Problem A. The Smart Bomb

Input file:	bomb.in
Output file:	bomb.out
Time limit:	1 second
Memory limit:	64 megabytes

The military scientists of Flatland have recently developed the new smart bomb. The main feature of the bomb is that its explosion power can be regulated after the production.

After the bomb was added to the arsenal, the generals have decided to run a series of test explosions. The equipment department has provided them with three bombs to explode. Since bureaucracy is a significant problem in the modern Flatland, the generals have decided to make all three explosions in one day. They selected the points where the bombs must be placed, and started to choose the destructing power of the bombs to be exploded.

Of course, the generals want to explode as powerful bombs as possible. On the other hand, to analyze the results of the experiment, the craters of the explosions must not intersect. Each crater is a circle, its radius is proportional to the power of the bomb exploded.

Now the generals want to know what is the maximal total power of the bombs they can explode. Help them to find that out! The total power of the bombs is the sum of their powers.

Input

The input file contains three lines, each contains two integer numbers — the coordinates of the location where the corresponding bomb must be exploded. Coordinates do not exceed 10^9 by their absolute value. Points do not belong to the same line.

Output

Output three real numbers — for each bomb output the desired radius of the crater after its explosion. The sum of the radii must be maximal possible, but craters must not intersect (although they may touch each other). Your answer must be accurate up to 10^{-4}.

Example

bomb.in	bomb.out
0 0 4 0 4 3	3.000000000 1.000000000 2.000000000

Problem B. I Just Called...

Input file:	`called.in`
Output file:	`called.out`
Time limit:	1 second
Memory limit:	64 megabytes

You are hired by *CBOOPS* company that is planning to create the new billing system for recently coming to the market company *MegaHrun*.

The country is divided to m regions that are united to n superregions. There are t towns in the country, each of them belongs to some region.

Phones numbers in the country consist of d digits. Each town has its own collection of phone *codes*. No code within the country is the prefix of another code. The code consists of two parts: the region code and the town code. For example, Gatchina in Leningrad region has a code `81362`, here `813` is Leningrad region code and `62` is the town code. Each region has exactly one code, while the town may have several codes. For example, St Petersburg region has region code `812` and St Petersburg itself has town codes `1`, `2`, `3`, and `5`. For some regions the town code may be empty, for example, Moscow region has code `095` and Moscow itself has no additional town codes. In this case we say that the town has 0 town codes.

Each call is characterized by the region from which it was made and the town it was made to.

There are four types of calls depending on the region from which it was made. The call can be made:

1. from home region,
2. from home network within home superregion,
3. from home network outside home superregion,
4. from alien network.

The network of *MegaHrun* exists in several regions of the country. One of these regions is *home* for the subscriber we are billing. A call from this region is of the first type. Calls from other regions covered by the network in this superregion are of the second type. Calls from covered by the network regions of other superregion are of the third type. Calls from regions not covered by the network are of the fourth type.

In turn, depending on the destination, the call can be local, regional, interregional, or long distance.

The call is local if it is made to the same town that the caller is. The call is regional if it is made to the same region. The call is interregional, if it is made to another region, covered by the network. In the other case the call is long distance.

Your task is given the tariff options for all sixteen types of calls, and the descriptions of the calls, calculate the total cost of the calls. We will only consider outgoing calls.

Input

The first line of the input file contains t, m, n, and d ($1 \le t \le 10\,000$, $1 \le m \le 200$, $1 \le n \le 20$, $2 \le d \le 1000$). The following m lines describe regions, each region is described with an integer number s_i — the superregion it belongs to, and the sequence of at most $d-1$ digits — the region code.

After that t blocks follow, each describing one town. The first line of the description contains r_i — the region this town belongs to, and p_i — the number of town codes for this town ($0 \le p_i \le 100$). If $p_i > 0$, the second line contains codes of the town, separated by spaces ($p_i = 0$ means that the town has no additional code to region code). The total length of the region code and the town code does not exceed $d-1$.

The next line contains h — the number of the home region of the number to bill, and z — the number of regions covered by the *MegaHrun* network. The next line contains z integer numbers — the numbers of regions covered. The home region is guaranteed to be covered.

