

[illegible]

B. Mean Requests

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

In this problem you will have to deal with a real algorithm that is used in the VK social network.

As in any other company that creates high-loaded websites, the VK developers have to deal with request statistics regularly. An important indicator reflecting the load of the site is the mean number of requests for a certain period of time of T seconds (for example, $T = 60 \text{ seconds} = 1 \text{ min}$ and $T = 86400 \text{ seconds} = 1 \text{ day}$). For example, if this value drops dramatically, that shows that the site has access problem. If this value grows, that may be a reason to analyze the cause for the growth and add more servers to the website if it is really needed.

However, even such a natural problem as counting the mean number of queries for some period of time can be a challenge when you process the amount of data of a huge social network. That's why the developers have to use original techniques to solve problems approximately, but more effectively at the same time.

Let's consider the following formal model. We have a service that works for n seconds. We know the number of queries to this resource a_t at each moment of time t ($1 \leq t \leq n$). Let's formulate the following algorithm *of calculating the mean with exponential decay*. Let c be some real number, strictly larger than one.

```
// setting this constant value correctly can adjust
// the time range for which statistics will be calculated
double c = some_constant_value;

// as the result of the algorithm's performance this variable will contain
// the mean number of queries for the last
// T seconds by the current moment of time
double mean = 0.0;

for t = 1..n: // at each second, we do the following:
    // a_t is the number of queries that came at the last second;
    mean = (mean + a_t / T) / c;
```

Thus, the mean variable is recalculated each second using the number of queries that came at that second. We can make some mathematical calculations and prove that choosing the value of constant c correctly will make the value of `mean` not very different from the real mean value a_x at $t - T + 1 \leq x \leq t$.

The advantage of such approach is that it only uses the number of requests at the current moment of time and doesn't require storing the history of requests for a large time range. Also, it considers the recent values with the weight larger than the weight of the old ones, which helps to react to dramatic change in values quicker.

However before using the new theoretical approach in industrial programming, there is an obligatory step to make, that is, to test its credibility practically on given test data sets. Your task is to compare the data obtained as a result of the work of an approximate algorithm to the real data.

You are given n values a_t , integer T and real number c . Also, you are given m moments p_j ($1 \leq j \leq m$), where we are interested in the mean value of the number of queries for the last T seconds. Implement two algorithms. The first one should calculate the required value by definition, i.e. by the formula $\frac{a_{p_j-T+1} + a_{p_j-T+2} + \dots + a_{p_j}}{T}$. The second algorithm should calculate the mean value as is described above. Print both values and calculate the relative error of the second algorithm by the formula $\frac{|approx - real|}{real}$, where *approx* is the approximate value, obtained by the second algorithm, and *real* is the exact value obtained by the first algorithm.

Input

The first line contains integer n ($1 \leq n \leq 2 \cdot 10^5$), integer T ($1 \leq T \leq n$) and real number c ($1 < c \leq 100$) — the time range when the resource should work, the length of the time range during which we need the mean number of requests and the coefficient c of the work of approximate algorithm. Number c is given with exactly six digits after the decimal point.

The next line contains n integers a_t ($1 \leq a_t \leq 10^6$) — the number of queries to the service at each moment of time.

The next line contains integer m ($1 \leq m \leq n$) — the number of moments of time when we are interested in the mean number of queries for the last T seconds.

The next line contains m integers p_j ($T \leq p_j \leq n$), representing another moment of time for which we need statistics. Moments p_j are strictly increasing.

Output

Print m lines. The j -th line must contain three numbers *real*, *approx* and *error*, where:

- $real = \frac{a_{p_j-T+1} + a_{p_j-T+2} + \dots + a_{p_j}}{T}$ is the real mean number of queries for the last T seconds;
- approx* is calculated by the given algorithm and equals *mean* at the moment of time $t = p_j$ (that is, after implementing the p_j -th iteration of the cycle);

- $error = \frac{|approx - real|}{real}$ is the relative error of the approximate algorithm.

