Problem 1. Product

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

Given a list of numbers, determine whether one of these numbers equals the product of all the other numbers.

Input

The first line of the input file contains a single integer N — the number of elements in the initial list ($2 \le N \le 10^5$). The next line contains N space-separated numbers constituting the list. Each of these numbers is an integer not greater than 10^9 in absolute value.

Output

In the first line of the output file, print `Yes` or `No`, depending on whether some number from the list is equal to the product of all the other numbers — or not. If the answer is `Yes`, in the second line, print such a number. If several numbers satisfy the conditions of the problem, print any of them.

Example

input.txt	output.txt
3 2 4 2	Yes 4
2 2 3	No

Problem 2. Driving the Gnu

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

Adam Kazimirovicz is racing his car, the Gnu, at the speed of v, across the vast cathedral square. Suddenly, a priest appears in his way, trying to fool Adam. The priest is a segment of the length w, perpendicular to Adam's trajectory, with its center on the trajectory. The initial distance between the car and the priest is d.

Enter Ostap — he overrides Adam's panicked driving in an attempt to avoid the collision with the priest. For this purpose, he can accelerate the car at every moment in any direction. Note that at different moments in time, both the module and the direction of the acceleration can be changed. However, Ostap is not omnipotent; hence, the acceleration module at each moment in time cannot be greater than a.

Find the minimal possible value of a sufficient to avoid the collision.

Input

The first line of the input file contains an integer T — the number of test cases ($1 \le T \le 5 \cdot 10^4$).

The following T lines each contain three integers v, d and w — the initial velocity of the car, the initial distance from the Gnu to the priest, and the length of the segment ($1 \le v, d, w \le 1000$).

Output

The output file must contain T lines, the ith line must contain the answer to the ith test case. The answer is the minimal possible value a sufficient to avoid a collision.

The absolute or relative error of each answer must not be greater than 10^{-10}.

Example

input.txt	output.txt
1 10 3 1	10.510002209123385

Problem 3. Crazy minesweeper

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

In this problem, we will be playing a popular game, «Minesweeper».

The game takes place on a rectangular field with $H \times W$ cells. K mines are randomly placed in the cells of the field, and their locations are unknown to the player. The goal is to find the mines safely.

Initially, all cells are «closed». The player can «open» any cell in the field. If there is a mine in the opened cell, the game ends. Otherwise, the player is informed about the number of mines in the cells adjacent to that cell by their sides or corners. The game ends when precisely K cells remain closed in the field — i.e. the cells containing mines.

Often, the available information is insufficient to determine the precise location of the mines. In these cases, the player has to guess and tries to open a cell which may contain a mine. To minimize the effect of these dangerous situations, the player can make up to six mistakes by opening a cell with mine. On opening such cell, the player is informed about the mine, the cell remains closed, and the game continues. If you open a mine-containing cell for the seventh time, you will lose.

The game fields are generated randomly in the following manner. The jury defines the field size W by H, the number of mines K and the magic number S. After that, the following pseudorandom sequence of integers is generated:

$$T_0 = S$$

$$T_{i+1} = (48\,271 * T_i) \bmod 2\,147\,483\,647$$

Then the mine locations are obtained from this sequence. Its elements $T_1, T_2, T_3, \ldots$ are iterated in order. For each element T_i, the row index r_i and the column index c_i are obtained using formulas:

$$r_i = \left\lfloor \frac{T_i}{W} \right\rfloor \bmod H \qquad c_i = T_i \bmod W,$$

and the mine is set into the corresponding cell, if it is not yet there. The process of setting mines ends as soon as the number of mines becomes equals to K.

The rows and colums are numbered from zero.

Interaction Protocol

This is an interactive problem. Instead of working with file input-output, you will be working with a special program — an interactor. Interacting with this program is performed through the standard input-output streams.

Upon the start, the input stream of your program is given a line containing three integers H, W and K. In all tests, $H = 16$, $W = 30$, and number K is chosed randomly in range from 100 to 200 inclusive. The magic number S is not provided in the input, but jury chooses it randomly in range from 100 to 10 000 inclusive.

Next, your program must send opening queries to the standard output stream. Each query must consist of a single line with two numbers: the row index r ($0 \le r \le H - 1$) and column index c ($0 \le c \le W - 1$) of the cell which is opened.

When each query is sent, one of the following responses is possible:

- `Empty`, followed by an integer from 0 to 8 — you have opened an empty cell, and the number of mines in the adjacent cells is provided.
- `Boom` — you have attempted to open a cell with a mine and made one of the forgivable errors.
- `Lose` — you have stepped on a mine and lost.
- `Win` — you win. Yay!

The answers `Empty` and `Boom` mean that the game continues. After getting the answers `Lose` or `Win` your game must end without printing anything else into the standard output stream. You can open one cell several times. The number of queries must not exceed $2 \cdot W \cdot H$, otherwise the solution will get the Wrong Answer verdict. Make sure you print the line break symbol and flush the output stream buffer (the `flush` command of the programming language) after each printed command. Otherwise the solution can get the Timeout verdict.

