Problem 1. Complicated documents

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

Roman is a tech writer in a big IT company, Textawei. His work consists in writing technical documentations, rules, specifications, etc. The volumes of the texts he has written are immense. Everything was going great until one day he received a letter from his boss... The letter stressed the importance of keeping the information in the documents relevant and keep the style consistent, and there was an attachment with a set of rules — the description of the style. Moreover, they set up a whole system to check all text files and point out inconsistencies.

Roman rushed to view the report on his documents and saw around 100500 urgent problems. Despite his massive grief, Roman bravely decided to analyze his mistakes. It turned out that the majority of the inconsistencies had to do with a single rule, which stated: "Before and after each colon and dash there must be at least one space or beginning/end of line".

At this point, Roman realized that he could come up with a program to fix his documents, at least in regards to this particular rule. The program would add the minimal number of spaces to the text in such a way that the rule is satisfied.

Unfortunately, Roman is a tech writer and cannot code. Help him!

Input

The first line of the input file contains a single integer T — the number of lines in the text ($1 \le T \le 10\,000$). Next come T lines — the document text.

It is guaranteed that the total number of characters in the text is not greater than 10 000. All characters have ASCII-codes from 32 to 126 inclusive.

Output

Print the corrected text into the output file.

Examples

input.txt
2 This document describes a secret: new powerful product-Arkana! What is that? Let us start:
output.txt
This document describes a secret : new powerful product - Arkana! What is that? Let us start :

Commentary

No line in the example ends with a space character.

Problem 2. Antiwaist

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	2 seconds
Memory limit:	256 megabytes

Outstanding Siberian scientists received the Ig Nobel Prize in anatomy for their work titled «Waist and Antiwaist». As it always happens with groundbreaking scientific achievements, the new idea began to seep into other areas of science and technology. It turns out that the concept of antiwaist is useful in engineering: it allows to find the most durable parts of a structure. There is, however, a problem: engineers do not have a program yet to find the antiwaist in a solid body.

Let us consider a solid body. We can build a planar section of this body with any horizontal plane $z = \text{const}$. An *antiwaist* is a horizontal section with maximum section area.

A solid body is a closed regular set of points in a three-dimensional space. It cannot have infinitely thin elements. A body is defined by its boundary, which is presented as a triangulation.

The triangulation is provided in indexed format: as an array of vertices and an array of triangles. All vertices have integer coordinates. All vertices are different. Each triangle is defined by its vertices: their indices in the global array of vertices are provided. All three indices of a triangle are different. Every triangle has non-zero surface area. The indices of a triangle are enumerated in counter-clockwise order when looking from outside the body.

Since triangulation bounds a solid body, it has no self-intersections and no self-touchings. Two triangles can share a segment if they have two common vertices, or share a point if they have one common vertex. They cannot have any other common points. There are no non-manifold edges or vertices, i.e.: each edge belongs to two triangles; the set of triangles containing a vertex is linked by their sides into a single loop. A body can be disconnected and can have hollows.

Input

The first line of the input file contains two integers: V — the number of vertices, T — the number of triangles ($4 \le V, T \le 200\,000$).

The following V lines describe the vertices. Each line contains three integers — x, y and z coordinates of the vertex, respectively. The remaining T lines describe triangles. Each line contains three integers — the indices of the vertices of this triangle. Vertices are numbered in the order of description starting from zero.

The coordinates do not exceed 500 000 in absolute value. Right-handed coordinate system is used.

Output

Write the found antiwaist into the only line of the output file. Print two real numbers: the z-coordinate of the plane section and the surface area of this section. If there are several horizontal sections with maximum area, print the z-coordinate of any of them.

It is recommended to print real numbers with maximum precision. The absolute or relative error of the area of the printed section must not exceed 10^{-8}.

