FDUPC19 - Onsite Contest

2019.12.6, **PUBLIC**

- **A. Good Luck**
  @overflow
  @xzl: 229bytes / 1ms

- **B. Induced Sort**
  @induce
  @xzl: 396bytes / 1ms

- **C. Fancy Triangle**
  @render
  @xzl: 1.3KB / 305ms

- **D. Unrolled Linked List**
  @block-easy
  @fez: 283bytes / 131ms

- **E. Efficient Multiplication**
  @optimize
  @xzl: 1019bytes / 2ms

- **F. Genius Problem**
  @sequence
  @sll: 989bytes / 314ms

- **G. Finding Closure**
  @closure
  @xzl: 1.4KB / 600ms

- **H. Harmonious Set**
  @geometry2
  @xzl: 6.9KB / 2374ms

(This page is intentionally left blank)

A. Good Luck

Description

> *"Are you ready kids?"*
> *"Aye, aye, Caption!"*
> *"I CAN'T HEAR YOOOOOU!!!"*
> *"Aye, aye, CAPTIOOON!!!"*
> *"Ohhhhhhhhhhhhhhhhhhhh!!!"*

Welcome to Fudan Programming Contest! It is well-known that `int` in C language cannot hold arbitrary integers and only integers in the interval $[-2^{31},\ 2^{31}-1]$ can be represented by `int`s. If the sum of two `int`s can't be represented by an `int`, an *overflow* occurs. Now you're given two `int`s named a and b, please report whether the sum $a+b$ will cause an overflow.

Input

The input consists of one line containing two `int`s a and b, separated by a space.

Output

Print "`YES`" if the sum $a+b$ can't be held in an `int`. Print "`NO`" otherwise. (without quotes)

Constraints

$-2^{31} \leqslant a,\ b \leqslant 2^{31}-1$.

Sample 1

Sample Input

```
1926081719 926081719
```

Sample Output

```
YES
```

Explanation

$926081719 + 1926081719 = 2852163438 > 2147483647 = 2^{31}-1$.

Sample 2

Sample Input

```
-2147483643 -6
```

Sample Output

```
YES
```

Explanation

$(-2147483643) + (-6) = -2147483649 < -2147483648 = -2^{31}$.

Sample 3

Sample Input

```
1 1
```

Sample Output

```
NO
```

B. Induced Sort

Description

Suffix arrays are versatile data structures and have many practical applications in the discipline of bioinformatics. In 2009, Ge Nong et al. introduced an efficient linear construction algorithm for suffix arrays named SAIS. Don't be astonished by it since we are not about to discuss how SAIS works. Instead, we invite you to implement the first phase of the SAIS algorithm.

Let's review some basic concepts related to strings. Consider a *string* S consisting of some lowercase English letters (characters) "`a`" to "`z`". We denote $|S|$ the length of the string, i.e. the number of characters in S, and $S[i]$ the ith character in S. Let $S[l..r]$ be the *substring* containing exactly all characters from lth character to rth character inclusive in S. For example, if S is "`somtimesnaive`", $S[9..13]$ will be "`naive`". A *suffix* S_i of S starting at ith character is $S[i..|S|]$. In addition, lexicographical order on letters is the order they appear in the alphabet or dictionaries, e.g. letter "`x`" is lexicographically smaller than letter "`y`".

The algorithm first **appends a new character** "`$`" (which is lexicographically smaller than all English letters) to the end of the original string S and then tries to label each suffix S_i with either `S` (smaller) or `L` (larger). Let $m = |S|$. Note that the last character $S[m]$ is "`$`" after appending. The algorithm first labels S_m with `S` and for each i from $m-1$ to 1, if character $S[i+1]$ equals to $S[i]$, then the label on S_i will be the same as the label on S_{i+1}; if $S[i+1]$ is lexicographically smaller than $S[i]$, the algorithm will label S_i with `L`. In other cases, label S_i with `S`.

Given the string S, please calculate the number of suffixes labeled with `S` or `L` by the procedure described above.

Input

The first line contains one positive integer n denoting the length of the original string.

The second line contains n successive lowercase English letters representing the string S.

Output

Output one line consisting of two nonnegative integers: the number of suffixes labeled with `S` and the number of suffixes labeled with `L`, separated by a space.

Constraints

$1 \leqslant n \leqslant 100$.