Example

standard input stream	standard output stream
16 30 137 Boom Empty 2 Empty 0 Empty 2 Boom Empty 6 ... Win	0 0 0 2 15 29 14 27 15 26 4 15 ... 15 28

Example explanation

The fully uncovered field in the sample (which is also the first test) is:

```
X2223X20013X310011100123211XXX
24XX3X2001XXX2122X2111XXX22343
X4X32111235X43X5X43X34443X23X3
X3110113XX3X34XXXX33XXX334X5XX
110112X4X4322XX654X334XX2XXX5X
0001X22X23X323XX2X4X212222334X
1112211112XX1234323X20122101X3
X33X2221135533X3X222112XX2012X
2XX33XX22XXXX3X313X201X6X42121
234X445X335X442213X2013XXX2X10
X22X4XX22X33X3X11X22124X633232
X3325X4112X34X312221X3XX3X12XX
3X2X4X41124X4X323X113X534333X3
X2214XX22X3X312XX3203X5X4XX211
11014X53X22121223X102X4X4X4200
0001XX3X21001X101110112122X100
```

Problem 4. Error in code

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	5 seconds
Memory limit:	256 megabytes

A rookie programmer wrote a program in C++.

1. First, the program reads the incidence matrix of an unoriented graph for N nodes and writes it into an array of 32-bit integers g. After reading the value $g[u][v]$ equals one if the graph has an edge between the nodes u and v, or 9999, if there is no edge (for $1 \le u \ne v \le N$). The main diagonal in the matrix contains zeroes.
2. After reading the graph the following piece of code in the program is run:

```
for (int w = 1; w < N; w = w + 1) {
    for (int u = 1; u < N; u = u + 1) {
        for (int v = 1; v < N; v = v + 1) {
            g[u][v] = min(g[u][v], g[u][w] + g[w][v]);
        }
    }
}
```

3. After that, the resulting matrix is written into a file.

The programmmer launched his program on a set of graphs, and after some tedious waiting finally decided to take a look at the results. Which were, of course, disappointing. You guessed right — the programmer wanted to calculate the matrix of the shortest distances between all nodes in the graph. However, the program was flawed. The programmer can fix everything himself, but waiting for the program to recalculate the results takes too much precious time. We suggest that you write a program that takes the results of the unlucky programmer's code and returns a matrix of the shortest distances.

Input

The first line of the input file contains a single integer N ($1 \le N \le 2\,000$). The ith of the following N lines contains a sequence of N numbers, with the jth number equal to the value $g[i][j]$ after running the algorithm provided above ($0 \le g[i][j] \le 9999$).

Output

Print N lines. The ith line must contain a sequence of space-separated numbers, with the jth number equal to the length of the shortest path in the graph between the nodes i and j, or the number 9999, if there is no path between these two nodes.

Example

input.txt	output.txt
3 0 1 1 1 0 9999 1 9999 0	0 1 1 1 0 2 1 2 0

Problem 5. Birthday

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	4 seconds
Memory limit:	256 megabytes

Malvina wants to give Buratino a birthday present — an unoriented graph $G = (V, E)$. Buratino is turning k, and Malvina wants to reflect this date in the graph by splitting it into k parts, that is, by presenting the range of nodes of the graph V as k pairwise disjoint subranges V_1, V_2, ..., V_k. Naturally, in this partition, all of the subranges V_i must be non-empty.

To demonstrate the connection between the past k years, Malvina wants to split the graph in such a manner that for each of the edges uv, its end nodes u and v belonged to either the same or two adjacent subranges. Subranges are adjacent when V_i and V_{i+1} for $i = 1, 2, \ldots k - 1$.

Input

The first line of the input file contains two space-separated numbers N and k — the number of nodes in the graph ($1 \le N \le 2\,000$) and the number of parts ($1 \le k \le 2\,000$) that the graph must be divided into.

The graph is defined in an adjacency matrix G. The following N lines of the file each contain N symbols. The jth symbol of the ith line equals '`1`' if there is an edge between the nodes i and j, and '`0`' if there is no edge between these nodes. It is guaranteed that the matrix is symmetrical ($g_{i,j} = g_{j,i}$), and that that the matrix diagonal only contains zeroes ($g_{i,i} = 0$).

Output

In the first line of the output file, print "`Yep`", if the graph can be split into k parts in the desired manner. Next, print k lines. The ith line(i = 1, 2, ..., k) contains the description of the ith part: first, print the number of nodes in V_i, next print the numbers of the nodes belonging to V_i, separated by spaces, numbered in the same order as in the input data. If there are several ways of splitting the graph, print any one of them.

If it is impossible to split the graph into k parts in the desired way, print "`Nope`".

Examples

input.txt	output.txt
4 3 0110 1010 1101 0010	Yep 1 2 2 1 3 1 4
3 3 011 101 110	Nope

Problem 6. Gas penalties

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	3 seconds
Memory limit:	256 megabytes

Pasha has entered a desert navigation contest, «World Sensei of Orienteering 2018» (WSO 2018). The desert can be viewed as an endless plane. The desert has been prepared for the contest, with n checkpoints numbered, in numerical order, from 1 to n, the ith checkpoint having the coordinates (x_i, y_i).