The following four lines contain four integer numbers each — the cost per minute of the local, regional, interregional, and long distance call for calls from home, home within superregion, home outside superregion, and alien network respectively (costs are positive and do not exceed 100 000).

The next line contains c — the number of calls made ($1 \le c \le 10\,000$). Each of the following c lines contains an integer number t_i — the town from which the call was made, d digits — the phone number of the call, and l_i — the length of the call in minutes ($1 \le l_i \le 1000$). If the phone number does not belong to any of the towns, it is considered wrong and costs nothing.

The size of the input file does not exceed 512 kilobytes.

Output

Output the total cost of all calls.

Example

called.in	called.out
6 4 2 10 1 812 1 813 2 095 2 096 1 4 1 2 3 5 1 1 4 2 1 62 2 2 6133 614 3 0 4 2 11 12 1 2 1 3 10 20 30 40 20 40 50 60 30 30 30 30 60 70 100 120 13 1 8121234567 10 1 8136255555 1 1 8136133333 5 1 8136139999 1 1 0953456789 5 1 0961234567 5 1 0969876543 4 2 8121234567 20 2 8136134567 5 5 8121234567 10 5 0953456789 5 6 0957654321 4 6 8121234567 20	3940

Problem C. Order-Preserving Codes

Input file:	codes.in
Output file:	codes.out
Time limit:	1 second
Memory limit:	64 megabytes

Binary code is a mapping of characters of some alphabet to the set of finite length bit sequences. For example, standard ASCII code is a fixed length code, where each character is encoded using 8 bits.

Variable length codes are often used to compress texts taking into account the frequencies of occurence of different characters. Characters that occur more often get shorter codes, while characters occuring less often — longer ones.

To ensure unique decoding of variable length codes so called *prefix codes* are usually used. In a prefix code no code sequence is a proper prefix of another sequence. Prefix code can be easily decoded scanning the encoded sequence from left to right, since no code is the prefix of another, one always knows where the code for the current character ends and the new character starts.

Among prefix codes, the optimal code is known, so called Huffman code. It provides the shortest possible length of the text among all prefix codes that separatly encode each character with an integer number of bits.

However, as many other codes, Huffman code does not preserve character order. That is, Huffman codes for lexicographically ordered characters are not necessarily lexicographicaly ordered.

In this problem you are asked to develop a prefix code that would be optimal for the given text among all order-preserving prefix codes. Code is called order-preserving if for any two characters the code sequence for the character that goes earlier in the alphabet is lexicographically smaller.

Since text itself is not essential for finding the code, only the number of occurences of each character is important, only this data is given.

Input

The first line of the input file contains n — the number of characters in the alphabet ($2 \le n \le 2000$). The next line contains n integer numbers — the number of occurences of the characters in the text for which the code must be developed (numbers are positive and do not exceed 10^9). Characters are described in the alphabetical order.

Output

Output n bit sequences, one on a line — the optimal order-preserving prefix code for the described text.

Example

codes.in	codes.out
5 1 8 2 3 1	00 01 10 110 111

Problem D. More Divisors

Input file:	divisors.in
Output file:	divisors.out
Time limit:	1 second
Memory limit:	64 megabytes

Everybody knows that we use decimal notation, i.e. the base of our notation is 10. Historians say that it is so because men have ten fingers. Maybe they are right. However, this is often not very convenient, ten has only four divisors — 1, 2, 5 and 10. Thus, fractions like 1/3, 1/4 or 1/6 have inconvenient decimal representation. In this sense the notation with base 12, 24, or even 60 would be much more convenient.

The main reason for it is that the number of divisors of these numbers is much greater — 6, 8 and 12 respectively. A good quiestion is: what is the number not exceeding n that has the greatest possible number of divisors? This is the question you have to answer.

Input

The input file n ($1 \le n \le 10^{16}$).

Output

Output positive integer number that does not exceed n and has the greatest possible number of divisors. If there are several such numbers, output the smallest one.

