Tsinghua Bootcamp 2024. Day 4

A. Card Game

1 second, 1024 megabytes

You are playing a computer game, and you need to defend against attacking monsters. You have a collection of cards. The main parameter of the hero in this game is their *strength*.

At the beginning of each battle, the strength is equal to 1. In the battle, you can use cards of three types:

1. A card that increases your strength. There are n such cards in the collection; the i-th of them increases the strength by a_i;
2. A card that doubles your strength — there are d such cards in the collection;
3. A card whose effects do not affect your strength — there are many such cards in the collection, and their quantity can be considered unlimited.

You will sequentially battle with q monsters. To battle each monster, you need to create a deck of cards from the given collection. At the beginning of the battle, you draw the initial set of cards from the assembled deck.

Before the battle with the j-th monster, you will know three parameters m_j, s_j, and x_j:

1. m_j is the size of the deck you must assemble from the collection of cards for battle with the j-th monster;
2. s_j is the number of cards randomly and uniformly drawn from the deck at the beginning of the battle with the j-th monster;
3. x_j is the minimum strength needed to defeat the j-th monster on the first turn.

Since the monsters are very strong, you want to assemble a deck to win on the first turn. For this, for any random set of s_j cards drawn from the assembled deck at the beginning of the battle, there should be a possibility to make your hero's strength at least x_j using these cards (after the cards are drawn, you may use them in an arbitrary order).

You start each battle with your strength equal to 1, and you want to know if you can guarantee victory on the first turn.

Input

The first line contains two integers n and d — the number of cards that increase and double the strength, respectively ($1 \le n, d \le 2 \cdot 10^5$).

The next line contains n integers a_i, where a_i is the value by which the i-th card of the first type increases your strength ($1 \le a_i \le 10^6$).

The third line contains an integer q — the number of monsters you will have to battle ($1 \le q \le 2 \cdot 10^5$).

Each of the next q lines contains three integers m_j, s_j, and x_j — the size of the deck, the initial number of cards drawn from the deck, and the minimum required strength to be gathered on the first turn in the battle with the j-th monster ($1 \le s_j \le m_j \le 2 \cdot 10^5$; $1 \le x_j \le 10^6$).

Output

Your program should output q lines, each containing the string "YES" if it is possible to create the required deck, and "NO" otherwise.

input

```
4 2
6 2 3 2
3
7 3 8
4 2 5
4 3 15
```

output

```
NO
YES
YES
```

input

```
3 200
3 5 9
3
4 3 24
4 3 25
100 97 999999
```

output

```
YES
NO
YES
```

Consider the first sample test.

You have 2 cards that double the strength and 4 cards that increase the strength $[2, 2, 3, 6]$.

- In the first battle, it is necessary to have strength of at least 8 using 3 starting cards out of 7 in the deck.
  It is advantageous to put all the doubling and all the strength-increasing cards in the deck, as well as one card that does not affect the strength.

  Consider the worst-case scenarios for the starting set of cards: a card that does not affect the strength and

  - strength-increasing cards of 2 and 2 — the total strength is 5;
  - strength-increasing cards of 2 and 3 — the total strength is 6;
  - two doubling cards — the total strength is 4;
  - a strength-increasing card of 2 and a doubling card — the total strength is 6.

- In the second battle, it is necessary to have strength of at least 5 using 2 starting cards out of 4 in the deck.
  One of the possible deck configurations that always defeats the monster is

  - one doubling card;
  - and strength-increasing cards of 2, 2, and 3.

  In this case, the starting hand will contain

  - either any two strength-increasing cards — the total strength will be at least $1 + 2 + 2 = 5$;
  - or any strength-increasing card and a doubling card — the total strength will be at least $(1 + 2) \cdot 2 = 6$.

- In the third battle, it is necessary to gather at least 15 strength using 3 starting cards out of 4 in the deck.
  One of the possible deck configurations that always defeats the monster is

  - two doubling cards;
  - and strength-increasing cards of 3 and 6.