The contest rules are the following:

1. All participants are sent to the start checkpoint with the number s ($1 \leqslant s \leqslant n$).
2. Each participant is given a truck with a gas tank capacity of v and fuel efficiency of one liter per length unit (meaning that covering a unit of distance takes a the same amount of fuel in liters). Initially, the tank is empty.
3. The participants can only fill the tank at checkpoints. Buying one liter of fuel at ith checkpoint costs p_i penalty points. It is not required to buy integer amount of fuel, you can buy non-integer amount if you like.
4. The participants are informed about the number of the final checkpoint f ($1 \leqslant f \leqslant n$, $f \neq s$). The goal of the participants is to reach the final checkpoint f from the initial checkpoint s with as few penalty points as possible.

Pasha is in the dark on which checkpoints s and f will be chosen by the organizing committee, so he needs to know the mean value of the number of penalty points needed. Being a true professional, he always choses the route which needs the minimal number of penalty points. All possible variants of s and f should be considered equiprobable.

Write a program that will help Pasha determine the mean value.

Input

The first line of the input data contains two integers n and v — the number of checkpoints and gas tank capacity, respectively ($2 \leqslant n \leqslant 200$, $1 \leqslant v \leqslant 10^5$).

Each of the next n lines contains a checkpoint with three integers x_i, y_i and p_i — the coordinates of the checkpoint and the number of penalty points for a liter of fuel at it, respectively ($-10^4 \leqslant x_i, y_i \leqslant 10^4$, $1 \leqslant p_i \leqslant 10^6$). No two checkpoints are the same.

It is guaranteed that the capacity of the gas tank is enough to reach any checkpoint from any other checkpoint, possibly refuelling on the way.

Output

The only line of the output data must contain a single real number — the number of penalty points required on average. The answer will be considered correct when its absolute or relative error is less than 10^{-6}.

Example

input.txt	output.txt
5 3 0 0 3 0 2 2 1 1 5 3 1 2 2 2 4	6.634062043
3 100000 0 0 1000 1 1 1 2 0 1000	943.751850623645282

Example explanation

There are only 20 variants of the start and final checkpoints in the first example. In all but four variants, the optimal route consists in buying the necessary volume of gas at the first checkpoint and proceeding directly to the end point.

In the case of driving from (0, 2) to (3, 1) or back, it is impossible to take a direct drive because of the insufficient tank capacity. The optimal variant is to fill the tank maximally at the starting point and driving through the checkpoint (2, 2), adding some fuel there. The total penalty in this case will be $2 \cdot 3 + 4 \cdot (\sqrt{2} - 1)$.

Similarly, in case of going from (0, 0) to (3, 1) and back, the optimal solution is to drive through the checkpoint (1, 1) for a refill.

In the second example it is sometimes worth to drive through the intermediate checkpoint to buy fuel there, even though gas tank is large enough to sustain a direct route.

Problem 7. Level check

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	6 seconds
Memory limit:	256 megabytes

Vasya still likes to play computer games, and he is still working as a game tester. Little has changed since our last meeting: he has been brooding about learning to code, but hasn't made any decisive moves yet. As the result, he is still cramped into a tiny room, testing the same browser game.

A while ago, gameplay designers finally decided that levels needed nonlinear gameplay. Now the *base level map* of the game is a set of N passable cells in an endless rectangular grid. The player, weapons and monsters are always located in the passable cells. In one game turn, the player can move from his current cell to any passable cell adjacent to it. Two cells are considered adjacent if they share a common side.

Unfortunately, for Vasya, this groundbreaking policy change is nothing but headache. He is currently busy testing the first level of the game. The problem is, in the beginning of the game, the player has no weapon, so any battle with a monster is doomed to end in a stupid pointless *failure*. Being a game tester, Vasya must ensure that the player is guaranteed to find at least some weapon before his first encounter with a monster. The assumed IQ of the target auditory is lurking near rock bottom, as usual. Hence if there is even an infinitesimal chance to fail, the future players of the game will do that, guaranteed.

Unfortunately, the designers take an unhealthy amount of perfectionism regarding the first level. Even though the base map of the level has already been approved and fixed, every day, Vasya keeps receiving a bunch of new variants of object layout for this level. *Object layout* defines which cells of the level contain monsters, weapons, and the initial position of the player. For each object layout, Vasya must determine whether it is prone to the above mentioned pointless failure or not. Help Vasya: write a program which will quickly analyze the layouts, while Vasya is busy enjoying the intricate gameplay of the second PoE. Mind that when a player moves to a cell with weapon, the weapon is automatically taken, and when the player moves to a cell with a monster, the fight is inevitable.

Input

The first line of the input file contains a single integer N — the number of passable cells in the base level map ($1 \le N \le 3 \cdot 10^5$). The following N lines describe these cells: each line contains two integers x and y — the cell coordinates ($0 \le x, y \le 10^6$). It is guaranteed that all these cells are different.

The next line contains a single integer T — the number of object layouts to be analyzed ($1 \le T \le 10^5$). It is followed by T blocks, with each block describing one object layout.

