Problem Bugatti. Cutting the Cake

Input file:	`input.txt` or standard input
Output file:	`output.txt` or standard output
Time limit:	2 seconds
Memory limit:	512 megabytes

This problem can only be solved in the C++ programming language.

The organizers of the Closed Olympiad for school students in programming decided to order a large cake for the evening banquet. They ordered a cake with n candles numbered from 0 to $n-1$. It is known that the candles are located at different points on the plane, no three candles lie on the same line, and no four candles form a trapezoid (a quadrilateral with two parallel sides).

To ensure that everyone gets a piece of cake, the organizers want to pre-construct a way to cut the cake into the maximum number of pieces. Unfortunately, you do not know the positions of the candles on the cake, and the only way to find out what the cake looks like is by corresponding with the pastry shop via email.

In one message, you can request to weigh no more than four triangular pieces, with vertices at the candles:

- You provide two lists of triangles `left` and `right`, containing no more than four triangles in total. The triangles may intersect, share common candles, and even coincide.
- The pastry shop will cut the triangular pieces according to the requested lists from several copies of the cake and place the pieces from `left` on the left scale and the pieces from `right` on the right scale.
- You will receive the result of the weighing. We will consider that the weight of a piece equals its area. As a result, you will find out which of the triangle lists has a greater total area, or if the areas are equal.

After several weighings, you must find a way to cut the cake. Each piece must be a convex polygon with vertices at the candles. The vertices of the piece must be listed in the order of traversal either clockwise or counterclockwise. Any two pieces must not intersect at internal points. You must return a list of pieces whose total area is maximally possible. Also, in some groups, it is required to maximize the number of pieces.

Solution Format

This is an unusual problem. It has a testing format with a grader, where you need to implement only the function `solve` with the solution. This function will be called by the testing program of the jury (the grader), and the return value of the function will be accepted as the solution to the problem.

In particular, this means that the code you submit **must not contain input or output**. Your code **must not** contain a `main` function. If necessary, you can implement any number of helper functions, structures, classes, and global variables, but all the code of your solution must be in one file.

You must implement the following function:

```
std::vector<std::vector<int>> solve(int n);
```

The function `solve` takes a single integer n — the number of candles.

In the implementation of the function `solve`, you can use the function `compare`, which is implemented by the grader:

```
int compare(const std::vector<Triangle>& left, const std::vector<Triangle>& right);
```

This function takes two non-empty lists of triangles with a total size of no more than 4. It returns -1 if the total area of the triangles in `left` is less than the total area of the triangles in `right`, 0 if the

areas are equal, and 1 otherwise. If an incorrect request is made or the limit on the number of requests is exhausted, the program will terminate automatically.

To access the function `compare` in your solution, the first line of your code must include the header file with the following line:

```
#include "triangles.h"
```

In this file, the structure `Triangle` used in the function `compare` is defined as follows:

```
struct Triangle {
    int i, j, k;

    Triangle() = default;
    Triangle(int i_, int j_, int k_) : i(i_), j(j_), k(k_) {}
};
```

This structure describes a single triangular piece, where the parameters `i`, `j`, and `k` describe the indices of the candles that form the vertices of the triangular piece. The indices of the candles must be distinct for a single triangular piece, but they can repeat in multiple triangles.

You do not need to include the definition of the structure `Triangle` in the code you submit; it will be automatically taken from the header file.

All parameters (indices of candles in requests, as well as in the result you return) are specified in **0-indexing**.

Your function `solve` should use calls to the function `compare` to find any partitioning of the cake into convex pieces, where any two pieces do not intersect at internal points, the total area of the pieces is maximized, and possibly the number of pieces is maximized.

The function `solve` should return a list of pieces into which the cake is divided. Each piece is described by a structure `std::vector<int>`, consisting of the candles that form the convex polygon. The candles must be listed in **the order of traversal along the boundary of the polygon** (either clockwise or counterclockwise). Any two pieces must not intersect at internal points. You must return a list of pieces whose total area is maximally possible, and in some groups, it is additionally required to maximize the number of pieces while maintaining the condition of maximizing the total area.

During one run of the grader, the **jury will make exactly one call to the function `solve`**. The **grader is not adaptive**, meaning that the positions of the candles are fixed in advance and do not depend on the implementation of the function `solve`.