Sample 1

Sample Input

```
6
aaaaaa
```

Sample Output

```
1 6
```

Explanation

Here are the details of labels on suffixes in string S:

```
letter: a a a a a a $
 label: L L L L L L S
```

Note that the last character "$" also counts.

Sample 2

Sample Input

```
16
mmiissiissiippii
```

Sample Output

```
7 10
```

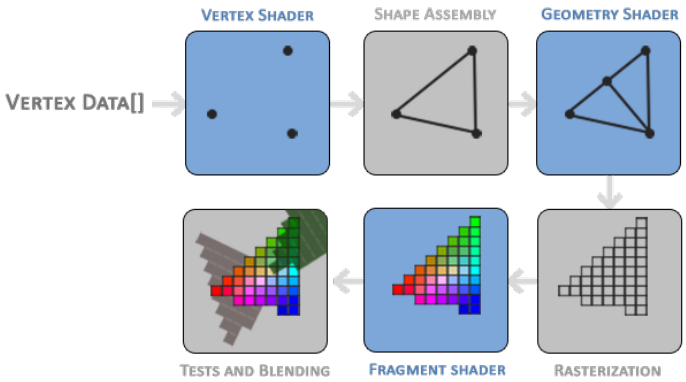
Explanation

Details of labels:

```
letter: m m i i s s i i s s i i p p i i $
 label: L L S S L L S S L L S S L L L L S
```

C. Fancy Triangle

Description



OpenGL graphics pipeline. (*image from learnopengl.com*)

Many youths might once dream of making a 3D game independently, so they attempted to learn Direct3D or OpenGL. However, they would probably found everything in graphics programming is so obscure and complicated. What's worse, they realized that all fancy 3D game worlds are just made of tons of commonplace triangles!

Let's demonstrate how powerful triangles are. In this problem, you are required to *draw* some triangles on a black-white screen. A $W \times H$ screen can be regarded as a matrix of H rows and W columns of *pixels*, where each pixel is either white or black. **Initially all pixels are black**. The coordinates of the center of pixel in the ith **column** and jth **row** is $(i,\ j)$. Therefore, the center of pixel in the top left corner is $(1,\ 1)$ while the center of pixel in the bottom right corner is $(W,\ H)$.

A triangle is described by three vertices $(x_1,\ y_1)$, $(x_2,\ y_2)$ and $(x_3,\ y_3)$, either in clockwise or counterclockwise order. Each vertex lies on some center of pixel in the screen. To draw the triangle, turn all pixels whose centers are inside the triangle (**borders of triangles included**) to white, regardless of what colors of pixels are before. By the way, this process is called *rasterization*. There are n triangles pending to be drawn. Please show the final state of the screen.

Input

The first line contains three positive integers W, H and n: the width of the screen, the height of the screen and the number of triangles to be drawn.

For the next n lines, each line contains six positive integers x_1, y_1, x_2, y_2, x_3, y_3, which describes a triangle with vertices $(x_1,\ y_1)$, $(x_2,\ y_2)$ and $(x_3,\ y_3)$.

Output

Print H lines. Each line consists of W characters. If the pixel centered at $(i,\ j)$ is white, the ith character of jth line is "#". Otherwise, if the pixel centered at $(i,\ j)$ is black, the ith character of jth line is "`.`".

Constraints

$1 \leqslant W \leqslant 1920,\ \ 1 \leqslant H \leqslant 1080,\ \ 1 \leqslant n \leqslant 50.$

$1 \leqslant x_1,\ x_2,\ x_3 \leqslant W,\ \ 1 \leqslant y_1,\ y_2,\ y_3 \leqslant H.$

It is guaranteed that all triangles are non-degenerated.

Sample 1

Sample Input

```
10 10 1
1 1 10 10 3 7
```

Sample Output

```
#.........
.#........
.##.......
.###......
..###.....
..####....
..#####...
.....###..
.......##.
.........#
```

Sample 2

Sample Input

```
45 16 6
38 6 8 16 45 16
5 15 30 4 25 13
45 8 2 16 1 12
13 12 42 12 6 5
21 14 18 2 33 8
16 11 24 13 1 8
```

Sample Output

```
.............................................
..................#..........................
..................##.........................
...................#####......#..............
.....#............#######..##................
......#####.......##########.........#.......
.......#########...###########....####.......
#.......#################################.....#
.....#...################################.....
..........################################.....
...........###############################....
##############################################...
.#############################################...
.##############################################..
.######...#####################################.
.#.....#######################################
```

