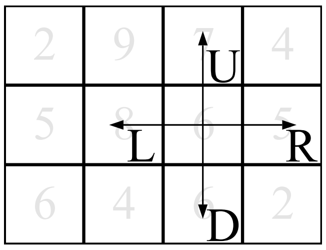
Problem 1. Trampler

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

Trampler the Jackrabbit loves trampling fields. He's just found a good new field. As seen on the map, the field is a rectangle H cells high and W cells wide. When Trampler gets into a cell, he tramples it down to the last twig. It takes him a while to do that, and that time is defined for each specific cell.

Trampler is not sure where to begin, but he's already figured out the sequence of moves. He's scribbled it down on a piece of paper, marking each elemental move with a letter.



There are different kinds of moves:

- letter `'U'`: move from the current field cell to the cell directly adjacent above;
- letter `'R'`: move to the adjacent cell on the right;
- letter `'L'`: move to the adjacent cell on the left;
- letter `'D'`: move to the adjacent cell below.

For each cell, Trampler needs to know the time necessary to implement his plan, if he begins his trampling route from that cell. He will spend some time in the initial cell and in each consecutive cell, even if that cell will have been trampled already.

He is not planning to leave the field until he is done. If starting at a certain cell means having to leave the field eventually, this cell is no good for a starting point. Trampler wants such cells marked with 0.

Input

The first line of the input file defines an integer T — the number of test cases ($1 \le T \le 100$). It is followed by T blocks.

The first line of a block contains two integers H and W — the height and width of the field, respectively ($1 \le H, W \le 100$).

The second line of the block contains a non-empty sequence of moves. Each move is represented by one of the symbols 'L', 'R', 'U', 'D'. The number of moves is less than or equals 100.

Each of the following H lines contains W integers — the time the jackrabbit spends in a specific field cell. These numbers fall in the range from 1 through 1 000.

The total number of cells in all test cases is less than or equals 10^5.

Output

For each test case, print H lines of W integers — the answers for each of the field cells.

Examples

input.txt	output.txt

1
3 4
RDRUL
2 9 7 4
5 8 6 5
6 4 6 2
	41 38 0 0
	37 33 0 0
	0 0 0 0

2
1 1
D
1
5 5
RLDULRUD
4 7 8 5 8
7 9 9 5 7
3 8 5 3 6
1 6 6 9 3
5 2 1 6 1
	0
	0 0 0 0 0
	0 76 72 49 0
	0 63 51 40 0
	0 47 51 63 0
	0 0 0 0 0

Example explanation

The illustration in the problem statement shows the field from the first example and the directions of moves.

Problem 2. Video buffering

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

You are given a video in the MPEG-2 format. For each frame of a video, it is known how much time it takes to decode it. Find the minimal necessary buffer size, such that the whole video could be demonstrated without lags and skipped frames.

The video file contains a sequence of frames in the order they must be shown on the screen. The video must be shown at constant frame rate: the time interval d between two adjacent frames is specified in the file. If we number the frames from zero up, the 0-th frame must be shown at the moment of time 0, the first frame at time d, the second frame at time $2d$, the third frame at time $3d$, etc.

For a video frame to be shown on the screen it has to be decoded first. All frames which have been decoded and are being decoded must be stored in the buffer, except for the frames which will never be needed anymore for sure. The buffer size is set in frames and remains fixed throughout the demonstration of the video. Usually, a frame can be discarded from the buffer once it has been shown, however, this is not always the case, because sometimes in order to decode one frame, you have to have some other frames in the buffer in decoded state. To begin decoding frame f, it is necessary to:

1. have all frames, upon which the frame f depends, decoded in the buffer, and
2. have free slot in the buffer, where the frame is placed for the period of decoding.

Decoding time is specified for each individual frame. No more than one frame can be decoded at any given moment. A decoded frame f can be discarded from the buffer only if:

1. the frame f has already been shown, and
2. all frames dependent on the frame f have already been decoded.

There are three types of frames in the video file:

- `I`-frame. This frame does not depend on anything.
- `P`-frame. Depends on the preceding frame of the type `I` or `P` (the nearest one).
- `B`-frame. Depends on the preceding frame of the type `I` or `P`, and on the following frame of the type `I` or `P` (both frames being the nearest).

For instance, if there are five frames with the types `IBBBP`, the first frame is completely independent, the last frame depends on the first frame, and the remaining frames depend on both the first and the last frame.