Examples

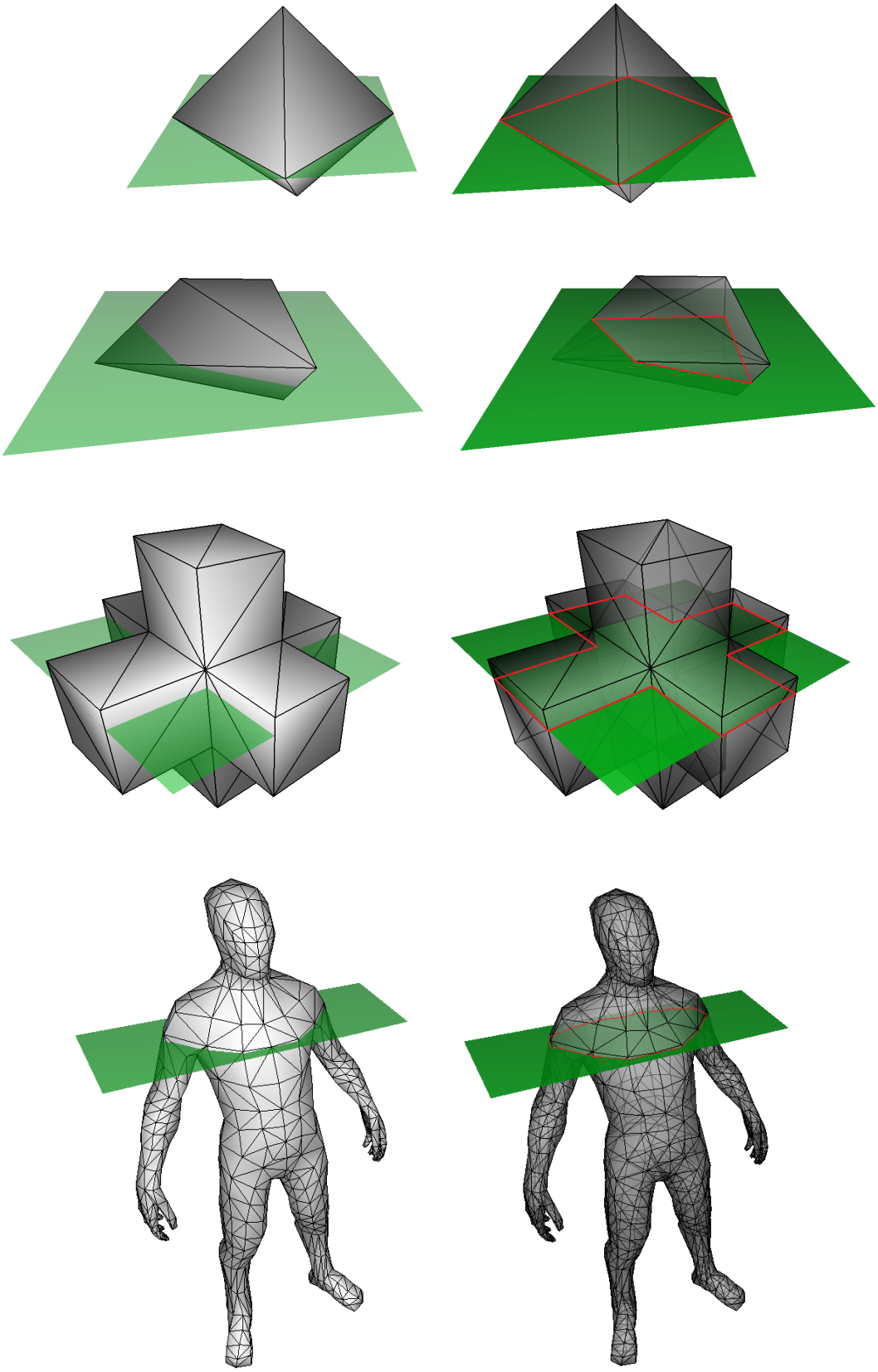
The input data for the last two examples in the table are **not** shown. They can be found in the samples archive, available for download near the problem statements.

input.txt	output.txt
6 8 1 0 0 -1 0 0 0 1 0 0 -1 0 0 0 1 0 0 -1 2 0 5 1 4 2 0 2 4 5 0 3 1 3 4 3 1 5 2 5 1 4 3 0	0 2
6 8 0 0 0 2 0 0 0 1 0 0 0 1 1 0 1 0 2 1 1 0 2 4 5 3 0 1 3 3 1 4 3 5 0 5 2 0 4 1 5 5 1 2	0.5 1.5
32 60 ... The complete data are in the samples archive!	0.7913756716346761 5
896 1788 ... The complete data are in the samples archive!	-3318.6080203949518 4003775.6378878844

Example explanation

In the first example, the body is a regular octahedron with its center at the origin, and all its vertices lie on the coordinate axes. The maximum section area is achieved at the middle section $z = 0$, which is a square with the side of $\sqrt{2}$. In the second example, the largest section is also obtained at the middle by z. In the third example, there is a cross made up of six cubic protrusions. The section of maximum area is obtained at $0 \le z \le 1$, being a planar cross. In the last example,

the body is a human body. You can see the usual location of the antiwaist in a human. Illustrations of sample tests are provided below.