  In this case, the starting hand will contain

  - either a strength-increasing card of 3 and two doubling cards — the total strength will be $(1 + 3) \cdot 2^2 = 16$;
  - or strength-increasing cards of 3 and 6, but one doubling card — the total strength will be $(1 + 3 + 6) \cdot 2 = 20$.

B. Symmetric Race

4 seconds, 1024 megabytes

Every year in Bytecity there is a paired race through n locations numbered from 1 to n. Participants are grouped in pairs with the first participant in the pair running along the *blue* route and the second participant running along the *red* route.

For each of the two routes it is known in which direction to move and which location has to be ultimately reached. Specifically, along the blue route one must proceed from location i directly to location b_i, and along the red route — directly to r_i. The final target of the blue route is located at location B and the target or the red route is located at location R. After arriving at the final location of the route, there is no need to move further.

Three pairs participate in the final round. Both participants of each pair start from the same location, but different pairs must start from different locations. It is known that for each participant of the finals moving along the blue route from location i to location b_i or along the red route from i to r_i takes exactly one minute for any i. All participants start at time 0 and at the moment of each participant's arrival at their final location, the duration of their race is recorded.

It is not guaranteed that location B is reachable from any location along the blue route or that location R is reachable from any location along the red route, but according to the rules of the race the starting locations are necessarily chosen so that both B and R are reachable from them. Also the rules of the race do not prohibit the starting location from being equal to B or R.

When the finals were over, it turned out that the race durations of the participants running along the blue route coincide with the race times of the participants running along the red route as multisets (same as a set in which each element can occur more than once). In other words, if $t_{i,j}$ is the arrival time at the final location of the j-th participant of the i-th pair then $\{t_{1,1}, t_{2,1}, t_{3,1}\} = \{t_{1,2}, t_{2,2}, t_{3,2}\}$.

Find the number of unordered sets of three different locations from which the pairs could have started, such that the above condition on their race durations is satisfied. Two sets of three locations are considered different if there is at least one location in one of the sets that is not in the other.

Input

The first line of input contains three integers n, the number of locations ($3 \le n \le 10^5$), B, the location of the target of the blue route ($1 \le B \le n$), and R, the location of the target of the red route ($1 \le R \le n$). Note that it is not guaranteed that $B \neq R$.

The second line contains n integers b_i representing the numbers of the next locations along the blue route for each location from the first to the n-th ($1 \le b_i \le n; b_i \neq i$). The third line contains n integers r_i in the same format representing the numbers of the next locations along the red route ($1 \le r_i \le n; r_i \neq i$).

Output

Output a single integer — the number of sets of three different locations that could have been the starting locations satisfying the described conditions for the race result.

input
4 4 4 2 3 4 1 2 3 4 1
output
4

input
4 1 4 2 3 1 1 4 3 4 2
output
2

input
5 1 3 2 5 2 1 1 5 1 5 1 3
output
1

In the first example blue and red routes coincide, so any three different locations form a suitable set.

In the third example the only suitable set of locations is $(1, 2, 3)$. Then along the blue route the runners will spend 0, 2 and 3 minutes, respectively, and along the red route they will spend 2, 3 and 0 minutes, respectively.

C. Array Partitioning

1 second, 512 megabytes

You are given an array $A = [a_1, a_2, \ldots, a_n]$, containing n positive integers.

You need to color the elements of the array in two colors such that there are no two elements x and y of the same color such that x is divisible by y and the equality $\frac{x}{y} = p$ holds, where p is a *prime* number. It is guaranteed that such a coloring exists.

Recall that an integer $p > 1$ is called prime if it has exactly two divisors: 1 and p.

Input

The first line contains a single integer n ($1 \le n \le 10^5$) — the number of elements in the array.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^6$) — the elements of the array.

Output

Output the description of the partitioning of the array into two sets in the following format.

Output n integers, where the i-th integer equals 1 if the element a_i should be colored in the first color, and 2 if the element a_i should be colored in the second color.

If there are multiple suitable colorings, you may output any of them.

input
4 1 2 3 4
output
2 1 1 2

input
1 20
output
1

In the first example, there are two elements of the first color: 2 and 3, and two elements of the second color: 1 and 4. The elements of the first color do not divide each other. 4 divides 1, but their ratio is not a prime number.

