Problem A. (a, b)-Tower *(Division 2)*

Input file:	`abtower.in`
Output file:	`abtower.out`
Time limit:	1 second
Memory limit:	256 mebibytes

Little Artem is pondering the following question: how to write only few digits and yet get a very large number? One of the answers, of course, is to write a tower of powers. Recall that a tower of powers is calculated from top to bottom: in an expression of the form x^{y^z}, first, y^z is calculated, and after that, x is raised to that power. For example, the small tower 2^{3^4} is equal to the huge number $2^{3 \cdot 3 \cdot 3 \cdot 3} = 2^{81} = 2\,417\,851\,639\,229\,258\,349\,412\,352$.

Artem wrote down integers from a to b diagonally, moving upwards and to the right. This resulted in a tower of powers:

$$a^{(a+1)^{\cdot^{\cdot^{\cdot^{(b-1)^b}}}}}.$$

Help him find the last decimal digit of this, possibly huge, number.

Input

The first line of input contains two integers a and b, the parameters of the tower ($1 \le a \le b \le 10$).

Output

Print one integer: the last decimal digit of the number

$$a^{(a+1)^{\cdot^{\cdot^{\cdot^{(b-1)^b}}}}}.$$

Examples

abtower.in	abtower.out
2 4	2
3 3	3

Explanations

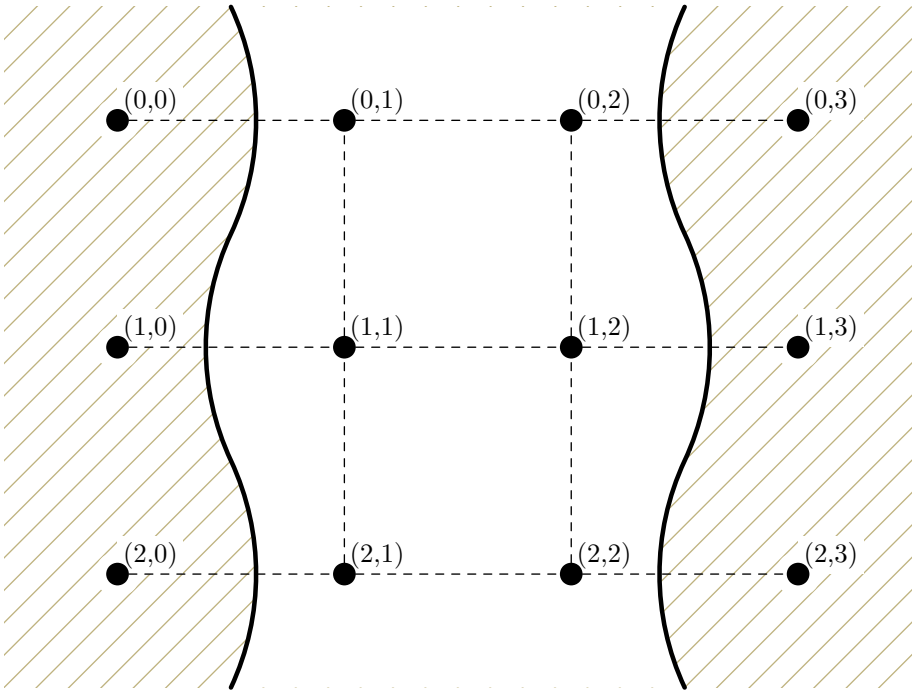
In the first example, the number is $2^{3^4} = 2^{81} = 2\,417\,851\,639\,229\,258\,349\,412\,35\mathbf{2}$. The last decimal digit of this number is 2.

In the second example, the number is just **3**, and the last and only decimal digit of this number is 3.

Problem B. Bridges Construction *(Division 2)*

Input file:	`connection-probability.in`
Output file:	`connection-probability.out`
Time limit:	1 second
Memory limit:	256 mebibytes

In one country, a tender must be held for all important contracts. "Cat in Tank, Inc." won a contract for constructing a system of bridges over a river. The contract states that the system must connect $R \cdot C$ pile foundations which form a grid of R rows by C columns on the river. The first column of piles is located on the left side, and the last column is located on the right side.



