Problem 1. Signal processing

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 MB

A space research group spent decades searching for extraterrestrial intelligence in an attempt to contact it. To find intelligence of any sort, the group tried direct observation of space objects through telescopes, scanning radio frequencies, transmitting all sorts of radio messages, sending unmanned spacecrafts loaded with probes, etc. However, every time the search came to nothing — the search did not yield any results which would point to the presence of extraterrestrial intelligence with certainty. However, some members of the group found some of the signals received by their research equipment suspicious, and they decided to analyze them thoroughly. The signals were recorded as sequences of 0's and 1's. A problem occurred: if the signals had been sent by a civilization from outer space, there was no way to know what coding method they had used and how they were supposed to interpret these sequences.

They decided to try the easiest ways first and recode the signals from binary to hexadecimal. The result of such processing for a signal, which is a sequence of the length L, must contain $\lceil \frac{L}{4} \rceil$ hex digits. Digits from zero to nine are represented by the digits '`0`'-'`9`', and digits from ten to fifteen — by the capital Latin letters '`A`', '`B`', '`C`', '`D`', '`E`' and '`F`', respectively. Let's consider that least significant digits are located to the right both in the unprocessed sequence and in the processing result.

Input

The first line contains an integer: N — the number of signals which were found suspicious ($1 \le N \le 100$).

Each of the following N lines contains a single signal represented as a line of zeroes and ones. Each of these lines is non-empty and is no longer than 250.

Output

For each signal, print the result of its processing in a separate line.

Example

standard input	standard output
3 100 00001 0111010010101101100000001000	4 01 3A56C08

Problem 2. Innovative keyboard

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 MB

A multikeyboard is a conventional keyboard with multiple outputs, each of which can be connected to a separate PC. The creators of the device state that the use of the multikeyboads makes lives of its users much easier.

A while ago, a multikeyboard was installed into the server room of a very serious and seriously secret company. It was attached to N computers. Consequently, each push of the button prints the same symbol on all machines. But a problem occured: the multikeyboard is made from sub-par materials, and if you unplug a cable from a computer, this ruins the socket and you cannot reconnect it again.

The workday is in full swing. You must type a specific string on each of the computers. You are only allowed to use the multikeyboard: you can only press buttons with latin alphabet symbols and disconnect connecting cables from the computers. In the end of your typing session, each computer must have precisely the desired string printed on its screen.

Determine the minimum number of button strokes on the multikeyboard to type all defined strings on the computers.

Input

The first line of the input files contains an integer T — the number of tests ($1 \le T \le 100$). It is followed by T blocks.

The first line of a block contains an integer N — the number of computers to which the multikeyboard is attached ($1 \le N \le 10^5$). Each ith of the following lines defines a nonempty string s_i, which must be printed on the ith computer.

The total length of all strings in all tests is no greater than 10^5. Strings can only contain small Latin letters.

Output

In the output file, print T lines. The ith line must contain an answer to the ith test case. If it is impossible to print the defined strings, the answer must be the word `Impossible`. Otherwise print a single integer — the minimum number of button strokes necessary.

Example

standard input	standard output
2 3 aba abacaba abca 2 hello hell	Impossible 5

Problem 3. Rocket

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2.5 seconds
	5 seconds (for Java)
Memory limit:	512 MB

Dima is playing a new game: Kerbal Space Program. In the game you build amazing large spaceships and launch them into the outer space. Dima likes to break records. He wants to make his rockets as light as possible.

A rocket consists of a sequence of N stages and a payload at the top. The payload has the given mass of M_p. Each stage is either a single or a set of identical solid-fuel engines installed parallel to each other. Each engine has the following parameters: mass — M, thrust — T, fuel-specific impulse — I_{sp}. If K identical engines are installed parallel to each other in a single stage, they have a mass of $K \cdot M$, a fuel-specific impulse of I_{sp} and work synchronously, generating a thrust of $K \cdot T$.

Consider all engines ideal, i.e. the mass of each engine is negligibly small compared to the mass of its fuel. In other words, an engine with a mass of M has a fuel load of M and the engine itself has zero mass.

Here's how the launch happens. At any moment of time during acceleration, only the engines of one stage work: the first stage of those which have not yet been jettisoned. When engines of a stage work out all their fuel, the stage is jettisoned, and engines of the next stage start. First, the first stage is "fired", i.e. all its engines begin to work. When the engines of the first stage run out of fuel, it is jettisoned, and the second stage fires. When all fuel has been burnt out in the second stage, the stage is jettisoned, and the third stage switches on. This goes on until all stages are jettisoned.

The increment of the rocket velocity while the fuel is being burned out in a single stage is defined using the Tsiolkovsky formula:

$$\Delta V = I_{sp} \ln \frac{M_{start}}{M_{end}}$$

where: ΔV — incremental velocity of the rocket, I_{sp} — fuel-specific impulse for the engines of the working stage, M_{start} — initial mass of the whole rocket (at the moment of stage ignition), M_{end} — the final mass of the whole rocket (at the moment the stage is jettisoned), ln — natural logarithm function, i.e. of logarithm to the base e. The final speed of the rocket equals the sum of all these increments for all stages of the rocket (initially, the speed of the rocket is zero).

The rocket must accelerate the payload to a speed equal or greater than orbital velocity V_*. Moreover, the relation of the thrust of the working engines to the current mass of the rocket must not drop lower than the constant $c = 8/5$. Since stages are fired successively, it is enough to check that the condition is satisfied at the moment of ignition of each stage: when engines work, their fuel burns out, the thrust remains constant, and their mass gradually decreases. Dima is not interested in heavy rockets with a mass over M_{max} — this is too bad to even consider as a viable option.