Testing

You are provided with a solution template `triangles.cpp`, as well as a header file `triangles.h`, containing the definitions of the functions `solve` and `compare`. For convenience in testing, you are provided with a grader — the file `grader.cpp`. This file implements reading input data from the standard input stream, calling the function `solve`, and outputting the result of the function `solve` to the standard output stream. In the testing system, the grader may differ.

To compile your code `triangles.cpp` in C++, use the command

```
g++ -std=c++20 grader.cpp triangles.cpp -o grader
```

After executing this command, an executable file of the grader `grader` or `grader.exe` will be created, depending on your operating system, which you can run to input tests in the specified format.

If compilation via commands causes you difficulties, for local testing, you can copy the implementation of the function `solve` into the file `grader.cpp` (it must be inserted before the function `main`) and run the file `grader.cpp`. In this case, before submitting the solution to the testing system, you will need to leave only the implementation of the function `solve`, **remembering to include the header file at the beginning of the code** (this is done by adding the line `#include "triangles.h"` at the beginning

of the code you submit). If you are using any C++ libraries, their inclusion must also be added at the beginning of the submitted code.

In case of receiving a compilation error verdict, ensure that **the code you submit does not contain a `main` function, nor does it contain definitions of the function `compare` and the structure `Triangle`**. You can only use the function `compare` and the structure `Triangle` in the code you submit.

Input

The grader reads the test in the following format:

The first line contains an integer n ($4 \le n \le 10\,000$) — the number of candles.

In the next n lines, there are two integers x_i, y_i ($-10^9 \le x_i, y_i \le 10^9$) — the coordinates of the i-th candle.

Output

The grader outputs the results of the function `solve` — the list of found pieces.

In the first line, the number of found polygons (the size of the vector returned by the function `solve`) is printed.

In the following lines for each polygon (an element in the vector returned by the function `solve`), the size of the polygon is printed first, followed by the line of indices of the candles that are the vertices of the polygon (elements of each `std::vector<int>` in the return value of the function `solve`). Each polygon must be convex, and its vertices must be listed in the order of traversal (either clockwise or counterclockwise). No two polygons should intersect at internal points.

In the file `grader.cpp`, there is a variable `verbose`, which is initially set to 0. By increasing its value, the grader will write more detailed information about your solution and its requests.

Examples

input	output
4 1 2 1 4 0 0 3 -1	3 3 0 1 2 3 0 2 3 3 0 1 3
5 -1 -1 4 4 4 -2 1 2 -2 2	1 4 0 4 1 2
6 2 2 0 -2 -1 3 -2 0 7 0 2 -3	4 3 2 4 3 3 3 4 5 3 1 3 5 3 0 2 4

Note

In the first example, a possible sequence of function calls could look like this:

Initially, the function `solve(4)` is called.

From the function `solve`, the function

```
compare({Triangle(0, 1, 2)}, {Triangle(1, 2, 3)})
```

is called.

During the execution of the function, the sizes of the triangular pieces formed by the triangle with vertices at candles numbered 0, 1, and 2 and the triangle with vertices at candles 1, 2, and 3 are compared.

Since the first triangle is smaller than the second, the function returns -1.

Next, from the function `solve`, the function

```
compare({Triangle(1, 2, 3)}, {Triangle(0, 1, 2), Triangle(0, 1, 3)})
```

is called.

The response will return 1, as the area of the triangle with vertices at candles 1, 2, and 3 is greater than the total area of the triangle with vertices at candles 0, 1, and 2 and the triangle with vertices at candles 0, 1, and 3.

Next, from the function `solve`, the function

```
compare({Triangle(1, 2, 3)}, {Triangle(0, 1, 2), Triangle(2, 3, 0), Triangle(0, 1, 3)})
```

is called.

The response will return 0.

In the end, the function `solve` returns `{{0, 1, 2}, {0, 2, 3}, {0, 1, 3}}` — a description of the partitioning of the cake into convex pieces that have the maximum possible total area and do not intersect at internal points.

In the first and third examples, the found method of cutting the cake has the maximum possible number of pieces, so such answers would be considered correct in groups with maximization. In the second test, the number of pieces is not maximally possible, so such an answer would be considered correct in groups without maximization, but would receive 0 points in groups with maximization.

Note that the first test fits the additional constraints of groups 1 and 2, while the third test fits the additional constraints of group 5.