D. Ultra mex

3 seconds, 512 megabytes

Consider A — a set of non-negative integers. The minimum non-negative integer that does not appear in A is denoted as $\operatorname{mex}(A)$. For example, $\operatorname{mex}(\{0, 1, 2, 4, 5, 9\}) = 3$. This function is often used, for example, in game theory.

We define another operation on the set A, which contains the number 0. This operation will be called "ultra". Let $m = \operatorname{mex}(A)$. Note that $m > 0$. We will construct a new set $\operatorname{ultra}(A)$ as follows: apply "bitwise exclusive or" with the number $(m - 1)$ to all elements of A. For example, $\operatorname{ultra}(\{0, 1, 2, 4, 5, 9\}) = \{0 \oplus 2, 1 \oplus 2, 2 \oplus 2, 4 \oplus 2, 5 \oplus 2, 9 \oplus 2\} = \{2, 3, 0, 6, 7, 11\}$. It can be shown that if the set A contains 0, then the set $\operatorname{ultra}(A)$ also contains 0.

Let us choose the set A_0, consisting of integers from 0 to $2^k - 1$ and containing 0. Consider the following sequence:

- $m_0 = \operatorname{mex}(A_0)$, $A_1 = \operatorname{ultra}(A_0)$
- $m_1 = \operatorname{mex}(A_1)$, $A_2 = \operatorname{ultra}(A_1)$
- ...
- $m_i = \operatorname{mex}(A_i)$, $A_{i+1} = \operatorname{ultra}(A_i)$
- ...

We will call the set A_0 mex-stable if starting from some index l, the numbers m_i stop changing. That is, for all $i \ge l$, we have $m_i = m_l$. The number m_l will be called the mex-limit of the set A_0.

You are given the numbers k, n, and p. Calculate the number of sets A_0 that:

- Consist of n distinct numbers from 0 to $2^k - 1$ (the number 0 must be included in A_0);
- Are mex-stable;
- The mex-limit of A_0 equals p.

Since the answer may be large, output it modulo M. It is guaranteed that $(M - 1)$ is divisible by 2^{18}.

Input

The first line contains one integer M — the modulus to compute the answer ($3 \le M \le 10^9$; $(M - 1)$ is divisible by 2^{18}). It is guaranteed that M is a prime number.

The second line contains one integer t — the number of test cases ($1 \le t \le 10^5$).

For each test case, a single line contains three integers k, n, and p ($1 \le k \le 17$; $1 \le n, p \le 2^k$).

Output

For each test case, output one integer on a new line — the number of sought sets A, taken modulo M.

input

```
998244353
6
3 2 1
3 2 2
3 2 3
3 2 4
3 5 1
4 6 1
```

output

```
6
1
0
0
29
2461
```

There are a total of 7 mex-stable sets of size 2 from numbers 0 to 7: $\{0,1\}$, $\{0,2\}$, $\{0,3\}$, $\{0,4\}$, $\{0,5\}$, $\{0,6\}$, $\{0,7\}$.

For $\{0,1\}$: $\mathrm{mex}(\{0,1\})$ = 2, $\mathrm{ultra}(\{0,1\}) = \{0 \oplus 1, 1 \oplus 1\} = \{1,0\} = \{0,1\}$, thus $A_1 = A_0$. Therefore, the mex-limit will be equal to 2.

For all other sets, $m_0 = \mathrm{mex}(A_0) = 1$, for them during the ultra calculation, XOR is performed with the number 0, so $\mathrm{ultra}(A_0) = A_0$. Thus, for them, the mex-limit equals $\mathrm{mex}(A_0) = 1$.

E. Airport Codes

4 seconds, 1024 megabytes

We will denote the length of a string s as $|s|$. For a string s, we will call any string that can be obtained from it by deleting any number of characters its *subsequence*. In other words, t is a subsequence of s if there exist $1 \le i_1 < i_2 < \ldots < i_{|t|} \le |s|$ such that $t_j = s_{i_j}$ for all j.

