A 684 - Integral Determinant

Time limit: 3.000 seconds

Write a program to find the determinant of an integral square matrix. Note that the determinant of a square matrix can be defined recursively as follows: the determinant of a 1 $\times$ 1 matrix $M = (a_{1,1})$ is just the value $|M| = a_{1,1}$; further, the determinant of an $n \times n$ matrix is

$$|M| = \sum_{i=1}^{n} (-1)^{i+1} \cdot |M_{1,j}|.$$

Here the notation $M_{1,i}$ is the $(n-1) \times (n-1)$ matrix by removing the first row and the i^{th} column of the original $n \times n$ matrix M.

A straightforward method of calculating the determinant of an $n \times n$ matrix by the recursive method will end up with $n!$ multiplications, a very time-consuming algorithm. To give you a feeling about this, note that 15! = 1,307,674,368,000. To reduce the time complexity, there are two ways of modifying the original matrix for easier computation.

1. Exchanging two columns (or rows) of a matrix will change the sign of the determinant; for example

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} = - \begin{vmatrix} 2 & 1 \\ 4 & 3 \end{vmatrix}$$

2. Multiplying one column by any constant, and add them to another column will not change the value of the determinant; for example:

$$\begin{vmatrix} 2 & 1 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = - \begin{vmatrix} 2 & 1 & 3+2\times 2 \\ 4 & 5 & 6+4\times 2 \\ 7 & 8 & 9+7\times 2 \end{vmatrix}$$

Using the above methods, you shall be able to write a program for computing the determinants of matrices, even for a size like 30 $\times$ 30, very efficiently. Below is an example to show how this can be done:

$$\begin{vmatrix} 2 & 3 & 5 \\ 1 & 6 & 7 \\ 4 & 8 & 9 \end{vmatrix} = \begin{vmatrix} 2 & 3-2 & 5-2\times 2 \\ 1 & 6-1 & 7-1\times 2 \\ 4 & 8-4 & 9-4\times 2 \end{vmatrix} = \begin{vmatrix} 2 & 1 & 1 \\ 1 & 5 & 5 \\ 4 & 4 & 1 \end{vmatrix} = - \begin{vmatrix} 1 & 1 & 2 \\ 5 & 5 & 1 \\ 1 & 4 & 4 \end{vmatrix}$$

$$= -\begin{vmatrix} 1 & 1-1 & 2-1\times 2 \\ 4 & 5-5 & 1-5\times 2 \\ 1 & 4-1 & 4-1\times 2 \end{vmatrix} = -\begin{vmatrix} 1 & 0 & 0 \\ 5 & 0 & -9 \\ 1 & 3 & 2 \end{vmatrix} = -\begin{vmatrix} 0 & -9 \\ 3 & 2 \end{vmatrix} = -27$$

Note that the answer shall be an *integer*. That is, all the operations needed are just integer operations; by reducing to floating numbers would result in the round-off errors, which will be considered as the **wrong** answer. Do not worry about the problem of integral overflows problem. You can assume that the given data set will not cause the integer overflow problem. What is emphasized here is the required integer precision. Anyway use of floating numbers is not forbidden.

Input

em Several sets of integral matrices. The inputs are just a list of integers. Within each set, the first integer (in a single line) represents the size of the matrix, *n*, which can be as large as 30, indication an $n \times n$ matrix. After *n*, there will be *n* lines representing the *n* rows of the matrix; each line (row) contains exactly *n* integers. Thus, there is totally n^2 integers for the particular matrix. These matrices will occur repeatedly in the input as the pattern described above. An integer *n* = 0 (zero) signifies the end of input.

Output

For each matrix of the input, calculate its (integral) determinant and output it in a line. Output a single star (*) to signify the end of outputs.

Sample Input

```
2
5 2
3 4
3
2 3 5
1 6 7
4 8 9
0
```

Sample Output

```
14
-27
*
```

B 634 - Polygon

Modern graphic computer programs can, among other, even more stunning capabilities, fill a closed region. Though not all of them can protect the user from accidentally choosing to fill the background rather than the inner part. Besides being a channel hopper at home your boss' favourite hobby is colouring the pictures, you cannot protest long about adding this magnificent protection feature to his graphic program.

