Problem A. Alien Communication Masterclass

Input file:	`acm.in`
Output file:	`acm.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Andrea is a famous science fiction writer, who runs masterclasses for her beloved readers. The most popular one is the Alien Communication Masterclass (ACM), where she teaches how to behave if you encounter alien life forms or at least alien artifacts.

One of the lectures concerns retrieving useful information based on aliens' writings. Andrea teaches that based on alien mathematical formulas, one could derive the base of the numeral system used by the aliens, which in turn might give some knowledge about aliens' organisms. (For example, we use numeral system with base 10, due to the fact that we have ten fingers on our upper extremities).

Suppose for simplicity that aliens use the same digits as we do, and they understand and denote addition, subtraction, multiplication, parentheses and equality the same way as we do.

For her lecture, Andrea wants an example of a mathematical equality that holds in numeral systems with bases $a_1, a_2, \cdots, a_n$, but doesn't hold in numeral systems with bases $b_1, b_2, \cdots, b_m$. Provide her with one such formula.

Input

The first line of the input file contains two integer numbers, n and m ($1 \le n, m \le 8$). The second line contains n numbers, $a_1, a_2, \cdots, a_n$. The third line contains m numbers, $b_1, b_2, \cdots, b_m$.

All a_i and b_i are distinct and lie between 2 and 10, inclusive.

Output

Output any syntactically correct mathematical equality that holds in numeral systems with bases $a_1, a_2, \cdots, a_n$, but doesn't hold in numeral systems with bases $b_1, b_2, \cdots, b_m$. The equality can contain only digits 0 through 9, addition ('`+`'), subtraction and unary negation ('`-`'), multiplication ('`*`'), parentheses ('`(`' and '`)`') and equality sign ('`=`'). There must be exactly one equality sign in the output.

Any whitespace characters in the output file will be ignored. The number of non-whitespace characters in the output file must not exceed 10 000.

Examples

acm.in	acm.out
1 2 2 3 9	(10 - 1) * (10 - 1) + 1 = 10
2 2 9 10 2 3	2 + 2 = 4

Problem B. Bug2

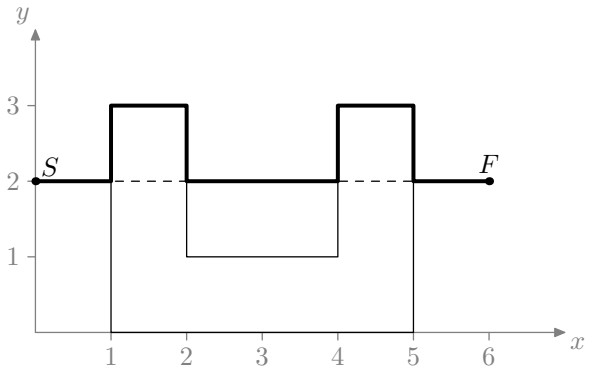
Input file:	`bug2.in`
Output file:	`bug2.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

There is a variety of navigation problems around us. This is a problem about an algorithm called *Bug2*.

Bug algorithms solve the following navigation problem. There is a two-dimensional map containing obstacles of an arbitrary shape, and start and finish points are given. There is also an "agent", who initially stands at the start point S, and its task is to reach the finish point F. It knows the coordinates of the finish point, and at any moment of time it can determine its own coordinates. The agent has $O(1)$ memory, so it cannot store the map of obstacles. The only way it can get information about the outer world is to detect whether it touches an obstacle. The Agent is able to move around the obstacle following its boundary. The problem is to reach the finish point when it is possible, and correctly report the fact of unreachability otherwise.

The *Bug2* algorithm, which belongs to the family of Bug algorithms, works as follows:

1. Move towards F until one of the following happens:
   - The finish point is reached. Then the algorithm stops.
   - An obstacle is encountered. Then go to step 2.
2. Define the current point as H. Move around the boundary of the obstacle in the clockwise direction until one of the following happens:
   - The finish point is reached. Then the algorithm stops.
   - The point H is reached. Then the finish point is unreachable, and the algorithm stops.
   - A point L is reached, which lies on the line SF, $|LF| < |HF|$ and it is possible to move towards F without hitting an obstacle immediately. In this case, go to step 1.



One may prove the correctness of an algorithm, that is, that it reaches the finish point in finite time (that is, the length of the resulting path is finite) if it is possible, and reports that the finish point is unreachable in finite time otherwise.