According to the initial plan, the system must consist of all bridges which connect all neighboring piles located in same rows and all neighboring piles located in same columns except for leftmost and rightmost columns (who constructs bridges along the coast anyway?). Well, it turned out to be not that easy. "Cat in Tank, Inc." specializes in programming and has no clue about bridge construction. In order to increase the chance of completing the project, they hired a separate contractor for every bridge. For each bridge, the probability that it will be finished on time by its contractor is also known.

Construction is considered complete if it is possible to get from the left side of the river to the right side using only the bridges built on time. Your task is to calculate the probability of work completion.

Input

The first line contains two integers: R and C (in this version of the problem, $R = 3$ and $C = 4$).

Next R lines contain $C - 1$ numbers each: probabilities of timely completion of the bridges connecting neighboring piles located in the same row. Here, j-th number on i-th of these lines is the probability of building the bridge between piles $(i - 1, j - 1)$ and $(i - 1, j)$ on time.

Next $R - 1$ lines contain $C - 2$ numbers each: probabilities of timely completion of the bridges connecting neighboring piles located in the same column. Here, j-th number on i-th line is the probability of building the bridge between piles $(i - 1, j)$ and (i, j) on time.

All probabilities are integer percentages between 1 and 99.

Output

Let the expected probability of timely completion of a system of bridges that connects left and right sides of the river be equal to $\frac{p}{q}$ where p and q are coprime. Output a single integer: the number $(p \cdot q^{-1}) \bmod (10^9 + 7)$. It is guaranteed that q is not a multiple of $10^9 + 7$ in all tests.

Examples

connection-probability.in	connection-probability.out
3 4 50 50 50 50 50 50 50 50 50 50 50 50 50	500000004

Note

One may perform all probability-related calculations in a finite field of order $10^9 + 7$ instead of the field of reals. It is possible to prove that such modification is equivalent to calculation of p and q and taking them modulo $10^9 + 7$ in the end.

The answer for the example is $\frac{1}{2}$.

Problem C. Equivalence Relation *(Division 2)*

Input file:	`equivalence.in`
Output file:	`equivalence.out`
Time limit:	1 second
Memory limit:	256 mebibytes

Egor is studying equivalence relations.

Equivalence relation on set S is such subset of $R = \{(x, y) | x \in S, y \in S\}$ that three following conditions hold:

- for each x, it is true that $(x, x) \in R$;
- if $(x, y) \in R$, then $(y, x) \in R$;
- if $(x, y) \in R$ and $(y, z) \in R$, then $(x, z) \in R$.

It is easy to show that, given these conditions, set S can be split into several classes of elements so that elements of each class are pairwise equivalent and elements of different classes are not equivalent. Such classes are called *equivalence classes*.

Let us consider the following equivalence relation on strings of zeroes and ones: strings are called equivalent when one can be transformed to another using the following operations.

- Remove or insert two consecutive zeroes in any position of the string.
- Remove or insert three consecutive ones in any position of the string.

Operations can be performed any number of times in any order.

You are given a set of strings. Print its partition into equivalence classes.

Input

The first line contains an integer n ($2 \le n \le 4096$). Each of the next n lines contains one string from the set. All strings are non-empty, and their total length is at most 50 000 characters.

Output

On the first line, print one integer k: the number of equivalence classes. On the next k lines, print the classes themselves. Each class must be printed in the following form: print the numbers of its elements in increasing order. Classes must be printed in the order of increasing of their miminal elements. Strings are numbered from one in the order they are given.