Also, we will say that the string s is *lexicographically smaller* than the string t if either s is a prefix of t, or the first differing character in s is smaller than the corresponding character in t. More formally, s is smaller than t if for some i it is true that $s_1 = t_1$, ..., $s_{i-1} = t_{i-1}$ and $s_i < t_i$ or $i = |s| + 1$.

You are given a long string s containing the names of different cities as its substrings. The *airport code* for a city can be chosen as follows:

1. Let the name of the city be formed by the characters s from the l-th to the r-th inclusive; let's denote it as t;
2. An integer k from 1 to $|t|$ is chosen;
3. The airport in this city is assigned a code equal to the lexicographically smallest subsequence of t of length k.

For example, the code for Paris ("`PARIS`") of length 3 is "`AIS`", the code of length 4 is "`ARIS`", and the code of length 5 is the same as the city name.

You receive q queries, each of which is defined by two triples (l_1, r_1, k_1) and (l_2, r_2, k_2). For each query, output which of the two airport codes is lexicographically smaller: the code of length k_1 of the city formed by the characters of s from the l_1-th to the r_1-th inclusive, or the code of length k_2 of the city at positions from l_2 to r_2 inclusive.

Input

The first line of input contains a string s consisting of capital letters of the English alphabet (characters from '`A`' to '`Z`') — the original string containing the names of cities ($1 \le |s| \le 10^6$).

The second line of input contains a single integer q — the number of queries to answer ($1 \le q \le 10^5$).

In the i-th of the following q lines, integers $l_{i,1}$, $r_{i,1}$, $k_{i,1}$, and $l_{i,2}$, $r_{i,2}$, and $k_{i,2}$ representing the boundaries of the city names and the lengths of their airport codes participating in the i-th query are listed ($1 \le l_{i,j} \le r_{i,j} \le |s|$; $1 \le k_{i,j} \le r_{i,j} - l_{i,j} + 1$).

Output

For each query, output on a separate line the symbol '<' if the first code from the query is lexicographically smaller, '>' if the second one is smaller, and '=' if the two codes from the query are equal.

input

```
SPBOSTONTARIOWASTRAHANNOVER
10
7 13 3 12 15 1
7 13 3 12 15 2
1 3 2 11 13 2
1 3 2 11 13 3
15 22 3 20 27 1
15 22 3 20 27 3
3 8 2 1 3 1
3 8 2 1 3 3
11 13 2 12 15 2
11 13 2 12 15 4
```

output

```
>
<
>
<
>
<
>
<
>
<
```

input

```
CCCBBBAAA
7
1 9 3 4 7 1
1 6 3 4 9 3
3 4 1 6 7 1
4 9 3 4 9 4
1 7 1 2 9 4
1 9 9 1 9 8
4 6 3 1 6 3
```

output

```
>
>
>
<
<
>
=
```

In the first example, the queries are grouped in pairs:

1. For "`ONTARIO`" the code of length 3 is "`AIO`", and for "`IOWA`" the codes of lengths 1 and 2 are "`A`" and "`IA`", respectively;
2. For "`SPB`" the code of length 2 is "`PB`", and for "`RIO`" the codes of lengths 2 and 3 are "`IO`" and "`RIO`", respectively;
3. For "`ASTRAHAN`" the code of length 3 is "`AAA`", and for "`HANNOVER`" the codes of lengths 1 and 3 are "`A`" and "`AER`", respectively;
4. For "`BOSTON`" the code of length 2 is "`BN`", and for "`SPB`" the codes of lengths 1 and 3 are "`B`" and "`SPB`", respectively;
5. For "`RIO`" the code of length 2 is "`IO`", and for "`IOWA`" the codes of lengths 2 and 4 are "`IA`" and "`IOWA`", respectively.

F. Bacteria

1 second, 512 megabytes

In a biological laboratory, an experiment is being conducted. At the beginning, the scientists have n *frozen* bacteria, numbered from 1 to n.

According to the experiment plan, the frozen bacterium with number i will enter the Petri dish after a_i seconds from the start of the experiment. If there are multiple such bacteria, they all enter at the same time.

