Problem A. 1024 Stack Edition

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	512 megabytes

Recently Ksyusha bought an application "1024 Stack Edition" (turn-based arcade) in an online shop. As you might guess from the title, the game is about some stack filled up with powers of two. According to the rules, the player can do one of the following three actions on each move:

- Add a random number to the top of the stack.
- Delete the number from the top of stack.
- If the two numbers on the top of the stack are equal, replace them with their sum.

If the player chooses to add a random number to the top of the stack, this number will be equal to 1 with probability p and equal to 2 with the remaining probability $(1-p)$ independently of other generated random numbers. If a player chooses to remove the number from the top of the stack, she pays a fine of one coin. Note that any other action except removing the number from the top of the stack does not lead to paying the fine. If the stack consists of a single number 1024 and nothing else, the player wins the game. When it happens, the less the player paid as fine, the better.

Once Ksyusha had left her smartphone unattended. After coming back, she found out that somebody played "1024 Stack Edition". Ksyusha does not care who was it and why. She is interested in calculating the expected value of the fine that she will pay while completing the game (given that she plays optimally). Your task is to find this number.

Input

The first line of input contains two integers: the initial size of the stack N ($0 \le N \le 100\,000$) and the probability p ($0 < p \le 100$) expressed **as a percentage**. The second line contains the sequence of N positive integers s_i ($s_i = 2^h$ where h is an integer and $0 \le h \le 10$) that are initially located in the stack, starting with the deepest elements (the top of the stack is the last number in the line). Note that the stack may be empty, which means there will be no numbers on the second line. Also note that there is no upper bound on the size of the stack, only the initial size is given.

Output

Print the expected number of coins that Ksyusha will need to pay while finishing the game given in the input. The absolute or relative error should not exceed 10^{-6}.

Examples

stdin	stdout
10 50 512 256 128 64 32 16 8 4 2 1	1.000000000
9 50 512 256 128 64 32 16 8 4 2	0.500000000

Problem B. String Problem

Input file:	stdin
Output file:	stdout
Time limit:	4 seconds
Memory limit:	512 megabytes

Search requests are great! An edit box and a search button order the whole internet in front of you, in search of an answer. Millions of search requests are carefully stored to allow answering the questions faster. To make a search, even faster users receive hints as they type the question. But what if we know nothing about the user?

You are working on an algorithm that will order two request hints in lexicographical order. Well, almost lexicographical.

Users can make typos while entering a search request. We say that string A is *almost less* than string B if string A is lexicographically less than string B, or we can replace one letter in A with another lowercase English letter so that the resulting string is lexicographically less than string B.

You are given a set of N words which are the search hints. Calculate the number of ordered pairs of hints in which the first hint is *almost less* than the second hint. The order of the words matters: if two hints are *almost less* than each other, you need to count both ordered pairs.

Input

The first line of input contains one integer N ($1 \le N \le 10^6$), the number of given search hints. Each of the next N lines contains a single word which is a search hint. Words consist only of lowercase English letters, and their total length does not exceed 10^6 characters.

Output

Print one integer: the number of pairs of search hints such that the first one is *almost less* than the second.

Examples

stdin	stdout
3 a b c	4
2 aab aba	2
2 aa aaa	1

Problem C. Dice

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	512 megabytes

You are given a cube. Each of its six faces has an integer from 1 to 6 corresponding to it. Different faces have different integers corresponding to them.

You are also given a hexomino: a planar polygon made of six equal-sized squares connected edge-to-edge. Each of its six squares also has an integer from 1 to 6 corresponding to it. Again, different squares have different integers corresponding to them.

Check if it is possible to fold the given hexomino into a cube which could be then rotated in such a way that the numbering of its faces corresponds to the numbering of the faces of the given cube. It is allowed to fold the hexomino along the borders of the squares, but it is not allowed to make cuts.

Input

The first line of input contains the integers corresponding to the faces of the cube. The faces are listed in the following order: front, back, left, right, top, bottom.

After that, the hexomino is given as an array of $K \times N$ squares: each of next K lines contains exactly N digits from 0 to 6 without spaces. Zeroes denote empty cells, and non-zero digits denote the cells of the hexomino.

You may assume that each row and each column of the array contains at least one non-zero digit, each non-zero digit from 1 to 6 can be found in the array exactly once, and non-zero digits form an edge-connected polygon.

Output

Print "Yes" if it is possible to obtain the given cube by folding the given hexomino, and "No" otherwise.

Examples

stdin	stdout
1 2 3 4 5 6 0100 3546 0200	Yes
1 6 2 5 3 4 1020 3546	No

Problem D. UFO

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	512 megabytes

Last night, an UFO crash landed in a small lake near Vasya's home.

UFO was constructed as an N-dimensional rectangular parallelepiped with integer lengths of edges. At the moment of crash, the joints and all the other parts of UFO except for titanium edges were burned out. The edges themselves however did not break.