A block begins with a line containing two integers: M — the number of monsters and W — the number of weapons ($0 \le M, W \le 10^5$). The next line contains the coordinates of the cell where the player is initially located. The following M lines contain the coordinates of cells with monsters, and finally, the last W lines of the block provide the coordinates of cells containing weapons. For each of these $(M + W + 1)$ cells two integers are provided: x and y; it is guaranteed that all these cells are passable and different.

The total number of monsters over all object layouts does not exceed 10^5. Similarly, the total

number of weapons over all object layouts does not exceed 10^5.

Output

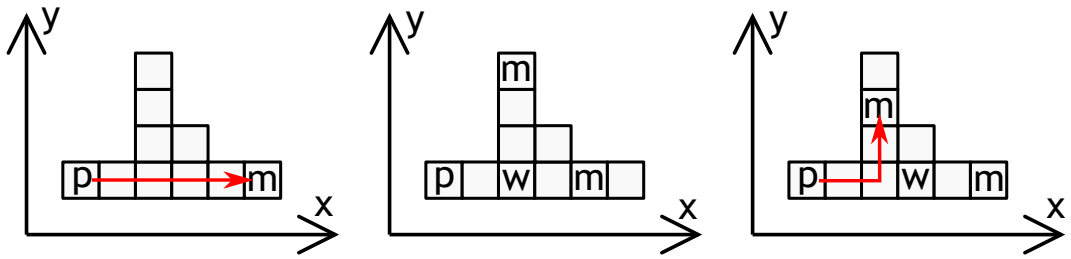
For each object layout print an answer to the problem in a separate line. Answers must be printed in the same order in which the object layouts are defined in the input file. If the player can fail, print the word `fail`, otherwise print the word `ok`.

Example

input.txt	output.txt
10 1 1 2 1 3 1 4 1 5 1 6 1 4 2 3 2 3 3 3 4 3 1 0 1 1 6 1 2 1 1 1 5 1 3 4 3 1 2 1 1 1 3 3 6 1 4 1	fail ok fail

Illustration

Three object layouts from the example (`p` — player, `m` — monster, `w` — weapon):



Problem 8. Intersection Graph

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	3 seconds
Memory limit:	256 megabytes

Innokentiy is still very fond of writing geometrical algorithms, but he has switched from triangular meshes to solid bodies. He wants to write an algorithm for building an intersection graph of two bodies, necessary for running, for instance, a boolean operation with these bodies. Note that only the boundaries of the bodies are considered during the building of the graph, and the interior volumes of the bodies are not considered here. Innokentiy knows that building an intersection graph of two arbitrary bodies is a challenging task in the CAD area. He has decided to take it easy: first, he is going to learn to build an intersection graph of two boxes, A and B.

A *box* (usually called axis-aligned box) is a rectangular parallelepiped with sides parallel to the coordinate axes. The boundary of the box consists of *topological elements* of three types:

- `V` — a vertex, which is, geometrically speaking, a point;
- `E` — an edge, which is a line segment;
- `F` — a face, which is a rectangle;

Each box has 8 vertices, 12 edges, and 6 faces.

For the purpose of this problem, let edges and faces be represented by the corresponding point sets **without boundary points**. Hence an edge is a line segment which does not contain its end points, and a face is a rectangle which does not contain its corners and sides. Based on this definition, the topological elements have no common points and together form precisely the whole boundary of the box.

An *intersection graph* consists of interrelated *intersection elements* defined in the following manner. Consider all possible pairs: a topological element u of the box A and a topological element v of the box B (a total of 676 pairs). For each such pair (u, v), find their intersection as intersection of two sets. If this set intersection is non-empty, then it defines the intersection element, which can be of one of the three following types:

- `p` — point: a single point;
- `c` — curve: a set of points forming a line segment;
- `s` — surface: a set of points forming a rectangle;

The intersection graph consists of all intersection elements acquired in this manner.

The intersection elements are interconnected. For each intersection curve, two intersection points located on its ends are saved. For each intersection point, a list of intersection curves incident to that point is saved. An intersection curve is considered incident to its end points and not incident to any other intersection points. For an intersection surface, the set of intersection curves located on its boundary (and thus bounding it) is saved.

Input

The first line of the input file contains a single integer T — the number of test cases ($1 \le T \le 5\,000$). It is followed by T blocks.

Each block consists of two lines: the first line defines the box A, while the second line defines the box B. A box is described by six integers x_1, y_1, z_1, x_2, y_2, z_2, which do not exceed 10^3 by absolute

value. The numbers x_1, y_1, z_1 define the coordinates of the «minimal» vertex, and the numbers x_2, y_2, z_2 define the coordinates of the «maximal» vertex ($x_1 < x_2$, $y_1 < y_2$, $z_1 < z_2$).

Output

The output file must contain T intersection graphs — the answers to the test cases from the input file. After each intersection graph, print `===` (three equality symbols) in a separate line. All numbers in the output file must be integers.

The description of an intersection graph contains a lot of links between intersection elements and topological elements. For the purpose of describing these links, you must assign numbers (or *indices*) to all elements. All topological elements of the box A must be numbered by integers from 1 to 26 in any order. Similarly, all topological elements of the box B must be numbered by integers from 1 to 26 in any order. Furthermore, all intersection elements must be numbered from 1 to K in the order of their appearance in the output file, where K is the total number of intersection elements.

