Problem A: Gate

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

Recently a local farmer bought the vast unused meadow near your house. Upon striking the deal, he decided he needed to construct a proper fence for the front side of his new property, and so he began right away by digging in N fence posts along this side of the meadow. As he contentedly looked at the completed work, he suddenly realized that he completely forgot to leave a large enough gap for a gate. To fix this, he might have to dig some of the posts out again.

However, he doesn't want to waste his labor, so he intends to dig out the minimum number of posts that suffices to create a large enough gap (between two posts, between a post and an edge of the field or between both edges of the field). To find out how many that is, he asked you for help.

Input

We identify the front side of the meadow with a segment of a coordinate axis. The input consists of

- one line containing N and W – the number of posts already dug in and the width a gap must have to be able to accommodate the gate, respectively ($0 \leq N \leq 3 \cdot 10^4,\ 0 \leq W \leq 6 \cdot 10^4$)
- one line containing L and R – the coordinates of the meadows' boundaries ($L < R$)
- one line containing N integers – the coordinates of currently dug in posts.

All given coordinates are integers whose absolute values don't exceed 30 000. Also, it is guaranteed that all posts are dug in between the given boundaries.

Output

Output the minimum number of posts he needs to dig out. If there is no solution output `-1`.

Sample input and output

Input	Output
3 2 2 6 3 4 5	1
3 2 1 6 4 3 5	0
3 5 1 7 5 3 4	3

Problem B: Minimum cover

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

Given a set of segments $[L_i, R_i]$ with integer end points L_i, R_i, you are to find a subset of it that completely covers the segment $[0, M]$ and, in doing so, consists of a minimal number of segments.

Input

The input consists of

- one line containing M $(1 \leq M \leq 5000)$
- several lines each describing a segment by a its integer endpoints L_i and R_i $(L_i < R_i, |L_i|, |R_i| \leq 5 \cdot 10^4)$. There are at most 10^5 segments.
- one line containing "0 0", indicating the end of the input.

Output

Output the minimum number of segments to cover the segment $[0, M]$. If there is no solution output "`No solution`"

Sample input and output

Input	Output
1 -1 0 -5 -3 2 5 0 0	`No solution`
1 0 1 0 0	1

Problem C: Kamikaze Kombat

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

You are one of the developers behind the extremely popular new fighting game *Kamikaze Kombat*. In the game, the players engage in long one-versus-one combats in which they can use a large variety of different moves, each denoted by a capital letter (e.g. `B` for a bite or `H` for a headbutt).

After one of the combatants is knocked out, points are awarded based on their performance. Besides the points awarded for certain combinations of moves, good timing etc., there is a special mechanism for bonus points. So far, the bonus is awarded whenever the (relatively long) pattern P occurred in the sequence of moves a player made during the fight. However, the *Kamikaze Kombat* community quickly reached the consensus that this mechanism was way too restrictive and began urging the developers (including you) to relax the requirement. In your last meeting, you and the rest of the developing team agreed that from now on the bonus should be awarded for each contiguous subsequence of a player's move sequence that has the same length as the pattern P and contains each move exactly as often as P contains it. Now it is up to you to implement a procedure that quickly calculates for any given move sequence how many of the new bonuses should be awarded for it.

Input

The input consists of

- one line containing N ($1 \leq N \leq 1000$) – the number of moves in the pattern P
- one line containing the pattern P – a string consisting of N uppercase letters
- one line containing M ($1 \leq M \leq 100$) – the number of player sequences you need to check

- M lines giving the queries, each of them containing first an integer n_i ($N \leq n_i \leq 10^5$) – the number of moves in the given player's move sequence – and then the move sequence itself, a string consisting of n_i uppercase letters.

Output

For each query output one line containing the number of bonuses that should be awarded for the given move sequence.

Sample input and output

Input	**Output**
2 AB 5 3 ACB 5 ABCAB 5 ABCBA 6 ABABAB 7 CDEFGHI	0 2 2 5 0
3 ABA 5 6 AAABBB 9 ABABABABA 9 ABCABCABC 3 AAB 11 AABAABAABAA	1 4 0 1 9

Problem D: Segment Tree

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

There are n non-empty segments $[\ell_1, r_1], ..., [\ell_n, r_n]$ with integer endpoints that are unique (i.e. $\#\{l_1, ..., l_N, r_1, ..., r_n\} = 2n$). They induce an undirected graph in the following way: For $1 \leq v \leq n$, segment $[\ell_v, r_v]$ is represented by vertex v and there is an edge between vertices v and u if and only if $[\ell_v, r_v]$ and $[\ell_u, r_u]$ intersect and neither segment lies fully inside the other. Given the segments, you are to determine whether this graph is a tree.

