Big Stonks (stonks)

Since he learned Rust, Dario is now able to write software that would have never been possible with inferior languages such as C/C++. For example, he recently wrote a software that can predict with perfect accuracy the stock market prices for the next N days. Knowing these prices, he is now looking for a strategy that will make him rich.

Dario selected K individual stocks, and for each of them he computed the buying price B and selling price S for each of the N consecutive days.

For example, let's say that Dario's list contains $K = 3$ stocks ([A]pple, [F]acebook, [G]oogle) and that he computed their prices for $N = 2$ consecutive days:

Day	Buy[A]	Sell[A]	Buy[F]	Sell[F]	Buy[G]	Sell[G]
1	5	3	6	2	5	4
2	8	6	7	6	6	5

The prices in the list are expressed in Euro, and are always integers. The total starting budget that Dario can invest is 1 Euro. For the purpose of this problem, we assume that it's possible to buy **any fraction** of a stock (e.g. you could buy $1/5$ of an Apple stock on the first day for 1 Euro)

Help Dario find the maximum profit that he can make by optimally investing 1 Euro over those N days.

☞ Among the attachments of this task you may find a template file `stonks.*` with a sample incomplete implementation.

Input

The first line contains two integers N and K.

N lines follow: the i-th line contains K pairs of integers: the buying price $B_{i,j}$ and selling price $S_{i,j}$ for the j-th stock on the i-th day (with $j = 0, 1, \ldots, K - 1$).

Output

You need to write a single line with an integer: the maximum possible amount of Euro that Dario can have at the end of the N days. The answer will be considered correct if the absolute or relative error is lower than 10^{-6}.

Constraints

- $1 \leq N \leq 3000$.
- $1 \leq K \leq 3000$.
- $1 \leq S_{i,j} \leq B_{i,j} \leq 100$ for each $i = 0 \ldots N-1$ and $j = 0 \ldots K-1$.
- It is guaranteed that the answer will fit in a 64 bit floating point variable (`double`).

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (25 points) $K = 1$.

– **Subtask 3** (42 points) $N, K \leq 300$.

– **Subtask 4** (33 points) No additional limitations.

Examples

input	output

`2 3`
`5 3   6 2   5 4`
`8 6   7 6   6 5`
	`1.2`

`3 3`
`5 3     6 2     5 4`
`8 6     7 6     6 5`
`14 13   15 14   11 9`
	`2.6`

Explanation

In the **first sample case**, Dario can buy $1/5$ of the first stock on the first day (when it costs 5 Euro) and sell it on the second day (when it sells for 6 Euro), scoring $1/5 \times 6 = 1.20$ Euro.

In the **second sample case** note that first two days are unchanged. One almost-optimal strategy in this case would be to reinvest the 1.20 Euro of the second day by buying $1.2/7 = 0.1714285714$ units of the second stock, which can be sold on the last day for $1.2/7 \times 14 = 2.40$ Euro. However, there is an even better solution: keep the $1/5$ units of the first stock bought on the first day, and sell it on the third day for $1/5 \times 13 = 2.60$ Euro!