Given a set of polygonal obstacles, a start and a finish point, determine the length of the path that an agent directed by *Bug2* algorithm will traverse.

Input

The first line of the input file contains five integer numbers n, x_S, y_S, x_F, y_F — the number of obstacles and the coordinates of start and finish points.

The rest of the input file describes obstacles. Each description starts with a line containing an integer m ($m \geq 3$) — the number of vertices in the obstacle. The following m lines contain pairs of integer numbers x_i, y_i — the coordinates of vertices of the obstacle, given in the clockwise direction. The obstacle does not have self-intersections or self-tangencies.

The total number of vertices in all the obstacles does not exceed 5000. No coordinate exceeds 10^6 by an absolute value.

No vertex of an obstacle lies on a line SF. Both start and finish point will lie strictly outside any obstacle. No two obstacles share common points. If there are two points A and B where obstacle boundaries intersect with the line SF, either $|AF| = |BF|$ or $\big||AF| - |BF|\big| > 10^{-6}$ will be true.

Output

Output the length of a path traversed by the agent directed by the described algorithm. The absolute or relative error of 10^{-6} is acceptable.

Example

bug2.in	bug2.out
1 0 2 6 2 8 1 0 1 3 2 3 2 1 4 1 4 3 5 3 5 0	10.0

Problem C. Commuting Functions

Input file:	commuting.in
Output file:	commuting.out
Time limit:	3 seconds
Memory limit:	256 megabytes

Two functions f and g ($f, g : X \to X$) are *commuting* if and only if $f(g(x)) = g(f(x))$ for each $x \in X$. For example, functions $f(x) = x + 1$ and $g(x) = x - 2$ are commuting, whereas functions $f(x) = x + 1$ and $g(x) = 2x$ are not commuting.

Each function h ($h : N_n \to N_n$, where $N_n = 1, 2, \ldots, n$ and n is positive integer) can be represented as a *value list* — a list in which the i-th element is equal to $h(i)$. For example, a function $h(x) = \lceil x/2 \rceil$ from N_5 to N_5 has the value list $[1, 1, 2, 2, 3]$.

The value lists are ordered lexicographically: list $[a_1 \ldots a_n]$ is smaller than list $[b_1 \ldots b_n]$ if and only if there exists such an index k that $a_k < b_k$, and $a_l = b_l$ for any index $l < k$.

The function f ($f : X \to X$) is *bijective* if for every y in X, there is exactly one x in X such that $f(x) = y$.

Given a bijective function f ($f : N_n \to N_n$, n is positive integer), find the function g that is commuting with f and has the lexicographically smallest possible value list.

Input

The first line contains single integer number n — the number of the elements in the value list of bijective function f ($1 \le n \le 5000$).

The second line of the input file contains the value list of the function f.

Output

The single line of the output file must contain n integer numbers — the value list of function g that commutes with the function f and has the lexicographically smallest value list.

Examples

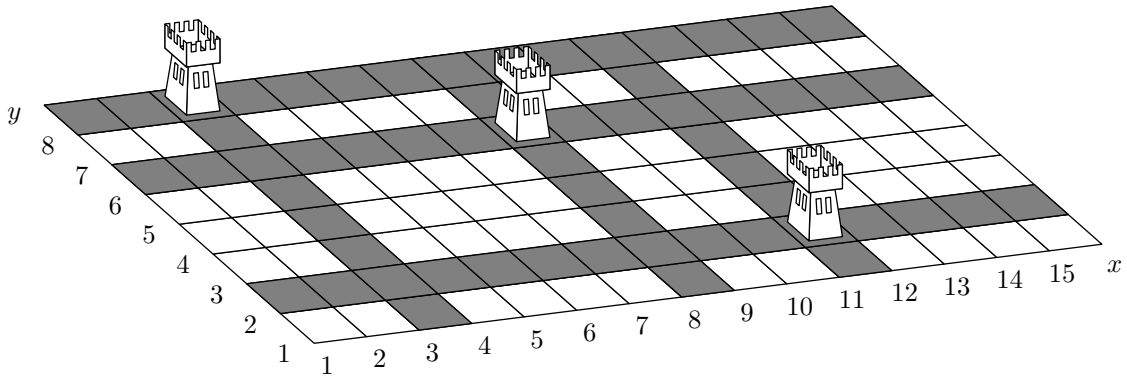
commuting.in	commuting.out
10 1 2 3 4 5 6 7 8 9 10	1 1 1 1 1 1 1 1 1 1
10 2 3 4 5 6 7 8 1 9 10	1 2 3 4 5 6 7 8 9 9

Problem D. Defense of a Kingdom

Input file:	`defense.in`
Output file:	`defense.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Theodore implements a new strategy game "Defense of a Kingdom". On each level player defends the Kingdom that is represented by a rectangular grid of cells. The player builds crossbow towers in some cells of the grid. The tower defends all the cells in the same row and the same column. No two towers share a row or a column.

