values.

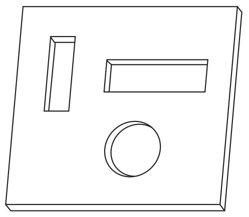


Fig. 1: Token for a vending machine

A scanner will produce an image of the token introduced by the client and a computer program will validate it. In the scanner image, metal appears as dark pixels (values near 0) and holes appear as light pixels (values near 255). There are two problems that must be solved in the validation process. The first problem is that, since the token is square, a client can introduce it in the machine slot in several possible ways. The second problem is due to the poor quality of the image generated by those cheap scanners, which will contain 'noise' (errors). To validate the token, the machine will compare the scanner output to a 'standard image' of the token, previously produced using a high quality scanner.

You must write a program which, given the standard image of a token and an image produced by the machine scanner, determines the confidence degree that the token introduced is a valid one. The confidence degree is the percentage of pixels in the scanner image whose intensity value differ by 100 or less from corresponding pixels in the standard image. As the token may have been introduced in several ways, we are interested in the highest possible confidence degree, considering all possible token positions.

Input

Your program should process several test cases. Each test case specifies the size of the token image and the pixel values for the standard and scanned images. The first line of a test case contains an integer L that indicates the size, in pixels, of the image ($1 \leq L \leq 400$). The next L lines will contain L integers each, representing the pixel values for the rows of the standard image. Following that, the next L lines will contain the pixel values for the rows of the scanned image.

The end of input is indicated by $L = 0$.

Output

For each test case your program should output a single line containing the confidence degree for the corresponding image. The confidence degree must be printed as a real number with two-digit precision, and the last decimal digit must be rounded. The input will not contain test cases where differences in rounding are significant.

Sample input

```
4
250 251 249 250
251 120 245 248
248 5 190 247
5 5 180 246
0 1 240 240
250 2 250 254
244 251 255 253
230 250 250 252
3
250 250 250
150 0 150
250 2 250
253 150 253
0 2 248
251 150 250
5
255 255 255 255 255
255 0 255 0 0
255 0 0 255 255
255 255 0 255 255
255 255 255 255 0
255 0 255 255 0
255 0 255 255 255
255 255 0 0 255
255 0 0 255 255
154 154 255 255 255
0
```

Output for the sample input

```
93.75
100.00
92.00
```

Problem H

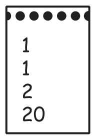
Supermarket

Input file: market.in

Mr. Jones is an exemplary husband. Every Saturday morning Mrs. Jones gives him a list of goods to be bought from the supermarket and he buys exactly what he has been asked for, always choosing the brands with lowest prices. But Mr. Jones hates going to the supermarket on a Saturday, since its aisles are packed with shoppers. He wants to change the way he does his shopping. Instead of going to and fro to buy the products on his wife's list, he will try to get the goods on the list going through each aisle only once, picking up the products in the exact order given in the list. So he asked you to write a program to help him with his new style of shopping.

Given the information about products available in the supermarket together with their prices in the order in which they appear in Mr. Jones' way and the list of products given by his wife, your program must determine the least cost that he would pay.

Mr. Jones buys the products in the order in which they appear in Mrs. Jones' list and he never goes back as he walks down the aisles. Therefore, if he buys the *i*-th product on his way as the *j*-th item on the list, the next product to be bought is the *(j+1)*-th item of the list – and it must be bought from the products that come after *i* in his path. The figure below shows an example where products are identified by integers. Note that different brands of the same product may appear separately. In the example Mr. Jones must buy products 1,1,2,20 (notice that product 1 appears twice in the list). For the example, the least cost for Mr. Jones following his constraints is 21.30. Notice that with this new way of shopping it may be impossible for Mr. Jones to buy all the goods on Mrs. Jones list; in that case, your program should warn Mr. Jones.



(a) Mrs. Jones' list

2	**1**	**20**	**1**	**5**	**2**	**20**	**20**
0.29	0.30	0.15	1.0	0.05	10.00	20.00	10.00

(b) List of products with respective prices, and order they appear on Mr. Jones' way down the aisles

Input

Your program should process data for several shopping sessions. The first line in the description of a shopping session contains two integers *M* and *N*; *M* indicates the number of items in Mrs. Jones' list ($1 \le M \le 100$) and *N* represents the total number of products available in the supermarket ($1 \le N \le$ 100,000). The next line contains *M* integers X_i representing the list of products in Mrs. Jones' list ($1 \le X_i \le 100{,}000$, $1 \le i \le M$). Then *N* lines follow, representing the supermarket products in the order in which they appear in Mr. Jones' way. Each of those lines contains an integer *K* and a real *P* which represent respectively a product identifier and its price ($1 \le K \le 100{,}000$). The end of input is indicated by $M = N = 0$.

Output

For each shopping session in the input, your program should produce one line of output, containing the least cost that Mr. Jones would pay. If it is not possible to buy all the goods for the session, print the word ‘Impossible’. The cost must be printed as a real number with two-digit precision, and the last decimal digit must be rounded. The input will not contain test cases where differences in rounding are significant.

Sample input

```
4 8
1 1 2 20
2 0.29
1 0.30
20 0.15
1 1.00
5 0.05
2 10.00
20 20.00
20 10.00
2 5
1 2
3 1.00
4 1.00
2 0.01
1 1.00
2 1.50
2 3
1 2
2 0.05
1 10.00
1 3.00
0 0
```

Output for the sample input

```
21.30
2.50
Impossible
```