The first line of intersection graph description must contain three numbers: P — the number of intersection points, C — the number of intersection curves, and S — the number of intersection surfaces (naturally, $P + C + S = K$). After that the intersection elements must be described in the following order: first P intersection points, then C intersection curves, and last S intersection surfaces (one intersection element per line).

The description of each intersection element must begin with its index, follower by a three-letter code devised according to the rule:

1. The first letter — type of intersection element: point `p`, curve `c` or surface `s`.
2. The second letter — type of topological element u: vertex `V`, edge `E` or face `F`.
3. The third letter — type of topological element v.

Here u and v are topological elements from the boxes A and B, respectively, from which the intersection element in question was generated. Next, print two integers: the indices of these topological elements u and v in that order. Finally, depending on the type of the intersection element, print additional information.

For an intersection point, additionally provide: three coordinates of its location, the number of incident intersection curves, and the list of indices of these curves. The indices of incident curves may be printed in any order.

For an intersection curve, additionally provide the indices of the two intersection points located on its ends. These two numbers may be printed in any order.

For an intersection surface, additionally provide: the number of intersection curves on its boundary and the list of their indices. The indices of these bounding curves may be printed in any order.

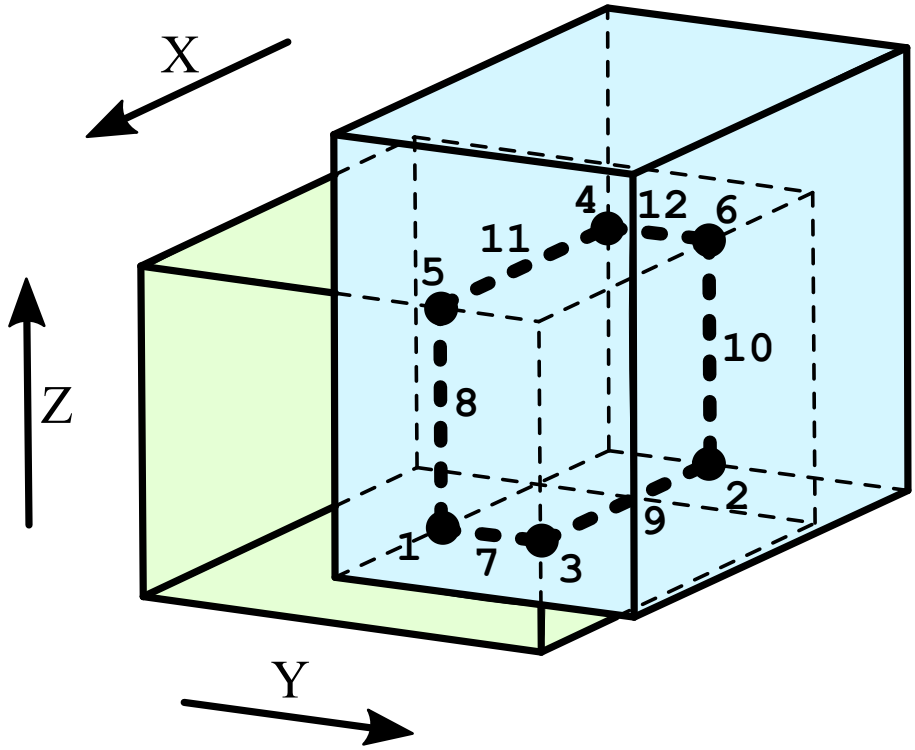
It is **not** necessary to separate the numbers in the file strictly by a single space: you can use several spaces in a row, like in the example below.