This means that your job is to write a program, which determines whether a point belong to a polygon, given the array of its vertices.

To make life a bit simpler you may assume that:

- all edges of the polygon are vertical or horizontal segments
- lengths of all the edges of the polygon are even integer numbers
- co-ordinates of at least one vertex are odd integer numbers
- both co-ordinates of any vortex cannot be divisible by 7 at the same time
- the investigated point P has both co-ordinates being even integer numbers
- the polygon has at most 1000 vertices
- co-ordinates of the vertices lay in the range: -10000..10000.

Input

Input data may consist of several data sets, each beginning with a number of polygon's vertices (n). Consecutive n lines contain co-ordinates of the vertices (x followed by y). Then go the co-ordinates of investigated point P. Input data end when you find 0 the number of polygon's vertices.

Output

For each polygon and each point P you should print one character (in separate lines): T when P belongs to the polygon or F otherwise.

Sample Input	Sample Output
4 1 1 1 3 3 3 3 1 2 2 12 1 1 1 9 3 9 3 5 5 5 5 9 7 9 7 1 5 1 5 3 3 3 3 1 4 2 0	T F

C 624 - CD

Time limit: 3.000 seconds

You have a long drive by car ahead. You have a tape recorder, but unfortunately your best music is on CDs. You need to have it on tapes so the problem to solve is: you have a tape *N* minutes long. How to choose tracks from CD to get most out of tape space and have as short unused space as possible.

Assumptions:

- number of tracks on the CD. does not exceed 20
- no track is longer than *N* minutes
- tracks do not repeat
- length of each track is expressed as an integer number
- *N* is also integer

Program should find the set of tracks which fills the tape best and print it in the same sequence as the tracks are stored on the CD

Input

Any number of lines. Each one contains value *N*, (after space) number of tracks and durations of the tracks. For example from first line in sample data: *N*=5, number of tracks=3, first track lasts for 1 minute, second one 3 minutes, next one 4 minutes

Output

Set of tracks (and durations) which are the correct solutions and string ``sum:" and sum of duration times.

Sample Input

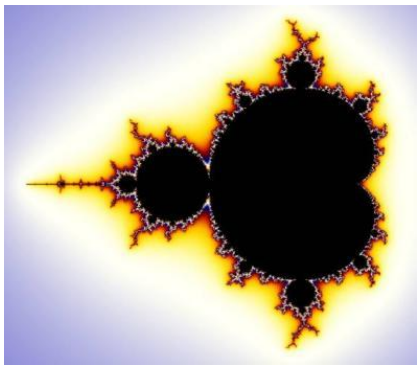
```
5 3 1 3 4
10 4 9 8 4 2
20 4 10 5 7 4
90 8 10 23 1 2 3 4 5 7
45 8 4 10 44 43 12 9 8 2
```

Sample Output

```
1 4 sum:5
8 2 sum:10
10 5 4 sum:19
10 23 1 2 3 4 5 7 sum:55
4 10 12 9 8 2 sum:45
```

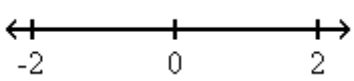
D 918 - ASCII Mandelbrot

The Mandelbrot set, named after Benoit Mandelbrot, is a **fractal**. Fractals are beautiful objects that display self-similarity at various scales. Magnifying a fractal reveals small-scale details similar to the large-scale characteristics. Although the Mandelbrot set is self-similar at magnified scales, the small scale details are not identical to the whole. In fact, the Mandelbrot set is infinitely complex. Yet the process of generating it is based on an extremely simple equation involving complex numbers.

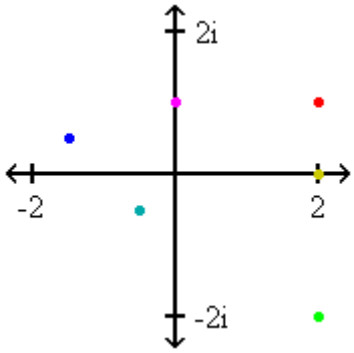


The Mandelbrot set is a mathematical set, a collection of **complex numbers**. Remember that complex numbers have two parts: a real one and an imaginary one. The imaginary part is equal to a real number times a special number called ***i***. For example, a valid complex number is `2.5+3i`.