Explanation

Picasso's masterpiece :)

Sample 3

Sample Input

```
32 32 26
8 8 8 9 9 9
8 8 9 8 9 9
10 6 10 7 22 7
10 6 22 6 22 7
23 8 23 9 24 9
23 8 24 8 24 9
6 10 6 22 7 22
6 10 7 10 7 22
8 23 8 24 9 24
8 23 9 23 9 24
25 10 25 22 26 22
25 10 26 10 26 22
23 23 23 24 24 24
23 23 24 23 24 24
10 25 10 26 22 26
10 25 22 25 22 26
13 13 13 15 11 15
13 13 11 13 11 15
18 13 18 15 20 15
18 13 20 13 20 15
14 21 14 22 19 22
19 22 19 21 14 21
11 29 19 29 13 27
13 27 19 27 21 29
11 29 11 32 21 32
21 32 21 29 11 29
```

Sample Output

```
.............................
.............................
.............................
.............................
.............................
.........#############.........
.........#############.........
.......##.............##.......
.......##.............##.......
.....##.................##.....
.....##.................##.....
.....##.................##.....
.....##...###....###....##.....
.....##...###....###....##.....
.....##...###....###....##.....
.....##.................##.....
.....##.................##.....
.....##.................##.....
.....##.................##.....
.....##.................##.....
.....##......######.....##.....
.....##......######.....##.....
.......##.............##.......
.......##.............##.......
.........#############.........
.........#############.........
...........#######.............
.........#########.............
.......###########.............
.......###########.............
.......###########.............
.......###########.............
```

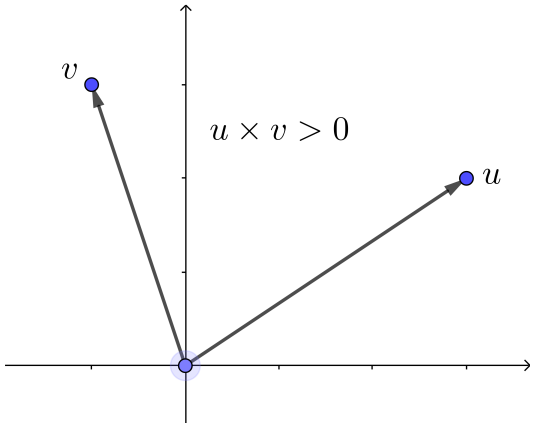
Explanation

Hint

The *cross product* (vector product, directed area product) of two vector $(x_1,\ y_1)$ and $(x_2,\ y_2)$ on 2-dimensional plane is defined as:

$$(x_1,\ y_1) \times (x_2,\ y_2) = x_1 y_2 - x_2 y_1$$

It has some nice properties. For two nonzero vectors u and v, if v is on the left side of u, then $u \times v > 0$. If u is parallel to v, $u \times v = 0$.



Meanwhile, cross products are useful in calculating areas of polygons. Particularly, for a triangle ABC and an **arbitrary point** P on the plane, the next formula gives the area of the triangle, regardless of where P locates:

$$S_{\triangle ABC} = \frac{1}{2} \left| \overrightarrow{PA} \times \overrightarrow{PB} + \overrightarrow{PB} \times \overrightarrow{PC} + \overrightarrow{PC} \times \overrightarrow{PA} \right|$$

This equality is usually adapted to decide whether P is inside the triangle ABC or not.

D. Unrolled Linked List

Description

Learning *unrolled linked list* recently, Codeplay came up with a problem (un)related to it.

We assume that an *unrolled linked list* consists of n blocks. Every block contains k units of memory. The first a_i units of the ith block store valid information ($a_i \leqslant k$). Thus, it leaves $k - a_i$ units of waste in the ith block. So we need to merge some of the blocks to improve this situation.

The mergence of the ith block and the jth block ($i < j$) is operated as follows: join valid information in the jth block to the back of valid information in the ith block, forming $a_i + a_j$ units of valid information in the ith block. Then delete the jth block.

Note that the ith and the jth block can be merged if and only if they are adjacent to each other (which means $|j - i| = 1$), and $a_i + a_j \leqslant k$. Because of the limited storage of the computer, please calculate the least number of blocks Codeplay can get by merging blocks as much as possible.