Example

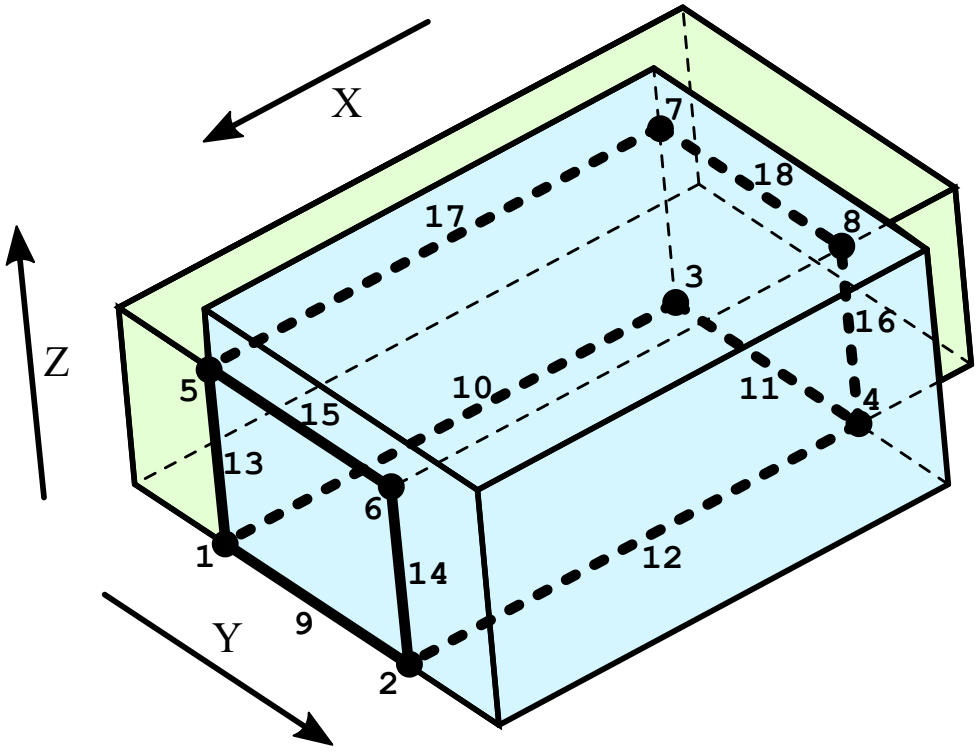
input.txt	output.txt
2 0 0 0 5 4 3 2 3 1 7 6 5 0 2 6 10 8 9 2 4 6 10 10 10	6 6 0 1 pFE 14 2 5 3 1 2 7 8 2 pFE 16 4 2 4 1 2 9 10 3 pEF 17 5 5 4 1 2 7 9 4 pFE 22 10 2 3 3 2 11 12 5 pEF 23 11 5 3 3 2 8 11 6 pEF 25 13 2 4 3 2 10 12 7 cFF 14 5 1 3 8 cFF 14 11 1 5 9 cFF 16 5 2 3 10 cFF 16 13 2 6 11 cFF 22 11 4 5 12 cFF 22 13 4 6 === 8 10 2 1 pEV 6 3 10 4 6 3 9 10 13 2 pVE 9 6 10 8 6 3 9 12 14 3 pFV 5 1 2 4 6 2 10 11 4 pEE 8 4 2 8 6 3 11 12 16 5 pEE 23 12 10 4 9 3 13 15 17 6 pVF 26 14 10 8 9 2 14 15 7 pFE 22 10 2 4 9 2 17 18 8 pEF 25 13 2 8 9 2 16 18 9 cEE 6 6 1 2 10 cFE 5 2 1 3 11 cFE 5 4 3 4 12 cEF 8 5 2 4 13 cFE 14 12 1 5 14 cEF 17 14 2 6 15 cEF 23 14 5 6 16 cFF 16 13 4 8 17 cFF 22 11 5 7 18 cFF 22 13 7 8 19 sFF 5 5 4 9 10 11 12 20 sFF 14 14 4 9 13 14 15 ===

Illustration

In the first test case, the boxes poke though each other with a corner. The intersection graph is shown with a bold line. It consists of six points and six curves, obscured to the observer by the box B. The indices of the intersection elements are labeled.



In the second test case, the boxes share the lower and the front faces, resulting in the intersection surfaces 19 and 20, respectively (they are **not** shown in the picture). All remaining elements are shown in bold, and their indices are provided.



Problem 9. In search of the chair

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	5 seconds
Memory limit:	256 megabytes

Kisa and Osya, in their search for the last, twelfth chair, found themselves on the planet Plop. They know their precise coordinates and the coordinates of the chair, and they want to get that chair, at last! But alas — they can move anywhere on the planet surface, with the exception of several forbidden circular areas — anyone who sets foot there is locked in a cage and forced to sing the "Mommy, what do I do now" song. This may seem like a minor nuisance first, however, our heroes will not tolerate any delays — and their singing may impress the Master, and he could end up keeping them locked up forever... For this reason, they need to avoid these areas. Your task is to help them find the shortest path from their current location to the chair, avoiding any forbidden areas.

Input

The first line of the input file contains two numbers: R — the planet radius($1000 \le R \le 10000$) and N — the number of the forbidden round areas ($0 \le N \le 20$). Next come N lines, each containing three numbers: ϕ_i, ψ_i — the latitude and longitude of the area center, respectively, and r_i — its radius ($1 \le r_i \le R/2$). The last two lines contain the coordinates ϕ_s, ψ_s and ϕ_t, ψ_t — the latitude and longitude of the starting point and the finish point, respectively.

All numbers in the input data are integers. Latitudes and longitudes are set in degrees. The latitude lies in the range of -90 degrees to 90 degrees, and the longitude lies in the range from -180 to 180 degrees.

A forbidden area with a radius of r contains all points reachable from the area center via a path shorter than r along planet surface.

It is guaranteed that the surface distance from the starting point to any point in any forbidden area, as well as to the finish point, is less than or equals $\frac{3}{2}R$. Forbidden areas can overlap. Centers of forbidden areas are all different. It is guaranteed that the starting and finish point do not fall into any forbidden areas; moreover, the minimal distance from these points to any forbidden area is at least $R/1000$. It is also guaranteed that changing any feature of any forbidden area by no more than 10^{-3} will not affect the mutual position of the areas (i.e. if they overlapped, they will stay overlapped, and vice versa).

