Efficient generation of "true" random numbers is an interesting problem in Computer Science. The reasonably fast random number generators (RNGs) that are provided in code libraries are usually merely "pseudorandom". They have a finite cycle length and their low-order bits can be predictable to some extent.

Even if the RNGs themselves could be perfect, their common usage is not. For instance, if you have a function that returns a true random integer between 1 and 100 inclusive, and you want to choose between three events, each with probability 1/3, you cannot do it with just one RNG call.

However, if we allow for making more RNG calls if necessary, it is possible to achieve a perfectly uniform distribution over the three events. In fact, any perfect RNG can be used to choose among ANY set of events and associated probabilities with perfect accuracy.

Assume that you have access to a perfect RNG that returns numbers between 1 and R inclusive, and the desired probabilities of N disjoint events. You must determine an algorithm that uses the RNG to choose among the events with exactly the correct probability, while minimizing the expected (average) number of RNG calls.

Input

Input will consist of at most 32 cases, each consisting of two lines. The first line will contain two integers, R and N, satisfying $2 \leq R \leq 1000$ and $2 \leq N \leq 1000$. The second line will contain N pairs of integers a_i and b_i, with $1 \leq a_i < b_i \leq 1000$. The desired probability of event i is a_i/b_i. The sum of the N probabilities will be 1.

Input will be terminated by a line containing two zeros.

Output

For each case, output, rounded to six fractional digits, the minimum expected number of random number generator calls required to decide among the events.

Sample Input

```
100 3
1 3 1 3 1 3
2 4
1 4 1 4 1 4 1 4
0 0
```

Sample Output

```
1.010101
2.000000
```