Input

The first line contains two positive integers n and k: the number of blocks and the maximum number of units of information that one block can store.

The second line contains n nonnegative integers $a_1,\ a_2,\ ...,\ a_n$.

Output

Print the minimum number of blocks after merging blocks.

Constraints

$1 \leqslant n \leqslant 10^6, 1 \leqslant k \leqslant 10^9$.

For each $i = 1,\ 2,\ ...,\ n, 0 \leqslant a_i \leqslant k$.

Sample 1

Sample Input

```
3 2
1 1 1
```

Sample Output

```
2
```

Explanation

The first and the second block can be merged into one block with two units of information.

Sample 2

Sample Input

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

Sample Output

```
2
```

Explanation

After the first and the second block merged, the second block (the third in the original list) can be merged with the third block (the fourth in the original list), resulting in a list with only two blocks.

Sample 3

Sample Input

```
4 10086
1 1 1 1
```

Sample Output

```
1
```

E. Efficient Multiplication

Description

Integer multiplications are ubiquitous nowadays on many platforms and they occur a lot when we programmes. However, on most microprocessors today, `imul`, the instruction for integer multiplications, is still slightly slower than the `add` instruction (for integer additions) and other instructions for bitwise operations (e.g. left logical shift and exclusive or), with respect to both CPU *clock cycles* and *throughput*. In particular, one useful utility appeared in many computer programs is multiplying a variable by a given constant, which is illustrated by the code snippet written in C:

```
// imul.c
unsigned long imul(unsigned long x) {
    return x * D;  // D is constant
}
```

Since the constant D is known, compilers have the opportunity to emit alternative assembly code in place of the original one-instruction `imul` implementation to generate more efficient machine code on specific platforms. For instance, on most x86-64 based operating systems (which might be the one in front of you), if we set D to 16 and compile the code with `-O9` flag (fully optimized??????) enabled, GCC will possibly dump the following assembly code:

```
sal $4, %rax
```

which means to shift out 4 bits on the left of the variable x (x is stored in the *register* `%rax`. **A register resembles a 64-bit unsigned integer variable in C language**). This is equivalent to multiply x by $16 = 2^4$ if not overflowed. You can examine this by checking out the result of `x << 4` in C.

The story continues. One day, with great efforts and prefect preparation, you finally succeeded in becoming an honorable system software architect in Giant Gromah™ Corporation (GGC), the leader of semiconductor industry, founded in Superember 31st, 9102. On the first day of work, your crazy mentor assigns you a weird task in order to help you familiarize yourself with the workflows and collaborations in the huge team. This task asks you to design an algorithm for the Giant Gromah™ Compiler Collection (GGCC) to generate highly optimized machine code for multiplications with constants, i.e. the C function `imul` mentioned in the front of this problem statement, on z64 microprocessor architecture (an obsolete sequential architecture since 8102, which **executes all instructions one by one** chronologically).

That is to say, you're going to simulate an integer multiplication "`x * D`" for specific constant `D` by some basic instructions. For brevity, we assume the unsigned integer variable x is initially moved to the register `%rax` before the multiplication, so all instructions should operate on the only register `%rax` and no other registers are involved in this process. Meanwhile, the result of multiplication should be stored in register `%rax` as well.

You can implement the multiplication by using one or more instructions described below:

- "`imul S, T`" (Integer Multiplication): multiply T by S. The result is saved to register T. S is either a register or a constant. "`imul $D, %rax`" is the default implementation of an integer multiplication `x * D`. Note that the expression `$D` means D is a constant value (an immediate

value, more precisely) rather than a register like `%rax`.

- "`sal $k, T`" (Left Arithmetic/Logical Shift): shift out k bits of the bit representation of T on the left and k zeros are supposed to be shifted in on the right. The result is saved to register T. For example, if T contains merely 8 bits and $T = 11010111_{(2)}$, after executing "`sal $3, T`", $T = 10111000_{(2)}$.
- "`lea (A, B, $k), T`" (Load Effective Address): evaluate the expression $A + kB$ and save the result to register T. Both A and B **must be** registers and A can sometimes be omitted (written as "`lea (, B, $k), T`"), in which case this instruction simply evaluates kB and save the result to T. For some reasons pertaining to memory systems, k can only be 1, 2, 4 or 8 and other values are not permitted.