Problem 3. Find the radio operator

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

Stierlitz is awaiting a new assistant — a radio operator girl about to be parachuted from a plane. But the pilots missed the spot in the pitch black night, and dropped the girl in the wrong spot. The only saving grace is that this spot is a point on a plane with integer coordinates... Stierlitz has no choice — he must find the girl. Crashing through thickets with a flashlight is not a becoming pastime for a Standartenfuerher, but alas, he has got no one to delegate it to. One of his deputies is enjoying a vacation, another one has hit the bottle, and the third one, albeit done with his vacation — and the bottle — is good for nothing. To cope with the challenge, Stierlitz has borrowed a radio tracker from Herr Schellenberg and is driving across the field, trying to pinpoint the poor girl.

The tracker device, the latest invention of the self-taught locksmith genius Polesov, works in the following way: the car stops in a point on the plane and directs the antenna in a certain direction, measuring the strength of the signal. If Stierlitz is standing in the same spot as the radio operator, the device reads -1 regardless of the direction. Otherwise, the device shows a signal level of $P \cos \varphi / R^2$ if $\varphi < 90$ degrees, and zero if $\varphi \geq 90$ degrees. Here φ is the angle between the vector of the antenna direction and the vector pointing from Stierlitz's car towards the radio operator, R is the distance from the car to the radio operator, and P is an unknown constant depending on the radio equipment.

Stierlitz must pinpoint the location of the radio operator with less than 10 bearings; more bearings will summon Mueller's boys from the competing organization.

Interaction Protocol

This is an interactive task, and instead of file input-output you will be working with a special program, called interactor. You will interact with this program using the standard input-output streams. To make a query, print the question mark into the standard output stream followed by four space-separated integers x_0, y_0, x_d, x_d, where x_0, y_0 are the coordinates of Stierlitz's car, and x_d, y_d are the components of the antenna direction vector ($|x_0|, |y_0|, |x_d|, |y_d| \leq 10^4$). The direction vector must be nonzero, i.e. $|x_d| + |y_d| > 0$.

In response to the standard input stream a single real number is sent: the power of the signal shown by the device. If this value equals -1, the program must be stopped immediately, since the radio operator has been found.

The coordinates of the radio operator are integers and do not exceed 10^3 in absolute value. The constant P is a real number between 1 and 10^6.

Make sure you print the end-of-line character and flush the output stream buffer (function `flush` of the programming language) after each printed query. Otherwise the solution can get Timeout response.

Example

For reading convenience, commands in the example are separated by lines with the minus symbol. Your program must **not** print any minuses.

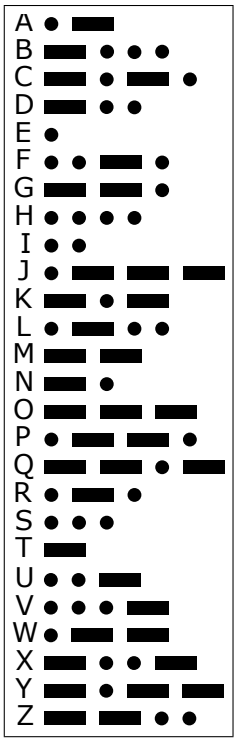
standard input stream	standard output stream
- 0.035999999999999972799536 - 0.000000000000000000000000 - 0.008999999999999993199884 - 0.089999999999999827915431 - 0.166666666666666574148081 - -1	? 0 0 0 1 - ? -2 -3 -1 1 - ? -4 -3 0 100 - ? 3 0 3 1 - ? 4 0 0 1 - ? 4 3 1 1

Example explanation

In the sample, the radio operator girl is located at point (4, 3) and the constant P is 1.5. Notice that interactor computes the power of the signal using double precision and prints it with many decimal points.

Problem 4. Morse code

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

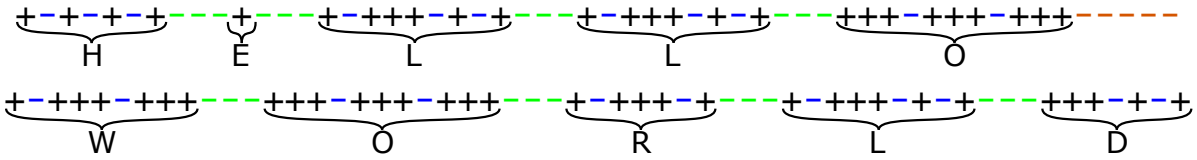


In Morse code, each letter is represented by a sequence of «dots» and «dashes». However, this alone is not enough to transmit text by radio. In addition, Morse code defines the duration of the dot and dash, as well as of the different pauses between them.