The number *i* was invented because no real number can be squared (multiplied by itself) and result in a negative number. The number *i* is defined to be the square root of `-1`. So when you square an imaginary number you can get a negative number. For example, `3i` squared is `-9`.

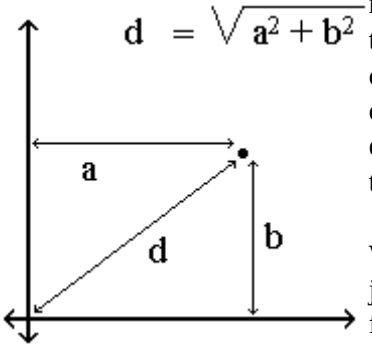


Real numbers can be represented on a one dimensional line called the real number line. Negative numbers like `-2` are plotted to the left of zero and positive numbers like `2` are plotted to the right of zero. Any real number can be graphed on the real number line.



Since complex numbers have two parts, a real one and an imaginary one, we need a second dimension to graph them. We simply add a vertical dimension to the real number line for the imaginary part. Since our graph is now two-dimensional, it is a plane, the complex number plane. We can graph any complex number on this plane. The dots on this graph represent the complex numbers: [2 + 1*i*], [-1.5 + 0.5*i*], [2 - 2*i*], [-0.5 - 0.5*i*], [0 + 1*i*], and [2 + 0*i*].

The Mandelbrot set is a set of complex numbers, so we graph it on the complex number plane. However, first we have to find many numbers that are part of the set. To do this we need a test that will determine if a given number is inside the set or outside the set. The test is based on the equation $\mathbf{Z = Z^2 + C}$. `C` represents a constant number, meaning that it does not change during the testing process. `C` is the number we are testing, the point on the complex plane that will be plotted when testing is complete. `Z` starts out as zero, but it changes as we repeatedly iterate this equation. With each iteration we create a new `Z` that is equal to the old `Z` squared plus the constant `C`. So the number `Z` keeps changing throughout the test.



We're not really interested in the actual value of `Z` as it changes, we just look at its magnitude. The magnitude of a number is its distance from zero. To calculate the magnitude of a complex number, we add the square of the number's distance from the x-axis (the horizontal real axis) to the square of the number's distance from the y-axis (the imaginary vertical axis) and take the square root of the result.

As we iterate our equation, `Z` changes and so does its magnitude. The magnitude of `Z` will do one of two things. It will either stay equal to or below `2` forever, or it will eventually surpass two (strictly bigger than

2). Once the magnitude of `Z` surpasses `2`, it will increase forever. In the first case, where the magnitude of `Z` stays small, the number we are testing is part of the Mandelbrot set. If the magnitude of `Z` eventually surpasses `2`, the number is not part of the Mandelbrot set.

As we test many complex numbers we can graph the ones that are part of the Mandelbrot set on the complex number plane. If we plot thousands of points, an image of the set will appear, as illustrated in the figure in the upper right corner. We can add color to the image if we add colors to the points that are not inside the set, according to how many iterations were required before the magnitude of `Z` surpassed two.

To make exciting images of tiny parts of the Mandelbrot set, we just zoom in on it, trying to perceive its full infinite beauty.

Problem

Your task is to plot a region of the Mandelbrot set using only simple text, this is, only with ASCII characters.

Input

The first line of input contains an integer **T** which is the number of test cases that follow.
Each test case is given in a line with the following format:

"CHARS MINI MAXI PRECI MINR MAXR PRECR", where

- **CHARS** represents the set of chars to use in the plotting, always enclosed in quotes, and always with size 12 (the set of chars never includes quotes and spaces);
- **MINI** and **MAXI** are two real numbers representing the lower and upper bound of the imaginary part in the plot;
- **MINR** and **MAXR** are two real numbers representing the lower and upper bound of the real part in the plot;
- **PRECI** and **PRECR** are two real numbers representing the precision that the plot must have in what respects to imaginary and real part, respectively.