Example

equivalence.in	equivalence.out
5 0101010101 101010110101011 0101 0000 111	4 1 2 3 4 5

Problem D. Formula-1 *(Division 2)*

Input file:	`hamilton-clique.in`
Output file:	`hamilton-clique.out`
Time limit:	1 second
Memory limit:	256 mebibytes

Famous Formula-1 driver Lewis Hamilton is currently training for the next race. He has a special circualar training ground for this. There are N checkpoints evenly placed on the perimeter and numbered from 1 to N. The main coach advised Lewis to drive between every pair of different checkpoints exactly once. The direction for each pair is not important.

Because Formula-1 tracks are circular, Lewis wants to drive every day through all checkpoints in some order and return back. He soon realized that he can do this no more than $\frac{N-1}{2}$ days in a row. He is very busy with his training, so he asked you to make a plan for each of this days or say that it is impossible. To prevent problems with rounding, the main coach made N odd.

Input

The first line contains one odd integer N ($3 \le N \le 500$).

Output

The first line must contain only one word "`Yes`" if it is possible to make such a plan, or "`No`" otherwise. In case the answer is positive, each of the next $\frac{N-1}{2}$ lines must contain N different integers from 1 to N: the circular tracks for each day. If there are several possible answers, print any one of them.

Example

hamilton-clique.in	hamilton-clique.out
5	Yes 1 2 3 4 5 1 3 5 2 4

Problem E. Ideal Photo *(Division 2)*

Input file:	`make-a-row.in`
Output file:	`make-a-row.out`
Time limit:	1 second
Memory limit:	256 mebibytes

It is that time of the year again... UEFA Champions League final match will start in just a few minutes. This year, Real Madrid CF will try to win the trophy back from Chelsea FC. Tradition states that both teams sing their national anthems first. To do this, all players, coaches and team staff (in total, there are N people in each team) form two lines, each team stands on their half of the field. The two halves are divided by the middle line $y = 0$. Real Madrid members will stand somewhere on the line $y = 1$, and Chelsea members on the line $y = -1$. From the photo taken by a helicopter, we can see that right now, i-th member of the Real has coordinates $(r_i, 1)$, and i-th member of Chelsea has coordinates $(c_i, -1)$.

After the anthems, everybody should stand on the middle line for group photo. The main photographer decided that all of them should form a row such that the distance between any two consecutive people in the row is exactly 1. You task is to say to every person where he or she should be standing, and they will move there. To make your task more interesting, you are additionally asked to minimize the sum of distances travelled by each person.

Input

The first line contains one integer N ($1 \le N \le 100$). The second line contains N integers r_i ($-10^6 \le r_i \le 10^6$). The third line contains N integers c_i ($-10^6 \le c_i \le 10^6$). All r_i are different, all c_i are also different.

Output

Output one real number: the minimal sum of distances travelled by each person.

Your answer will be considered correct if its absolute or relative error does not exceed 10^{-9}. In more detail, let us assume that your answer is a, and the answer of the jury is b. The checking program will consider your answer correct if $\frac{|a-b|}{\max(1,b)} \le 10^{-9}$.

Example

make-a-row.in	make-a-row.out
3 1 4 5 -1 3 5	7.3730791040629358

Explanation

In the example, the optimal way is to put people in the row with coordinates: $(0.3964673051113727, 0)$, $(1.3964673051113727, 0)$, ..., $(5.3964673051113727, 0)$.

Problem F. (p, q)-Knight *(Division 2)*

Input file:	`pqknight.in`
Output file:	`pqknight.out`
Time limit:	1 second
Memory limit:	256 mebibytes

A (p, q)-knight stands at some cell of a flat chessboard which is infinite in all directions. In a single step, such knight can move from cell (r, c) to any of the cells $(r \pm p, c \pm q)$ and $(r \pm q, c \pm p)$. In particular, if p and q are different positive integers, such knight has eight possible moves from each cell. The regular chess knight is a special case of a (p, q)-knight for $p = 1$ and $q = 2$ (or $p = 2$ and $q = 1$).

What is the minimum required number of steps a (p, q)-knight has to make to arrive at a cell which shares a single side with the starting cell?

Input

The first line of input contains two integers p and q, the parameters of the knight ($0 \le p, q \le 100$).