Input

The input consists of

- one line containing N ($1 \leq N \leq 5 \cdot 10^5$) – the number of segments
- N lines each giving a segment by means of its endpoints ℓ_i and r_i ($1 \leq \ell_i < r_i \leq 2n$).

Output

Output `YES` if the graph induced by the segments is a tree and `NO` otherwise.

Sample input and output

Input	Output
5 6 8 2 5 4 9 1 3 7 10	YES
5 6 8 2 5 4 10 1 3 7 9	NO
3 1 4 2 5 3 6	NO

Problem E: Educated Guess

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

You are playing `2D Physics`, a game that attempts to teach the player basic physics by translating it to two dimensions, visualizing it and letting the player solve small puzzles. Currently, you are trying to solve `Educated Guess`, a puzzle where a large polygon of known weight is about to be dropped into water of known density (weight per area) and you are supposed to guess two things: First you should predict how far the polygon will be immersed in water after being dropped and finding a stable position. Secondly, just to make sure you didn't use some dirty trick to arrive at the first result, you need to find as how many non-degenerate polygons the part under water will appear when this equilibrium is reached (e.g. for a convex polygon this is always one; for a comb with teeth pointing downwards and a density such that the shaft isn't immersed, it's the number of teeth). To make things a bit easier, the polygon cannot rotate, its orientation is constant throughout the procedure. You already figured out that in this stable position there will be an equilibrium between the buoyancy pushing the polygon up and the gravity pulling it down (as these are the only two forces considered in the puzzle). Furthermore, you remember the formula for the immersed area to be

$$A_{\text{imm}} = \frac{\rho_{\text{poly}}}{\rho_{\text{water}}} \cdot A_{\text{total}},$$

where A_{imm} is the immersed area, that is, the polygon area below the water surface, ρ_{poly} is the density of the polygon (in weight units per area unit), ρ_{water} is the density of the water (same unit) and A_{total} is the overall area of the polygon. It is known that $\rho_{\text{poly}} < \rho_{\text{water}}$. Unfortunately you are having some difficulties putting all of this together in your head. So you decided to write a little program...

Input

The input consists of

- one line containing two numbers W_{poly} and ρ_{water} ($0 < W_{\text{poly}} \leq 10^6$, $0 < \rho_{\text{water}} \leq 10$)– the total weight of the polygon and the density of the water as described above
- one line containing N ($3 \leq N \leq 10^5$) – the number of nodes in the polygon
- N lines giving the coordinates of the polygon before it is dropped, with the i-th line containing integers x_i and y_i ($0 < x_i, y_i \leq 10^7$) – the coordinates of the i-th node. The nodes are given in either clockwise or counter-clockwise order. The water surface is described by the x-axis, i.e. the set $\mathbb{R} \times \{0\} = \{(x, 0) \mid x \in \mathbb{R}\}$.

Output

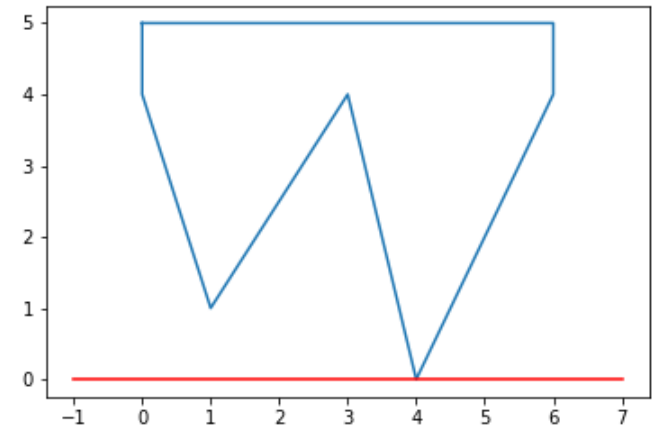
First output the y-coordinate of the lowest point of the polygon after it is dropped into the water and reaches the equilibrium between gravity and buoyancy (with precision at least 10^{-4}).