As soon as a frozen bacterium is in the Petri dish, it thaws and begins to *mature*. The maturation of bacterium number i takes t_i seconds. Once the bacterium has matured, it starts to reproduce: it immediately turns into two *mature* bacteria, and then each mature bacterium divides into two mature bacteria at the end of each second.

The size of the colony is defined as the total number of bacteria in the Petri dish. The goal of the experiment is to determine after how many seconds the size of the colony will be exactly m for the first time.

Help the scientists determine the required number of seconds or find out that the size of the colony will never be exactly m.

Input

The first line contains two integers n, m ($1 \le n \le 2 \cdot 10^5$, $1 \le m \le 10^9$) — the number of frozen bacteria and the desired size of the colony.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the times at which the frozen bacteria enter the Petri dish.

The third line contains n integers $t_1, t_2, \ldots, t_n$ ($1 \le t_i \le 10^9$) — the duration of maturation for the frozen bacteria.

Output

If the size of the colony will never equal m, output -1.

Otherwise, output the number of seconds after the start of the experiment when the size of the colony will first be exactly m.

input
4 11 3 5 1 10 2 9 2 13
output
5

input
13 124 5 6 8 8 1 6 4 6 4 7 10 3 9 5 2 10 5 2 1 1 4 8 3 4 1 9
output
8

G. Yet Another Problem about a Grasshopper

1 second, 256 megabytes

There are n columns standing in a row, numbered from left to right with integers from 1 to n. The height of the i-th column is equal to a_i. The grasshopper can only jump forward (from a column with a smaller number to a column with a larger number), and in one jump, it can skip at most k columns ahead. Formally, the grasshopper can jump from column i to column j if $i < j$ and $j - i \le k$.

The grasshopper wants to be as high as possible, so it aims to maximize the minimum height of the columns it visits.

Let $f(l, r)$ denote the maximum of the minimum heights of the visited columns over all paths of the grasshopper from column number l to column number r.

You are required to find the value of the sum $\sum_{l=1}^{n} \sum_{r=l}^{n} f(l, r)$. In other words, you need to find the sum of the values $f(l, r)$ for all pairs of columns l and r ($l \le r$).

Input

The first line contains two integers n and k ($2 \le n \le 2 \cdot 10^5, 1 \le k \le n - 1$) — the number of columns and the maximum distance the grasshopper can jump forward.

The second line contains n integers a_i ($1 \le a_i \le 10^8$) — the heights of the columns.

Output

Output a single integer — the value of the sum $\sum_{l=1}^{n} \sum_{r=l}^{n} f(l, r)$.

input
4 2 2 1 4 2
output
18

Consider the example from the statement.

- For the pair of columns $[1, 1]$, the optimal route consists only of the first column, hence $f(1, 1) = 2$.
- For the pair of columns $[1, 2]$, the optimal route consists of columns numbered 1 and 2, hence $f(1, 2) = 1$.
- For the pair of columns $[1, 3]$, the optimal route consists of columns numbered 1 and 3, hence $f(1, 3) = 2$.
- For the pair of columns $[1, 4]$, the optimal route consists of columns numbered 1, 3, and 4, hence $f(1, 4) = 2$.
- For the pair of columns $[2, 2]$, the optimal route consists only of the second column, hence $f(2, 2) = 1$.
- For the pair of columns $[2, 3]$, the optimal route consists of columns numbered 2 and 3, hence $f(2, 3) = 1$.
- For the pair of columns $[2, 4]$, the optimal route consists of columns numbered 2 and 4, hence $f(2, 4) = 1$.
- For the pair of columns $[3, 3]$, the optimal route consists only of the third column, hence $f(3, 3) = 4$.
- For the pair of columns $[3, 4]$, the optimal route consists of columns numbered 3 and 4, hence $f(3, 4) = 2$.
- For the pair of columns $[4, 4]$, the optimal route consists only of the fourth column, hence $f(4, 4) = 2$.

H. Good arrays

2 seconds, 1024 megabytes

Recently, Bob learned about integer division. Inspired by this sacred knowledge, he decided to learn more about arrays of positive integers which satisfy some divisibility conditions. More precisely, Bob calls an array $a = \{a_1, a_2, \ldots, a_n\}$ *good* iff for every i from 1 to $n - 1$, a_i is divisible by a_{i+1}. Please help him count the number of good arrays of length n consisting of integer numbers not greater than c.