The penalty of the position is a number of cells in the largest undefended rectangle. For example, the position shown on the picture has penalty 12.



Help Theodore write a program that calculates the penalty of the given position.

Input

The first line of the input file contains three integer numbers: w — width of the grid, h — height of the grid and n — number of crossbow towers ($1 \le w, h \le 40\,000$; $0 \le n \le \min(w, h)$).

Each of the following n lines of the input file contains two integer numbers x_i and y_i — the coordinates of the cell occupied by a tower ($1 \le x_i \le w$; $1 \le y_i \le h$).

Output

Output a single integer number — the number of cells in the largest rectangle that is not defended by the towers.

Example

defense.in	defense.out
15 8 3 3 8 11 2 8 6	12

Problem E. Explicit Formula

Input file:	`explicit.in`
Output file:	`explicit.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Consider 10 Boolean variables $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9$, and x_{10}. Consider all pairs and triplets of distinct variables among these ten. (There are 45 pairs and 120 triplets.) Count the number of pairs and triplets that contain at least one variable equal to 1. Set $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}) = 1$ if this number is odd and $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}) = 0$ if this number is even.

Here's an explicit formula that represents the function $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10})$ correctly:

$$f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}) = (x_1 \vee x_2) \oplus (x_1 \vee x_3) \oplus (x_1 \vee x_4) \oplus (x_1 \vee x_5) \oplus (x_1 \vee x_6) \oplus (x_1 \vee x_7) \oplus (x_1 \vee x_8) \oplus (x_1 \vee x_9) \oplus (x_1 \vee x_{10}) \oplus (x_2 \vee x_3) \oplus (x_2 \vee x_4) \oplus (x_2 \vee x_5) \oplus (x_2 \vee x_6) \oplus (x_2 \vee x_7) \oplus (x_2 \vee x_8) \oplus (x_2 \vee x_9) \oplus (x_2 \vee x_{10}) \oplus (x_3 \vee x_4) \oplus (x_3 \vee x_5) \oplus (x_3 \vee x_6) \oplus (x_3 \vee x_7) \oplus (x_3 \vee x_8) \oplus (x_3 \vee x_9) \oplus (x_3 \vee x_{10}) \oplus (x_4 \vee x_5) \oplus (x_4 \vee x_6) \oplus (x_4 \vee x_7) \oplus (x_4 \vee x_8) \oplus (x_4 \vee x_9) \oplus (x_4 \vee x_{10}) \oplus (x_5 \vee x_6) \oplus (x_5 \vee x_7) \oplus (x_5 \vee x_8) \oplus (x_5 \vee x_9) \oplus (x_5 \vee x_{10}) \oplus (x_6 \vee x_7) \oplus (x_6 \vee x_8) \oplus (x_6 \vee x_9) \oplus (x_6 \vee x_{10}) \oplus (x_7 \vee x_8) \oplus (x_7 \vee x_9) \oplus (x_7 \vee x_{10}) \oplus (x_8 \vee x_9) \oplus (x_8 \vee x_{10}) \oplus (x_9 \vee x_{10}) \oplus (x_1 \vee x_2 \vee x_3) \oplus (x_1 \vee x_2 \vee x_4) \oplus (x_1 \vee x_2 \vee x_5) \oplus (x_1 \vee x_2 \vee x_6) \oplus (x_1 \vee x_2 \vee x_7) \oplus (x_1 \vee x_2 \vee x_8) \oplus (x_1 \vee x_2 \vee x_9) \oplus (x_1 \vee x_2 \vee x_{10}) \oplus (x_1 \vee x_3 \vee x_4) \oplus (x_1 \vee x_3 \vee x_5) \oplus (x_1 \vee x_3 \vee x_6) \oplus (x_1 \vee x_3 \vee x_7) \oplus (x_1 \vee x_3 \vee x_8) \oplus (x_1 \vee x_3 \vee x_9) \oplus (x_1 \vee x_3 \vee x_{10}) \oplus (x_1 \vee x_4 \vee x_5) \oplus (x_1 \vee x_4 \vee x_6) \oplus (x_1 \vee x_4 \vee x_7) \oplus (x_1 \vee x_4 \vee x_8) \oplus (x_1 \vee