In this problem, we will work with digitized radio signal encoded on the sending side using Morse code. This signal is represented as a string of «plus» and «minus» characters. Each character defines whether there was sound during the corresponding time tick: plus means there was sound, and minus means there was no sound. For our purposes, assume that Morse code has five types of elements. These elements are written into signal according to the following table:

`+`	«dot»	there is sound during one tick
`+++`	«dash»	there is sound during three ticks
`-`	pause inside a letter	no sound during one tick
`---`	pause between letters	no sound during three ticks
`-----`	pause between words	no sound during **five** ticks

For instance, with the standard Latin Morse alphabet, the two words `HELLO WORLD` will be encoded into a signal with 63 elements and 109 ticks:



Note that there is a five-tick pause between the words, and that there are no pauses or special elements at the beginning and at the end of the text.

The automatic digitization procedure is known to produce errors, making the signal difficult to decode. The procedure can shorten or lengthen any element by one tick, except that a one-tick element cannot become any shorter.

You are given a dictionary and a digitized signal. Find the minimal number of errors that the digitization procedure could have made, and recover the original text, assuming that:

1. the original text contained only words from the dictionary,
2. Latin Morse alphabet is used,
3. there are no other errors apart from those described above.

Input

The first line of the input file contains two integers: M — the number of words in the dictionary, N — the number of ticks in the signal ($1 \le M, N \le 5\,000$).

The following M lines contain words from the dictionary, one per line. Each word is non-empty and contains only capital Latin letters. The sum of lengths of all these words does not exceed 5 000.

The remaining lines contain a digitized signal, being a sequence of «plus» and «minus» characters with the total length of N. The sequence can be split arbitrarily into several non-empty lines. The signal contains at least one «plus».

The Latin Morse alphabet in text form is available for download near these problem statements.

Output

If the signal provided in the input data could not have been produced under the given conditions, print a single number `-1`.

Otherwise, print a single integer $K \geq 0$ into the first line, being the minimum possible number of errors; into the second line, print the original text, which could produce the signal with this number of errors. The words of the text must be separated with exactly one space.

If there are several variants of the original text which could produce the signal with minimal number of errors, print the **lexicographically minimal** one among them. It is assumed that a space is smaller than any letter.

Examples

input.txt	output.txt
2 109 HELLO WORLD +-+-+-+---+---+-+++-+-+--- +-+++-+-+---+++-+++-+++----- +-+++-+++---+++-+++-+++--- +-+++-+---+-+++-+-+---+++-+-+	0 HELLO WORLD
3 115 WORLD PROGRAM HELLO +-+-+-+---++--+-++++-+-+--- +-++-+-++---+++-+++-+++------ +-+++-+++---+++-+++-++--- +-++++-+---++-+++-+-+---+++-+--++	12 HELLO WORLD
1 13 E +-----------+	-1
1 13 HELLO +--+--+--+--+	-1

Example explanation

The first sample contains exactly the example from the problem statement without any errors. In the second sample the text is the same, but twelve elements are cut/extended by one tick.

In the third sample there is an 11-tick pause, which is impossible to produce. The longest element is a 5-tick pause between words, which can become 6-tick long as the result of an error.

In the fourth sample, there are five dots, and they can make up the letters `E`, `I`, `S` and `H` in various combinations, however, there is only one word `HELLO` in the dictionary, which cannot be built using these letters alone.

Problem 5. Loggers Inc.

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	2 seconds
Memory limit:	256 megabytes

> *— Listen here, guys! So here's our woodlot. Cut the biggest one and drop them other logs on top of it. Tie them all up in a choker, and then you drag them outta here. Got it? — This kinda makes sense. Leave the regrowth bush. Don't worry about stumps. And mind the steel cable! This way, we make the most of it. Salary, too...*
>
> «The Girls/Devchata» [a 1962 Soviet movie]

Ilya Kovrigin and his team of lumberjacks came up with a new optimizing approach. The idea consists in felling several trees so that they fall down roughly in the same spot, to tie them up together and haul away with a tractor. For the purpose of this problem, define the maximum number of trees that can be felled in such a way that they can all be tied to a still-standing tractor.

Assume that the ground is a horizontal plane. There are N trees growing from the ground, and each tree is a directed segment which has a beginning and an end. A growing tree is a vertical segment with its beginning strictly on the ground. For each tree, its coordinates (x_i, y_i) and height h_i are provided.

When a tree is cut and felled, it becomes a segment on the ground plane with the beginning in the same point (x_i, y_i), and the unchanged length of h_i. The direction from the beginning of the felled tree towards its end is arbitrary and is selected by the logger.

The tractor can be driven to any point. You can only tie those trees to the tractor whose ends are not farther than R from the tractor. Assume that the tractor and trees do not stand in the way of each other, i.e. there is no movement restriction.