Each instruction takes some CPU clock cycles to complete its evaluation and will be encoded into a byte sequence when stored in executable files on the file system. We introduce the concept of **timing parameter** $t_{\texttt{imul}}$ denoting the number of clock cycles the instruction `imul` takes and let $l_{\texttt{imul}}$ be the length of the encoded byte sequence of instruction `imul`, i.e. the **encoding length**. Both parameters are positive integers. We also refer to similar parameters $t_{\texttt{sal}}$, $l_{\texttt{sal}}$ for instruction `sal` and $t_{\texttt{lea}}$, $l_{\texttt{lea}}$ for instruction `lea`. All parameters can be retrieved in the z64 architecture reference manual, so they are known in advance. Supposing your algorithm has emitted a sequence of instructions to calculate the result of multiplication, let t be the sum of timing parameters of all instructions appeared in the sequence and l be the sum of encoding lengths. Your mentor instructs you to maximize the so-called *efficiency index* K of the instruction sequence:

$$K = \frac{t_{\texttt{imul}}/t}{l/l_{\texttt{imul}}}$$

(where $t_{\texttt{imul}}/t$ stands for *speed-up ratio* while $l/l_{\texttt{imul}}$ is *inflation ratio*)

Will you manage to accomplish this task to satisfy your crazy mentor?

Input

The first line contains one positive integer D.

The second line contains two parameters $t_{\texttt{imul}}$ and $l_{\texttt{imul}}$ for instruction `imul`.

The third line contains two parameters $t_{\texttt{sal}}$ and $l_{\texttt{sal}}$ for instruction `sal`.

The fourth line contains two parameters $t_{\texttt{lea}}$ and $l_{\texttt{lea}}$ for instruction `lea`.

Output

Output one line containing a reduced fraction p/q representing the maximum efficiency index K, i.e. $K = p/q$. Note that you should ensure $\gcd(p,\ q) = 1$ and $p,\ q \in \mathbb{N}^+$. There are no whitespaces among the numerator, the separator and the denominator.

Constraints

$2 \leqslant D \leqslant 2^{32} - 1, 1 \leqslant t_{\texttt{imul}}, l_{\texttt{imul}}, t_{\texttt{sal}}, l_{\texttt{sal}}, t_{\texttt{lea}}, l_{\texttt{lea}} \leqslant 10^4$.

You may assume $x \leqslant 2^{32} - 1$ so that the register `%rax` contains enough bits to represent the result of the multiplication without any overflow.

Sample 1

Sample Input

```
16
100 6
70 5
80 4
```

Sample Output

```
12/7
```

Explanation

The same as the example shown in problem statement. The emitted assembly code is "`sal $4, %rax`".

Sample 2

Sample Input

```
8
100 6
70 5
80 4
```

Sample Output

```
15/8
```

Explanation

Use "`lea (, %rax, $8), %rax`" instead. That's cheaper.

Sample 3

Sample Input

```
12
100 20
70 3
80 4
```

Sample Output

```
40/21
```

Explanation

Consider the following assembly code:

```
sal $2, %rax
lea (%rax, %rax, $2), %rax
```

which will produce $12x$ in register `%rax`, as you should verify.

Sample 4

Sample Input

```
998244353
998 1926
244 0817
353 9102
```

Sample Output

```
1/1
```

Explanation

`imul` is fast enough in this case. Just emit "`imul $998244353, %rax`".

F. Genius Problem

Description

We love Gromah, and geniuses love enumerating sequences. A sequence of m integers $a_1,\ a_2,\ ...,\ a_m$ ($1 \leqslant a_1 \leqslant a_2 \leqslant \cdots \leqslant a_m \leqslant n$) is *good* if and only if each number is divisible by its predecessor in the sequence, i.e. $a_i \mid a_{i+1}$ for all $i = 1,\ 2,\ ...,\ m-1$. Given n and m, please count the number of good sequences of length m. As the answer may be extremely large, just print the answer modulo 998244353 ($2^{23} \times 7 \times 17 + 1$, a prime number).

Input

The input contains only one line with two positive integers n and m denoting the upper bound of integers in the sequence and the length of sequence respectively.

Output

Print one line containing the number of good sequences modulo 998244353.

Constraints

$1 \leqslant n,\ m \leqslant 5 \times 10^5$.