The game store hase T types of engines in stock. Dima is pretty rich and can buy any number of any engines. Find out the minimum possible takeoff mass of a rocket which can take the payload into the outer space - subject to all conditions.

Below is an example of a three-stage rocket. Every ith stage has the parameters $M^{(i)}$, $T^{(i)}$, $I_{sp}^{(i)}$. The total velocity gain for the whole acceleration time equals:

$$\Delta V = I_{sp}^{(1)} \ln \frac{M^{(1)} + M^{(2)} + M^{(3)} + M_p}{M^{(2)} + M^{(3)} + M_p} + I_{sp}^{(2)} \ln \frac{M^{(2)} + M^{(3)} + M_p}{M^{(3)} + M_p} + I_{sp}^{(3)} \ln \frac{M^{(3)} + M_p}{M_p}$$

Mind the lower limit on thrust:

$$\frac{T^{(1)}}{M^{(1)} + M^{(2)} + M^{(3)} + M_p}, \frac{T^{(2)}}{M^{(2)} + M^{(3)} + M_p}, \frac{T^{(3)}}{M^{(3)} + M_p} \geq \frac{8}{5} = c$$

Input

The first line contains two integers: T — the number of types of engines in the store ($1 \le T \le 3 \cdot 10^2$), M_{max} — maximum allowed mass of the whole rocket ($1 \le M_{max} \le 10^5$). The second line contains two more integers: M_p — the mass of the rocket's payload ($1 \le M_p \le M_{max}$), V_* — orbital veolcity ($1 \le V_* \le 10^9$).

The remaining T lines describe types of engines in the store, one per line. For each engine, three integers are given: M — mass of a single engine ($1 \le M \le 10^5$), T — thrust of a single engine ($1 \le T \le 2 \cdot 10^5$), I_{sp} — fuel-specific impulse ($1 \le I_{sp} \le 10^5$).

Output

If reaching the orbital velocity with the initial mass of the rocket M_{max} or less, print a single number -1.

Otherwise the first line of the output file must contain two integers: M_{start} — the initial mass of the rocket ($M_p \le M_{start} \le M_{max}$), N — the number of stages in the rocket ($N \ge 1$). Each of the following N lines must describe a single stage of the rocket. Stages are described in the order of firing, from the first to the last. Each stage description must contain two integers: j — the number of engine type in the stage ($1 \le j \le T$), k — the number of engines installed in the stage parallel to each other ($k \ge 1$). Types are numbered in the order of their description.

The jury guarantees that for all tests a change of orbital velocity by 0.001% **does not** affect the answer to the problem (i.e. the minimum launch mass of the rocket).

Example

standard input	standard output

2 100000
3 408
5 13 200
8 20 200
	24 2
	2 2
	1 1

Note

Note that the problem conditions ignore the Earth gravity in the formula of speed gain. Moreover, we assume that the rocket is accelerated along a straight line path which is not exactly true.

Base of natural logarithm: $e \approx 2.718281828459045$.

Problem 4. Universities

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 MB

Far, far away, once upon a time there was a state which introduced the Uniform state exam for its school graduates. Depending on their score, graduates can enter all sorts of higher education facilities - let's call them colleges. Every graduate can apply to several colleges simultaneously, and university officials have no way of knowing who will be their freshers - until the very last moment.

In the beginning of every new entrance season, there are N school graduates and M colleges. Let's assume that each college can accept an unlimited number of students, so there are no cut-off scores.

During the entrance season, every graduate can apply to any number of colleges, and with each college, he or she can do it on any day during the entrance season. Thanks to a poll in social networks we know exactly for each graduate all the colleges he or she will apply, and on what days. Applications are accepted early in the morning.

Every college must organize an entrance procedure in one of the days of the entrance season. When the entrance procedure happens, all free graduates who have already applied to this college are accepted. A graduate is considered free if he or she **has not** yet been accepted to any other college on any of the previous days. The entrance procedure happens in the afternoon, so a graduate can enter even if he or she applies on the day of the procedure. The accepted graduates can **not** enter any other college afterwards, and the accepting college can **not** accept any more graduates afterwards.

If several colleges where the graduate has applied, carry out the entrance procedure simultaneously, there is an uncertainty about which college the graduate will end up in. In this case, the graduate is free to choose from any of these colleges. In reality, the graduate always chooses the highest-ranking college.

Representatives of the Noname State University (NSU) are cooking up a strategy for the upcoming entrance period. They have assigned values to all graduates. Their intelligence reports approximate dates of other universities' entrance procedures. Unfortunately, the data is not exact; they only have probability estimates. For every college and every day the probability that the given college performs the entrance procedure on that given day is known. Consider that all colleges select their entrance procedure days completely independently from each other.

Determine the day for the entrance procedure in the NSU, such that the sum of values of all accepted graduates is maximum on average. Remember that the NSU is the highest-ranking college among all.

Input

The first line contains four integers: N — the number of graduates ($1 \le N \le 3\,000$), M — the number of colleges ($1 \le M \le 50$), K — the number of entrance applications ($1 \le K \le 30\,000$), T — the length of the entrance period in days ($1 \le T \le 10\,000$). Graduates and colleges are numbered successively, starting from one. The NSU has the number M.

The second line contains N integers V_i — the values of all graduates ($1 \le V_i \le 10^3$).

Every lth of the following K lines describes a single entry application. Each description contains three integers: P_l — the number of the applying graduate ($1 \le P_l \le N$), Q_l — the number of the college where he or she is applying ($1 \le Q_l \le M$), R_l — the number of the day when this happened ($1 \le R_l \le T$). It is guaranteed that every graduate applies to each college no more than once during the entry period. A graduate can apply to several colleges in a single day.