Input

The only input line contains two integers n and c ($1 \le n, c \le 5 \cdot 10^7$) — the length of the array and the maximum allowed value.

Output

Output a single integer — the total number of good arrays of length n consisting of positive integers not greater than c. As this number might be quite large, please output its remainder modulo $998\,244\,353$.

input
3 3
output
7

input
2 6
output
14

I. Plane stretching

10 seconds, 1024 megabytes

Jack is a big fan of geometry, so he bought himself a plane together with a set P of n distinct points; the i-th of them is located at (x_i, y_i).

It was extremely easy for Jack to find two points among them furthest away from each other. He quickly got bored and decided to come up with q real numbers α_1, α_2, α_3, ..., α_q. For each of these numbers, Jack is interested in the maximum possible distance between any two of the points if he scales the x-coordinate of each point by α_j. Formally speaking, he is interested in finding the two furthest points in a set $(x_i \cdot \alpha_j, y_i)$. Please help Jack!

Input

Each input contains multiple test cases. The first line contains a single integer t ($1 \le t \le 250\,000$) — the number of test cases. Then t test cases follow.

Each test case starts with two integers n and q ($2 \le n \le 5 \cdot 10^5, 1 \le q \le 5 \cdot 10^5$) — the number of points and the number of queries.

The following n lines contain the coordinates of each point x_i and y_i ($-10^9 \le x_i, y_i \le 10^9$). It is guaranteed that all points within a test case are distinct.

The following q lines contain the queries; each of them is identified by a single **real** number α_j ($1 \le \alpha_j \le 10^9$) — the scaling coefficients.

Let us denote the sum of values n_i among all test cases as N, and the sum of values q_i as Q. It is guaranteed that $N, Q \le 5 \cdot 10^5$.

Output

For each test case, output q real numbers: the answer to the i-th query. Your answer will be accepted if its absolute or relative error does not exceed 10^{-6}. More precisely, if a is your answer, and b is the judges' answer, then your answer will be considered correct in case $\frac{|a-b|}{\max(b,1)} \le 10^{-6}$.

input

```
2
5 2
0 0
1 1
0 2
-1 3
0 4
1
2.5
8 4
0 0
6 11
7 13
4 14
0 15
-4 14
-7 13
-6 11
2
1
1.25
1.5
```

output

```
4.000000
5.385165
28.000000
15.000000
17.500000
21.000000
```

J. Tree Cutting

2 seconds, 1024 megabytes

The organization "2D" is responsible for issuing permits for deforestation. It receives applications for cutting down trees located along a highway. There are n trees on the highway, the i-th of which grows at the coordinate x_i and has a height of h_i. The information about the existing trees is ordered such that $x_1 < x_2 < \ldots < x_{n-1} < x_n$.

Trees can be cut down one by one in the following manner. A tree is cut at the root and must then be felled either to the left or to the right. To avoid damaging the tree during its fall, it must not touch any not cut yet trees at a distance less than its height. In other words, if a tree at point x_i with height h_i is being felled to the right, there must not be any not cut yet tree at coordinate x_j such that $x_i < x_j < x_i + h_i$. If the same tree is being felled to the left, there must not be any not cut yet tree at coordinate x_j such that $x_i - h_i < x_j < x_i$.

There are important buildings located to the left of the tree at coordinate x_1 and to the right of the tree at coordinate x_n, so it is prohibited to fell trees in such a way that they fall outside the segment $[x_1, x_n]$. In other words, a tree at point x_i with height h_i cannot be felled to the left if $x_i - h_i < x_1$, and cannot be felled to the right if $x_i + h_i > x_n$.

The image above shows the first sample test. First, the second tree is felled to the right, then the third tree is felled to the left, and in the end, the first tree is felled to the right.

The organization has received q applications for tree cutting. Each application is specified by two numbers l_i and r_i, indicating that the applicant wants to cut down trees numbered from l_i to r_i, inclusive.