Input

The first line of the input file contains a single integer N — the number of trees in the lot, and a real number R — the radius of the tractor's grab ($1 \le N \le 250$, $\frac{1}{10} \le R \le 1\,000$). The other N lines describe trees. Each tree is described by three real numbers: the coordinates x_i, y_i on the plane and the height h_i ($|x_i|, |y_i| \le 1\,000$, $\frac{1}{10} \le h_i \le 1\,000$).

All real numbers have no more than 15 decimal points. It is guaranteed that no two trees stand closer than $\frac{1}{10}$ from each other. It is guaranteed that if you increase the tractor grabbing radius R by 10^{-3}, the maximum number of trees in the answer will not change.

Output

In the first line of the output file, print a single integer A — the maximum number of trees that can be tied to the tractor at once. In the second line, print two real numbers X^* and Y^* — the coordinates of the point where the tractor should be put. In the remaining A lines, point out which trees must be felled and how. In each of these lines, print an integer k_i — the number of the tree to be felled, and φ_i — the polar angle defining the direction of the felling ($1 \le k_i \le N$, $0 \le \varphi_i \le 360$).

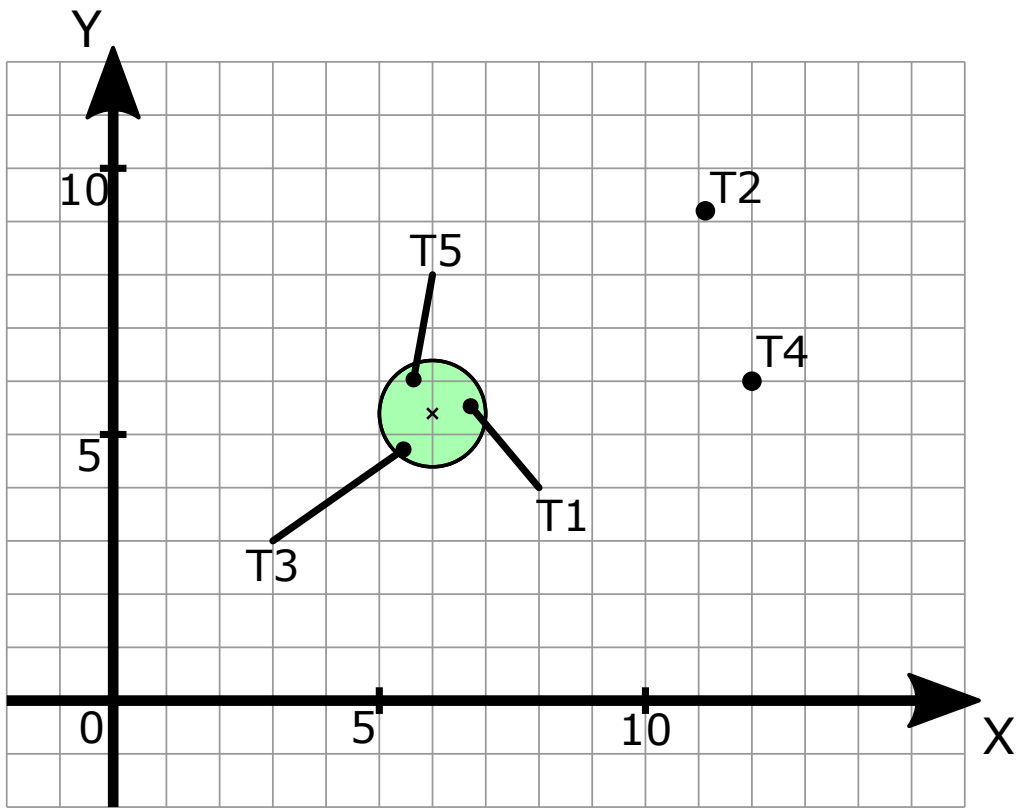
Trees are numbered in the order of description in the input file, starting from 1. The order of printing the trees is irrelevant. The polar angle is measured starting from the X axis in the direction of the Y axis in degrees and defines the direction of the felled tree from its beginning to the end.

It is recommended that all real numbers be printed with 15 decimal points. The checking program will check your solution with the radius R, increased by 10^{-6}.

Examples

input.txt	output.txt
5 1.0 8.0 4.0 2.0 11.13 9.19 0.55 3 3 3 12 6 1 6 8 2	3 6 5.41234567890 1 130 5 259.726501937845610 3 35

Illustration



Problem 6. Caching approximations

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	2 seconds
	4 seconds (for Java)
Memory limit:	256 megabytes

In high-precision geometric modeling it is necessary to store various complex curves, such as for instance an intersection curve of two surfaces. The complexity arises because such curves are virtually impossible to represent in both precise and usable way: the representation is either approximate or is so complex that it cannot be used in many geometrical algorithms without rewriting them. This is why a mixed approach is commonly used: to retain the maximum precision, the curve is stored in the hard-to-use but precise format, but computational algorithms use a simple approximation of the curve generated with sufficient precision.