x_4 \vee x_9) \oplus (x_1 \vee x_4 \vee x_{10}) \oplus (x_1 \vee x_5 \vee x_6) \oplus (x_1 \vee x_5 \vee x_7) \oplus (x_1 \vee x_5 \vee x_8) \oplus (x_1 \vee x_5 \vee x_9) \oplus (x_1 \vee x_5 \vee x_{10}) \oplus (x_1 \vee x_6 \vee x_7) \oplus (x_1 \vee x_6 \vee x_8) \oplus (x_1 \vee x_6 \vee x_9) \oplus (x_1 \vee x_6 \vee x_{10}) \oplus (x_1 \vee x_7 \vee x_8) \oplus (x_1 \vee x_7 \vee x_9) \oplus (x_1 \vee x_7 \vee x_{10}) \oplus (x_1 \vee x_8 \vee x_9) \oplus (x_1 \vee x_8 \vee x_{10}) \oplus (x_1 \vee x_9 \vee x_{10}) \oplus (x_2 \vee x_3 \vee x_4) \oplus (x_2 \vee x_3 \vee x_5) \oplus (x_2 \vee x_3 \vee x_6) \oplus (x_2 \vee x_3 \vee x_7) \oplus (x_2 \vee x_3 \vee x_8) \oplus (x_2 \vee x_3 \vee x_9) \oplus (x_2 \vee x_3 \vee x_{10}) \oplus (x_2 \vee x_4 \vee x_5) \oplus (x_2 \vee x_4 \vee x_6) \oplus (x_2 \vee x_4 \vee x_7) \oplus (x_2 \vee x_4 \vee x_8) \oplus (x_2 \vee x_4 \vee x_9) \oplus (x_2 \vee x_4 \vee x_{10}) \oplus (x_2 \vee x_5 \vee x_6) \oplus (x_2 \vee x_5 \vee x_7) \oplus (x_2 \vee x_5 \vee x_8) \oplus (x_2 \vee x_5 \vee x_9) \oplus (x_2 \vee x_5 \vee x_{10}) \oplus (x_2 \vee x_6 \vee x_7) \oplus (x_2 \vee x_6 \vee x_8) \oplus (x_2 \vee x_6 \vee x_9) \oplus (x_2 \vee x_6 \vee x_{10}) \oplus (x_2 \vee x_7 \vee x_8) \oplus (x_2 \vee x_7 \vee x_9) \oplus (x_2 \vee x_7 \vee x_{10}) \oplus (x_2 \vee x_8 \vee x_9) \oplus (x_2 \vee x_8 \vee x_{10}) \oplus (x_2 \vee x_9 \vee x_{10}) \oplus (x_3 \vee x_4 \vee x_5) \oplus (x_3 \vee x_4 \vee x_6) \oplus (x_3 \vee x_4 \vee x_7) \oplus (x_3 \vee x_4 \vee x_8) \oplus (x_3 \vee x_4 \vee x_9) \oplus (x_3 \vee x_4 \vee x_{10}) \oplus (x_3 \vee x_5 \vee x_6) \oplus (x_3 \vee x_5 \vee x_7) \oplus (x_3 \vee x_5 \vee x_8) \oplus (x_3 \vee x_5 \vee x_9) \oplus (x_3 \vee x_5 \vee x_{10}) \oplus (x_3 \vee x_6 \vee x_7) \oplus (x_3 \vee x_6 \vee x_8) \oplus (x_3 \vee x_6 \vee x_9) \oplus (x_3 \vee x_6 \vee x_{10}) \oplus (x_3 \vee x_7 \vee x_8) \oplus (x_3 \vee x_7 \vee x_9) \oplus (x_3 \vee x_7 \vee x_{10}) \oplus (x_3 \vee x_8 \vee x_9) \oplus (x_3 \vee x_8 \vee x_{10}) \oplus (x_3 \vee x_9 \vee x_{10}) \oplus (x_4 \vee x_5 \vee x_6) \oplus (x_4 \vee x_5 \vee x_7) \oplus (x_4 \vee x_5 \vee x_8) \oplus (x_4 \vee x_5 \vee x_9) \oplus (x_4 \vee x_5 \vee x_{10}) \oplus (x_4 \vee x_6 \vee x_7) \oplus (x_4 \vee x_6 \vee x_8) \oplus (x_4 \vee x_6 \vee x_9) \oplus (x_4 \vee x_6 \vee x_{10}) \oplus (x_4 \vee x_7 \vee x_8) \oplus (x_4 \vee x_7 \vee x_9) \oplus (x_4 \vee x_7 \vee x_{10}) \oplus (x_4 \vee x_8 \vee x_9) \oplus (x_4 \vee x_8 \vee x_{10}) \oplus (x_4 \vee x_9 \vee x_{10}) \oplus (x_5 \vee x_6 \vee x_7) \oplus (x_5 \vee x_6 \vee x_8) \oplus (x_5 \vee x_6 \vee x_9) \oplus (x_5 \vee x_6 \vee x_{10}) \oplus (x_5 \vee x_7 \vee x_8) \oplus (x_5 \vee x_7 \vee x_9) \oplus (x_5 \vee x_7 \vee x_{10}) \oplus (x_5 \vee x_8 \vee x_9) \oplus (x_5 \vee x_8 \vee x_{10}) \oplus (x_5 \vee x_9 \vee x_{10}) \oplus (x_6 \vee x_7 \vee x_8) \oplus (x_6 \vee x_7 \vee x_9) \oplus (x_6 \vee x_7 \vee x_{10}) \oplus (x_6 \vee x_8 \vee x_9) \oplus (x_6 \vee x_8 \vee x_{10}) \oplus (x_6 \vee x_9 \vee x_{10}) \oplus (x_7 \vee x_8 \vee x_9) \oplus (x_7 \vee x_8 \vee x_{10}) \oplus (x_7 \vee x_9 \vee x_{10}) \oplus (x_8 \vee x_9 \vee x_{10})$$