Sample 1

Sample Input

```
3 2
```

Sample Output

```
5
```

Explanation

Five good sequences are:

- $1,\ 1$
- $2,\ 2$
- $3,\ 3$
- $1,\ 2$
- $1,\ 3$

Sample 2

Sample Input

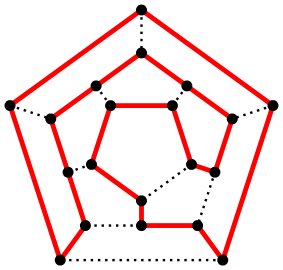
```
6 4
```

Sample Output

```
39
```

G. Finding Closure

Description



Hamiltonian graph. (*image from Wikipedia*)

In the mathematical field of graph theory, a Hamiltonian cycle is a cycle in an undirected or directed graph that visits each vertex exactly once. If a graph has a Hamiltonian cycle, it's a Hamiltonian graph. Characterizations of Hamiltonian graphs have been studied for many decades and mathematicians discovered plenty of properties related to such graphs. The best vertex degree characterization of Hamiltonian graphs was provided in 1972 by the *Bondy-Chvátal Theorem*:

> (**Bondy-Chvátal**) If two different vertices u and v are not adjacent in graph $G = (V,\ E)$ and the sum of their degrees $\mathrm{d}(u) + \mathrm{d}(v) \geqslant |V|$, then G is Hamiltonian iff. $G + \{(u,\ v)\}$ is Hamiltonian.

This theorem later introduced the concept of K-closure of an undirected graph. For an undirected graph G, the degree $\mathrm{d}(u)$ of vertex u is the number of edges incident to u. If there are two nonadjacent vertices u and v such that the sum of their degrees $\mathrm{d}(u) + \mathrm{d}(v)$ is not less than a parameter K, we add the edge $(u,\ v)$ to graph G. Repeat this procedure until no more edge can be added. The final graph is called the K-closure of the original graph. We can prove that K-closure is unique for any undirected graph regardless of the order of edges being added.

Given a simple undirected graph G and the parameter K, please compute the K-closure of G.

Input

The first line contains two nonnegative integers n and K: the number of vertices in graph G and the parameter. Vertices are numbered from 1 to n.

The next n lines describe the adjacent matrix M of graph G. Each line contains a string consisting of "`0`" and "`1`" of length n. For two vertices i and j, there is an edge $(i,\ j)$ if and only if the jth character of i th line is "`1`".

Output

Print n lines representing the adjacent matrix of K-closure of graph G. The format of adjacent matrix is the same as the one described in the "Input" section.

Constraints

$1 \leqslant n \leqslant 5000,\ \ 0 \leqslant K \leqslant 2n-2$.

It is guaranteed that M is a valid adjacent matrix, i.e. for any i, $M_{i,\,i} = 0$, and for any i and j, $M_{i,\,j} = M_{j,\,i}$. (Therefore M is a symmetric matrix whose main diagonal is all zeros)

Sample 1

Sample Input

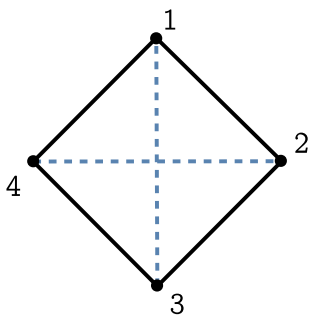
```
4 4
0101
1010
0101
1010
```

Sample Output

```
0111
1011
1101
1110
```

Explanation

Graph G is a cycle with four vertices. Blue dashed edges in the image below are newly added in K-closure.



Sample 2

Sample Input

```
4 5
0101
1010
0101
1010
```

Sample Output

```
0101
1010
0101
1010
```

Sample 3

Sample Input

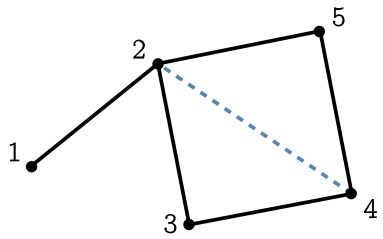
```
5 5
01000
10101
01010
00101
01010
```

Sample Output

```
01000
10111
01010
01101
01010
```

Explanation

Only one edge $(2,\ 4)$ is added to graph G.