What you must do is to plot the following graph:

```
[MINI      , MINR] [MINI      , MINR+PRECR] (...)
[MINI+PRECI , MINR] [MINI+PRECI , MINR+PRECR] (...)
[MINI+2*PRECI, MINR] [MINI+2*PRECI, MINR+PRECR] (...)
(...)          (...)          (...)
(...)          (...)          [A,B]
```

where [A,B] are the last coordinates that are smaller or equal than [MAXI,MAXR].

Each one of this coordinates must be plotted as a single char. If one iteration was required before the magnitude of Z surpassed two, then the first char of CHARS is plotted, if two iterations are needed, then the second char is plotted and so on until 12 iterations. If after twelve iterations the magnitude of Z has not surpassed Z, then the char " " (space) is plotted.

Output

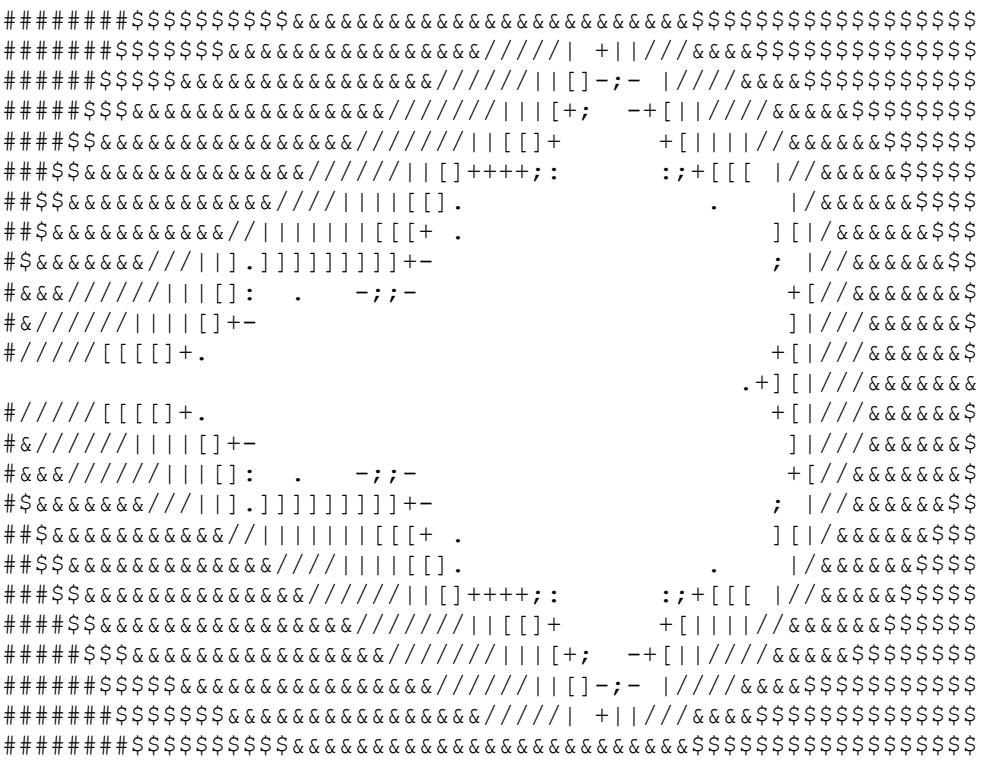
The output consists of the required number of lines and columns to plot the specified region of the Mandelbrot set. All lines in output should be terminated with a newline.

Different test cases should be separated by a single blank line.

Sample Input

```
2
"#$&/|[]+;:-." -1.2 1.2 0.1 -2 1 0.05
"1234567890AB" -1.2 -0.8 0.02 -0.5 0.5 0.02
```

Sample Output



E 914 - Jumping Champion

`Time limit: 3.000 seconds`

Professor Ma.L. (Math Lover) loves everything related to prime numbers. Remember that *a prime is a positive number bigger than one and only divisible by 1 and itself*. He is now working on a property of a set of primes called the jumping champion. An integer *N* is called the "**jumping champion**" if it is the most frequently occurring difference between consecutive primes.