In this formula $\vee$ stands for logical or, and $\oplus$ stands for exclusive or (xor). Remember that in C++ and Java these two binary operators are denoted as "`||`" and "`^`".

Given the values of $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}$, calculate the value of $f(x_1, x_2, \ldots, x_{10})$.

Input

The input file contains 10 numbers $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9$, and x_{10}. Each of them is either 0 or 1.

Output

Output a single value — $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10})$.

Example

explicit.in	explicit.out
1 0 0 1 0 0 1 0 0 1	0

Problem F. Frames

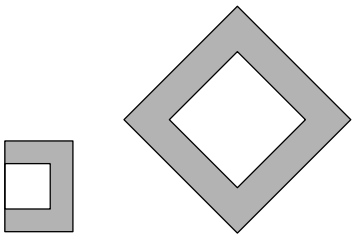
Input file:	`frames.in`
Output file:	`frames.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Vasya and Petya are playing an interesting game. Rules are pretty easy: there are two frames, the players have to make a translation of the second frame in such a way that the area of the intersection of the frames would be as large as possible. Both players think for a minute, and write down the translation vector of the second frame. The player whose vector gives a larger intersection area wins.

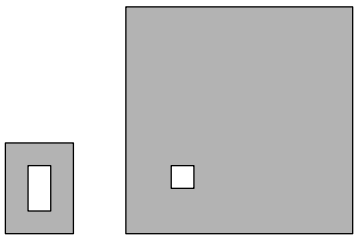
The game has many subtle cases, so Vasya wants to cheat and write a program that finds the best translation vector.

For this game the *frame* is a difference of two rectangles: the inner one and the outer one. The inner rectangle lies strictly inside the outer one. Sides of both rectangles are parallel to the coordinate axes.

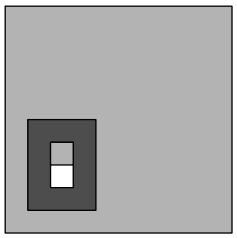
To make the definition more clear let us consider some examples.



Incorrect frames



Correct Frames



Frames intersection

The area of the frame is $(W \cdot H - w \cdot h)$, where W, H — dimensions of the outer rectangle and w, h — dimensions of the inner one $(0 < w < W; 0 < h < H)$.

Write a program that finds a translation of one frame relative to another that results in maximum frames intersection area.