Scoring

The tests for this problem consist of 11 groups. The rules by which points are awarded for groups are described below. Note that passing the tests from the statement is not required for some groups. The final score for each group is equal to the maximum score obtained for that group of tests across all submissions.

The score for a solution for a group of tests is equal to the minimum score obtained across all tests in the group.

- If your solution makes an incorrect call to `compare` or returns an answer that does not satisfy the mandatory conditions, it will receive 0 points for the test.
- In each test, your solution is allowed to make no more than $2 \cdot 10^6$ calls to the function `compare`. If the limit is exceeded, your solution will receive 0 points for the test.
- In some groups, it is required to maximize the number of pieces in the answer. In such groups, if the number of pieces is not maximally possible, your solution will receive 0 points for the test.

If none of the described conditions for the test are violated, the answer for the test is considered correct. Let the full score for the group of tests be p. Some groups are evaluated using a formula, while others are evaluated without using a formula:

- If the group is evaluated without using a formula, the score for the test is equal to p.
- Otherwise, if the group is evaluated using a formula, let q be the number of calls to `compare` made by your solution. Then the score for the test is equal to $\left\lfloor p \cdot \frac{20n}{\max(q, 20n)} \right\rfloor$.

Group	Full score	Score		Additional constraints	Required groups	Comments
		Max.	Formula	n		
0	0	no	no	–	–	Tests from the statement.
1	13	no	no	$n = 4$	–	
2	4	yes	no	$n = 4$	1	
3	13	no	no	–	–	$(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$ are in the set, the rest are random points inside this triangle
4	8	yes	no	–	3	$(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$ are in the set, the rest are random points inside this triangle
5	11	yes	yes	–	–	points are the vertices of a convex polygon
6	9	no	no	$n \le 100$	–	random points
7	8	yes	no	$n \le 100$	6	random points
8	9	no	yes	–	–	random points
9	8	yes	yes	–	–	random points
10	9	no	yes	–	–	
11	8	yes	yes	–	–	

In groups 6–9, all points were chosen randomly and independently from the square $[-10^9, 10^9] \times [-10^9, 10^9]$.

In groups 3–4, points other than $(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$ were chosen randomly and independently from the triangle with vertices $(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$.

In these groups, among such random tests, only those where all points are distinct, no three lie on the same line, and no four form a trapezoid are retained.

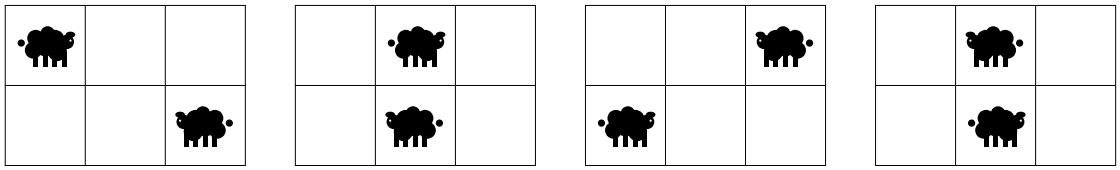
Problem Ferrari. March of the Sheep

Input file:	`input.txt` or standard input
Output file:	`output.txt` or standard output
Time limit:	1 second
Memory limit:	512 megabytes

Sasha is tired of the heavy city life and decided to move to the countryside. To occupy his time, he decided to raise sheep. For this purpose, he acquired a rectangular field in the village, which can be represented as a grid of size $n \times m$, where the rows are numbered from top to bottom with numbers from 1 to n, and the columns are numbered from left to right with numbers from 1 to m.

On his field, Sasha bought n sheep, each of which he assigned a whole row. Initially, the i-th sheep was placed in the cell with coordinates (i, a_i) (in row number i and column number a_i). Sasha found out that the sheep move only horizontally according to the following rules:

- If there is exactly one column in the field, the sheep do not move.
- Initially, the i-th sheep is in cell (i, a_i), and each sheep has an initial direction in which it moves — either left or right. There is no sheep that is in the first column and moves left, nor is there any sheep that is in the last column and moves right.
- Every second, each sheep moves one column to the left if it is moving left; otherwise, it moves one column to the right.
- If after moving a sheep ends up in column 1 and is moving left, or if a sheep ends up in the last column and is moving right, it changes its direction to the opposite. Thus, the sheep never leave the boundaries of the field.



