문제 1. 소 사방치기

입력 파일: hopscotch.in
출력 파일: hopscotch.out
시간 제한: 2 seconds
메모리 제한: 512 megabytes

인간이 사방치기를 좋아하는 것 처럼, 농부 존의 소들도 소들이 할 수 있는 사방치기 놀이를 만들었다. 거의 1톤에 달하는 꼴사라운 소들의 사방치기는 재앙이었지만, 매일 소들이 사방치기를 하는 것을 막을 수는 없었다.

게임은 $R \times C$ ($2 \leq R \leq 750$, $2 \leq C \leq 750$) 격자에서 진행되며, 각 격자칸에는 1 이상 K 이하의 ($1 \leq K \leq R \times C$) 정수가 붙어있다. 소들은 왼쪽 위 격자칸에서 시작해서 오른쪽 아래 격자칸으로 점프를 통해 이동하며,

1. 현재 있는 칸과 다른 수가 붙어 있으며,
2. 현재 있는 칸보다 적어도 한 행 만큼 아래에 있으며,
3. 현재 있는 칸보다 적어도 한 열 만큼 오른쪽에 있는 칸

인 경우에만 점프가 가능하다,

소들을 위해 왼쪽 위 격자칸에서 오른쪽 아래 격자칸으로 가는 서로 다른 경우의 수를 구해주자.

입력 형식

첫째 줄에는 R, C, K가 주어진다. 다음 R개의 줄에는 1 이상 K 이하의 C 개의 수가 주어진다.

출력 형식

왼쪽 위 격자칸에서 오른쪽 아래 격자칸으로 가는 서로 다른 경우의 수를 1000000007로 나눈 나머지를 출력하여라.

제한

hopscotch.in	hopscotch.out
4 4 4 1 1 1 1 1 3 2 1 1 2 4 1 1 1 1 1	5

문제 2. 검열

입력 파일:	`censor.in`
출력 파일:	`censor.out`
시간 제한:	2 seconds
메모리 제한:	256 megabytes

농부 존은 "발굽 관리하는 법" 이라는 잡지를 구독해서, 소들이 목장에서 기다리는 동안 읽을 것들을 준비했다. 유감스럽게도, 잡지의 최신간은 완벽하게 스테이크를 굽는 법 같은 것들이 있었고, 농부 존의 입장에서는 별로 소들에게 보여주고 싶지 않은 것이었다 (잡지는 좋은 편집장이 필요하다.)

농부 존은 잡지로 부터 모든 글을 모아서 최대 10^5 길이의 문자열 S를 만들었다. 또, S로 부터 검열할 문자열 $t_1 \cdots t_N$을 나열했다. 그래서 농부 존은 문자열 S의 부분문자열로 등장하는 가장 처음 (시작 인덱스가 가장 작은) 검열할 문자열을 찾아서 S로 부터 지웠다. 그리고 이 작업을 반복해서 S로 부터 검열할 문자열이 없을 때 까지 반복했다. 원래는 없었지만 단어를 지운 후에 새로운 문자열이 등장할 수 있음에 유의하여라.

검열할 문자열들은 어떤 하나가 다른 하나의 부분문자열이 아니어서, 가장 처음 등장하는 검열할 문자열이 유일하게 결정된다.

문자열 S의 검열이 끝난 결과를 농부 존에게 알려주자.

입력 형식

첫째 줄에는 S가 주어진다. 둘째 줄에는 검열 할 단어의 수 N이 주어진다. 다음 N개의 줄에는 문자열 $t_1 \cdots t_N$이 주어진다. 각 문자열은 소문자 알파벳 (a..z)으로 이루어져 있고, 길이를 모두 합치면 10^5를 넘지 않는다.

출력 형식

검열이 끝난 후 문자열 S를 출력하여라. 이 문자열이 빈 문자열이 아니라는 것이 보장되어 있다.

예제

censor.in	censor.out
begintheescapexecutionatthebreakofdawn 2 escape execution	beginthatthebreakofdawn

문제 3. 울타리 치기

입력 파일:	`fencing.in`
출력 파일:	`fencing.out`
시간 제한:	2 seconds
메모리 제한:	256 megabytes

농부 존은 당신에게 소들의 움직임을 제한하기 위해 지으려는 직선 모양 울타리 위치를 정하기 위한 도움을 요청했다. 그는 몇몇 울타리 위치를 정했고 이 위치들이 유용한지, 즉 모든 소들이 울타리의 한 쪽에 있는지 알려달라는 요청을 했다. 소가 정확히 울타리 위에 있을 경우 위치는 유용하지 않다. 농부 존은 울타리 위치에 관한 몇몇 질문을 하기로 했고, 유용한 위치이면 "`YES`", 아니면 "`NO`"로 대답해야 한다.

또한, 농부 존은 새로 소를 데리고 올 수 있다. 새로운 소가 들어오면 그 시점 이후에 들어오는 질문에 대해서 새로 들어온 소도 같은 위치에 있어야 한다.

입력 형식

첫째 줄에는 정수 N과 Q가 공백으로 구분되어 주어진다. ($1 \leq N \leq 100,000$, $1 \leq Q \leq 100,000$) 이는 각각 처음 소의 수와, 질의의 수를 의미한다.

다음 N개의 줄은 처음 소들의 위치를 나타낸다. 각 줄은 소의 위치를 의미하는 공백으로 구분된 두 정수 x, y가 주어진다.

다음 Q개의 줄은 새로운 소가 추가되거나 울타리가 유용한지에 관한 질의를 한다. 입력이 "1 x y" 이면 (x, y)에 소가 추가되었다는 것이다. 입력이 "2 A B C" 이면 $Ax + By = C$ 울타리가 유용한지 질문 하는 것을 의미한다.

모든 소들의 위치는 다르며 $-10^9 \leq x, y \leq 10^9$를 만족한다. 또한, 울타리에 관련된 질문은 $-10^9 \leq A, B \leq 10^9$, $-10^{18} \leq C \leq 10^{18}$을 만족한다. 어떤 질문도 $A = B = 0$인 경우는 없다.

출력 형식

각 울타리의 위치를 묻는 질문에 대해 유용한 위치이면 "`YES`", 아니면 "`NO`"를 출력한다.

예제

fencing.in	fencing.out
3 4 0 0 0 1 1 0 2 2 2 3 1 1 1 2 2 2 3 2 0 1 1	YES NO NO

참고 사항

직선 $2x + 2y = 3$를 기준으로 처음 세 마리의 소는 모두 같은 쪽에 있다. 하지만 $(1, 1)$에 있는 소가 추가 된 이후로 더 이상 이 직선에 해당하는 울타리는 유용하지 않다.

직선 $y = 1$에 해당하는 울타리는 $(0, 1)$과 $(1, 1)$에 있는 소가 정확히 울타리 위에 있기 때문에 유용하지 않다.

경고: 이 문제의 입력은 매우 크기 때문에, C++ 사용자들은 `scanf`나 `ios_base::sync_with_stdio(false)` 줄을 추가하여 입력을 빠르게 받는 것을 고려하십시오. Java 사용자들은 `java.util.Scanner`를 사용 하는 것을 피해야 합니다. 각 쿼리의 출력마다 (`std::endl` 등을 사용해서) 버퍼를 비우지 마세요.