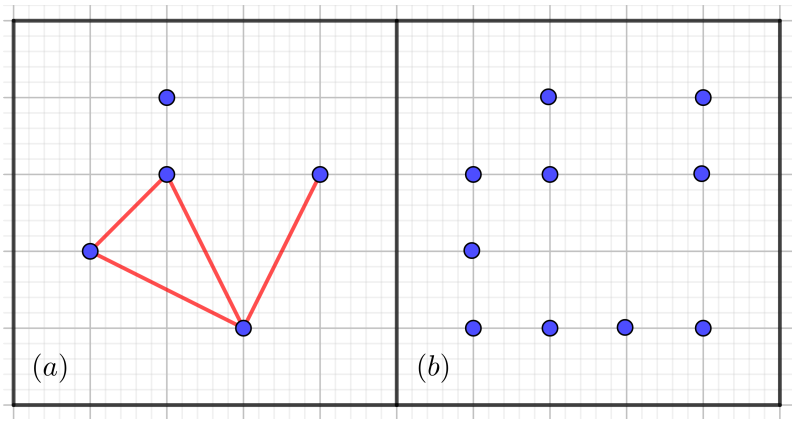


H. Harmonious Set

Description

> *"Unfortunately, the full behavior of splay trees can be described as either 'complete magic' or 'notoriously hard to understand,' depending on the reader's mood."*
>
> —— Dion Harmon et al.

Final challenge! A set S of points on 2-dimensional plane is said to be *harmonious* if for any two points A and B in S not on a common horizontal or vertical line, there is at least one point from $S\backslash\{A,\ B\}$ on the sides of the *axis-aligned* rectangle defined by A and B (in other words, the minimal rectangle parallel to axes containing A and B).



(a) an inharmonious set. Endpoints of red segments are unsatisfied pairs of points. (b) an example of harmonious set.

As a master of computer programming, you find it easy to determine whether a given set S is harmonious or not, but you find it really hard to generate a harmonious set! Thus, you wonder if you try to add a new point P to a harmonious set S such that $S' = S \cup \{P\}$, whether S' is still harmonious. Try to solve this problem! *YOU'RE THE WINNER!*

Input

This first lines contains four nonnegative integers n, q, C, D. n is the size of set S and q is the number of queries. C and D are used to generate two integer sequences A_i, B_i by the following pseudocode:

```
function MAGIC(x):
    x ← (x ^ (x << 13)) & 0xFFFFFFFF
    x ← (x ^ (x >> 17)) & 0xFFFFFFFF
    x ← (x ^ (x << 5)) & 0xFFFFFFFF
    return x
end function

let A[i], B[i] be two integer sequences of length q
A[1] ← C
B[1] ← D
for i from 2 to q:
    A[i] ← MAGIC(A[i - 1])
```

```
    B[i] ← MAGIC(B[i - 1])
end for
```

where "^", "<<", ">>" and "&" stand for *bitwise exclusive or*, *left logical shift*, *right logical shift* and *bitwise and* respectively.

For the next n lines each line contains two positive integer x, y denoting a point $(x,\ y)$ in S.

The last q lines are encrypted queries. The ith query corresponds to two nonnegative integers `x'[i]` and `y'[i]` in the ith line. The actual coordinates `x[i]` and `y[i]` of the point P_i of ith query are decoded as follows: if $i = 1$, then `x[1] = x'[1]` and `y[1] = y'[1]`; if $i > 1$ and the answer to $(i-1)$th query is "yes", then `x[i] = x'[i] ^ (A[i] & 0xFFFFF)` and `y[i] = y'[i] ^ (A[i] & 0xFFFFF)`. Otherwise `x[i] = x'[i] ^ (B[i] & 0xFFFFF)` and `y[i] = y'[i] ^ (B[i] & 0xFFFFF)`.

Output

For the ith query, print "`YES`" if $S \cup \{P_i\}$ is harmonious or "`NO`" otherwise in the ith line.

Constraints

$1 \leqslant n \leqslant 5 \times 10^6, 1 \leqslant q \leqslant 10^6, 0 \leqslant C,\ D \leqslant 2^{32} - 1$.

All coordinates are positive integers and no greater than 10^6. It is guaranteed that the first n points are pairwise distinct and S is initially harmonious. For each query, the point corresponding to it does not appear in S.