Output

Print one integer: the minimum required number of steps a (p, q)-knight has to make to arrive at a cell which shares a single side with the starting cell. If the knight can never arrive at such a cell, print the number -1.

Examples

pqknight.in	pqknight.out
2 1	3
1 1	-1

Explanations

In the first example, the regular chess $(2, 1)$-knight can move from any cell (r, c) to eight possible cells in a single step: $(r-2, c-1)$, $(r-1, c-2)$, $(r+1, c-2)$, $(r+2, c-1)$, $(r+2, c+1)$, $(r+1, c+2)$, $(r-1, c+2)$, and $(r-2, c+1)$. If, for example, the knight starts at cell $(0, 0)$, the first step can bring him to $(1, 2)$, the second one to $(-1, 1)$, and the third one to $(1, 0)$. The cell $(1, 0)$ shares a single side with the starting cell $(0, 0)$. The task is completed in three steps. It can be checked that two or less steps will not be enough.

In the second example, the $(1, 1)$-knight can move from any cell (r, c) to four possible cells in a single step: $(r-1, c-1)$, $(r-1, c+1)$, $(r+1, c+1)$, and $(r+1, c-1)$. If the cells are painted in checkerboard order (a white cell shares sides with black cells, a black cell with white cells), such knight can only visit cells of the same color as the starting cell, but the target cells have different color.

Problem G. Random Wormholes *(Division 2)*

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

This is an interactive problem.

In the Foggy Way galaxy, there is an infinite number of stars which are numbered by integers. As a means of fast travel between the stars, one can use wormholes: hyperspace tunnels connecting the stars. However, not every pair of stars is connected by such a tunnel.

When the demiurge created Foggy Way, he set the common probability p of existence of a direct tunnel from any star i to any star j, and then left the details to chance. The probability of existence of each tunnel does not depend on the probabilities for other tunnels. All tunnels are directed, so when $i \neq j$, the tunnel from i to j is different from the tunnel from j to i, and each of them exists with probability p.

A starship is orbiting the star number 0. The navigational computer of the starship has limited functionality: it accepts requests in the form "move to star x by a straight tunnel" where x is an integer from 0 to $2^{32} - 1$. After receiving such request, if there indeed exists a direct tunnel from the star the ship is currently orbiting to star x, the starship moves to star x, and the navigation panel shows the word "`yes`". Otherwise, the panel responds with "`no`", and the starship remains in place.

The captain of the starship wants to arrive to the star number 1. You are the navigator of the starship. Help the captain get to the required star!

Interaction Protocol

Your solution must print each request on a separate line to the standard output stream. A request is a single integer x from 0 to $2^{32} - 1$: the number of the star to move to if there exists a direct tunnel from the current star leading there.

To prevent output buffering, flush the output buffer after each request: this can be done by using, for example, `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

The response, which is either "`yes`" or "`no`", is given to your solution via the standard input stream, also on a separate line.

As soon as the starship arrives to the star number 1, your solution must immediately terminate gracefully.

After 30 000 requests to navigational computer, it runs out of electricity and does not respond to further requests. In this case, the mission is failed.

In each test for this problem, the probability p is fixed as a percentage from 3 to 99. After that, for each possible pair of stars, the existence of a tunnel between them is also fixed.

Example

participant's requests	checking program's answers
1	no
3	yes
3	no
1	yes

Explanation

Please note that the participant's **output** is displayed to the **left**, and the following **input** to the **right**.

The example illustrates the first test in the testing system. The probability of existence of each tunnel is 50%. There is no direct tunnel from star 0 to star 1. However, the starship successfully moves from star 0 to star 3. After that, the navigator tries to move from star 3 to itself, but there is no such tunnel. Finally, it turns out that the tunnel from star 3 to star 1 exists, and the mission is accomplished.

Problem H. Second Maximum *(Division 2)*

Input file:	`secondmax.in`
Output file:	`secondmax.out`
Time limit:	1 second
Memory limit:	256 mebibytes

"Cat in Tank, Inc." wants to take over the world again. This time, they are going to use a special device which generates random real numbers from l to r.