During the processing of an application, only trees numbered from l_i to r_i may be cut down. The cut tree may be knocked down into the territory to the left of the tree numbered l_i or to the right of the tree numbered r_i, but not beyond the segment $[x_1, x_n]$. It is not allowed to touch trees numbered outside the range from l_i to r_i.

For each application, it is required to calculate the maximum number of trees with numbers in the specified range that can be cut down without damaging any tree during the fall. Each application must be processed independently of the others.

Input

The first line contains two integers n and q ($1 \le n, q \le 5 \cdot 10^5$).

Each of the following n lines describes a tree and contains two integers x_i, h_i ($1 \le x_i \le 10^9$; $1 \le h_i \le 10^9$).

It is guaranteed that $x_1 < x_2 < \ldots < x_n$.

Next, there are q descriptions of applications, one per line. Line i contains two integers l_i, r_i ($1 \le l_i \le r_i \le n$).

Output

For each application, output the maximum number of trees that can be cut down without damaging any tree in the process.

input

```
3 3
1 3
3 1
4 2
1 1
2 3
1 3
```

output

```
0
2
3
```

input

```
5 3
1 5
3 1
4 2
5 3
6 1
1 5
5 5
1 1
```

output

```
5
1
0
```

input

```
1 1
100 100
1 1
```

output

```
0
```

K. Skyscrapers

1 second, 1024 megabytes

The city manager of the city N wants to plan a new residential area.

The residential area can be represented as a square grid of $n \times n$ cells, with a skyscraper to be built in each cell. The city hall has already approved the list of heights for the future skyscrapers.

However, skyscrapers cannot be built randomly: each skyscraper must be sufficiently *illuminated*. The illumination parameter for a skyscraper is the number of directions (up, down, left, right) such that there are no strictly taller skyscrapers in the corresponding direction.

You are asked to determine whether it is possible to arrange the n^2 skyscrapers of the specified heights on the grid, while meeting the illumination requirement: for each skyscraper, its illumination parameter must be at least k.

Input

The first line contains two integers n and k — the length of the side of the square grid and the illumination requirement, respectively ($1 \le n \le 1000$; $0 \le k \le 4$).

The second line contains an integer m — the number of types of skyscrapers ($1 \le m \le 10^4$).

Each of the next m lines contains two integers h_i and cnt_i — the height of the i-th type of skyscraper and the number of skyscrapers with that height, respectively ($1 \le h_i \le 10^9$; $1 \le cnt_i \le n^2$).

It is guaranteed that $\sum_{i=1}^{m} cnt_i = n^2$ and $h_i \ne h_j$ for $i \ne j$.

Output

In a single line, output "YES" if there exists an arrangement of the n^2 skyscrapers of the specified heights on the $n \times n$ grid, satisfying the illumination requirements with parameter k, and "NO" otherwise.

input

```
4 3
4
10 4
1 4
2 3
3 5
```

output

```
YES
```

input

```
4 3
4
10 6
1 3
2 3
3 4
```

output

```
NO
```

input

```
4 2
4
10 6
1 3
2 3
3 4
```

output

```
YES
```

One of the possible arrangements of skyscrapers in the first sample test is

1	2	2	2
1	10	10	3
1	10	10	3
1	3	3	3

In this case

- for the skyscrapers with height 1, the unilluminated side will be "right";
- for the skyscrapers with height 2, the unilluminated side will be "down";
- for the skyscrapers with height 3, the unilluminated sides will be "up" (for the skyscrapers in the bottom row) and "left" (for the skyscrapers in the rightmost column);
- for the skyscrapers with height 10, there will be no unilluminated sides.

In the second sample test it can be shown that there is no correct arrangement for the given set of skyscrapers and grid size.

In the third sample test one of the possible arrangements of skyscrapers is

1	1	3	3
1	10	10	3
2	10	10	3
2	2	10	10

In this case

- for the skyscrapers with height 1, the unilluminated sides will be "right" and "down";
- for the skyscrapers with height 2, the unilluminated sides will be "right" and "up" (for the second column);
- for the skyscrapers with height 3, the unilluminated sides will be "down" and "left" (for the skyscrapers in the second and third rows);
- for the skyscrapers with height 10, there will be no unilluminated sides.