The decoding sequence is strictly predefined. Frames must be decoded in the same order as they are to be shown on the screen. The only exception is when a `B`-frame is encountered, which depends on a later frame. In this case the later frame must be decoded first, if it has not been decoded yet. For instance, if frames of the types `IBBPBBI` are given, they must be decoded in the following order: 0, 3, 1, 2, 6, 4, 5.

Find the minimal buffer size required to show all frames of the video strictly on time. It is allowed to begin the decoding any time before the beginning of the video demonstration. Assume that showing a frame on screen occurs instantaneously and takes no time, as well as placing and discarding a frame in the buffer.

Input

The first line of the input file contains two integers: N — the number of frames, d — the time between adjacent frames in microseconds ($3 \le N \le 2 \cdot 10^5$, $1 \le d \le 10^9$).

The remaining N lines describe the frames in the order they must be shown. Each frame is defined by its type (`I`, `P` or `B`) and the amount of time in microseconds required to decode it. The decoding time is an integer in the range from 1 to 10^9 inclusive.

It is guaranteed that the first frame has the type `I`, and the last frame has the type `I` or `P`.

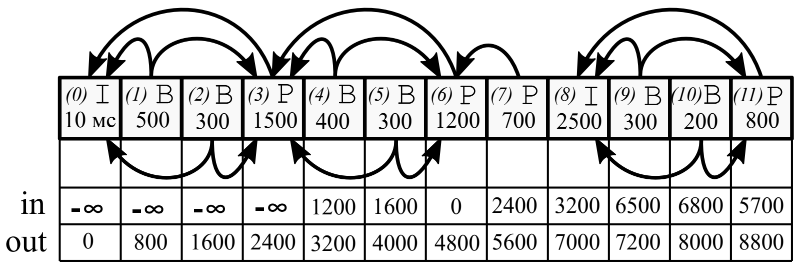
Output

The output file must contain a single integer: the minimal size of the buffer in frames, such that all video is demonstrated without lags.

Example

input.txt	output.txt
12 800 I 10000 B 500 B 300 P 1500 B 400 B 300 P 1200 P 700 I 2500 B 300 B 200 P 800	4

Example explanation



decoding order: 0, 3, 1, 2, 6, 4, 5, 7, 8, 11, 9, 10;

The illustration shows a method of playing the video using a 4-frame buffer. Arrows show frame dependencies, and the in/out lines show when the frame is placed into the buffer and when it is discarded from the buffer. All frames are numbered from zero up.

Problem 3. Package

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

To install packages in the Vinux OS, Vasya uses Vum, the modern package manager.

Each package has several versions that can be installed. Only one of the application versions can be installed at once. Let's call a specific version of a specific application a *package*. Since specific versions of different applications can be mutually incompatible, there is such thing as *conflicts* in Vum. Every conflict is a set of packages, from which only one package can be installed. Attempting to install more than one package from the conflict list leads to a crash, and Vasya does not want that.

Different versions of any application can conflict (or not). Several versions of an application can be in a conflict, including all versions of the application.

Since the conflict description system in Vum is rather primitive, each package can be listed only in one conflict.

Vasya wants to install N different applications without crashing his Vinux. He must choose the application versions without breaking the rules above, but Vum's exhaustive dependency resolution algorithms cannot cope with this task. Vasya is asking for help with his package manager.

Input

The first line of the input file contains an integer N — the number of applications that Vasya wants to install ($1 \le N \le 200$).

The following N lines provide the descriptions of the applications, containing their version lists. An application description consists of the following space-separated fields:

- application name — a line of lower case Latin letters from 1 to 10 symbols long;
- number of versions — a positive integer smaller than or equalling 1 000;
- numbers of versions — positive integers listed in the ascending order.

The numbers of applications are smaller than or equal 100 000. All application names are different.

The following line contains the number K — the number of conflicts ($0 \le K \le 500$).

The following K lines describe the conflicts. A conflict description begins with a positive integer T — the number of packages in the conflict. Next, T space-separated packages are listed. For each package, the application name is provided, followed by the version number.

It is guaranteed that packages listed in the conflicts are never repeated, and that all versions are mentioned in the application descriptions.