The remaining T lines describe the intelligence data on other colleges, with $M - 1$ real numbers in each line. In the tth of these lines, the jth number P_{tj} defines the probability of the jth college performing the entry procedure on the tth day. It is guaranteed that the sum of defined probabilities for each college is strictly one. The real numbers $0 \le P_{tj} \le 1$ are defined as a decimal fraction with no more than four digits after the decimal point.

Output

The output file must contain two numbers: X — the maximum average value of the sum of values of graduates accepted to the NSU (a real number) and t^* — the number of the day to perform the entry procedure for this to happen (an integer, $1 \le t^* \le T$).

The relative error of your answer X should be equal to or less than 10^{-8}.

Example

standard input	standard output

4 4 7 6
4 666 5 3
1 1 2
3 1 5
4 4 6
1 4 4
4 2 4
3 4 3
3 2 2
0.2 0 0.01
0.2 0.7 0.17
0.2 0 0.03
0.2 0 0.25
0.1 0.3 0.09
0.1 0 0.45
	3.9 4

Example explanation

Note that the exceptionally valuable second graduate left to study abroad, and that no one wants to study in the third college.

Problem 5. Sheet music

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

Storing sheet music is one of the routine worries of every musician. Luckily, nowadays you can buy a file folder with transparent pockets in any store. This way you can store your sheets safely and play them regularly. But rearranging sheets in these folders is notoriously time-consuming.

Given: a folder with pockets numbered from 1 to M. Each pocket can be either empty or contain a single sheet from a musical composition. A single pocket **can not** contain contain several sheets, because this way they are easy to lose.

The orchestra plays a total of N compositions, which are numbered from 1 to N for convenience. Each composition has one or several sheets of music. The musician stores all sheets for all compositions in a single folder, with exactly one instance of each sheet. To make sheets easy to manage, each is marked with the number of the composition and the number of the sheet in the composition. For instance, "7: Passacaglia, 2/3" on the sheet means that it is the second sheet of the total of three sheets of the composition named "Passacaglia" (being composition number seven).

If a composition takes up several sheets, the musician has to "flip pages" when playing. There is no time to think and search during playing, so any musician stores the sheets from a single composition strictly in order (in adjacent pockets). For example, if the sheets of "Passacaglia" take up three pages, and the second sheet is stored in the sixth pocket, that means that the first sheet is stored in the fifth pocket, and the third sheet — in the seventh pocket. If that rule is broken in the folder, an epic fail may happen during the next concert!

Number	Compositions are sorted	Initial state
(1)	—	—
(2)	1: March 1/2	1: March 1/2
(3)	1: March 2/2	1: March 2/2
(4)	—	—
(5)	2: Passacaglia 1/3	—
(6)	2: Passacaglia 2/3	4: Nocturne 1/1
(7)	2: Passacaglia 3/3	—
(8)	3: Waltz 1/2	2: Passacaglia 1/3
(9)	3: Waltz 2/2	2: Passacaglia 2/3
(10)	—	2: Passacaglia 3/3
(11)	4: Nocturne 1/1	3: Waltz 1/2
(12)	—	3: Waltz 2/2

To find a specific composition in the folder, musicians use an algorithm of binary or interpolated search (even though usually they do not know these terms). Naturally, these approaches can only be used when sheets in the folder are arranged in the ascending order by composition number. However, sometimes compositions have to be arranged in a different order, for example, before major concerts. Naturally, in such cases the rule that the sheets of a single composition must go in order is observed as well. You are to find the optimal way of rearranging sheets in the folder in such a way that in the end, they are arranged by composition number in ascending order.

For instance, let's examine two states of a folder with twelve pockets (see the table above). On the right, the initial state of the sheets is given. The sheets are not sorted by composition. On the left, the same sheets are arranged so that the number of the composition is ascending. Other ways of arranging sheets are possible, such that the compositions are sorted properly.

To avoid drowning in heaps of paper, the musician rearranges sheets by performing chains of repositions. A reposition chain (i_1, i_2, i_3, …, i_k) with $k \geq 2$ is performed in the following way:

1. Extract sheet from the pocket number i_1.
2. Extract sheet from the pocket number i_2 and place the sheet from the pocket i_1 there.
3. Extract sheet from the pocket number i_3 and place the sheet from the pocket i_2 there.

...

k. Place the sheet got in the previous step into an empty pocket i_k.

A pocket can be used in a reposition chain several times.

Every time the musician moves a sheet, there is a risk it will tear the pocket. It is necessary to find a way to minimize the number of extractions and insertions when rearranging sheets.

For instance, in the example above you can get an ordered folder from the initial state with the following sequence of reposition chains:

- (6 9 6)
- (8 5)
- (10 7)
- (9 11 8)
- (12 9)

The total number of insertion operations in this case is 7, though you can achieve the same results with only 6 insertions.

Input

The first line contains two integers: M — the number of pockets in the folder ($1 \le M \le 2 \cdot 10^5$), N — the number of compositions ($1 \le N \le 10^5$).

The remaining M lines describe the initial state of the folder, one pocket per line in the numeration order. Every jth of these lines contains two integers describing the sheet in the jth pocket: C_j — the number of the composition ($1 \le C_j \le N$), and P_j — the number of sheet in the composition ($1 \le P_j \le M$). If the pocket is empty, both numbers equal -1.

It is guaranteed that the folder contains sheets for all compositions from 1 to N, and for each jth composition all sheets numbered from 1 to L_j are present, and there is only a single copy of each sheet in the folder. Moreover, for every composition, its sheets are arranged in order within the folder.