Here is an example of the sheep's movement, where initially the first sheep was in column 1 and moved to the right, and the second was in column 3 and moved to the left. The first picture shows the state at the beginning, the second after one second, the third after 2 seconds from the start of movement, and the fourth after 3 seconds from the start of movement.

Sasha does not like that the sheep move so chaotically; he wants all the sheep to move uniformly. This means that all sheep should be in the same column and have the same direction of movement. To achieve this, Sasha can trim his field several (possibly zero) times as follows:

- He chooses a time t (how many seconds have passed since the initial start of the sheep's movement) and the number of columns x that will remain in the field after trimming.
- All sheep at time t must strictly lie within the field of size $n \times x$. This means that for any i from 1 to n, the i-th sheep at time t must be in cell (i, y_i), where $1 \le y_i \le x$. Otherwise, such a trimming of the field is not allowed.
- After this operation, starting from time t, the number of columns on the field decreases to x.
- Each sheep that is in the last column and moves to the right changes its direction to the opposite.

Detailed examples of the trimming process are described in the notes section.

Sasha wants to know if it is possible to achieve that all sheep move uniformly by trimming the field several times. If this is possible, Sasha wants to know what the maximum number of columns that can remain on the field is and how exactly he can achieve this. Help Sasha solve this problem!

Input

The first line contains one integer T ($1 \le T \le 3$) — a parameter indicating what information about the answer needs to be output. A detailed description is provided in the "Output format".

The second line contains two integers n and m ($2 \le n \le 200,000$, $2 \le m \le 10^9$) — the number of rows and columns of the field.

The third line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le m$) — the column numbers where the sheep are initially located.

The fourth line contains a string s of length n, consisting only of the characters `L`, `R`, where the i-th character is `L` if the i-th sheep is initially moving left, and `R` if the i-th sheep is initially moving right. It is guaranteed that no sheep in the first column is moving left, and no sheep in the m-th column is moving right.

Output

In the first line, output "No"(without quotes) if Sasha cannot achieve that all sheep move uniformly. Otherwise, output "Yes"(without quotes).

If $T = 2$ or $T = 3$, then in the second line output the maximum number of columns that Sasha can leave on the field so that all sheep move uniformly. If $T = 1$, then **this should not be output**.

If $T = 3$, in the third line output the number q ($0 \le q \le 10^6$) — how many times Sasha will need to trim the field. In the next q lines, output the description of the trimmings.

For the description of each trimming of the field, output in one line two integers t and x ($0 \le t \le 10^{18}$, $1 \le x < m$), where t is the time at which Sasha needs to trim the field (from the very beginning of the sheep's movement), and x is the number of columns that will remain on the field after this operation.

The operations must be output in non-decreasing order of t. In each subsequent trimming, x must be less than in the previous one.

If there are multiple suitable sequences of trimmings, any of them can be output.

It can be shown that under the given constraints, if a solution exists, then there exists a solution that fits within the given limits.

In case $T = 1$ or $T = 2$, do not output the description of the trimmings.

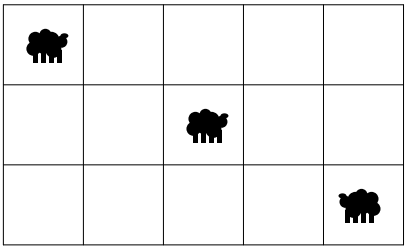
Examples

input	output
3 2 3 1 3 RL	Yes 2 1 3 2
3 2 3 1 2 RL	No
3 3 5 1 3 5 RRL	Yes 3 2 1 4 2 3
2 3 5 1 3 5 RRL	Yes 3
1 3 5 1 3 5 RRL	Yes
3 3 7 3 3 5 RRL	Yes 4 3 0 6 0 5 1 4

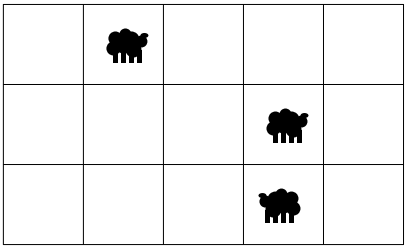
Note

Consider the third example:

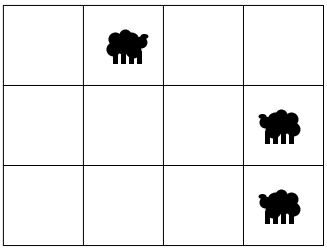
The states of the sheep at zero seconds.