Example

divisors.in	divisors.out
10	6
20	12
100	60

Problem E. Long Dominoes

Input file:	dominoes.in
Output file:	dominoes.out
Time limit:	1 second
Memory limit:	64 megabytes

Find the number of ways to tile an $m \times n$ rectangle with long dominoes — 3×1 rectangles.

Each domino must be completely within the rectangle, dominoes must not overlap (of course, they may touch each other), each point of the rectangle must be covered.

Input

The input file contains m and n ($1 \le m \le 9$, $1 \le n \le 30$).

Output

Output the number of ways to tile an $m \times n$ rectangle with long dominoes.

Example

dominoes.in	dominoes.out
3 3	2
3 10	28

Problem F. The Magic Wheel

Input file:	`magic.in`
Output file:	`magic.out`
Time limit:	1 second
Memory limit:	64 megabytes

The great magician Zahruman is going to create the great artefact — the magic wheel. The magic wheel is a circular cylinder with radius r and height h. Along each of the two circular bases of the wheel there are n special lugs. Also there are n small loops located along a circle surronding the cylinder in the middle.

To turn the wheel into the magic artefact, one needs to connect pairs of lugs on the base circles by magic threads. Each lug must be connected to exactly one lug from the other circle. Each thread in turn must pass through one of the loops of the central circle. There must be exactly one thread passing through each of the loops.

Since magic thread is a rather expensive thing, Zahruman wants to use as little of it as possible. So he wants to know what is the minimal possible length of the n threads. And of course, to construct the wheel with the thread of that length he needs to know which pairs of lugs should be connected, and which loop should each thread pass through.

Each thread must run along the lateral face of a cylinder, it is not allowed to run thread along cylinder base.

Input

The first line of the input file contains n ($1 \le n \le 1\,000$), r and h (r and h are integer, $1 \le r, h \le 100$).

The second describes lugs on the top base circle of the cylinder, it contains n real numbers — polar angles of the lugs in radians. No two lugs coincide.

The third line describes loops on the center circle in the same format. The fourth line describes lugs on the bottom base circle. All polar angles are in range $[-2\pi, 2\pi]$ and measured from some common direction orthogonal to the cylinder axis.

Output

On the first line of the output file print the total length of all magic threads. After that output n lines, each with three integer numbers: a_i, b_i and c_i — the number of the lug on the top base circle, the number of the loop on the center circle, and the number of the lug on the bottom base circle to use for the i-th thread.

Your answer must be accurate up to 10^{-4}.

Example

magic.in	magic.out
3 10 4 0 1.57079632679489662 3.14159265358979324 -1.57079632679489662 0 3.14159265358979324 3.14159265358979324 0 1.57079632679489662	67.3391013548167057 1 1 2 2 2 3 3 3 1

Problem G. Cracking SSH

Input file:	`ssh.in`
Output file:	`ssh.out`
Time limit:	1 second
Memory limit:	64 megabytes

Famous secure shell (ssh) protocol is often used to provide remote access to Unix systems. In ssh protocol all communcations with the server are encrypted using a strong cipher, so it is considered essentially impossible to eavesdrop them.

However, cryptoanalysts have recently found a vulnerability that can be used to learn the user's password when the ssh session is established. The drawback is that when the characters are typed slowly, it is possible that each character is sent to the server in his own network packet. Analyzing the time intervals between consecutive packets and comparing them to typical intervals between typing various characters by the user, it may be possible to determine the most probable password.

You are given the time intervals between consecutive packets in some password sending session and the typical intervals between typing all possible pairs of characters. Your task is to determine the most probable password, assuming that each character of the password was sent in its own packet.

The probability of some string to be the password is determined in the following way. Let the sequence of time intervals given be $a_1, a_2, \ldots, a_{l-1}$. Let the typical time interval between typing characters c and d be t_{cd}. For the password $p = p_1 p_2 \cdots p_l$ its *unlikeness* to the given intervals sequence is

$$U(p) = |a_1 - t_{p_1 p_2}| + |a_2 - t_{p_2 p_3}| + \ldots + |a_{l-1} - t_{p_{l-1} p_l}|.$$

The less is the unlikeness of the password — the more probable it is.