Input

Each frame is described by four points — two opposite corners of the outer rectangle, followed by two opposite corners of the inner rectangle. Points are described by their coordinates — pairs of integer numbers x and y. Coordinates do not exceed 10^8 by absolute value.

The first line of the input file contains the description of the first frame.

The second line of the input file contains the description of the second frame.

Output

The first line of the output file must contain an integer number A — the maximal possible intersection area of the given two frames achievable by a translation.

The second line of the output file must contain a pair of integer numbers x and y — coordinates of the translation vector of the second frame that provides the intersection area A. Coordinates must not exceed 10^{18} by the absolute value.

Example

frames.in	frames.out
2 2 5 6 3 3 4 5 0 0 10 10 2 2 3 3	10 1 1

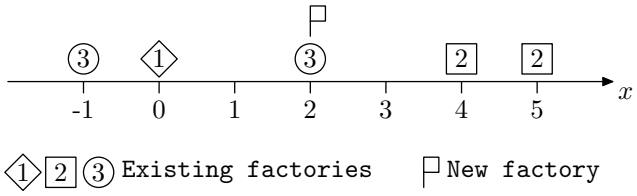
Problem G. Gadgets Factory

Input file:	gadgets.in
Output file:	gadgets.out
Time limit:	3 seconds
Memory limit:	256 megabytes

Mr. Smith is a very rich gadgets fan. As soon as he realized that he cannot buy all the gadgets he wants just because they are not yet produced, he decided to build his own Gadgets Factory.

The Gadgets Factory will be built at a place called 'Silicon Road". This road concentrates production of highly technological parts required to produce gadgets. The Silicon Road is straight and the factories are placed very close to it, so the road can be considered as an axis, and factories can be considered as points on it.

There are n parts needed to produce gadgets, and m factories that produce these parts. Mr. Smith wants to minimize the sum of squared distances to the nearest factories that produce each of required parts. Help him to find all the points in which that sum is minimal.



Input

The first line of the input file contains integer numbers n and m ($1 \le n \le m \le 5000$).

Next m lines contain pairs of integer numbers x_i and p_i, x_i is the coordinate of i-th factory, and p_i is the identifier of the part that it produces ($|x_i| \le 100\,000$; $x_i \le x_{i+1}$; $1 \le p_i \le n$).

For each required part there is at least one factory that produces it.

Output

The first line of the output file should contain an integer number k — the number of points where the Gadgets Factory can be built.

Next k lines should contain these points in ascending order. The values should be accurate within an absolute error of 10^{-6}.

Examples

gadgets.in	gadgets.out
3 5 -1 3 0 1 2 3 4 2 5 2	1 2.0
2 5 1 1 2 2 3 1 4 2 5 1	4 1.5 2.5 3.5 4.5

Problem H. Horrible Truth

Input file:	`horrible.in`
Output file:	`horrible.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

In a Famous TV Show "Find Out" there are n characters and only one Horrible Truth. To make the series breathtaking all way long, the screenplay writer decided that every episode should show exactly one important event.

There are three types of the *important events* in this series:

- character A finds out the Truth;
- character A finds out that the other character B knows the Truth;
- character A finds out that the other character B doesn't know the Truth.

Initially, nobody knows the Truth. All events must be correct, and every fact found out must be true. If some character finds out some fact, she cannot find it out once again.

Moreover, to give the audience some sense of action, the writer does not want an episode to show the important event of the same type as in the previous episode.

Your task is to determine the maximal possible number of episodes in the series and to create an example of a screenplay plan.

Input

The only line of the input contains a single integer n — the number of characters in the TV show ($1 \le n \le 100$).

Output

In the first line of the output file output a single integer k — the maximal possible number of episodes in the series. Then write k lines, each containing a description of an episode. For the episode in which character A (characters are numbered 1 through n) finds out the Truth, write the line "A 0". For an episode in which character A finds out that character B knows the Truth, write the line "A B". Similarly, for an episode in which character A finds out that character B doesn't know the Truth, write the line "A -B".

If there are several plans providing the maximal possible number of episodes, output any one of them.

Example

horrible.in	horrible.out
3	13 2 -1 1 0 2 1 1 -2 3 1 3 -2 2 0 1 2 1 -3 3 2 2 -3 3 0 1 3

Problem I. Ideal Contest

Input file:	`ideal.in`
Output file:	`ideal.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

It's a pity that there is no higher-order contest for ACM ICPC regionals and subregionals; probably this is because it's very hard to rank them. But now we have an idea how to do this! The idea is based on a notion of *negidealness* — a number showing non-conformity of the contest results with "ideal" contest criteria. It is a weighted sum of the following specific negidealnesses (penalties).