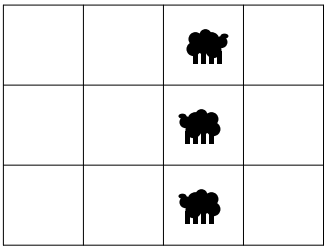
The state of the sheep at one second.



Trimmed the field to 4 cells.



The state of the sheep at two seconds.



Scoring

The tests for this problem consist of ten groups. Points for each group are awarded only if all tests of the group and all tests of some of the previous groups are passed. Note that passing the tests from the statement is not required for some groups. **Offline testing** means that the results of testing your solution on this group will become available only after the end of the competition. The final score for each group equals the maximum score obtained for that group of tests across all submissions.

Group	Points	Additional constraints			Necessary groups	Comment
		T	n	m		
0	0	–	–	–	–	Tests from the statement.
1	8	$T \le 1$	–	–	–	
2	11	$T \le 2$	$n \le 3$	$m \le 4$	–	
3	9	–	$n = 2$	–	–	$a_1 = 1, a_2 = m$
4	12	–	$n = 2$	$m \le 200,000$	–	
5	8	–	$n = 2$	–	3, 4	
6	14	$T \le 2$	$n \le 1000$	$m \le 1000$	2	
7	10	$T \le 2$	–	–	1, 2, 6	
8	9	–	–		–	The string s contains only the character `R`.
9	12	–	$n \le 1000$	$m \le 1000$	0, 2, 6	
10	7	–	–	–	0 – 9	**Offline testing.**

Problem Lamborghini. Decoration

Input file:	`input.txt` or standard input
Output file:	`output.txt` or standard output
Time limit:	1 second
Memory limit:	512 megabytes

During the preparation for the Closed Olympiad of School Students in Programming, the organizers decided to decorate the auditorium. For this purpose, $2n$ pegs were hammered into a straight line on the wall. All pegs are in different positions. Between them, n strings were stretched, the i-th of which was stretched between the pegs at distances l_i and r_i from the start of the wall ($l_i < r_i$). It is known that each peg has exactly one string stretched to it.

The strings can undergo operations called *restringing*. In one *restringing* operation, you can choose two strings (l_i, r_i) and (l_j, r_j), remove them from the pegs, and then hang two new strings, using each of the four freed pegs exactly once. That is, from the four freed pegs, two new pairs are formed, between which new strings are stretched.

The strings stretched between the pegs (l_i, r_i) and (l_j, r_j) intersect if the segments $[l_i, r_i]$ and $[l_j, r_j]$ have at least one common point. A configuration of strings is considered to have beauty at least k if there exists a set of k strings, each pair of which intersects. Note that if a configuration has beauty k, it also has beauty at least $k-1,\ k-2,\ \ldots,\ 0$.

The organizers of the Olympiad have q queries for obtaining a configuration of a certain beauty as a result of several *restringing* operations. In the i-th query, they would like to achieve beauty at least k_i. For each query, the organizers want to know the minimum number of *restringing* operations required to achieve this. The queries are independent of each other, meaning that the *restringing* operations between queries are not preserved.

Input

The first line contains two integers n and q ($1 \le q \le n \le 200\,000$) — the number of strings and the number of queries.

The next n lines describe the strings. In the i-th line, there are two integers l_i and r_i ($1 \le l_i < r_i \le 10^9$) — the numbers of the pegs connected by the i-th string. It is guaranteed that each peg appears exactly once.

The next line contains q integers $k_1, k_2, \ldots, k_q$ ($1 \le k_i \le n$) — the desired beauty sizes in the queries from the organizers. It is guaranteed that all k_i are distinct.

Output

For each query, output a single non-negative integer — the minimum number of *restringing* operations that need to be performed to obtain a configuration of strings with the required beauty as specified in the query.