Output

The first line of the output file contains two integers: A — the minimum possible number of insertion operations and Q — the number of chains in the optimal solution.

Each of the following Q lines must contain a single reposition chain in the order of their execution. A description of a reposition chain begins from an integer k — the length of the chain ($k \ge 2$), followed by k integers $i_1, i_2, \ldots, i_k$ — the numbers of pockets in the chain in the order of execution of operations ($1 \le i_1, i_2, \ldots, i_k \le M$).

Example

standard input	standard output

12 4
-1 -1
1 1
1 2
-1 -1
-1 -1
4 1
-1 -1
2 1
2 2
2 3
3 1
3 2
	6 4
	4 6 11 8 5
	2 10 7
	2 9 6
	2 12 9

Problem 6. Split the sandwich

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

The children of Volga are facing a difficult quest. Shortly before the inspection of the Great Combinator, Alchen gave them a bread-and-butter sandwich and left to feed the mischievous guest with whatever God would send. The brethren want to split the sandwith with one linear cut, halving both the bread and butter precisely. But the trouble is, them poor souls ain't got no educatin' and this task is too much for them. Let us help them.

The abovementioned sandwich is a vertical cylinder $x^2 + y^2 \leq 1$, transsected by three planes $z = 0$, $z = a_1x + b_1y + c_1$ and $z = a_2x + b_2y + c_2$. A layer of bread lies between the first and second plane; and a layer of butter lies between the second and third plane.

Calculate three real coefficients of the vertical section equation: $ax + by = c$.

Input

The input file contains six real numbers $a_1, b_1, c_1, a_2, b_2, c_2$, defining the coefficients in plane equations which limit the bread and butter layers. All numbers have an absolute value of no more than 100 and are given with no more than six digits after the decimal point.

It is guaranteed that everywhere within the unit sandwich cylinder the second plane lies strictily above the first, and the third lies strictly above the second, i.e. $0 < a_1x + b_1y + c_1 < a_2x + b_2y + c_2$ for all $x^2 + y^2 \leq 1$.

Output

The output file must contain three real numbers a, b and c, defining the required position of the section (which is a vertical plane $ax + by = c$), which splits both the bread and the butter in half. The inequality $\frac{1}{10} \leq a^2 + b^2 \leq 10$ must hold true.

The printed plane will be accepted if the volume ratio of the parts of each substance (bread and butter) will differ from one by no more than 10^{-8}.

If such a section is impossible, print a single number -1.

Examples

standard input
`1 0 3 -1 0 20`
standard output
`0.000000000000 1.000000000000 0.000000000000`

standard input
`10 10 50 -10 2 100`
standard output
`-0.514495755428 0.857492925713 0.022850559544`

Problem 7. Collisions search

Input file:	*standard input*
Output file:	*standard output*
Time limit:	6 seconds
Memory limit:	256 MB

Stephen is taking part in the development of a game mod with the idTech4 engine. The game requires finding collisions in the trajectories of instantly moving objects (such as bullets), but methods of finding such collisions in the game are laggy. Help Stephen implement an efficient algorithm for finding collisions.

There are G geometric models numbered with numbers from 1 to G. Each model consists of a set of triangles in a three-dimensional space. It is guaranteed that the triangles are not degenerate. There are no other guarantees, e.g. of topological correctness, closedness, absence of self-intersections, etc. The size of all geometric models is roughly unit, i.e. for each of these models the tight bounding box (the smallest box parallel to the axes and containing the model) will have the longest side of the length $\frac{1}{2} \le L \le 1$.

A game region with the coordinates $0 \le x \le M$, $0 \le y \le N$, $0 \le z \le K$ is specified in the three-dimensional space. It contains $S = (M \cdot N \cdot K)$ game objects numbered from 1 to S. Each object is defined by the number of its geometric model and the coordinates of its center. The geometric model of the object is shifted in such manner that the origin of the model's coordinate system gets precisely into the object center. In other words, the object consists precisely of the triangles which are present in its geometric model, but all vertices of these triangles are shifted by the vector v, which is pointing from the global origin to the center of the object. Game objects may intersect.

Process Q queries for finding collisions. Each query contains a ray set by the coordinates of its initial point and its unit direction vector. Intersect the ray with all the triangles of all the game objects and locate the intersection point nearest to the initial point of the ray (among all the intersection points).

All jury tests except the sample one were generated in the following way:

1. The same set of 10 geometric models pictured on the next page is used. All models correspond to real game objects. They can be downloaded during the contest.
2. Object centers are generated pseudo-randomly with uniform distribution over the game region.
3. Initial points of the rays are generated pseudo-randomly with uniform distribution over the game region.
4. Direction vectors of the rays are generated pseudo-randomly with uniform distribution over the unit sphere.

Input

The first line contains an integer G — the number of geometric models ($1 \le G \le 10$). Next, G geometric models are described in their numeration order.

The description of each model begins with two integers: V — the number of points defined ($3 \le V \le 700$), T — the number of triangles in the model ($1 \le T \le 1\,000$). The following V lines contain the coordinates of V points. Points are numbered in the order of description starting from one. The ith of lines contains the coordinates of the ith point as real numbers P_x^i, P_y^i and P_z^i ($0 \le P_x^i, P_y^i, P_z^i \le 1$). The following T lines describe the model triangles. Every jth of these lines contains three integers a_j, b_j and c_j, defining the indices of three points, which are the vertices of the jth triangle ($1 \le a_j < b_j < c_j \le V$).