The numbers you printed will be compared to the correct numbers with the relative or absolute error 10^{-4} . It is recommended to print the numbers with at least five digits after the decimal point.

Sample test(s)

input
1 1 2.000000 1 1 1
output
1.000000 0.500000 0.500000

input
11 4 1.250000 9 11 7 5 15 6 6 6 6 6 8 4 5 6 7 8 9 10 11
output
8.000000 4.449600 0.443800 9.500000 6.559680 0.309507 8.250000 6.447744 0.218455 8.000000 6.358195 0.205226 8.250000 6.286556 0.237993 6.000000 6.229245 0.038207 6.000000 6.183396 0.030566 6.000000 6.146717 0.024453

input
13 4 1.250000 3 3 3 3 20 3 3 3 3 3 3 10 4 5 6 7 8 9 10 11 12 13
output
3.000000 1.771200 0.409600 3.000000 2.016960 0.327680 7.250000 5.613568 0.225715 7.250000 5.090854 0.297813 7.250000 4.672684 0.355492 7.250000 4.338147 0.401635 3.000000 4.070517 0.356839 3.000000 3.856414 0.285471 3.000000 3.685131 0.228377 3.000000 3.548105 0.182702

C. Name Quest

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

A Martian boy is named s — he has got this name quite recently from his parents for his coming of age birthday. Now he enjoys looking for his name everywhere. If he sees that he can obtain his name from some string by removing zero or more letters (at that, the remaining letters remain in the same order), he gets happy. For example, if $s = \text{«aba»}$, then strings «baobab» , «aabbab» , «helloabahello» make him very happy and strings «aab» , «baaa» and «helloabhello» do not.

However rather than being happy once, he loves twice as much being happy twice! So, when he got string t as a present, he wanted to cut it in two parts (the left part and the right part) so that each part made him happy.

Help s determine the number of distinct ways to cut the given string t into two parts in the required manner.

Input

The first line contains string s , consisting of lowercase English letters. The length of string s is from 1 to 1000 letters.

The second line contains string t , that also consists of lowercase English letters. The length of string t is from 1 to 10^6 letters.

Output

Print the sought number of ways to cut string t in two so that each part made s happy.

Sample test(s)

input
aba baobababbah
output
2

input
mars sunvenusearthmarsjupitersaturnuranusneptune
output
0

D. Statistics of Recompressing Videos

time limit per test: 3 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

A social network for dogs called DH (DogHouse) has k special servers to recompress uploaded videos of cute cats. After each video is uploaded, it should be recompressed on one (any) of the servers, and only after that it can be saved in the social network.

We know that each server takes one second to recompress a one minute fragment. Thus, any server takes m seconds to recompress a m minute video.

We know the time when each of the n videos were uploaded to the network (in seconds starting from the moment all servers started working). All videos appear at different moments of time and they are recompressed in the order they appear. If some video appeared at time s , then its recompressing can start at that very moment, immediately. Some videos can await recompressing when all the servers are busy. In this case, as soon as a server is available, it immediately starts recompressing another video. The videos that await recompressing go in a queue. If by the moment the videos started being recompressed some servers are available, then any of them starts recompressing the video.

For each video find the moment it stops being recompressed.

Input

The first line of the input contains integers n and k ($1 \leq n, k \leq 5 \cdot 10^5$) — the number of videos and servers, respectively.

Next n lines contain the descriptions of the videos as pairs of integers s_i, m_i ($1 \leq s_i, m_i \leq 10^9$), where s_i is the time in seconds when the i -th video appeared and m_i is its duration in minutes. It is guaranteed that all the s_i 's are distinct and the videos are given in the chronological order of upload, that is in the order of increasing s_i .

Output

Print n numbers e_1, e_2, \dots, e_n , where e_i is the time in seconds after the servers start working, when the i -th video will be recompressed.

Sample test(s)

input
3 2 1 5 2 5 3 5
output
6 7 11

input
6 1 1 1000000000 2 1000000000 3 1000000000 4 1000000000 5 1000000000 6 3
output
1000000001 2000000001 3000000001 4000000001 5000000001 5000000004