Perhaps the most convenient approximation is the Hermite cubic spline, which is easy to generate and easy to work with. Unfortunately, as approximation precision is improved, then the cubic spline takes more memory to store, takes more time to generate, and all algorithms work slower with it. The cubic spline approximation has error of fourth order, meaning that a 16-fold improvement of precision doubles the size of the approximation.

For the purpose of this problem, we shall assume that an approximation with precision ε takes M bytes of memory, where M is calculated as:

$$M = \frac{s}{\sqrt[4]{\varepsilon}}$$

Such approximation takes T CPU cycles to generate:

$$T = aM + b$$

Here, s, a and b are some given constants.

To avoid generating the same approximation for a curve many times, it is sometimes "cached", i.e. saved along with the curve for later use. In this problem, there is only one complex curve, and you must find the best way of caching its approximations, such that the given N operations are executed as fast as possible.

The operations must be executed strictly in the same order as they are given. No operation can be executed directly on a complex curve; each operation must be given an acceptable approximation of the curve instead. Each operation has a given "tolerance" δ, which defines the requirement on input approximation precision. A curve approximation generated with the precision ε is good enough for an operation with the tolerance δ only if $\varepsilon \le \delta$, otherwise it cannot be used to execute the operation.

The operation execution time is calculated using the formula:

$$T = cM + d$$

Here c and d are certain constants defined independently for each operation, and M is the size of the curve approximation used. Recall that the size of approximation is calculated from its precision ε according to the formula provided above.

Find the minimum time necessary to complete all the operations under three different caching policies:

1. **Caching off**: there is no way to save curve approximations between operations. Once an operation is complete, the used approximation is discarded, and a new approximation must be generated for the next operation.

2. **Cache one approximation**: each time a curve approximation is generated, it is saved in the cache, evicting the old approximation if it is present. The saved approximation can be used in future operations as many times as desired until a new approximation is generated. Note that the old approximation is always evicted, even if the new approximation is less precise, i.e. you cannot generate an approximation without caching it.

3. **Unlimited caching**: all generated approximations are saved in cache in unlimited quantity. All of them are available for use in the future. To execute an operation, any of the previously generated approximations can be chosen, provided that it is precise enough for the operation of course.

In all three variants, the cache is initially empty and does not contain any approximations. Before each operation, you can generate an approximation with any desired precision $\varepsilon > 0$, or you can choose not to generate anything. The time it takes to generate approximations is added to the total operation execution time which you need to minimize.

Input

The first line of the input file contains an integer N — the number of operations to be executed ($1 \le N \le 10\,000$). The second line contains three real numbers s, a and b — constants affecting the size and the generation time of approximations ($10^{-3} \le s \le 10^3$, $10^{-4} \le a, b \le 10^4$).

The remaining N lines describe operations. Each line contains three real numbers: δ — the required precision for the operation, and c, d — constants affecting the operation execution time ($10^{-12} \le \delta \le 1$, $10^{-4} \le c, d \le 10^4$).

It is guaranteed that the tolerances δ of any two operations either are exactly equal or differ by at least 10^{-3} in relative sense. Real numbers in the input file can be represented in exponential format, e.g. `1e-10`

Output

The output file must contain three answers; each answer contains $N+1$ lines. Between the answers, print a line of three "equals" symbols. The first answer is for the case when caching is off, the second is for the case when one approximation is cached, and the third is for the case of unlimited caching.

An answer begins with a line with one real number τ, which is the total time of generating all approximations and executing all operations. Every ith of the remaining N lines must state whether a new curve approximation is generated before the ith operation, and if so, how precise that approximation should be ($1 \le i \le N$). If there is no need to generate an approximation, print -1 into the line; if an approximation must be generated, print a real number ε, defining the precision of the new approximation.

The checking program will check the operation tolerance requirement with a **relative** precision of 10^{-12} and the total work time with the relative precision of 10^{-8}. It is recommended to print all real numbers in **exponential** format with 15 decimal places.