The following line of the input data defines three integers M, N and K, which define the size of the game region, and the number of game objects. ($1 \le M, N, K \le 50$). The following ($M \cdot N \cdot K$) lines describe game objects as numbered, one per line. Each of these lines contains an integer q_l — the number of the geometric model of the object ($1 \le q_l \le G$), and three real numbers C_x^l, C_y^l, C_z^l — the coordinates of the object center ($0 \le C_x^l \le M$, $0 \le C_y^l \le N$, $0 \le C_z^l \le K$).

The following line contains an integer Q — the number of queries to be processed ($1 \le Q \le 10\,000$). Each of the following Q lines describes a single query and contains six real numbers $O_x^q, O_y^q, O_z^q, D_x^q, D_y^q, D_z^q$. The first three numbers are the coordinates of the ray origin ($0 \le O_x^q \le M$, $0 \le O_y^q \le N$, $0 \le O_z^q \le K$), the remaining three numbers are the components of the unit direction vector ($-1 \le D_x^q, D_y^q, D_z^q \le 1$, $(D_x^q)^2 + (D_y^q)^2 + (D_z^q)^2 = 1$).

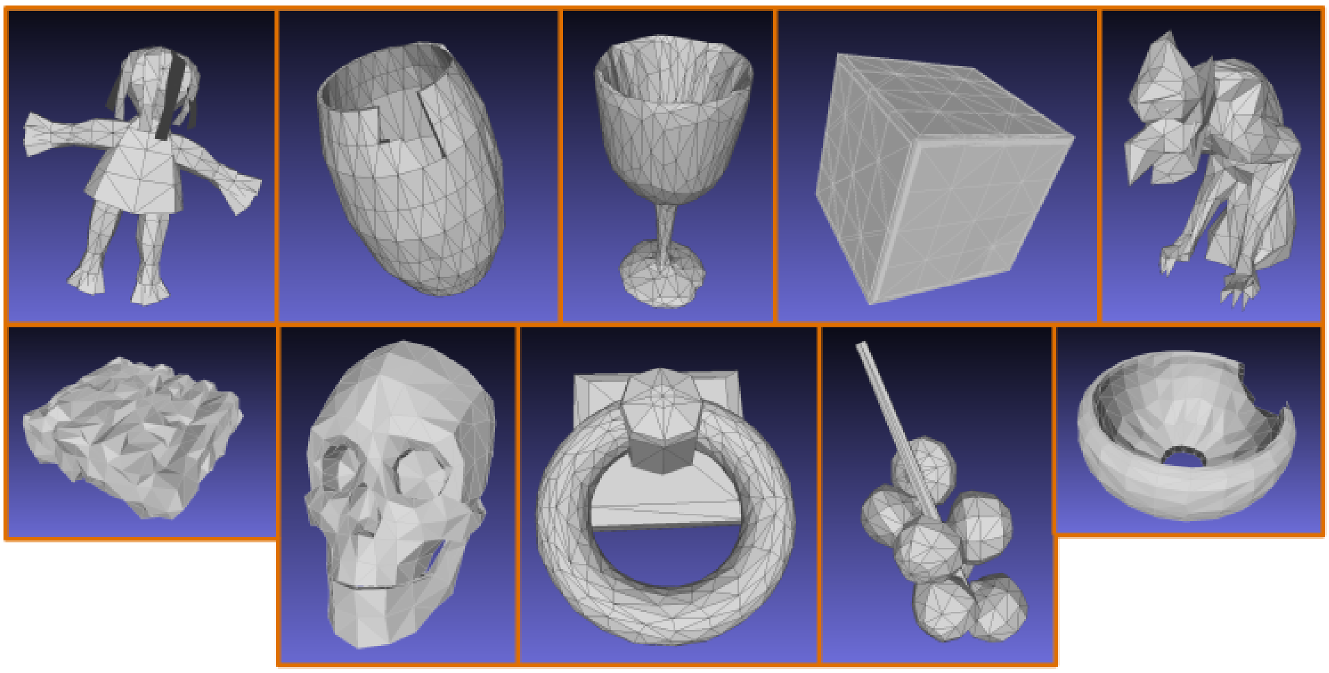
All real numbers are given with no more than 8 digits after the decimal point.

Output

Print Q lines with answers to queries in the order of their description in the input data. Each answer should contain the coordinates of the sought-for intersection point I_x^q, I_y^q and I_z^q (printed as real numbers). Note: the intersection point may be located outside the game region. If the ray does not intersect with any objects, print three numbers -1.

A query answer is considered correct if the absolute of relative error of the intersection point coordinates is no greater than 10^{-5}. A full answer to the test is accepted if the number of incorrect answers to queries is no greater than one thousandth of the number of queries (rounded upwards).

Below are images of the models used in the jury tests. You can download an archive with these models together with samples using Samples ZIP tab — file `models.zip` in the root of the samples archive The file `all.txt` contains descriptions of all models strictly according to the format of the input data (up to the numbers M, N, K exclusive).



Example

standard input

```
2
8 12
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
1 2 3
2 3 4
5 6 7
6 7 8
1 2 5
2 5 6
3 4 7
4 7 8
1 3 5
3 5 7
2 4 6
4 6 8
3 1
0.5 0 0
0 0.5 0
0 0 0.5
1 2 3
3 1 1
2 0 0 0.5
1 0.5 0.5 0.00
2 1 0 0.5
5
0 0 0.5 0.66666667 0.33333333 0.66666667
0 0.1 0.6 1 0 0
1 0.1 0.6 1 0 0
1.2 1 0.7 0 -1 0
0 0 0 -0.48 -0.6 -0.64
```

standard output

```
0.2 0.1 0.70
0.3 0.1 0.60
1.3 0.1 0.60
1.2 0.5 0.70
-1 -1 -1
```

Problem 8. Chair quest

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 MB

Fleeing from the earthquake, the concessionaires abandoned the chairs in a scattered mess. Now they must find their hard-earned belongings.