For example, consider the consecutive primes `2 3 5 7 11`. The differences between primes are `1 2 2 4`. Therefore, for this set of primes, the jumping champion is exactly 2 (occurring two times).

He would really like to know for any set of primes what is their corresponding jumping champion. Could you help him?

Problem

Your task is to write a program that, given a lower and an upper bound, calculates the jumping champion of all the primes numbers that are in the defined limits (the upper and lower bound are considered themselves to be inside the limit).

Input

The first line of input contains an integer **T** which is the number of test cases that follow.

Each test case is given on a line with two integer numbers **L** and **U** ($0 \leq L \leq U \leq 1000000$), separated by a single space, which represent the lower and upper limits (respectively) to consider.

Output

The output consists of **T** lines, one for each case.

The i^{th} line contains:

- `"The jumping champion is NUM"` - if the jumping champion for the i^{th} case can be found and it is NUM ;
- `"No jumping champion"` - if no single jumping champion can be found (if there are less than two primes in the interval or if there is more than one difference occurring a maximum number of times)

Sample Input

```
3
2 11
2 5
30 50
```

Sample Output

```
The jumping champion is 2
No jumping champion
The jumping champion is 4
```

F 10129 - Play on Words

`Time limit: 3.000 seconds`

Some of the secret doors contain a very interesting word puzzle. The team of archaeologists has to solve it to open that doors. Because there is no other way to open the doors, the puzzle is very important for us.

There is a large number of magnetic plates on every door. Every plate has one word written on it. The plates must be arranged into a sequence in such a way that every word begins with the same letter as the previous word ends. For example, the word ``acm'' can be followed by the word ``motorola''. Your task is to write a computer program that will read the list of words and determine whether it is possible to arrange all of the plates in a sequence (according to the given rule) and consequently to open the door.

Input Specification

The input consists of *T* test cases. The number of them (*T*) is given on the first line of the input file. Each test case begins with a line containing a single integer number *N*that indicates the number of plates (*1 <= N <= 100000*). Then exactly *N*lines follow, each containing a single word. Each word contains at least two and at most 1000 lowercase characters, that means only letters '`a`' through '`z`' will appear in the word. The same word may appear several times in the list.

Output Specification

Your program has to determine whether it is possible to arrange all the plates in a sequence such that the first letter of each word is equal to the last letter of the previous word. All the plates from the list must be used, each exactly once. The words mentioned several times must be used that number of times.

If there exists such an ordering of plates, your program should print the sentence "`Ordering is possible.`". Otherwise, output the sentence "`The door cannot be opened.`".

Sample Input

```
3
2
acm
ibm
3
acm
malform
mouse
2
ok
ok
```

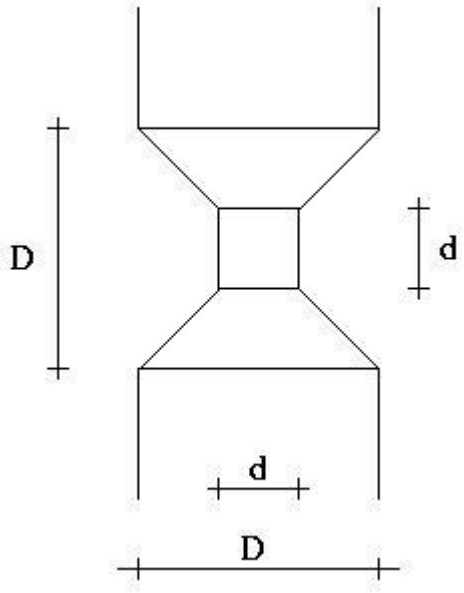
Output for the Sample Input

```
The door cannot be opened.
Ordering is possible.
```

G 10297 - Beavergnaw

When chomping a tree the beaver cuts a very specific shape out of the tree trunk. What is left in the tree trunk looks like two frustums of a cone joined by a cylinder with the diameter the same as its height. A very curious beaver tries not to demolish a tree but rather sort out what should be the diameter of the cylinder joining the frustums such that he chomped out certain amount of wood. You are to help him to do the calculations.