Output

If it is impossible to install all N applications without conflicts, the only line of the output file must contain the word `No`. If it is possible, the first line must contain the word `Yes`, and the second line must list space-separated numbers of versions that must be installed to achieve that.

The versions for N applications must be printed in the same order in which the applications are described in the input file. If there are several possible ways of selecting the applications, print any one of them.

Examples

input.txt	output.txt
3 vim 3 1 2 3 nano 2 5 8 python 2 2 3 2 2 vim 3 nano 8 3 vim 1 nano 5 vim 2	Yes 2 8 2
2 firefox 1 38 chrome 1 46 1 2 firefox 38 chrome 46	No

Example explanation

The first example contains two conflicts. The first conflict prevents installing the third version of vim and the eigth version of nano simultaneously. The second conflict states that only one of the following can be installed — 1st version ov vim, 5th version of nano, and 2nd version of vim, or nothing at all. The correct solutions will be the following:

- 1 8 2
- 1 8 3
- 2 8 2
- 2 8 3
- 3 5 2
- 3 5 3

In the second example, there are two applications, each with one version. However, the only conflict forbids installing both applications together.

Problem 4. Vasya's graph

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

Vasya has got a graph. The graph has N vertices, but it's got no edges yet. Vasya cares a lot about the future graph structure: he knows K pairs of vertices $\{u_j, v_j\}$, such that if there is a path between these vertices in the graph, the *irredeemable* will happen to the graph. Vasya must prevent it at all costs.

Vasya has made a list of M **unoriented** edges. Vasya will examine the edges in the preset order. Considering the next edge he will surely put it into the graph, if possible. If adding another edge will cause the *irredeemable*, Vasya will simply discard such an edge. Your task is to find out which edges are good for the graph and which ones must end up in the trash.

Input

The first line of the input file contains three integers N, K and M ($1 \leq N \leq 10^5, 0 \leq K, M \leq 10^5$).

It is followed by K lines, with the i-th line containing two integers u_i and v_i — the numbers of conflicting vertices, which should not have any paths between them ($1 \leq u_i < v_i \leq N$). The conflicting vertex pairs are unique.

Next come M lines with the i-th line containing two integers $\tilde{u}_i$ and $\tilde{v}_i$ — the numbers of vertices of the edge which can be added to the graph ($1 \leq \tilde{u}_i < \tilde{v}_i \leq N$). These edges are provided in the order of examination. Edges in the list are unique.

Output

The first line of the output file must contain the number of edges that Vasya can accomodate into the graph. The second line must contain space-separated numbers of edges in the ascending order.

Examples

input.txt	output.txt

3 1 3
1 2
1 2
2 3
1 3
	1
	2

5 2 6
1 2
2 3
1 3
2 4
3 4
1 4
4 5
1 5
	3
	1 2 5

Problem 5. Quadratic equation

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

Ted has just been accepted to the university, afer passing the exams successfully. During his preparation for the exams, Ted solved several thousands of similar problems. For instance, he solved tons of quadratic equations, again and again and again. He's decided to make up his own problems, because after the exams, he's free to do anything! Ted is wondering – why does one always have to find the roots of the equation, and not vice versa – to find the equation for a root? First, he decided to experiment with what has been his favorite type of equations since the ninth grade — the quadratic equations. But he discovered a new problem here – he is not knowledgeable enough, because this problem is not typical.

Help Ted solve this new task!

Input

The first line of the input file contains an integer T — the number of test cases ($1 \le T \le 100$). It is followed by T lines – one per each case.

Each line contains one non-zero integer Y — the root of a quadratic equation ($1 \le |Y| \le 10^6$).

Output

The output file must contain T lines, and the i-th line must have the answer to the i-th test case. The answer must contain three space-separated **integers** A_i, B_i, C_i, such that the i-th input number is a root of the quadratic equation $A_iX^2 + B_iX + C_i = 0$.

All coefficients must be **non-zero** and **not greater than** 10^6 **in absolute value**. If the required equation is not unique, print the coefficients of any of them. It is guaranteed that at least one equation exists in each case.

Example

input.txt	output.txt
2 1 -2	2 -3 1 3 2 -8

Problem 6. Alignment

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
	3 seconds (for Java)
Memory limit:	**64 megabytes**

Petya is studying the C language and has just reached the topic about structures. He has always been amazed by the fact that fields of a structure do not have to be placed in the memory strictly one after another: sometimes there is "padding" between adjacent fields. This way, the size of a structure may depend on the order in which its components are listed!