Next output the number of non-degenerate polygons as which the immersed part of the polygon appears when disregarding the part above the surface. Note that, for the purposes of this problem, we call a polygon non-degenerate if it has non-zero area and non-zero width everywhere (e.g. if the polygon from the sample was immersed precisely up to the vertex with original coordinates $(3, 4)$, this part of the answer would still be 2, but if it was immersed any further, there would be a non-zero-width-part connecting the two components, so the answer would become 1).

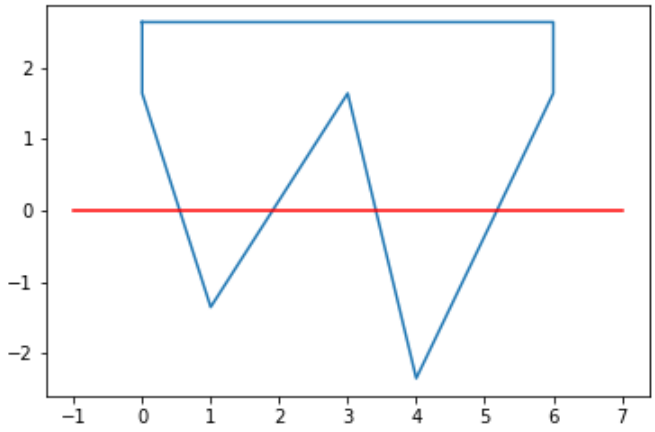
Sample input and output

Input	Output
3.0 1.0 7 0 5 6 5 6 4 4 0 3 4 1 1 0 4	-2.35571371 2

Initial configuration:



Configuration after the polygon is dropped and stabilizes its position:



Problem F: Patio

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

As the summers tend to get hotter and hotter, you decided to equip your patio with two large, convex, polygonal canvasses. You already made some sketches of where exactly to put them, but you are having doubts they will provide enough shadow, especially since the areas they shield from the sun have some overlap. Thus, before you start implementing your plan, you want to calculate the ratio of shadowy area to overall area of the patio that would result from it, and – just to be safe – the shadowy area itself.

Input

The input consists of

- one line containing four integers n, m ($3 \leq n, m \leq 1000$) – the number of corners in the canvasses – and w, l ($2 \leq w, l \leq 10^7$) – the width and length of the patio (it encompasses the area $[0, w] \times [0, l]$)
- n lines giving the corner coordinates of the first canvas in clockwise order – all corners are located in integer coordinates and for each corner (x_i, y_i) it is guaranteed that $0 \leq x_i \leq w$ and $0 \leq y_i \leq l$
- m lines giving the corner coordinates of the second canvas in clockwise order – all corners are located in integer coordinates and for each corner (x_i, y_i) it is guaranteed that $0 \leq x_i \leq w$ and $0 \leq y_i \leq l$

It is guaranteed that the canvasses are convex and that there is some overlap (with non-zero area) between them. Furthermore, the corner coordinates are unique (i.e. there is no point in which both canvasses have a corner).

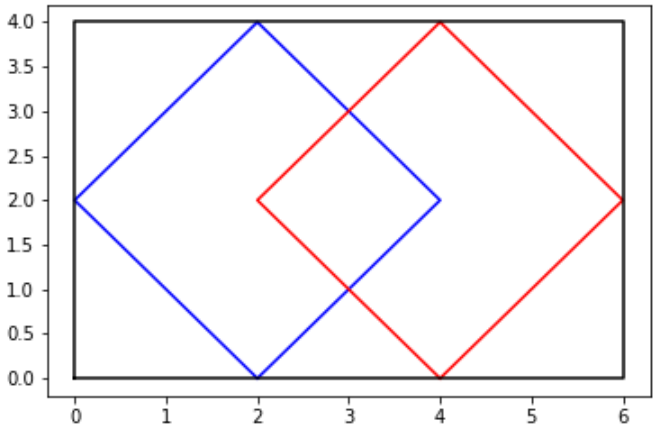
Output

First print fraction between shadowy area and total area under the given plan. Then output the shadowy area, by first printing the number s of its corners and then s lines giving the corners in clock-wise order, beginning with the lexicographically smallest one (to make the result unique). Ratio and coordinates should have an absolute or relative precision of 10^{-4}.

Sample input and output

Input	Output
4 4 6 4 2 0 0 2 2 4 4 2 4 0 2 2 4 4 6 2	0.58333333 8 0 2 2 4 3 3 4 4 6 2 4 0 3 1 2 0

Visualization of the Sample:



Remark: $0.58\overline{3} = 14/24$