Input

The first line of the input file contains l — the length of the password, and m — the number of different characters that can be used in password ($2 \le l \le 100$, $2 \le m \le 26$). The characters used in the password are the first m small letters of the English alphabet.

The second line of the input file contains $l - 1$ integer numbers: $a_1, a_2, \ldots a_{l-1}$ ($1 \le a_i \le 1000$). The following m lines contain m integer numbers each and represent the typical intervals between typing the characters, j-th number of the i-th line is the interval between typing i-th and j-th characters of the alphabet ($1 \le t_{ij} \le 1000$).

Output

Output the most probable password. If there are several possible answers, output any one.

Example

ssh.in	ssh.out
7 3 3 4 4 6 3 5 1 3 4 5 1 2 6 3 1	abacaba

Problem H. Periodic Tilings

Input file:	`tilings.in`
Output file:	`tilings.out`
Time limit:	1 second
Memory limit:	64 megabytes

In the middle of the XX-th century Roger Penrose impressed many mathematicians showing that there exists a set of tiles that can cover the whole plane, but only aperiodically. On the contrary, in this problem we are interested in periodic tilings, more of that, in regular tilings.

Consider a connected set of unit squares. It is called a *polyomino*. A polyomino is said to tile the plane *in a regular way*, if one can cover the whole plane with non-overlapping copies of the selected polyomino in such a way that:

- all copies can be transformed one to another using translation only, rotations and flips are not allowed;
- each copy has the same surrounding polyominoes, that is, if we select some polyomino and translate the whole plane so that the top-left corner of the selected polyomino is in the coordinates center, the picture is the same for all polyominoes.

Given a polyomino, find out if it is possible to tile the plane with it in a regular way. You may find useful the following fact: if it is possible to tile the plane with some polyomino in a regular way, then in the tiling each polyomino has a common border of non-zero length with at most six other polyominoes.

Input

The input file contains a part of the plane that contains the given polyomino, represented by the characters '`.`' (dot) denoting empty spaces and '`*`' (asterisk) denoting squares that belong to the polyomino. Input file contains at most 50 lines, each contains at most 50 characters.

The polyomino is constructed of at most 30 unit squares.

Output

Output "`YES`" if it is possible to tile the plane in a regular way with the polyomino given, or "`NO`" if it is impossible.

Example

tilings.in	tilings.out
.... .*.. .**.	YES
.... .*.. .**.* ..***	NO
.... .***. ..*.. .***. ..*..	YES

Problem I. Trade

Input file:	`trade.in`
Output file:	`trade.out`
Time limit:	1 second
Memory limit:	64 megabytes

In the Middle Ages m European cities imported many goods from n Arabian cities. Due to continous feudal wars, European cities did not trade with each other, so is some European city needed some Arabian goods, the special trade route was established for this particular trade.

Studying the manuscripts historians have found out that each European city imported goods from at least two Arabian cities, and each Arabian city exported goods to at least two European cities. They have also investigated different factors and identified all potential trade routes (trade routes between some pairs of cities were impossible due to various reasons).

Now historians wonder, what is the minimal possible number of trade routes, that could have existed. Help them to find that out.

Input

The first line of the input file contains m, n, and p — the number of European and Arabian cities respectively, and the number of potential trade routes ($1 \le m, n \le 300$, $1 \le p \le nm$). The following p lines describe potential trade routes, each description consists of two numbers — the European and the Arabian city connected by the route.

Output

On the first line of the output file print k — the minimal possible number of trade routes that could have existed. After that output k numbers — some minimal set of routes that might have existed to satisfy all conditions. Routes are numbered starting from 1 as they are given in the input file.

If historians must have made a mistake and it is impossible to satisfy the specified conditions, print -1 on the first and the only line of the output file.

Example

trade.in	trade.out
5 5 14 1 2 1 3 1 4 1 5 2 1 2 5 3 1 3 5 4 1 4 5 5 1 5 2 5 3 5 4	12 1 2 3 5 6 7 8 9 10 12 13 14

Problem J. Counting Triangulations

Input file:	`tria.in`
Output file:	`tria.out`
Time limit:	1 second
Memory limit:	64 megabytes

The *triangulation* of the set of the points on the plane is the set of triangles, satisfying the following conditions:

1. no triangle is degenerate;
2. no two triangles have a common interior point (common border points are allowed);
3. all vertices of each triangle are the points from the given set;
4. each point within the convex hull of the given points belongs to some triangle;
5. no point of the given set belongs to the interior of a triangle.