In the morning, Vasya came to the lake and found K titanium sticks. Assuming that those sticks are edges of an UFO and that some of those edges are still under water, find out the minimum possible number of dimensions N of the place where this UFO was from.

Input

The first line of input contains one integer K ($1 \le K \le 10^6$) which is the number of sticks found by Vasya. The second line contains K space-separated integers a_i ($1 \le a_i \le 10^6$) which are the lengths of the sticks.

Output

Print one integer: the minimum possible number of dimensions N of the place where UFO came from.

Examples

stdin	stdout
3 1 2 3	3
3 1 2 2	2

Problem E. Unextendable Matching

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	512 megabytes

Artemka likes matchings very much.

A *matching* in an undirected graph is a set of pairwise disjoint edges.

A matching is called *unextendable* if it is impossible to add an edge into it without excluding any edges which are already present in the matching.

An undirected graph is called *bipartite* if it is possible to divide its vertices into two sets in such a way that each edge connects vertices from different sets.

Artemka has a bipartite graph. His task is to count the number of distinct unextendable matchings in the graph modulo $1\,000\,000\,007$ ($10^9 + 7$). Your task is to help him.

Input

The first line contains three integers n_1, n_2 and m ($1 \le n_1 \le 10$, $1 \le n_2 \le 100$, $0 \le m \le n_1 \cdot n_2$): the number of vertices in two parts and the number of edges, respectively. The next m lines contain the description of edges of the given graph. Each of these lines contains two integers v_1 and v_2 ($1 \le v_1 \le n_1$, $1 \le v_2 \le n_2$) which mean that an edge connects vertex v_1 of first part and vertex v_2 of second part. It is guaranteed that all the edges are unique.

Output

Print a single integer: the number of unextendable matchings modulo $1\,000\,000\,007$ ($10^9 + 7$).

Examples

stdin	stdout
2 3 4 1 1 1 2 1 3 2 2	3
2 2 4 1 1 1 2 2 1 2 2	2

Problem F. Musical World

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	512 megabytes

In the brand new Yandex.Music service, personal online radios are now even better than before. They take into account every single track a user listened to, and create a pool of tracks the user may like. Player randomly chooses one track from this pool to play, then another and so on.

But Yandex servers are so fast that they can predict recommendations not only for the currently playing track, but also for all recommended songs at once.

So, a user chooses the first track, then the algorithm builds recommendations for N songs forward. Each song in i-th generation can add at most K_i ($1 \le K_i \le 5$) new songs in generation $(i+1)$, and will add exactly j songs with probability $p_{i,j}$ for each j from 0 to K_i. The recommendations based on each single song are independent of other songs in the same generation.

The research was conducted on indie music, so consider that all songs are different. To calculate the variety of recommended music, the algorithm needs to compare each pair of tracks. To estimate the amount of memory required to run this algorithm, calculate the expected value of the number of pairs in the recommendation pool in generation $(N+1)$.

Input

The first line of input holds the number of generations N ($1 \le N \le 10^5$) you need to calculate the recommendations for.

Each of the next N lines holds the probability distribution for the number of recommended songs in generation. The i-th of these lines starts with an integer K_i ($1 \le K_i \le 5$) which is the maximum possible number of songs recommended based on one song in generation i. It is followed by (K_i+1) integers $a_{i,j}$ ($1 \le a_{i,j} \le 1000$) each of which is a non-normalized probability to recommend j songs in generation $(i+1)$ based on one song in generation i. Normalized probability $p_{i,j}$ can be calculated as $\frac{a_{i,j}}{\sum_{l=0}^{k} a_{i,l}}$.

The first generation contains only the first song, the second generation contains all the songs recommended based on the first song, the third generation contains all songs recommended based on each song in generation 2, and so on.

Output

As the answer for the problem can be very large, calculate it as an irreducible fraction $\frac{A}{B}$ and output $(A \cdot B^{-1}) \bmod (10^9+7)$. Here, B^{-1} is the multiplicative inverse of B modulo 10^9+7. The input constraints guarantee that B does not divide by 10^9+7, so this expression is properly defined.

Examples

stdin	stdout
2 2 1 1 1 2 1 1 1	666666672
2 2 1 1 1 2 1 1 2	520833338
2 2 1 1 2 2 1 1 1	916666674

Note

In the first example, both generations produce 0 to 2 tracks with probability of $\frac{1}{3}$.

Generation 3 will hold the following number of songs:

- 0 with probability $\frac{13}{27}$,
- 1 with probability $\frac{5}{27}$,
- 2 with probability $\frac{2}{9}$,
- 3 with probability $\frac{2}{27}$,
- 4 with probability $\frac{1}{27}$.

The expected value is therefore $\frac{2}{9}\binom{2}{2} + \frac{2}{27}\binom{3}{2} + \frac{1}{27}\binom{4}{2} = \frac{2}{3}$.
The answer for the second example is $\frac{49}{48}$, and for the third example it is $\frac{11}{12}$.