Output

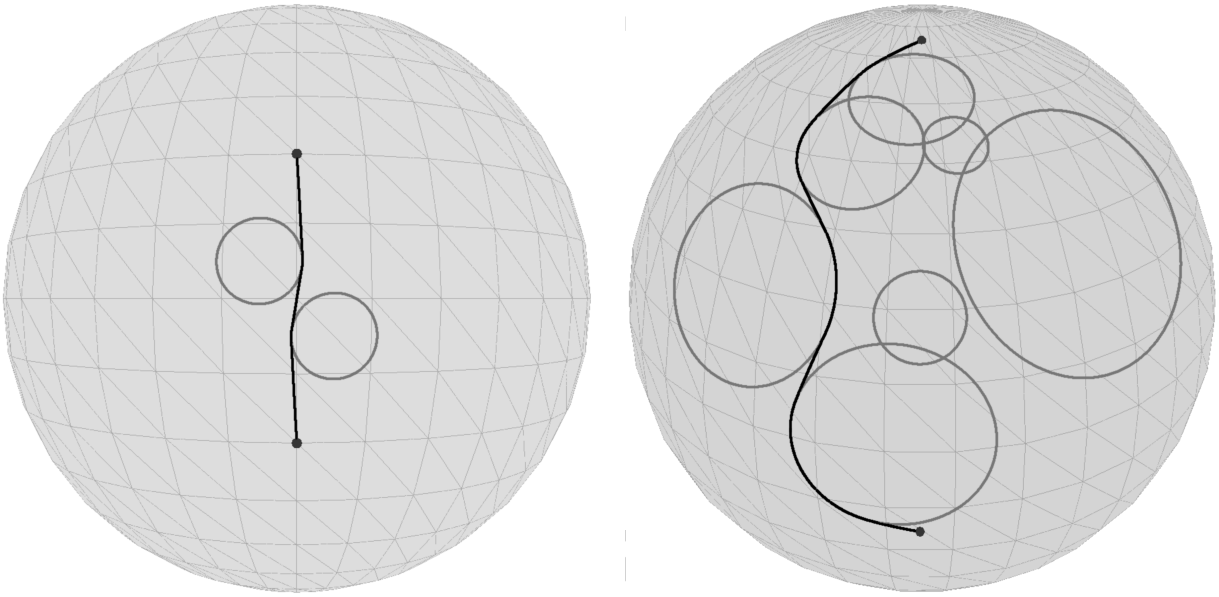
In the output file, print a single real number — the length of the shortest path. The absolute or relative error of the answer must not be greater than 10^{-8}. If there is no such path, print -1.

Example

input.txt	output.txt
1000 2 5 5 100 -5 -5 100 -20 0 20 0	700.830526321397656
1000 7 11 49 253 30 75 253 52 59 158 62 48 157 37 14 348 29 45 107 53 37 79 -5 45 75 45	1715.42276690714903

Illustration

Locations of the forbidden areas and the optimal paths on the planet surface for the first and the second tests respectively are shown below.



Problem 10. Subsequences

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	5 seconds
	8 seconds (for Java)
Memory limit:	512 megabytes

In the Datastring Research Corporation, the string s is called *good*, if the number of its various subsequences is even.

The string t is called a subsequence of the string s, if it can be derived from s by crossing out a certain set of symbols. The string s and the empty string are also considered to be subsequences of s. In a string of the length l, there is a total of 2^l subsequences, however, some of them may coincide. For instance, a three-letter string *abb* has only 6 distinct subsequences: empty, *a*, *b*, *ab*, *bb*, *abb*.

The string s was lying on the String Researcher's desk, and he was trying to figure out whether it was a good one. He did realize that counting the number of distinct subsequences is a tedious task, so, instead of tackling the task right away, he went on a coffee break — with donuts.

Upon his return, the String Researcher discovered that someone had made an $N - 1$ cut in the string s, and it fell apart into N non-empty substrings $s_1, s_2, \ldots, s_N$ scattered all over the table. And he absolutely does not remember the initial string s. However, he is curious about how many permutations $p = (p_1, \ldots, p_N)$ exist, such that the recovered string $s_{p_1} s_{p_2} \ldots s_{p_N}$ is good. Note that there is a total of $N!$ permutations (the factorial of N), and all these permutations are considered different, even if some of the substrings s_i match.

Input

The first line of the output file must contain a single integer N — the number of substrings ($2 \le N \le 20$). The ith of the following N lines contains the substring s_i.

All of the s_i strings are non-empty and consist exclusively of lower-case Latin letters. It is guaranteed that the sum of lengths of all the lines s_i is equal to or less than 10^5.

Output

In the only line of the output file, print a single integer number: the number of permutations such that the concatenation of the strings s_i in the order of permuation is a good one.

Example

input.txt	output.txt
3 a a baa	4

Example explanation

The string $a + a + baa$ has 14 subsequences, the string $a + baa + a$ has 13 subsequences, and the string $baa + a + a$ has 10 subsequences. Each of these lines can be obtained with the aid of only two permutations, since there are two occurrences of the substring a.

Problem 11. Office

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

There are several rooms in the office of the company N*, located along a long corridor. There are several workspaces in each room. There are lots or employees in N*, and every now and then they have to go either to one end of the corridor, or to its another end, running errands: that's where the most important places are, production-wise.