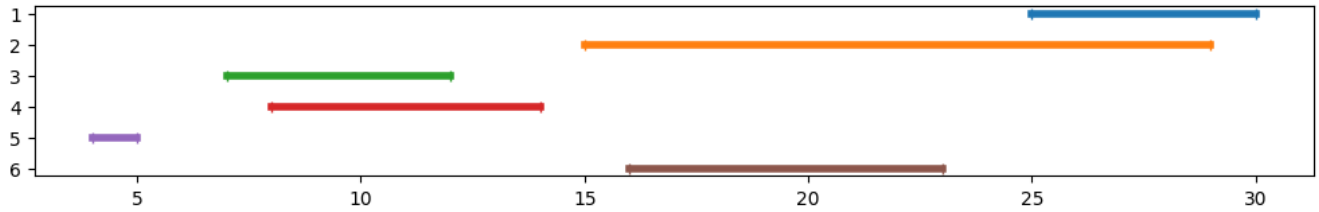
It is guaranteed that for each query, it is possible to perform several restringings to achieve a configuration of the required beauty.

Example

input	output
6 6 25 30 15 29 7 12 8 14 4 5 16 23 1 2 3 4 5 6	0 0 1 1 2 3

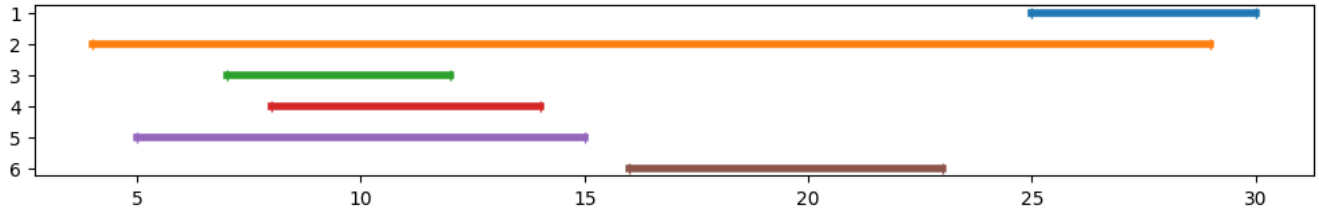
Note

In the first example, the strings initially have the following configuration:

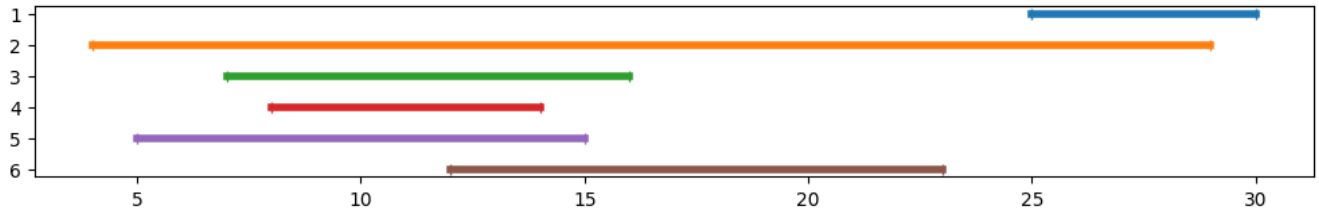


Since strings 3 and 4 already intersect, no operations are needed to achieve beauty at least 1 and beauty at least 2.

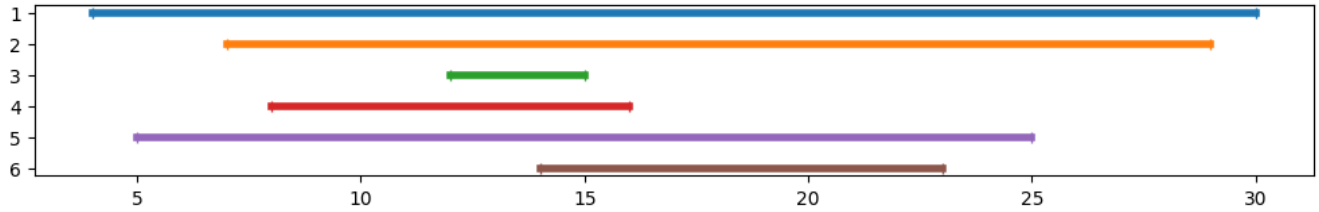
To achieve beauties 3 and 4, you can perform a *restringing* operation on strings 2 and 5 to obtain strings $(4, 29)$ and $(5, 15)$.



To achieve beauty 5, you can additionally perform an operation on strings 3 and 6 to obtain $(7, 16)$ and $(12, 23)$.