The device has a wonderful property: if one takes a segment located completely inside the segment from l to r, the probability of getting a random point inside the segment depends on the length of the segment only and equals to $\frac{length}{r-l}$.

The device will be used n times with possibly different values of l and r. We have no idea how this is connected with taking over the world, but it is crucial for the company to know the expected value of the second maximum among n occurred values. You certainly can calculate it, can't you?

Recall that the second maximum of a sequence is the second value in the sequence after we sort it in descending order.

Input

The first line contains an integer n ($2 \le n \le 50$). Next n lines contain two integers each: l_i and r_i ($-100 \le l_i < r_i \le 100$).

Output

Output a single line with a single real number on it: the expected value of the second maximum.

Your answer will be considered correct if its absolute or relative error does not exceed 10^{-6}. In more detail, let us assume that your answer is a, and the answer of the jury is b. The checking program will consider your answer correct if $\frac{|a-b|}{\max(1,b)} \le 10^{-6}$.

Example

secondmax.in	secondmax.out
3 0 1 0 1 -1 0	0.3333333333333333

Problem I. Ticket To Ride *(Division 2)*

Input file:	`ttr.in`
Output file:	`ttr.out`
Time limit:	1 second
Memory limit:	256 mebibytes

Oleg and his friends love to play the board game "Ticket To Ride". In this game, each player builds his own network of railways.

Attention! In this problem, the rules of "Ticket To Ride" are slightly different from the standard rules. We recommend you to read the statement carefully even if you know the standard rules.

To build the network of railways, players use colored Train Car cards as resources. There are eight types of colored Train Car cards, in quantities of 12 each, and 14 Locomotive cards (can be used as a card of any color). The colors of each type of Train Car cards match the colors of various routes between cities on the map: Purple, Blue, Orange, White, Green, Yellow, Black, and Red. Thus, there are 110 cards in the deck.

As the game starts, the deck is shuffled and laid on the table face down. The first five cards are opened and laid on the table face up in a row.

At the start of the game, each player has no cards. Players make moves in turn. Each player must choose one of two alternatives:

- take several resource cards, or
- build a new route using resource cards from the player's hand.

In one move, a player can take **exactly two** resource cards one by one, or **only one** resource card if he takes an open Locomotive card. Each of the cards taken is either one of the five open cards or the top card of the deck (in the latter case, other players won't see what card was taken). If the player takes one of the five open cards, it is immediately replaced by the next card of the deck. However, if the player takes an **open Locomotive** card, it must be the **only** resource card taken in this move.

When the deck becomes empty, all discarded cards are reshuffled and become the new deck. If at some point of time there are at least three Locomotive cards among the five open cards, all current open cards are immediately discarded, and after that, new five cards are drawn from the deck.

In one move, a player can build exactly one route using some or all cards from his hand. The cards which were used for building are immediately discarded and become known to other players. Each route requires a certain amount of resource cards of certain colors, depending on its type and parameters. A route can have one of the following types:

- simple route colored by one of the eight colors: to build it, a player needs *length* cards of color *color*;
- simple uncolored route: to build it, a player needs *length* cards of any one color (the player can choose any color he likes);
- ferry route: to build it, a player needs *length* cards of any one color (the player can choose any color he likes), and at least L of them must be Locomotive cards.

When building a route, a player can use Locomotive card as a Train Car card of any color. So, for building a simple blue route of length 3, a player can use three blue Train Car cards, or one blue Train Car cards and two Locomotive cards, or three Locomotive cards. And for building a simple uncolored route of length 5, a player can use four red Train Car cards and one Locomotive card, but can not use three blue Train Car cards and two orange Train Car cards.

During the game, it sometimes makes sense to know whether it is possible that a certain player can build a certain route. Obviously, the players don't know the order of the cards in the deck, and consequently don't always know which cards are in the hand of another player. But if one carefully tracks open and

discarded cards, sometimes one can be pretty sure that another player has not got enough resources to build a certain route.