The first thing they know is the number of chairs n. They also know that coordinates of all chairs are integers and are no greater than 10^6 in absolute value. No two chairs are in the same spot.

All that our hapless treasure hunters can do is to ask a query with randomly defined coordinates (x_q, y_q) and find out the number of chairs with coordinates (x_i, y_i) satisfying the conditions $x_i \le x_q$, $y_i \le y_q$.

Interaction Protocol

This is an interactive task. Instead of the usual file input-output, you will be working with the special program — an interactor. Interaction is done through the standard input-output streams.

First, your program reads a single integer n — the number of chairs ($1 \le n \le 100$). Next, the program can print queries for the number of chairs in the following format (without the quotation marks):

- "? x_q y_q", where x_q and y_q — integers with absolute value not greater than 10^7.

In response to the query a single integer k is provided to the input stream — the number of chairs with coordinates (x_i, y_i) satisfying the conditions $x_i \le x_q$, $y_i \le y_q$.

In the end your program must print an answer to the problem in the following format (without the quotation marks):

- "! x_1 y_1 ... x_n y_n", where x_i and y_i — integers defining the coordinates of the ith chair.

You can print the chairs in any order.

If the coordinates of chairs in the answer are wrong, the solution will receive Wrong Answer. The number of queries on counting chairs must not be greater than $50n$, otherwise the answer will be tagged Wrong Answer.

Make sure you print the line break symbol and flush the output stream buffer (the `flush` command of the language) after each query and when you produce the answer to the problem. Otherwise the solution may receive Timeout.

Example

standard input	standard output
2 2 1 1 1 0 0	? 1 1 ? 1 0 ? 0 1 ? 0 0 ? -1 0 ? 0 -1 ! 0 0 1 1

Problem 9. Vim

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
	6 seconds (for Java)
Memory limit:	256 MB

Commands described in this problem may differ from the commands in the actual Vim editor. Please read the problem attentively.

The Vim editor shows the text line by line. Lines can be of different length. A text can only consist of Latin lowercase letters and spaces.

The cursor is used to navigate the text. In Vim, the cursor is located **not** between symbols, but on a symbol. Its location in the text is defined by a pair of coordinates: the number of line and the position in the line. The location of the cursor always corresponds to a real symbol in the text: the number of the line cannot be lower than 1 or greater than the number of lines in the text, and the position in the line cannot be lower than 1 or greater than the length of the current line.

The location A in the text is considered preceding the location B, if:

- the number of the line A is smaller than the number of the line B;
- both locations have the same line number, but the position number of A is smaller than that of B.

It follows that the location A is considered following the position B, if B precedes A.

A word in the text is a sequence of Latin letters. Words are separated by one or several space symbols and line breaks. The beginning of a word is the first symbol of the word. The end of a word is the last symbol of the word.

Vim has several working modes. One of them is the command mode. In the command mode, you can use navigation keys:

- The key '`l`' moves the cursor one symbol to the right. If the cursor is positioned on the last symbol of the line, it does not move anywhere.
- The key '`h`' moves the cursor one symbol to the left. If the cursor is positioned on the first symbol of the line, it does not move anywhere.
- The key '`w`' moves the cursor to the beginning of the next word in the text: the earliest location corresponding to the beginning of a word following the current location of the cursor. If there is no such beginning of a word, the cursor does not move anywhere.
- The key '`b`' moves the cursor to the beginning of the current word: the latest location corresponding to the beginning of a word preceding the current location of the cursor. If there is no such beginning of the word, the cursor does not move anywhere.
- The key '`$`' moves the cursor to the last position of the current line.
- The key '`0`' (zero symbol) moves the cursor to the first position of the current line.
- The key '`j`' moves the cursor one line down. If it is the last line, the cursor does not move.
- The key '`k`' moves the cursor one line up. If it is the first line, the cursor does not move.

When moving up and down, the cursor remembers the initial position of moving between the lines — let's call it *start* position. If the cursor moves to a line with a length greater or equal to the number of the *start* position, the cursor's location will be in the *start* position in the new line. If the cursor moves to a line with a length smaller than or equal to the *start* position, the cursor's location will be in the end position of the new line. Similarly, with further pressing of the '`k`' or '`j`' keys the cursor chooses the position in the new line based **not** on the current position, but on the *start* position, from which its moving between the lines began. The memory about the *start* position is erased once a key different from '`k`' or '`j`' is pressed.

Let's examine an example of how the '`k`' key works. Given are three lines with lengths 16, 6 and 11, and the cursor is located in the third line at position 9. With the first press of ¡¡up¿¿, the cursor moves from

(3, 9) to (2, 6). With the second press of ¡¡up¿¿, the cursor moves from (2, 6) to (1, 9), since movement between lines began from the position 9. If you move the cursor left between the first ans second press, the movement of the cursor will look like this: $(3, 9) \to (2, 6) \to (2, 5) \to (1, 5)$.

You are an efficient programmer and you are using Vim as your main editor. Since you are so efficient, you want to do everything with the least effort. You are facing a task — to move the cursor from one location to another. You want to do it with the minimum possible number of key presses. Help yourself: write a program which finds the shortest sequence of key presses moving the cursor from the initial location to the final location in the text.

Input

The first line of the input file contains a single integer N ($N \geq 1$) — the number of lines in the text.