We will consider an idealized beaver chomping an idealized tree. Let us assume that the tree trunk is a cylinder of diameter **D** and that the beaver chomps on a segment of the trunk also of height **D**. What should be the diameter **d** of the inner cylinder such that the beaver chmped out **V** cubic units of wood?



Input contains multiple cases each presented on a separate line. Each line contains two integer numbers **D** and **V** separated by whitespace. **D** is the linear units and **V** is in cubic units. **V** will not exceed the maximum volume of wood that the beaver can chomp. A line with **D**=0 and **V**=0 follows the last case.

For each case, one line of output should be produced containing one number rounded to three fractional digits giving the value of **d** measured in linear units.

Sample input

```
10 250
20 2500
25 7000
50 50000
0 0
```

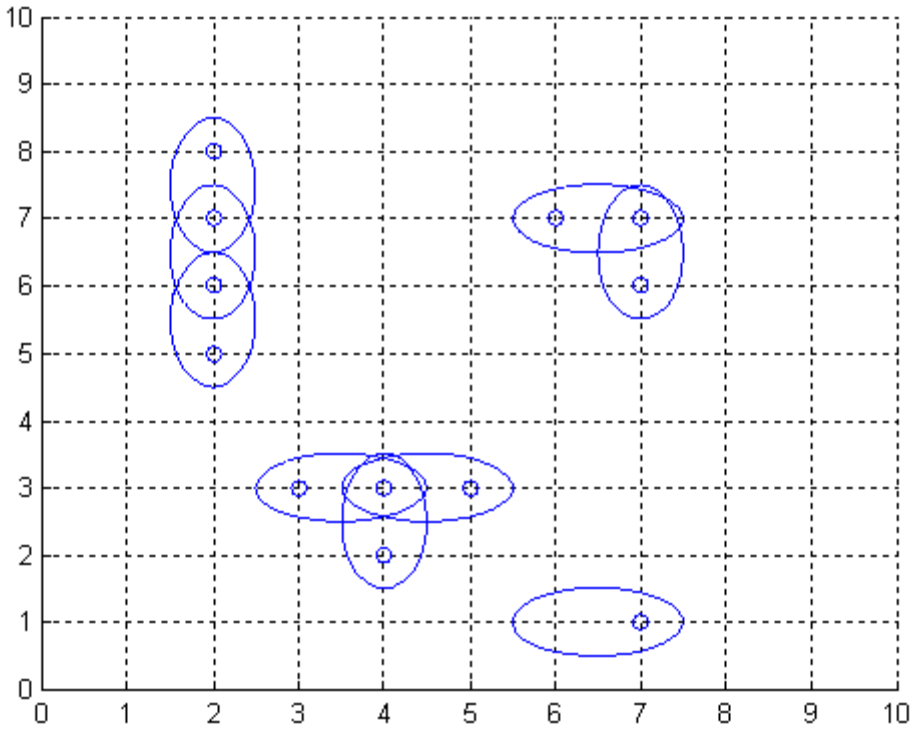
Output for sample input

```
8.054
14.775
13.115
30.901
```

H 10349 - Antenna Placement

`Time limit: 3.000 seconds`

The Global Aerial Research Centre has been allotted the task of building the fifth generation of mobile phone nets in Sweden. The most striking reason why they got the job, is their discovery of a new, highly noise resistant, antenna. It is called 4DAir, and comes in four types. Each type can only transmit and receive signals in a direction aligned with a (slightly skewed) latitudinal and longitudinal grid, because of the interacting electromagnetic field of the earth. The four types correspond to antennas operating in the directions north, west, south, and east, respectively. Below is an example picture of places of interest, depicted by twelve small rings, and nine 4DAir antennas depicted by ellipses covering them.



Obviously, it is desirable to use as few antennas as possible, but still provide coverage for each place of interest. We model the problem as follows: Let *A* be a rectangular matrix describing the surface of Sweden, where an entry of *A* either is a point of interest, which must be covered by at least one antenna, or empty space. Antennas can only be positioned at an entry in *A*. When an antenna is placed at row *r* and column *c*, this entry is considered covered, but also one of the neighbouring entries *(c+1, r)*, *(c, r+1)*, *(c-1, r)*, or *(c, r-1)*, is covered depending on the type chosen for this particular antenna. What is the least number of antennas for which there exists a placement in *A* such that all points of interest are covered?