A description of a structure in C is provided. It begins with a keyword `struct`, followed by a description of the structure fields in squiggle brackets, with a semicolon in the end. The description begins with the field type, followed by the field name and closed with a semicolon. It is guaranteed that the field type is separated from the field name by at least one space. The field name is a non-empty line containing upper and lower case Latin letters, digits and underscores, **never** beginning with a digit.

The defined type can be either a primitive type or a pointer to a primitive type. The following primitive types are allowed:

- `char` has the size of 1 byte.
- `short` has the size of 2 bytes.
- `int` and `float` have the size of 4 bytes.
- `int64_t` and `double` have the size of 8 bytes.

When a field is a pointer, the type begins with a primitive type followed by one or several asterisks. Regardless of the primitive type and the number of asterisks, any pointer is 8 bytes in size, i.e. we are presuming a 64-bit platform.

The structure is placed in the memory in the following manner. The first field of the structure is located at an address with a maximum alignment: assume its address divides by 8. Each consecutive field is located at an address such that:

1. it is located after the preceding one,
2. it is correctly aligned, i.e. its address divides by its size,
3. its address is minimal provided the first two conditions are satisfied.

Adjacent fields may have a padding of 1 to 7 bytes.

The last field can be followed by several empty bytes. This is necessary to enable the storage of an array of such structures, i.e. when it place a number of consecutive identical structures in the memory. In the end of the structure, a minimum number of empty bytes is added, such that in array of any size all fields of all structures be correctly aligned. The size of the whole structure equals the sum of sizes of all fields and sizes of all paddings.

Petya can switch the locations of fields in the structure. He wants to know the minimal an maximal possible sizes of the structure. He is also curious about what is the average size of the structure if he swaps all fields at random.

Input

The input file describes a single structure according to the rules defined in the problem statement.

Any number of spaces and/or line break symbols can be added anywhere in the description, if such addition does not cut through a name of a primitive type, field name, or keyword. It is guaranteed that the structure has 1 to 100 fields, and the total number of symbols in the description is smaller than or equals 5 000. All names of structure fields are different from each other.

Output

The only line of the output file must contain three space-separated numbers: minimal possible structure size, maximal possible structure size, and average structure size. The first two numbers must be integers and the third number must be real.

The absolute or relative error of the third number must be lower than or equal 10^{-9}.

Example

input.txt	output.txt

`struct  {`
`  int x;`
`  int64_t* *   Y;`
 	
`  char   _temp1;`
`} ;`
	`16 24 18.6666666666666`

Example explanation

The field `x` has a size of 4 bytes, the field `Y` has a size of 8 bytes, and the field `_temp1` has a size of 1 byte.

If the fields are not shuffled, then the field `x` will be occupied by the bytes 0-3. We will have to add 4 empty bytes for the field `Y` to be aligned to 8 bytes, occupying the bytes 8-15. The field `_temp1` will have the address 16 and will take up one byte, and we will have to add 7 empty bytes. Without the empty bytes, the field `Y` of the second structure in the array cannot be correctly alighed. The total size amounts to 24 bytes.

If we choose the order of the fields, we get the size of 24 bytes only when the field `Y` is located between the two other fields. This happens with 2 cases of all 6 possible permutations. In other cases, the size of the structure equals 16 bytes.

Problem 7. Rocket

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

The secret design bureau of the X Institution, or just the *Bureau*, is working on an ambitious project: they are building a rocket which must reach the Sun and land on its suface. The rocket design is very complicated, however, the blueprints are ready, and a set of the necessary structural parts has been approved. For the i-th detail, the Bureau is considering K_i appropriate basic materials, each characterized by two valuesm_{ij} and c_{ij}. m_{ij} is the mass of the detail made completely of that particular basic material, and c_{ij} is its cost in this case.

The material for the i-th detail can be one of the K_i basic materials or an alloy of two basic materials. In case of an alloy of two basic materials with the masses m_1, m_2 and the costs c_1, c_2 in the proportion of $\alpha \in (0; 1)$, the resulting part has the mass $\alpha m_1 + (1 - \alpha)m_2$ and the cost $\alpha c_1 + (1 - \alpha)c_2$. The engineers are not sure about the durability of three-material alloys. Perhaps we shouldn't bother with them, either.