Vainness penalty V. Each team should solve at least one problem. If a team solves no problems, a penalty of $1/T$ (where T is the number of teams that participated in the contest) for each such team is added.

Oversimplification penalty O. There should be no team with all the problems solved. If some teams solve all the problems, a penalty of $1/T$ is added for each such team.

Evenness penalty E. The number of problems solved by different teams should increase evenly. If the difference in problems solved between two adjacent (in the standings) teams is greater than 1, then the penalty of $1/P$ (where P is the total number of problems) is added for each skipped number of problems solved. E.g., if a team solves 5 problems, and the next team solves 1 problem, then the penalty of $3/P$ should be added, since no team solved 2, 3 or 4 problems.

Unsolvability penalty U. Every problem should be solved by at least one team. If a problem is not solved, a penalty of $1/P$ for each such problem is added.

Instability penalties $I_1, I_2, \ldots, I_P$. If a problem p was solved by a team, then this problem should be solved by all the teams ranked above. For each team which did not solve problem p ranked above the lowest-ranked team that did solve problem p a penalty of $1/T$ is added to I_p.

The *total negidealness* N equals $1.03V + 3.141O + 2.171E + 1.414U + (I_1 + I_2 + \cdots + I_P)/P$.

Write a program that finds the negidealness of the given results table.

Input

The input file contains a contest results table in plain ASCII. The only whitespace symbol in the table is a space. There is always at least one space separating columns. The problems are named with capital English letters in the alphabetical order. There are at most 26 problems and at most 300 teams.

Output

The output data should contain the penalties for each criterion (values V, O, E, U, $I_1, \ldots, I_P$) and the total negidealness. All the real numbers should be precise up to 3 digits after the decimal point.

Example

ideal.in	ideal.out
The contest header may contain arbitrary number of lines Team A B C D E = Time R --- Revda STU + + +2 +1 -9 4 9274 1 Girvas NU #1 + + -1 . -11 2 321 2 Kargopol SU + -3 + . -4 2 321 2 Utorgosh SU . . . + -5 1 122 4 Dubrovno SU . + -1 . -4 1 123 5 Girvas NU - 2 . . . -5 -99 0 0 6	Vainness = 0.167 Oversimplification = 0.000 Evenness = 0.200 Unsolvability = 0.200 Instability 1 = 0.000 Instability 2 = 0.333 Instability 3 = 0.000 Instability 4 = 0.333 Instability 5 = 0.000 Negidealness = 1.022

Problem J. Journey

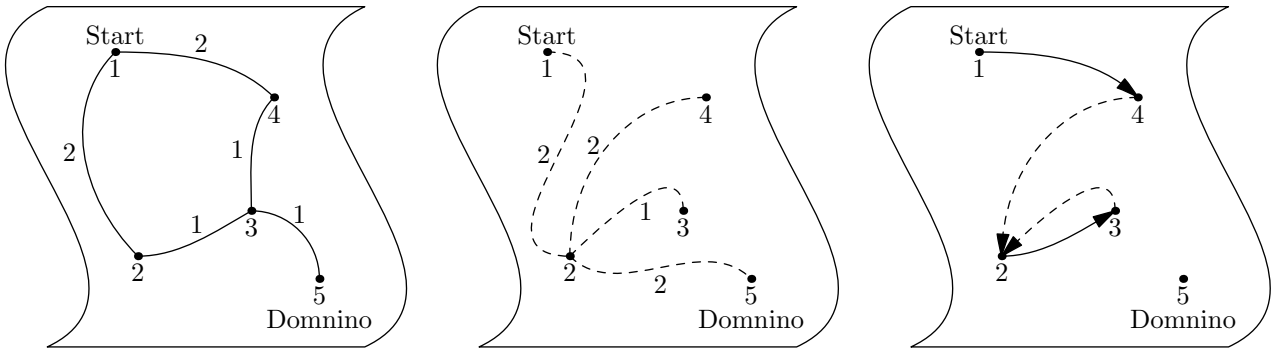
Input file:	`journey.in`
Output file:	`journey.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

The army of Rzeczpospolita is moving from the city Kostroma to the village Domnino. Two hetmans, Stefan and Konstantin, lead the army.