To make all 6 strings intersect, you can perform operations on strings 1 and 5, 2 and 3, 4 and 6 to obtain the following configuration:



Scoring

The tests for this problem consist of six groups. Points for each group are awarded only if all tests of the group and all tests of some previous groups are passed. Note that passing the tests from the statement is not required for some groups.

Group	Points	Additional constraints			Required groups	Comment
		n	q	k_i		
0	0	–	–	–	–	Tests from the statement.
1	14	$n \le 100$	–	–	0	–
2	16	$n \le 3000$	–	–	0, 1	–
3	13	–	–	–	–	Strings do not intersect pairwise.
4	25	–	$q = 1$	$k_i = n$	–	–
5	17	–	$q = 1$	$k_i \le 10$	–	–
6	15	–	–	–	0 – 5	–

Problem McLaren. Monsters and Swords

Input file:	`input.txt` or standard input
Output file:	`output.txt` or standard output
Time limit:	1 second
Memory limit:	512 megabytes

Given n monsters arranged in a row, for each monster, two values are known: h_i — the health of the monster and r_i — the reward for defeating this monster, expressed in coins. The knight must defeat all the monsters.

For fighting the monsters, the knight has m types of swords, and for each type of sword, its characteristics are known: s_j — the strength of the sword and c_j — the price of the sword in coins. To purchase a sword with price c_j, the knight must have at least c_j coins. After purchasing a sword, the knight's coin count decreases by c_j. Initially, the knight has x coins.

After purchasing a sword, it can be used no more than k times in battles against the monsters. Any type of sword can be purchased an arbitrary number of times. A sword with strength s_j can kill a monster with health h_i if $s_j \geq h_i$. At any moment, the knight can only possess one sword, meaning that after purchasing a new sword, the old sword can no longer be used (but it can be purchased again in the future).

The monsters must be defeated in a fixed order: from the first to the last. It is required to determine whether the knight can do this.

Input

The first line contains four integers n, m, k, and x ($1 \leq n, m \leq 500\,000$, $1 \leq k \leq n$, $1 \leq x \leq 10^9$) — the number of monsters, the number of sword types, the maximum number of times a sword can be used, and the initial number of coins the knight has.

In the following n lines, the descriptions of the monsters are given. In the i-th line, there are two integers: h_i and r_i ($1 \leq h_i, r_i \leq 10^9$) — the health of the i-th monster and the reward for defeating it.

In the next m lines, the characteristics of the swords are described. In the j-th line, there are two integers s_j and c_j ($1 \leq s_j, c_j \leq 10^9$) — the strength and price of the j-th sword.

Output

Print «`Yes`» (without quotes) if the knight can defeat all the monsters, and «`No`» (without quotes) otherwise.

Examples

input	output
3 3 2 5 1 1 5 5 9 5 9 10 1 1 5 1	Yes
5 5 5 6 3 6 2 4 1 1 4 4 8 4 2 7 7 7 3 4 5 9 6 4	No

Note

In the first example, the knight can first buy the third sword, using it to defeat the first and second monsters. After that, he will have $5 - 1 + 1 + 5 = 10$ coins. This allows him to buy the first sword and defeat the last monster.

In the second example, the knight cannot defeat all the monsters because there is no type of sword that can defeat the fifth monster.

Scoring

The tests for this problem consist of 9 groups. Points for each group are awarded only if all tests of the group and all tests of some previous groups are passed. Note that passing the tests from the statement is not required for some groups. Offline testing means that the results of testing your solution on this group will only be available after the competition ends. The final score for each group is equal to the maximum score obtained for this group of tests across all submitted solutions.

Group	Points	Additional constraints			Required groups	Comment
		n	m	k		
0	0	–	–	–	–	Tests from the statement.
1	11	–	–	$k = 1$	–	–
2	9	$n \le 100$	$m \le 100$	$k = n$	–	–
3	14	$n \le 100\,000$	$m \le 3000$	$k = n$	2	–
4	16	–	–	$k = n$	2, 3	–
5	7	$n \le 400$	$m \le 400$	–	0, 2	–
6	8	$n \le 3000$	$m \le 3000$	–	0, 2, 5	–
7	10	$n \le 150\,000$	$m \le 150\,000$	–	0, 2, 3, 5, 6	–
8	12	$n \le 300\,000$	$m \le 300\,000$	–	0, 2, 3, 5 – 7	–
9	13	–	–	–	0 – 8	**Offline testing.**