Oleg decided to write a "Ticket To Ride" application where players can get answers to such questions. Of course, the program must take into account what information is known and what information is hidden from the player asking the question. Can you help Oleg to cope with this task?

Input

In this problem, there are eight core colors. The following names are used to describe them: `purple`, `blue`, `orange`, `white`, `green`, `yellow`, `black`, and `red`. To describe each card taken from the deck, we use the name of its color or the word `locomotive`.

The first line contains two integers n and m: the number of players and the number of following lines with game description ($2 \le n \le 5$, $2 \le m \le 10\,000$).

The next line contains the description of cards which were put on the table at the start of the game. This line takes the form `open` <*five_cards*>$\{1, 6\}$ where *five_cards* are five words separated by spaces, describing the five cards which were laid out on the table. Here and forth, the numbers in curly brackets indicate the range of the number of possible repetitions of the preceding element, in this case, *five_cards*. If there are at least three Locomotive cards on the table, new five cards are laid out, and so on. It is guaranteed that in the tests for this problem, this situation arises no more than five times in a row. That is why the description of the `open` command can contain 5, 10, 15, 20, 25 or 30 words.

After that, there are descriptions of the actions of the players and their questions, one per line.

- `take blind` <*card_color*>: the player took the top card from the deck face down, and this card was *card_color*;
- `take` <*i*> <*card_color*> <*five_cards*>$\{0, 5\}$: the player took i-th open card, and it was replaced by card *card_color*; additionally, this command supplies from zero to five lists of five cards for the case when the open cards contain three or more Locomotive cards;
- `build` <*color*> <*length*> <*card_numbers_list*>: the player built a route colored by *color* with length *length* using cards *card_numbers_list*;
- `build any` <*length*> <*card_numbers_list*>: the player built an uncolored route with length *length* using cards *card_numbers_list*;
- `build ferry` <*length*> <*L*> <*card_numbers_list*>: the player built a ferry route with length *length* which required L Locomotive cards using cards *card_numbers_list*;
- `?` <*player*> <*color*> <*length*>: can the player number *player* build a route colored by *color* with length *length*;
- `?` <*player*> `any` <*length*>: can the player number *player* build an uncolored route with length *length*;
- `?` <*player*> `ferry` <*length*> <*L*>: can the player number *player* build a ferry route with length *length* which requires L Locomotive cards.

In all these lines, the parameters satisfy the following restrictions: $1 \le i \le 5$, $1 \le player \le n$, $1 \le length \le 26$, $1 \le L \le 14$, $L \le length$. The value of *color* is the name of one of the eight colors. A list *card_numbers_list* consists of *length* integers separated by spaces: the numbers of cards from a player's hand used for building a route. These numbers are listed in ascending order. Each player always keeps the cards in his hand in the order in which he took them. Cards in a player's hand are numbered from one starting with the card which was taken before the rest.

Note that players take turns in sequence, so in the next action record, the number of the player making the move is not specified: this information would be redundant. Each question is asked by the player who has to do the next action. Asking questions does not count as an action.

It is guaranteed that the described sequence of actions is correct and the players followed all the rules described in this task. It is guaranteed that each `build` command contains the correct data to build a route successfully.

Output

For each question "`?`", print one of the two possible answers: "`No`" if the player in question surely can not build such a route, or "`Maybe`" otherwise.

Example

ttr.in

```
3 16
open locomotive blue locomotive locomotive blue blue red purple locomotive blue
take blind blue
take blind blue
take 4 red
take blind blue
take 2 white
? 3 any 2
? 3 blue 2
build blue 1 1
? 1 blue 4
? 1 purple 1
? 2 ferry 1 1
? 2 red 1
take blind yellow
take blind orange
? 2 ferry 3 1
```

ttr.out

```
Maybe
No
No
Maybe
Maybe
Maybe
Maybe
```