Input

On the first row of input is a single positive integer *n*, specifying the number of scenarios that follow. Each scenario begins with a row containing two positive integers *h* and *w*, with $1<h<40$ and $0<w<10$. Thereafter is a matrix presented, describing the points of interest in Sweden in the form of *h* lines, each containing *w* characters from the set [‘*’,’o’]. A ‘*’-character symbolises a point of interest, whereas a ‘o’-character represents open space.

Output

For each scenario, output the minimum number of antennas necessary to cover all ‘*’-entries in the scenario’s matrix, on a row of its own.

Sample Input

```
2
7 9
ooo**oooo
**oo*ooo*
o*oo**o**
ooooooooo
*******oo
o*o*oo*oo
*******oo
10 1
*
*
*
o
*
*
*
*
*
*
```

Sample Output

```
17
5
```

I 11500 - Vampires

Felipinho is thrilled with his new RPG game, about wars between clans of vampires. In this game he plays a vampire that repeatedly comes into combat against vampires from other clans. Such battles are won or lost based on some characteristics of the opponents, with the help of a standard six-faced dice.

For simplicity, we will consider only the fight between two vampires, Vampire 1 and Vampire 2. Each vampire has a vital energy (denoted respectively by EV_1 and EV_2). Besides, an attack force AT and a damage capacity D are also given.

The combat is fought in turns, in the following way. At each turn, the dice is rolled; if the result value is less than or equal to AT, Vampire 1 wins the turn, otherwise Vampire 2 wins. The winner then sucks the value D from the loser's vital energy. That is, D points are subtracted from the loser's vital energy and added to the winner's vital energy. The combat continues until one of the fighters has EV less than or equal to zero.

For example, suppose $EV_1=7$, $EV_2=5$, $AT=2$ and $D=4$. The dice is rolled and the result value is 3. Then, Vampire 2 wins the turn, and therefore 4 points are subtracted from EV_1 and added to EV_2. The new values for the vital energies would be $EV_1=3$ and $EV_2=9$. Notice that, if in the next turn Vampire 2 wins again, the combat ends.

The values of AT and D are constant throughout the combat; only EV_1 and EV_2 vary. Despite loving the game, Felipinho thinks that the combats are too long, and suspects that, given the initial values of EV_1, EV_2, AT and D, it is possible to determine the probability of one of the players winning the combat, and that could help shorten the combat time.

Felipinho has asked your help to verify his suspicion.

The Input

The input contains several test cases. Each test case is given in one single line, containing four integers EV_1, EV_2, AT and D separated by spaces ($1 \le EV_1, EV_2 \le 10$, $1 \le AT \le 5$ and $1 \le D \le 10$).

The end of input is indicated by one line containing only four zeros, separated by spaces.

The Output

For each test case in the input, your program must print a single line. The line must contain a real number representing, in terms of percentages, the probability that Vampire 1 wins the combat. The result must be printed as a real number with exactly one decimal figure.

Sample Input

```
1 1 3 1
1 2 1 1
8 5 3 1
7 5 2 4
0 0 0 0
```

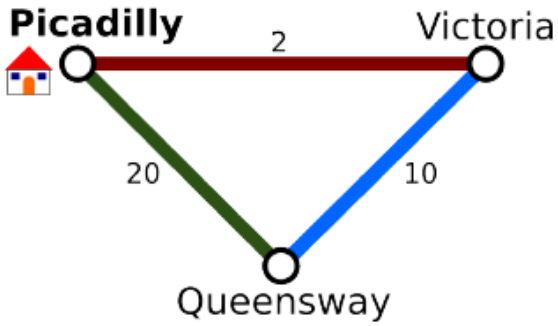
Sample Output

```
50.0
3.2
61.5
20.0
```

J 11710 - Expensive subway

Time limit: 4.000 seconds

Peter lives in *Expensive City*, one of the most expensive cities in the world. Peter has not got enough money to buy a car, and the buses in Expensive City are pretty bad, so he uses the subway to go to work. Up to now, the subway was very cheap: you could travel anywhere with just one $2 ticket. Last month, the managers decided that it was too cheap so they invented the EFS (Expensive Fare System). With this system, users can only buy monthly tickets between adjacent stations, which allows them to move between these stations any number of times. The price of the monthly ticket varies between stations, so the decision of which tickets to buy must be taken carefully.