At some point, the CEO's decided it was time to optimize the arrangement and hired consultants to figure out the best way to place the employees in the workspace. First, the consultants measured the length of the corridor, which they found to be equal to L, and suggested the following criterion: the placement is considered optimal if the total distance the employees travel along the coridor in one day is minimal. If an employee works in a room located at a distance p from the beginning of the corridor, then, having walked to the beginning of the corridor an back, he has covered the distance $2p$, and a single run to the end of the corridor and back equals $2(L-p)$. Further analysis revealed the number of runs to the beginning and to the end of the corridor per day for each employee.

Now, when all data are available, you must calculate the optimal placement of all employees. Naturally, the number of employees who can be placed in each room is limited by its workplace capacity.

Input

The first line of the input file contains three integers: N — the number of rooms($1 \le N \le 10^5$), M — the number of employees ($1 \le M \le 10^5$), L — the length of the corridor ($2 \le L \le 10^8$).

Each of the following N lines contains two integers describing the ith room: P_i — the distance from the beginning of the corridor to the room ($0 < P_i < L$), C_i — the workplace capacity of the room ($1 \le C_i \le 10^5$).

Each of the following M lines contains two integers A_i and B_i, describing the ith employee — the number of times per day he or she runs from the room to the beginning and the end of the corridor, respectively ($0 \le A_i, B_i \le 10^5$).

Total workplace number does not exceed 10^6. It is guaranteed that there is enough workplaces for all employees.

Output

In the first line, print an integer — the total distance the employees run per day in case of the optimal placement. The following N lines must contain information about such placement. In the ith of these lines, first print an integer S_i — the number of employees who will work in the ith room. Next, in the same line print S_i integers — the numbers of these employees in arbitrary order. If several solutions are possible, print any one of them. Employees are numbered in the order of their appearance in the input file, beginning from one.

Example

input.txt	output.txt
4 9 5 1 2 2 3 3 4 4 2 3 0 3 1 3 1 0 2 1 2 1 2 1 1 2 2 3 3	128 2 1 3 2 2 9 3 6 7 8 2 4 5

Problem 12. Recursive circuit

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	3 seconds
Memory limit:	256 megabytes

In the layout of a recursive microcircuit, there is a total of N contact points, with some pairs of contacts connected directly with conductive paths. In addition, there is a total of S sub-circuits within the circuit, each being an exact copy of the circuit in question.

There are three types of contacts in the circuit layout:

1. Input contacts of the circuit (K contacts). These are the only contacts connecting the circuit to external conductive paths.
2. Input contacts of the nested sub-circuits ($S \cdot K$ contacts).
3. Auxillary contacts.

All these contacts can be connected with each other by conductive paths without any restrictions.

Signals travel along conductive paths. When a signal reaches a contact, it can follow any of the paths connected to this contact. If an external signal reaches an input contact of the sub-circuit, it can enter the sub-circuit and travel further along its paths. If an internal signal reaches an input contact of the sub-circuit, it can leave the sub-circuit and travel outside the circuit (if there are paths outside, and if the external circuit is itself a sub-circuit of another circuit).

Consider the most external circuit, with nothing else on the outside. Determine whether two specified input contacts of the circuit are connected with paths. The contacts are connected if a signal can reach one contact from the other, possibly entering a number of various sub-circuits a finite number of times.

Apart from the fact of being connected, find out how deep the signal must go into the sub-circuits to reach one contact from the other. The external circuit has a nesting depth of 0; for its sub-circuits, the nesting depth is 1, and their sub-circuits, in turn, have a nesting depth of 2, etc. For an arbitrary path of the signal, the critical depth can be defined, which is the maximum nesting level among all circuits, whose conductive paths are travelled by the signal.

Determine the minimal value of the critical depth for the path between two given input contacts of the external circuit.

Input

The first line contains four integers: N — the number of contacts in the circuit layout, K — the number of input contacts of the circuit, S — the number of sub-circuits in the circuit, M — the number of conductive paths in the circuit layout ($1 \le K \le 100\,000$, $0 \le S \le 1\,000$, $K(S+1) \le N \le 100\,000$, $0 \le M \le 100\,000$).

The following M lines define the conductive paths in the circuit layout. Each path is defined by two integers a and b — the numbers of contacts in the layout directly connected by the path ($1 \le a \ne b \le N$).

The contacts in the circuit layout are numbered in numerical order from 1 to N. The input contacts are numbered from 1 to K. The input contacts of the tth sub-circuit are numbered from $tK+1$ to $tK+K$ (for $1 \le t \le S$). The jth input contact on the tth circuit layout is the $(tK+j)$th contact on the external circuit layout. The remaining contacts, if such exist, are auxillary.

The following line contains an integer Q — the number of queries ($1 \le Q \le 100\,000$). Each of the remaining Q lines contain one query that needs an answer. Each query is defined by two integers u and v — the numbers of the input contacts of the external circuit ($1 \le u \ne v \le K$).

Output

In the output file, print Q integers, one number per line. Every rth number must be an answer to the rth query: the nesting depth necessary to get from one of the input contacts to another. If there is no way to reach the other input contact, print the number -1 instead of the depth value.

Example

input.txt	output.txt
16 5 2 7 1 16 2 16 6 16 7 11 3 12 4 13 5 15 4 1 2 2 3 3 4 4 5	0 1 2 -1

Illustration