Your task is to decide for each of the N parts, which material it should be made of in order to minimize its cost. The rocket must be able to take off, so the total mass of all parts must not be greater than M.

Input

The first line of the input file contains two positive integers N and M — the number of parts necessary for building the rocket and the maximal total mass of all parts ($1 \le M \le 10^9$).

Next come N blocks of lines. The i-th block provides the description of the i-th part. The description of the i-th block begins from a line containing a positive integer K_i. It is followed by K_i lines, with the j-th line containing two integers m_{ij} and c_{ij} — the mass and cost of the i-th part made entirely of the j-th recommended basic material ($1 \le m_{ij}, c_{ij} \le 10^9$).

It is guaranteed that $\sum\limits_{i=1}^{N} K_i \le 10^5$. It is also guaranteed that there is a solution.

Output

The first line of the output file must contain the minimal possible cost of the rocket as a real number. The i-th of the following lines must contain the description of the optimal material for the i-th part ($1 \le i \le N$).

The description of the material for the i-th part must have the following structure.

If the i-th part must be made of a basic material, the description line has the format "1 A", where the integer A is the number of the basic material ($1 \le A \le K_i$).

If the i-th part must be made of an alloy of two basic materials, the description line must have the format "2 A B X Y", where the integers A and B are the numbers of basic materials ($1 \le A, B \le K_i$), and the integers X и Y are the numenator and denominator of the alloy proportion ($0 < X < Y \le 10^9$, $\alpha = \frac{X}{Y}$).

It is required for proportions of all alloys in your answer to have the same denominator Y. It is guaranteed that solution satisfying this condition exists. The relative or absolute error of your answer (the minimal cost of the rocket) must be less than or equal 10^{-12}.

Examples

input.txt	output.txt

2 11
3
4 3
6 3
7 8
4
9 5
10 3
6 5
7 6
	7.5000000000
	1 1
	2 3 2 3 4

2 4
2
1 2
2 3
2
3 2
2 5
	4.0
	1 1
	1 1

Problem 8. Cabbage

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

"The hungry" Children of the Volga Plain are known to be very fond of pickled cabbage. However, each of them has a favorite cabbage variety and would not eat any other kind. The preferences seem to be random. Different Children can like either different or the same varieties of cabbage. To make everyone happy, portions must be the same. Alchen, the chief, wants to make the portions as big as possible.

Initially, Alchen has a certain stock of each of the cabbage varieties, and a certain sum of money. He can buy extra cabbage with this money, a different amount of each variety. The prices are well-known. However, he won't be able to sell the cabbage he's already got.

Help Alchen to figure out the best portion size for his proteges.

Input

The first line of the input file contains a single integer T — the number of test cases ($1 \le T \le 100$). It is followed by T blocks.

The first line of a block contains three integers: N — the number of pickled cabbage varieties ($1 \le N \le 10^5$), M — the number of the hungry Children ($1 \le M \le 10^5$), S — the sum of money allocated for buying extra pickled cabbage ($1 \le S \le 10^9$).

The second line of a block contains M integers T_i, where T_i is the number of the pickled cabbage variety preferred by the i-th Child of the Volga Plain ($1 \le T_i \le N$).

Each of the following N lines contains two integers: A_i — the initially available amount of cabbage of the i-th variety, in kilograms ($0 \le A_i \le 10^4$), and C_i — the price of a kilogram of cabbage of this variety ($1 \le C_i \le 10^4$).

The sum M for all test cases is smaller than or equals 10^5, and the sum N for all test cases is smaller than or equals 10^5.

Output

The output file must contain T lines, and the i-th line must contain the answer to the i-th test case. The answer to a test is the maximum possible portion size, in kilograms.

The absolute or relative error of each answer must be smaller than or equal 10^{-9}.

Examples

input.txt	output.txt
1	2.777777777778
3 7 37	
3 3 2 3 1 2 3	
2 2	
1 6	
3 1	
2	8.5
2 3 17	1
1 2 1	
50 3	
0 2	
1 2 1	
1 1	
1 1	

Problem 9. Segments

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

Innokentiy's job consists in writing all sorts of algorithms for processing triangular meshes. He is struggling with his most recent task, which requires quick building of plane sections of triangulations. Innokentiy realized that the key challenge of the task can be reduced to the following one-dimensional problem. Given a set of segments on the coordinate line, answer the following type of queries: for a specified point, find all segments containing it. Help Innokentiy solve the problem.