Examples

input.txt	output.txt
1 1.12e-1 0.25 1.37 1.2345e-3 0.57e+2 37.019	72.5964530936900883967007684379 28 1.2345e-3 === 7.259645309369008e+1 0.12345e-2 === 0.7259645309369008e+2 0.0012345
4 1 2 1 1e-4 1e-3 1 1e-12 1 1 1e-8 1e+3 1 0.0625e-8 0.1 1	103648.01 1e-4 1e-12 1e-8 0.0625e-8 === 103628 1e-12 -1 1e-8 0.0625e-8 === 103306.1 1e-08 1e-12 -1 -1

Example explanation

In the first example, there is only one operation, so the answer is the same regardless of the caching policy; in any case, you must generate the curve approximation with a precision equal to the operation tolerance. Note that the size of the approximation is not integer here, and no rounding happens. In the output file, several different ways of printing the answer are shown.

Consider the second example with "one approximation caching" policy. It makes sense to generate an approximation with the precision of 10^{-12} from the very beginning, and use it in the first two operations, because the execution time of the first operation is barely affected by the size of the approximation used, meaning that it would be cheaper to use a more precise approximation instead of generating another smaller approximation. The third operation strongly depends on the size of the approximation, so it makes sense to regenerate the approximation with the worst precision allowed. However, such approximation cannot be used for the fourth operation, so a new and a bit more precise approximation has to be generated for it.

On the other hand, the unlimited caching policy makes it reasonable to reuse the initially generated approximation with the precision of 10^{-12} for the fourth operation. Moreover, it is possible to generate the approximation with the precision 10^{-8} earlier in order to use it for the first operation too.

Problem 7. Scheduler

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	2 seconds
Memory limit:	256 megabytes

There are T processes running in the multitasking operating system «Squirrel OS». Each of the T processes has a given priority p_i, which affects how often the process is run. Since the system only has a single core in a single processor to run things, it is facing a challenge of distributing the CPU time between these processes, taking their priorites into account.

The algorithm of defining which process is run at each time moment can be described in the following way. For each process, in addition to the priority p_i, there is also a counter t_i. Initially all t_i equal 0. Then every second:

1. processes with the maximum value of $p_i + t_i$ are chosen.
2. among such processes, the process with the minimum number i is chosen.
3. the chosen process i is run for one second.
4. for the chosen process i the value t_i is set to 0.
5. for all other processes, the value t_i is increased by 1.

Model the work of the operating system for T seconds and calculate for how many seconds each process was run. Assume that all calculations and switches between processes are instant, so the running time for each process in seconds is an integer.

Input

The first line contains two space-separated integers N and T — the number of processes in the operating system ($1 \le N \le 10^5$) and the number of seconds to be modeled ($1 \le T \le 10^6$).

The second line contains N space-separated integers p_i — the process priorities ($0 \le p_i \le 10^5$).

Output

In the only line of the output file, print N space-separated integers — for how many seconds each of the processes was run.

Examples

input.txt	output.txt
3 10 3 4 5	3 3 4

Problem 8. Text editor

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	3 seconds
	5 seconds (for Java)
Memory limit:	256 megabytes

When using a simple text editor, the user often needs the reorder of lines in a specific way. If the lines are long, and there is a need to minimize errors, the easiest way to achieve this is by moving chunks of text from one place to another (the "cut/paste" method). You are an efficient programmer. Determine the minimum time necessary to shuffle the lines in the desired way.

The file contains N lines of text. The editor has a single cursor with $N + 1$ possible positions: before the first line of the text, after the last line of the text, and inbetween any two adjacent lines. Pressing "up" or "down" moves the cursor one line up or down, respectively. You cannot move the cursor outside the text.

The Shift key is used to select lines. When it is pressed, the editor remembers the current position of the cursor; when it is released, the editor selects all the lines between the current cursor position and the remembered one. After releasing Shift you must press Ctrl+X to cut the selected lines, i.e. copy them into the clipboard and simultaneously remove them from the text. To paste (insert) back the cut fragment of text from the clipboard, press Ctrl+V after moving the cursor to the desired position using the "up" and "down" keys. After pasting, the text fragment is removed from the clipboard. When a text fragment is cut, all the following lines are automatically moved upward, and when it is pasted, they are moved downward. So there are never any empty lines in the text. When a fragment is cut, the cursor is moved up into the line where the cut fragment began unless it is already there; when a fragment is pasted, the cursor is moved downwards and ends up right after the inserted text.

Every action takes some predefined time, depending on user skills. Initially, the cursor is located before the first line of the text.

Input

In the first line of the input file there is one integer N, being the number of lines in the text editor ($2 \le N \le 8$). In the second line six integers are given, defining how much milliseconds each action takes ($1 \le t_i \le 100$). The list of actions is given below.

N integers in the third line define the initial order of the lines. They are different and lie in the range between 1 and N. N integers in the fourth line define the order in which the lines must be placed in the end. They are also different and lie in the range between 1 and N.