The following N lines contain the text. For convenience, all space symbols in the text are substituted with the point symbol (ASCII 46). The text consists of lowercase Latin letters and spaces (substituted with full points). Each line of the text contains a minimum of one symbol, but it can have no letters. The length of the text (the total length of all lines) is no greater than 10^5.

The following line contains the number Q ($1 \leq Q \leq 10^3$) — the number of cursor move queries.

Each query is described in a separate line. A query consists of 4 integers describing the initial and the final location of the cursor: row_s, col_s, row_f and col_f — the number of the line and position in the line of the initial cursor location and the number of the line and position in the line of the final cursor location, respectively. Lines and positions are numbered from 1 up. It is guaranteed that the provided coordinates are correct and correspond to real positions in the text. The initial and final positions of the cursor are not the same.

The product of the text length to the number of queries is not greater than 10^7.

Output

An answer to each query is the shortest sequence of key presses which moves the cursor from the initial location to the final location. For each query, print an answer on a separate line. If there are several possible shortest sequences, print any of them.

Example

standard input	standard output
3 abc.def q qewrewqr 4 3 7 3 5 1 2 1 5 1 2 1 6 1 1 3 5	hh w $h wjj

Problem 10. Acceleration

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 MB

Viktor Polesov is fleeing from a raging crowd on his wonderbike. To survive, he must drive L kilometers as fast as possible.

The contraption is powered by fire extinguishers which can be switched on in arbitrary order, strictly one at a time. At any moment of time no more than one fire extinguisher can be active. An extinguisher works for t microseconds and in that time, accelerates the bike by a meters per second squared. If none of the extinguishers work, the bike moves at a constant speed.

Initially, the speed of the bike is zero.

Input

The first line of the input file contains a single integer M — the number of tests ($1 \le M \le 10^4$).

Next follow M blocks. In each of these blocks, the first line contains two integers N and L — the number of extinguishers and the length of the road in kilometers, respectively ($1 \le N \le 10^5, 1 \le L \le 10^9$). Next come N lines with descriptions of the extinguishers. Each of them contains two integers a_i and t_i — the acceleration in meters per second squared generated by the ith extinguisher and its working time in microseconds ($1 \le a_i \le 10^5, 1 \le t_i \le 10^5$), respectively.

The total number of extinguishers in all tests is no greater than 10^5.

Output

The output file must contain M real numbers, one per line. Each kth line is an answer to the kth test: the minimum time in seconds necessary to cover the road of the specified length.

Answers must be printed with absolute or relative error no greater than 10^{-8}.

Example

standard input	standard output

1
1 1
100000 100000
	0.15

Example explanation

The mechanic-intellectual turns on the single extinguisher right away and covers half of the way in one tenth of a second, reaching a truly orbital speed of ten kilometers per second. Keeping up this speed, he covers the remaining half of the way in one twentieth of a second.

Problem 11. Unlucky Island

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

An uncontacted tribe lives on the Unlucky Island. Each inhabitant of the island has a circle of *acquaintances*. These acquaintances are either *friends* or *enemies*. Tribespeople treat each other symmetrically: If the tribesperson A knows B, then B knows A; if A is a friend of B, then B is a friend of A; if A is an enemy of B, then B is an enemy of A.

When two tribespeople A and B meet, they must determine how to treat each other. If they know each other, they determine it right away by the type of relationship whether they are friends or enemies. Otherwise, the following ritual is performed.

First, they find out who of the two is the taller one. Let it be the tribesperson A. Next, A begins to list all his acquaintances in any order, and B listens until he hears a familiar name (i.e. the name of someone whom tribesperson B also knows). As soon as A names a common acquaintance C, the listing stops, and their relationship is determined by connection parity. The primeval principle of "the enemy of my enemy is my friend" works here: tribespeople A and B become friends if C is their common friend or their common enemy. Otherwise they become enemies.

If the A has listed all of his or her acquaintances and B has not recognized anyone, they part ways, shocked, as if nothing has happened, without the faintest idea of what should their relationship be.

Gennadiy Kozodoev, a renowned anthropologist, came to the island to study the local culture. He interviewed the indigenous people and built a relationship map. Uef is the tallest person in the tribe and he helps the anthropologist as an interpreter. Uef wants to be friends with his people and asks Gennadiy for help. Gennadiy decided to help by using his mighty intellect: he wants to make a list of Uef's acquaintances for the greeting ritual in such an order that the possibility that an arbitrary tribesperson becomes his friend is maximized.

Input

The first line contains two integers: N — the number of tribespeople on the island ($2 \le N \le 42$), M — the number of acquaintance pairs on the island ($1 \le M \le \frac{1}{2}N(N-1)$).

Each of the following M lines contains a single acquaintance description with three numbers u, v and t, where u and v are two people who are acquaintances with each other ($1 \le u < v \le N$), and t is the type of the relationship: $t = 0$ if they are friends and $t = 1$ if they are enemies. All tribespeople are numbered from 1 to N; Uef is number 1.

It is guaranteed that for each pair of tribespeople u and v the file contains no more than one description of their relationship.

Output

The first line must contain the maximal possibility that Uef becomes a friend of an arbitrary tribesperson after meeting him on the island. Note that Uef can meet any of the $N-1$ other tribespeople with equal probability. The probability must be printed as a real number between 0 and 1, and its absolute or relative error must be no greater than 10^{-8}.

The second line must contain the order of listing Uef's acquaintances that maximizes the probability of friendship.