To reflect the nature of real-life triangulations, the input data in this problem are generated randomly. Overall, N segments of different lengths are provided, with many short segments and fewer long segments. Each segment is defined by its center C and radius R, covering the interval from $C - R$ and $C + R$ inclusively. The center of each segment C is generated with uniform distribution between 0 and N. The radius R is generated using the following formula:

$$R = \min\left(\frac{1}{\text{rnd}}, N\right)$$

where rnd returns a random number from 0 to 1 with uniform distribution.

In addition, N point queries are provided. Each query is defined by a coordinate in the range from 0 and N, also generated randomly with a uniform distribution. For each query, find all segments containing the point, and print the number of these segments.

However, in the real-life problem, the answer must be returned instantly after receiving the point query. To model this situation, let's make the problem more dynamic.

First, let's assume that initially the line contains no segments. The input data lists $2N$ queries, half of them being segment addition queries, the other half being point queries. We will process these queries in the same order in which they are listed. When processing a point query, we consider only those segments which have already been added to the line by this moment.

Second, processing each point query changes all the remaining input data in the following way. Assume that all the segments containing point query P have numbers $k_0, k_1, k_2, \ldots, k_{t-1}$. Let $S = (k_0 + k_1 + \ldots + k_{t-1})$ be the sum of these numbers. Then the coordinates of all the not-yet-processed point queries changes in the following manner: If X is the old coordinate of the point, the new coordinate equals $(X + S) \bmod N$. In other words, they must be shifted cyclically by S, bringing all values within the range of 0 through N (excluding N). Similarly, the centers of all the segments not yet added to the line must be shifted exactly the same way.

The segments are numbered consecutively with numbers from 0 to $N - 1$ inclusive.

Input

The first line of the input file contains an integer N being the number of segments and the number of points ($3 \le N \le 10^5$). The remaining $2N$ lines describe the queries in the order of their processing: a total of exactly N queries for segment addition and N point queries.

A point query begins with the digit 1, followed by coordinate X of the point as a real number ($0 \le X < N$). A segment addition query begins with the digit 0, followed by two real numbers:

C being the center of the segment and R being the radius of the segment ($0 \le C < N, 1 \le R \le N$).

All real numbers are generated according to the rules described in the problem statement. Then they are rounded up to some number of digits after the decimal point. This number of digits may vary from one to six.

Output

The output file must contain N answers to point queries in the order of their processing, one answer per line. For each point query, print only the number of the found segments t. The found segments, or rather, their numbers, must be used for correcting the remaining queries in the manner described in the problem statement.

Example

input.txt	output.txt

5
0 3.1 5.0
1 2.2
0 0.8 1.3
0 0.5 1.3
0 3.6 1.2
1 3.8
1 2.0
1 3.8
0 3.6 1.6
1 2.5
	1
	2
	3
	2
	4

Example explanation

First, a segment $[-1.9; 8.1]$ is added. The immediately following point query belongs to this segment, hence the first answer equals 1. Since the segment's number is zero, no shift for the next queries is necessary.

Next, three segments are added with their centers and radii exactly as written in the file. Then the point query at $x = 3.8$ is processed, with an answer being two segments with numbers 0 and 3. Now a shift of 3 must be applied to all the remaining data.

The next point query in the file equals 2.0. However, after cyclic shift by 3 it becomes 0.0. The point $x = 0$ belongs to the segments 0, 1 and 2. Another cyclic shift of 3 must be added, which, considering the already performed shift, accumulates to a shift of 1.

The next point query is at $x = 4.8$ (after the shift) and belongs to the segments 0 and 3. Note that the point is exactly on the right end of the segment 3. After this query, the total cyclic shift equals 4.

Next, a segment with the center at $c = 2.6$ is added (having taken the cyclical shift into account). Finally, the last point query at $x = 1.5$ falls into the segments 0, 1, 2 and 4.

Problem 10. Civilization

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

The actions of Civilization V, a turn-based computer strategy, is situated in a hexagonal field — a two-dimensional grid of identical regular hexagon tiles.

Terrain and feature are specified for each tile. The terrain can be either a plain, a hill, or a mountain, and the feature can be grassland, forest or marsh. Rivers can flow along tile borders.