Output

Print two integers into the first line of the output file: T being the total time of executing the plan in milliseconds, and K being the number of actions in the plan. Next, print K actions in the order of their execution, one action per line. Actions are described in the following way:

- `Up` — press "up" (takes t_1 milliseconds).
- `Down` — press "down" (takes t_2 milliseconds).
- `Shift-Press` — press and hold Shift (takes t_3 milliseconds).
- `Shift-Release` — release Shift (takes t_4 milliseconds).

- `Ctrl+X` — press the combination Ctrl+X (takes t_5 milliseconds).
- `Ctrl+V` — press the combination Ctrl+V (takes t_6 milliseconds).

Examples

input.txt	output.txt
6 99 98 100 97 99 98 1 2 3 4 5 6 6 5 4 3 2 1	3252 33 Shift-Press Down Down Shift-Release Ctrl+X Down Down Ctrl+V Shift-Press Down Shift-Release Ctrl+X Down Ctrl+V Shift-Press Up Up Up Shift-Release Ctrl+X Up Up Ctrl+V Down Shift-Press Up Up Up Shift-Release Ctrl+X Up Up Ctrl+V

Example explanation

In the example, you must reverse six lines. Every action takes 100 milliseconds or a bit less. The optimal plan is recorded into an animated gif image, which you can download near these problem statements.

Problem 9. Crushing blow

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	2 seconds
Memory limit:	256 megabytes

Khodislav is playing D&D. At the moment, his character is fighting a monster and Khodislav, who is for some reason very sure in his attack, wants to destroy the enemy with the last crushing blow. His character has different weapons; the damage these weapons can cause is defined by rolling dice and characterized by three numbers n, f, and m, where n is the number of dice, f is the number of faces on each die, and m is a modifier. For instance, if $n = 3$, $f = 8$, $m = 5$, then you will need to roll three eight-faced dice, sum the results, and add five to the sum to define the damage; this is usually written as $3d8 + 5$.

To destroy the monster, a weapon must inflict damage of D or greater. Help Khodislav to choose a weapon for his character that will help him to kill the monster with maximum probability.

Dice rolls are independent; it is equally likely to roll each of the faces on a die. There are all numbers from 1 to f written on the faces of the die.

Input

In the first line of the input file, there is a single integer T — the number of tests ($1 \le T \le 5\,000$). It is followed by the description of T tests.

The first line of a test contains two integers: W — the number of character's weapons and D — the minimum required damage to the monster ($1 \le W \le 5\,000$, $1 \le D \le 250$).

The following W lines describe the weapons. Each line contains three integers: n — the number of the dice, f — the number of faces on each die, and m — the modifier ($1 \le n \le 10$, $2 \le f \le 20$, $-10 \le m \le 10$).

It is guaranteed that the total number of weapons in all tests is not greater than 5 000.

Output

For each test, print a single real number on a separate line — the maximum probability of inflicting damage no less than D with a single blow. The absolute error of the answers must not exceed 10^{-11}.

Examples

input.txt	output.txt
2 2 2 1 20 -3 2 2 0 1 11 1 20 -10	1 0
3 1 6 1 6 0 1 7 2 6 0 1 100 10 20 10	0.166666666666667 0.583333333333333 0.799600378342187

Problem 10. Antiplagiarism

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

Students of a specialized arts school must paint a picture as their graduation qualification work. In the course of their study, the students have mastered a technique of painting on a square canvas using two types of brushstroke: «asterisk» and «hash». Using this technique, the prodigies must paint a square masterpiece of the size $N \times N$, using only the two types of strokes.

This is a tedious task, and every year a few students decide they could benefit from the hard work of the previous generations of their alumni-to-be. However, their imagination is quite limited. A student takes someone else's picture and applies the following operations several times: 1) rotate the image 90 degrees, 2) mirror it along the vertical or horizontal axis. After that, he submits the result as his own picture. Some students have even tried presenting unchanged old works as their own.

Their professors can feel something is wrong, but unfortunately, it is beyond their own powers to find out for sure whether the paintings are plagiarized or not. It's time to put an end to this disgraceful business and write a program that would automate the plagiarism check by defining whether a painting is a copy or not, thus helping the professors to find cheating students.

Input

The first line of the input file contains a single integer: N is the size of the side of the square canvas ($1 \le N \le 500$). Each of the following N lines contains N symbols `*` or `#`, denoting the types of the corresponding strokes of the first painting. Next comes an empty line. The next N lines describe the second painting in the same format.

Output

The only line of the output file is the word `YES`, if one of the paintings is plagiarized from another, or `NO` if not.

Examples

input.txt	output.txt
1 * # 	NO
3 #** #*# ### ### **# *##	YES