A set of triangles is called *pretriangulation* if it satisfies all these conditions except possibly the last. A pretriangulation is called *minimal* if it contains minimal possible number of triangles among all pretriangulations of the given set.

A triangulation is called *consecutive* if it can be obtained from some minimal pretriangulation by successively applying the following *split* operation: choose a point that belongs to the interior of some triangle and connect it to the vertices of this triangle, splitting it to three smaller ones.

Given the set of the points on the plane, no three of which are lying on the same line, find the number of its consecutive triangulations. Two triangulations are different if they are different as sets of fixed triangles.

Input

The first line of the input file contains n — the number of points ($3 \le n \le 50$). The following n lines contain two integer numbers each — coordinates of the points (no coordinate exceeds 10^4 by its absolute value).

Output

Output the number of different consecutive trianguations of the given points set.

Example

tria.in	tria.out
4 0 0 3 3 3 0 0 3	2
4 0 0 3 3 3 0 2 1	1

Problem K. Unfair Contest

Input file:	`unfair.in`
Output file:	`unfair.out`
Time limit:	1 second
Memory limit:	64 megabytes

Chief Judge of the Galactic Programming Contest in 3141–3157 has recently published his memoirs in which he writes about sensational facts. The jury of the contest used to make problem sets specially designed for some teams to win. In the book the judge described the technics used in details. In this problem you are asked to implement the algorithm used by the judges.

Before the contest the jury prepares m problems, each characterized with its *intellectuality* I_i, *technics requirements* A_i, *code length* L_i, and *ordinariness* O_i. The goal of the jury is to select n problems for the contest.

Each team participating in the contest is in turn characterized by its *theoretical background* T_j, *programming technics* Z_j, *typing speed* V_j, and *contests practice* C_j.

The j-th team can solve i-th problem if and only if $T_j + C_j$ exceeds $I_i - O_i$. The team solves problems one after one, in the ascending order of expected solution time. Expected solution time for the problem is

$$e_{i,j} = \lceil I_i/O_i \rceil + \max(\lceil A_i/C_j \rceil, 5).$$

But the actual time required to solve the problem is

$$t_{i,j} = \max(I_i - T_j, 0) + \lceil A_i/(Z_j + C_j) \rceil + \lceil L_i/V_j \rceil.$$

If two problems have the same expected solution time, we consider the team lucky, and suggest that if first solves the problem with smaller actual required time.

If the team does not finish to solve the problem within the contest length l, it does not solve it. If the team finished to solve problem exactly when the contest is over, we consider that it has solved the problem.

The teams are finally arranged in the descending order by the number of problems solved, and if the number is equal, in the ascending order by the penalty time. The penalty time is the sum of penalties for each solved problem. The penalty time for the problem is the time from the beginning of the contest till the moment the problem is solved. Teams with equal both number of solved problems and penalty time have the same highest possible for them rank.

The unsuccesful runs are ignored in the model, because it is difficult to predict them.

Given the descriptions of m problems and t teams, select n such problems, that the team 1 gets the highest possible rank.

Input

The first line of the input file contains m, n, t and l ($1 \le m \le 16$, $1 \le n \le 12$, $n \le m$, $1 \le t \le 20$, $10 \le l \le 500$).

The following m lines describe problems, each line contains I_i, A_i, L_i and O_i (all numbers are integer, ranging from 1 to 1000, except L_i which is from 100 to 50 000).

The following t lines describe teams, each line contains T_j, Z_j, V_j and C_j (all numbers are integer, ranging from 1 to 1000).

Output

Output n integer numbers in the ascending order — the problems that must be selected.

Example

unfair.in	unfair.out
3 2 4 60 20 40 2200 10 60 10 600 10 10 20 1500 40 100 10 100 10 100 30 70 2 20 20 300 10 30 1 100 1	2 3