Units can move along the field tiles. A unit takes up exactly one tile. On each turn of the game, each unit is given a specific number of movement points (MPs). By default, moving to an adjacent tile from any tile of the field takes one MP. However, depending on the initial tile and the target tile, the cost of the move can increase:

- on moving from a plain tile to a hill tile the cost increases by 1 MP;
- on moving to a forest tile or marsh tile the cost increases by 1 MP;
- crossing a river takes up all MPs of the current turn.

Mountain tiles are unpassable — you cannot move into tile with mountain.

While the MPs value of a unit is positive, it can move between tiles on the current turn. If the cost of a desired move exceeds the available amount of the unit's MPs, it is still allowed to make the move. However, as soon as the unit's MP becomes less than or equals zero, the current turn ends for the unit. In this case, the unit can continue moving only on the next turn, when the number of MPs is restored to its initial value.

Find the shortest path from the starting tile to the target tile with a minimum number of turns. Both finished turn when all MPs were spent and the last turn which may still have a positive number of MPs are taken into account. There is **no** need to maximize the remaining MPs of the last turn.

Input

The first line of the input file contains two integers W and H — the width and the height of the field ($1 \leq W,\ H \leq 100$). Units cannot leave the game field. The game field includes tiles with coordinates $(x,\ y)$ where $0 \leq x < W$ and $0 \leq y < H$ and none other.

The explanation of the example contains an illustration showing a hexagonal field with numbered tiles. The field consists of hexagonal tiles forming H rows. Each row consists of W tiles, with all tiles oriented in such a manner that there are hexagon nodes at the top and bottom, and on the left and right there are edges shared with adjacent tiles in the row. Each consecutive row is shifted relative to the previous row by a half-tiles. The Ox axis goes along the top row of tiles, from left to right. The Oy axis is at 120 degrees relative to the Ox axis.

The following H lines contain descriptions of the game field tiles. Each line contains descriptions of W tiles, separated by spaces. A tile description consists of two symbols: the first symbol describes the terrain and can have one of the following values:

- '`p`' — plain
- '`h`' — hill
- '`m`' — mountain

and the second describes the feature:

- ‘g’ — grassland
- ‘f’ — forest
- ‘m’ — marsh

The next line contains a single integer R — the number of rivers in the field ($0 \le R \le 10^4$). The following R lines each contain 4 integers x_A, y_A, x_B and y_B — the coordinates of two adjacent tiles A and B ($0 \le x_A, x_B < W$, $0 \le y_A, y_B < H$), meaning there is a river flowing along their shared edge.

The line before the last one contains a single integer M ($1 \le M \le 10^5$) — the number of MPs of the unit available at the current, first turn. This is the MP value of the unit which will be restored on each new turn.

The last line contains 4 integers x_s, y_s, x_f and y_f ($0 \le x_s, x_f < W$, $0 \le y_s, y_f < H$) — the coordinates of the initial (x_s, y_s) and the target (x_f, y_f) tiles. It is guaranteed that these coordinates are not located in tiles with mountains, and that the starting tiles and the target tile are different.

Output

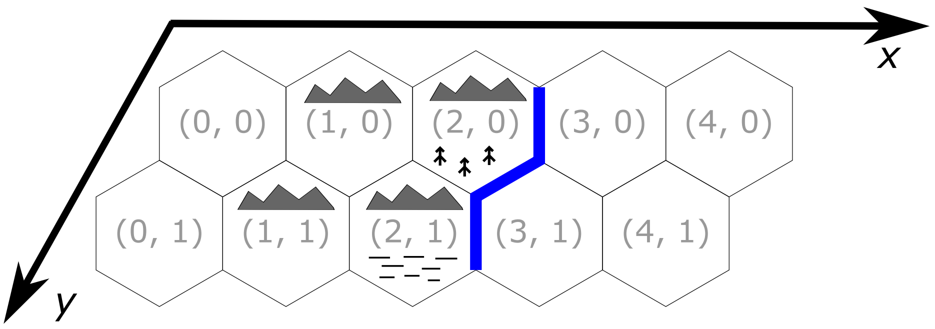
The first line of the output file must contain the phrase `Come this way`, if there is a path between the starting tile and the target tile, or the phrase `They shall not pass` otherwise.

If the path exists, the next line must contain the minimal number of turns necessary to reach the target tile.