Stefan procured the roadmap of Kostroma province, and every night he routes the army from one village to the other along some road. Konstantin bought the map of secret trails between villages in advance, and every day he leads the march along the one of such trails. Each hetman asks their guide Ivan Susanin for a route before each march.

The length of each road is indicated on Konstantin's map. So Konstantin knows the minimal distance from each village to the Domnino village according to his map. Similarly Stefan knows the minimal distance from each village to Domnino village along trails on his map.

Ivan Susanin does not want to be disclosed as a secret agent, so each time he chooses a road (for Konstantin) or a trail (for Stefan) so that the minimal distance to the Domnino village according to the map owned by the asking hetman is strictly decreasing.



Stefan's map | Konstantin' map | Susanin's route

Help Ivan to find the longest possible route to the Domnino village.

Input

The first line of the input file contains three integer numbers n, s and t — number of villages in Kostroma province, and numbers of start and Domnino village ($2 \le n \le 1000$; $1 \le s, t \le n$). Villages are numbered from 1 to n. Start and Domnino villages are distinct.

Two blocks follow, the first one describing Stefan's map, and the second one describing Konstantin's map.

The first line of each block contains an integer number m — the number of roads/trails between villages ($n - 1 \le m \le 100\,000$). Each of the following m lines contains three integer numbers a, b, and l — describing the road/trail between villages a and b of length l ($1 \le a, b \le n$; $1 \le l \le 10^6$).

Rzeczpospolita army can move in any direction along a road or a trail. It's guaranteed that one can travel from any village to any other using each of the maps. The army starts its movement in the evening from the village number s and moves one road each night and one trail each day.

Output

Output the total length of the longest route that Ivan Susanin can arrange for Rzeczpospolita army before reaching the Domnino village (along the roads and trails). If Ivan Susanin can route the army forever without reaching the Domnino village, output the number "`-1`".

Examples

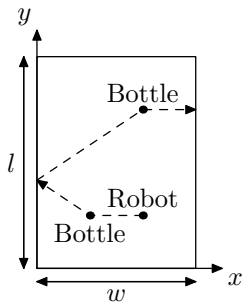
journey.in	journey.out
5 1 5 5 1 2 2 1 4 2 2 3 1 3 4 1 5 3 1 4 1 2 2 2 4 2 2 3 1 2 5 2	-1
3 1 3 4 1 2 10 2 3 10 1 3 20 2 3 30 4 2 1 10 1 3 10 1 1 10 2 3 10	20

Problem K. Kitchen Robot

Input file:	`kitchen.in`
Output file:	`kitchen.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Robots are becoming more and more popular. They are used nowadays not only in manufacturing plants, but also at home. One programmer with his friends decided to create their own home robot. As you may know most programmers like to drink beer when they gather together for a party. After the party there are a lot of empty bottles left on the table. So, it was decided to program robot to collect empty bottles from the table.

The table is a rectangle with the length l and width w. Robot starts at the point (x_r, y_r) and n bottles are located at points (x_i, y_i) for $i = 1, 2, \ldots, n$. To collect a bottle robot must move to the point where the bottle is located, take it, and then bring to some point on the border of the table to dispose it. Robot can hold only one bottle at the moment and for simplicity of the control program it is allowed to release bottle only at the border of the table.



You can assume that sizes of robot and bottles are negligibly small (robot and bottles are points), so the robot holding a bottle is allowed to move through the point where another bottle is located.

One of the subroutines of the robot control program is the route planning. You are to write the program to determine the minimal length of robot route needed to collect all the bottles from the table.

Input

The first line of the input file contains two integer numbers w and l — the width and the length of the table ($2 \le w, l \le 1000$).

The second line of the input contains an integer number n — the number of bottles on the table ($1 \le n \le 5$). Each of the following n lines contains two integer numbers x_i and y_i — coordinates of the i-th bottle ($0 < x_i < w$; $0 < y_i < l$). No two bottles are located at the same point.

The last line of the input file contains two integer numbers x_r and y_r — coordinates of the robot's initial position ($0 < x_r < w$; $0 < y_r < l$). Robot is not located at the same point with a bottle.

Output

Output the length of the shortest route of the robot. Your answer should be accurate within an absolute error of 10^{-6}.

Example

kitchen.in	kitchen.out
3 4 2 1 1 2 3 2 1	5.60555127546399