L. Table Game

1 second, 512 megabytes

Consider a table consisting of h rows and w columns. We denote $A_{i,j}$ to be the number located at the intersection of the i-th row and the j-th column. It is guaranteed that $A_{i,j}$ are non-negative integers.

One may apply a sequence of zero or more changes to the table. Each change should be one of the following:

- Choose a column and remove it from the table (after this operation, columns strictly to the left and to the right of the chosen one will become adjacent).
- Choose a row and remove it from the table (after this operation, rows strictly to the bottom and to the top of the chosen one will become adjacent).

Determine whether it is possible to perform such operations that after applying them to the table, the sum of its elements becomes equal to s.

Input

The first line of input contains two integers h and w — the number of rows and columns of the table A respectively ($1 \le h, w \le 15$).

Each of the next h lines contains w integers — elements of table A ($0 \le A_{i,j} \le 10^9$).

The last line of input contains a single integer s — the desired sum ($1 \le s \le 10^{18}$).

Output

If it is impossible to get a table with a sum of elements equal to s from the initial one, print "NO".

Otherwise:

- In the first line of output, print "YES".
- In the second line, print a single integer k — the number of operations you need to perform to get a table with a sum of elements equal to s.
- In each of the next k lines, print two integers t_j, i_j (where $1 \le j \le k$). If $t_j = 1$, the j-th operation removes row i_j in *initial* numeration from the table. Otherwise, $t_j = 2$ and the j-th operation should remove column i_j from the table.

input

```
3 3
1 2 3
2 3 1
3 1 2
8
```

output

```
YES
2
1 3
2 3
```

input

```
2 3
2 2 2
2 2 2
5
```

output

```
NO
```

input

```
5 5
1 2 1 4 5
2 5 4 1 2
4 2 4 3 1
5 5 3 2 4
1 2 4 5 2
34
```

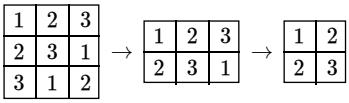
output

```
YES
3
1 4
1 5
2 1
```

In sample input 1, the initial table is as follows:

1	2	3
2	3	1
3	1	2

By erasing the third row and the third column, one can get a table with a sum of elements 8:

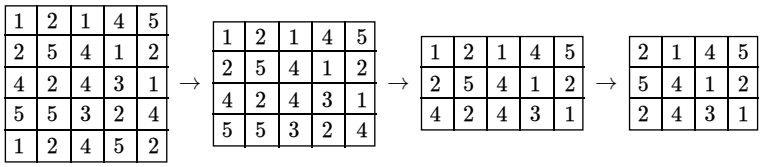


In sample input 2, it can be shown that it is impossible to get a table with the desired sum of elements from the initial one.

In sample input 3, the initial table is as follows:

1	2	1	4	5
2	5	4	1	2
4	2	4	3	1
5	5	3	2	4
1	2	4	5	2

By erasing rows four and five and row one, we can get a table with a sum of elements 34:



M. Game on a Tree

1 second, 512 megabytes

Alice and Bob are playing a game on a tree (a connected graph without cycles). They take turns, with Alice going first. On their first turn, each player places their piece on any vertex of the tree, but Bob cannot place his piece on the vertex where Alice's piece is located.

On each subsequent turn, a player moves their piece from one vertex to an adjacent vertex along an edge. Players cannot move their piece to a vertex that has ever had or currently has one of the pieces. The player who cannot make a move loses.

Determine whether Alice can win with optimal play from both players.

Input

The first line contains an integer t — the number of test cases ($1 \le t \le 10^5$).

The first line of each test case contains one integer n — the number of vertices in the tree ($2 \le n \le 10^5$).

The next $n - 1$ lines contain two integers a_i and b_i — the numbers of the vertices connected by an edge ($1 \le a_i, b_i \le n$). It is guaranteed that a tree is provided.

The sum of n across all test cases in one test does not exceed 10^5.

Output

For each test case, if Alice can win, print "`Yes`" on a new line; otherwise, print "`No`".

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