The following line must contain N — the number of tiles in the path, including the starting and target tiles. The following N lines must contain the coordinates of the tiles, beginning with the starting tile and ending with the target tile. If there are several possible optimal paths from the starting tile to the target tile, print any one of them.

Examples

The field from the third example looks like this:



(The table with examples is provided on the next page)

input.txt	output.txt

`3 1`
`pg pm hf`
`0`
`7`
`0 0 2 0`
	`Come this way`
	`1`
	`3`
	`0 0`
	`1 0`
	`2 0`

`3 1`
`pf mg hm`
`0`
`7`
`0 0 2 0`
	`They shall not pass`

`5 2`
`pg hg hf pg pg`
`pg hg hm pg pg`
`3`
`2 0 3 0`
`2 0 3 1`
`2 1 3 1`
`4`
`0 1 4 0`
	`Come this way`
	`3`
	`6`
	`0 1`
	`0 0`
	`1 0`
	`2 0`
	`3 0`
	`4 0`

Example explanation

In the first example, the only way from the starting tile to the target tile requires 5 MPs:

- The move from $(0, 0)$ to $(1, 0)$ requires 2 MPs because of the marsh at $(1, 0)$.
- The move from $(1, 0)$ to $(2, 0)$ requires 3 MPs because of the forest and hill in the target tile.

Hence moving can end on the first turn.

In the second example, it is impossible to get to the target tile due to a mountain in the way.

The field from the third example is provided on the previous page of the problem statement. One of the options of moving to the target tile is as follows:

- The move from $(0, 1)$ to $(0, 0)$ requires 1 MP (base cost).
- The move from $(0, 0)$ to $(1, 0)$ requires 2 MPs due to a hill at $(1, 0)$.
- The move from $(1, 0)$ to $(2, 0)$ requires 2 MPs due to a forest at $(2, 0)$. There is no increase in the cost here since the tile $(1, 0)$ is also a hill. The first turn ends here.
- Due to negative number of MPs, to continue movement unit should start the second turn. The move from $(2, 0)$ to $(3, 0)$ ends the second turn because of the crossing the river.
- Third turn starts here and the move from $(3, 1)$ to $(4, 0)$ requires 1 MPs (base cost).

Problem 11. Ecliptic

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

The "Beamer", an unmanned spacecraft, is stationed in the Northern Hemisphere at the latitude of ϕ degrees and is ready to take off at any moment. It is planned that the "Beamer" be put into a solar orbit, so it must be launched at a moment when the launchpad is at the nearest to the ecliptic plane.

The planet rotates around its sun along a circular orbit. The ecliptic plane is the plane containing this orbit. The planet is a sphere rotating around its axis. The angle between the planet axis to the normal to the ecliptic plane is α degrees. The planet makes a full turn around its axis in 24 hours.

The control station is located on the planet's equator, at the same longitude with the "Beamer". It crosses the ecliptic plane every 12 hours. At *HH:MM:SS* the control station reports crossing the ecliptic plane, and afterwards, the control station and the "Beamer" are **separated from each other** by the ecliptic plane. Find the first moment in time after this report when the "Beamer" will be as close to the ecliptic plane as possible.

Input

The first line of the input file contains a single integer T — the number of tests ($1 \le T \le 5 \cdot 10^4$). Each of the following T **pairs** of lines contains a description of a consecutive test.

The first line of a test description contans two real numbers α and ϕ. The number α is the angle between the rotation axis of the planet and the perpendicular line to the ecliptic plane, set in degrees; the number ϕ is the latitute of the spacecraft location, set in degrees ($1.0 \le \alpha, \phi \le 89.0$). Both real numbers are set with no more than five digits after the decimal point.

The second line contains the time of the incoming message from the control station. The time is set in the 24-hour format *HH:MM:SS*, where *HH* is the number of hours from 0 to 23, *MM* is the number of minutes from 0 to 59, and *SS* — is the number of seconds from 0 to 59. Each of the three numbers contains strictly two digits, with possible leading zeroes.

Output

The output file must contain answers to the tests in the order of their appearance in the input file, one answer per line.

An answer to a test is the nearest moment in time when the spacecraft can be launched. Time must be printed in the same format as the time in the message from the control station. The printed time must differ from the accurate answer by no more than one second.

Example

input.txt	output.txt
2 23.44 15.0 12:00:00 20.0 16.11 23:00:00	14:32:41 02:30:04