Example

standard input	standard output

7 8
1 2 0
1 3 0
2 4 0
4 5 0
3 4 1
2 5 1
5 7 1
1 7 1
	0.666666666666667
	7 2 3

Note

Because of their acquaintance the meeting of 2 or 3 automatically results in being friends, and meeting 7 — in being enemies. Meeting 6 results in nothing, because he doesn't know anyone. The greeting ritual results in becoming friends with two strangers — 4 and 5: 5 becomes a friend after Uef naming 7 because of the "the enemy of my enemy is my friend" rule, 4 becomes a friend after Uef naming 2 because of the "the friend of my friend friend is also my friend" rule.

Problem 12. UTF-8

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 MB

Vasya the hacker is studying savegame files of a game he wants to hack. Files can contain arbitrary binary data, but Vasya assumes that they contain substrings of UTF-8 symbols.

The UTF-8 encoding is used for representing an arbitrary sequence of 31-bit values (symbols) as a sequence of bytes. Usually, only Unicode symbols are coded in UTF-8, but Vasya is sure that the game uses the whole range of values from 0 to $2^{31} - 1$, because all names in the game are written in procedure-generated languages. Vasya doubts that he will be able to use functions from the standard library of his programming language to decode the data.

UTF-8 is a prefix self-synchronized encoding. To encode a sequence of values in UTF-8, you must represent each value as a sequence of one to six bytes. Next, you must write down all resulting byte sequences in the order of writing the values in the initial sequence. Each individual value is encoded according to the table:

length	template	bits
1	`0xxxxxxx`	7
2	`110xxxxx 10xxxxxx`	11
3	`1110xxxx 10xxxxxx 10xxxxxx`	16
4	`11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`	21
5	`111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx`	26
6	`1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx`	31

Here, the first column contains the number of bytes in the representation, the second column contains the template, and the third row shows the number of the letters 'x' in the template. The template shows (in binary notation) the bytes, which are obtained from using it for representing the value: the digits '1' and '0' denote fixed bit values, and the letter 'x' denotes the bit, the state of which is defined by the coded value.

To encode a value V as a sequence of k bytes:

1. Write the number V in binary.
2. Find the kth row in the table and lock it for further steps.
3. If necessary, add leading zeroes to V so that the number of bits in V matches the number in the last column of the table.
4. Insert bits of the number V into the letters 'x' of the template. Bits are inserted from left to right, in the order from most significant to least significant.
5. As the result, the template turns into k bytes with eight bits in each — the result of encoding V.

Note that if the number V in binary has more bits than there are letters 'x' in the template of the length k, then it is impossible to encode it in k bytes. On the other hand, for a specific value of V it is often possible to choose the number of bytes k in several ways, and all these variants are possible.

For instance, the letter 'c' of the Cyrillic alphabet is represented by the value 1089 in Unicode, which translates into 10001000001 in binary. In UTF-8 this value can be written down in five ways:

- $\texttt{110}\underline{\texttt{10001}}\ \texttt{10}\underline{\texttt{000001}}$
- $\texttt{1110}\underline{\texttt{0000}}\ \texttt{10}\underline{\texttt{010001}}\ \texttt{10}\underline{\texttt{000001}}$
- $\texttt{11110}\underline{\texttt{000}}\ \texttt{10}\underline{\texttt{000000}}\ \texttt{10}\underline{\texttt{010001}}\ \texttt{10}\underline{\texttt{000001}}$
- $\texttt{111110}\underline{\texttt{00}}\ \texttt{10}\underline{\texttt{000000}}\ \texttt{10}\underline{\texttt{000000}}\ \texttt{10}\underline{\texttt{010001}}\ \texttt{10}\underline{\texttt{000001}}$
- $\texttt{1111110}\underline{\texttt{0}}\ \texttt{10}\underline{\texttt{000000}}\ \texttt{10}\underline{\texttt{000000}}\ \texttt{10}\underline{\texttt{000000}}\ \texttt{10}\underline{\texttt{010001}}\ \texttt{10}\underline{\texttt{000001}}$

Vasya has a save file, which is a sequence of bytes. He wants to find and decode all intervals of the file encoded with UTF-8. Help him write a program for this purpose. The task is to find all inclusion-wise maximal subsets of consequent bytes in the sequence which are sequences of values encoded in UTF-8. Note that two such subsets cannot overlap, i.e. such subsets cannot share bytes. For each subset, print the encoded values. You should **not** print subsets with less than three encoded values.

Input

The input data provide the contents of the savegame file: a nonempty sequence of bytes separated by spaces and/or line breaks. The value of each byte is defined by a two-digit hex number. Each digit can be from 0 to 15, with the digits 10, 11, 12, 13, 14 and 15 encoded by the capital Latin letters 'A', 'B', 'C', 'D', 'E' and 'F', respectively.

The number of bytes lies in the range from 1 to $2 \cdot 10^5$ inclusive.

Output

For each maximal decodable interval of the savegame file with three or more values, print the decoded values in a separate line. Each value is printed as a hex number without leading zeroes, and values are separated by spaces.

Examples

standard input	standard output

`C1 B3 E0 81 B3 F0 80 81 B3`
`F8 80 80 81 B3 FC 80 80 80`
`81 B3 80 C1 B3 E0 81 B3 F0`
`80 81 B3 F8 80 80 81 B3 FC 80`
`80 80 81 B3 E0 AF B5`
	`73 73 73 73 73`
	`73 73 73 73 73 BF5`

`FF 00 FF D1 81 F0 80 91 81`
`00 D1 81 FF F0 80 91 81 FF`
`FF 00 D1 81 F0 80 91 81`
	`441 441 0 441`
	`0 441 441`

Commentary

The current problem does **not** require any sequence of values encoded in UTF-8 to be represented by the minimum possible number of bytes, as required by the real-world UTF-8 specifications.