Sample 1

Sample Input

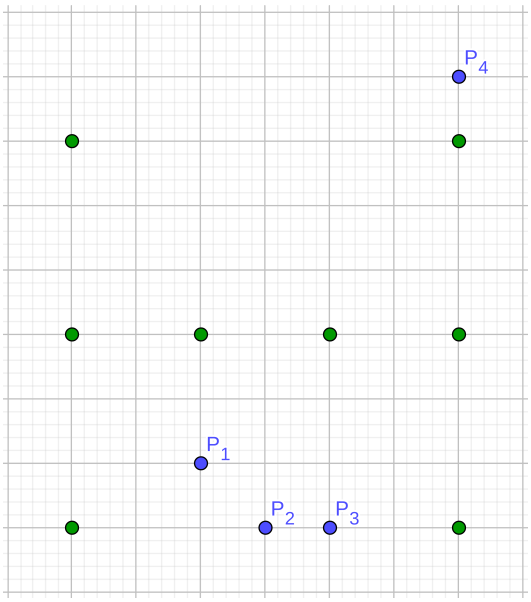
```
8 4 0 0
1 1
1 4
1 7
3 4
5 4
7 4
7 1
7 7
3 2
4 1
5 1
7 8
```

Sample Output

```
NO
NO
YES
YES
```

Explanation

Here is the visual representation of the sample. Points in green are those in set S, while P_1, P_2, P_3 and P_4 in blue are query points.



Sample 2

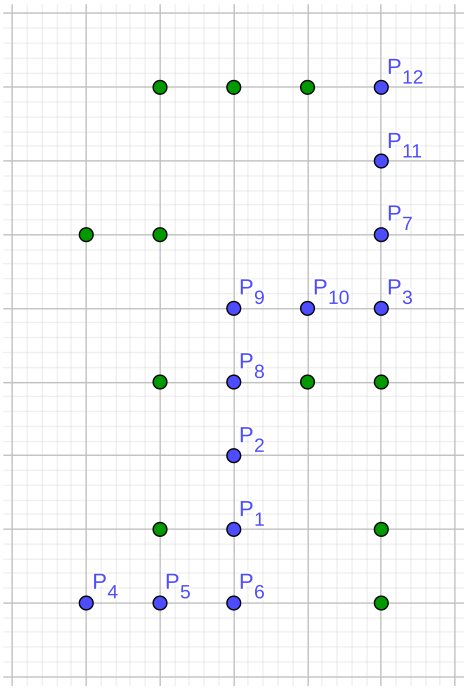
Sample Input

```
11 12 0 0
1 6
2 2
2 4
2 6
2 8
3 8
4 4
4 8
5 1
5 2
5 4
3 2
3 3
5 5
1 1
2 1
3 1
5 6
3 4
3 5
4 5
5 7
5 8
```

Sample Output

```
NO
NO
NO
NO
YES
NO
NO
YES
NO
NO
NO
YES
```

Explanation



Sample 3

Sample Input

```
11 12 3684530729 3270379979
1 6
2 2
2 4
2 6
2 8
3 8
4 4
4 8
5 1
5 2
5 4
3 2
683043 683043
47514 47514
226431 226431
997418 997417
356735 356733
299360 299363
193050 193053
531534 531528
668744 668745
74264 74266
225649 225660
```

Sample Output

```
NO
NO
NO
NO
YES
NO
NO
YES
NO
NO
NO
YES
```

Explanation

This is the encrypted version of the second sample.

Hint

Due to the large test data (approximately 80MB), we suggest taking more efficient measures to read from standard input. Here we provide a utility for fast reading integers from `stdin` in C++ language:

```
#include <cstdio>
#include <cctype>
#define _BUFFERSIZE 65536
static size_t _pos = _BUFFERSIZE, _len;
static char _buf[_BUFFERSIZE];
void _getc(char &c) {
    if (_pos == _BUFFERSIZE) {
        _pos = 0;
        _len = fread(_buf, 1, _BUFFERSIZE, stdin);
    }
    c = _pos < _len ? _buf[_pos++] : 0;
}
template <typename T>
void read(T &x) {
    x = 0;
    char c;
    do _getc(c); while (!isdigit(c));
    do {
        x = x * 10 + (c - '0');
        _getc(c);
    } while (isdigit(c));
}
```

Basic usage:

```
int main(int argc, char *argv[]) {
    int n, q;
    unsigned int C, D;
    // scanf("%d%d%u%u", &n, &q, &C, &D);
    read(n);
    read(q);
    read(C);
    read(D);
    return 0;
}
```

Notice: **DO NOT** mix `scanf/getchar` or `cin` with `read` provided by us. `printf/puts/putchar` and `cout` are not affected.