With the previous subway plan, the cheapest way to travel from Picadilly to Victoria and Queensway was to buy the monthly ticket Picadilly-Victoria and Queensway-Victoria, for a total cost of $12.

Peter is a salesperson, so he needs to be able to travel to any part of the city. He wants to spend as little money as possible, and here is where you come into the picture. He has hired you to write a program that, given the list of stations, the fares of the monthly tickets between pairs of stations and the station nearest Peter's home, returns the minimum amount of money Peter has to spend in order to travel to any other station. This program also has to return value if it is not possible to go from Peter's home station to all the rest, because in this case Peter will begin to consider using buses...

Input

The input consists of several test cases. A test case begins with a line containing two integers: $1 \leq s \leq 400$ (the number of stations) and $0 \leq c \leq 79800$ (the number of connections) separated by a single space. This is followed by s lines, each one containing the name of a subway station. These names will be strings of characters (uppercase or lowercase) without punctuation marks or whitespace characters, and with a maximum length of 10 characters. After the names of the stations there will be c lines showing the connections between stations. A connection allows people to travel from one station to the other in both directions. Each connection is represented as two strings indicating the names of the stations and a positive integer indicating the cost of the monthly ticket, all of which are separated by single spaces. All names of stations appearing in the connections will have previously appeared in the list of

s stations. The connections will all be different, and there will not be any connection from a station to itself. The test case will end with a line containing the name of the station from which Peter needs to travel to all the other stations.

The input finishes with the phantom test case `0 0', which must not be processed.

Output

For every test case, the output will be a line containing an integer, the minimum monthly price that Peter has pay to travel from the given station to all the others, or Impossible if it is not possible to travel to all the stations.

Sample Input

```
3 3
Picadilly
Victoria
Queensway
Picadilly Victoria 2
Queensway Victoria 10
Queensway Picadilly 20
Picadilly
4 2
Picadilly
Victoria
Queensway
Temple
Picadilly Victoria 2
Temple Queensway 100
Temple
0 0
```

Sample Output

```
12
Impossible
```

K 146 - ID Codes

Time limit: 3.000 seconds

It is 2084 and the year of Big Brother has finally arrived, albeit a century late. In order to exercise greater control over its citizens and thereby to counter a chronic breakdown in law and order, the Government decides on a radical measure--all citizens are to have a tiny microcomputer surgically implanted in their left wrists. This computer will contains all sorts of personal information as well as a transmitter which will allow people's movements to be logged and monitored by a central computer. (A desirable side effect of this process is that it will shorten the dole queue for plastic surgeons.)

An essential component of each computer will be a unique identification code, consisting of up to 50 characters drawn from the 26 lower case letters. The set of characters for any given code is chosen somewhat haphazardly. The complicated way in which the code is imprinted into the chip makes it much easier for the manufacturer to produce codes which are rearrangements of other codes than to produce new codes with a different selection of letters. Thus, once a set of letters has been chosen all possible codes derivable from it are used before changing the set.

For example, suppose it is decided that a code will contain exactly 3 occurrences of `a', 2 of `b' and 1 of `c', then three of the allowable 60 codes under these conditions are:

```
abaabc
abaacb
ababac
```

These three codes are listed from top to bottom in alphabetic order. Among all codes generated with this set of characters, these codes appear consecutively in this order.

Write a program to assist in the issuing of these identification codes. Your program will accept a sequence of no more than 50 lower case letters (which may contain repeated characters) and print the successor code if one exists or the message `No Successor' if the given code is the last in the sequence for that set of characters.

Input and Output

Input will consist of a series of lines each containing a string representing a code. The entire file will be terminated by a line consisting of a single `#`.

Output will consist of one line for each code read containing the successor code or the words `No Successor'.

Sample input

```
abaacb
cbbaa
#
```

Sample output

```
ababac
No Successor
```