A. LCM Challenge

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Some days ago, I learned the concept of LCM (least common multiple). I've played with it for several times and I want to make a big number with it.

But I also don't want to use many numbers, so I'll choose three positive integers (they don't have to be distinct) which are not greater than n. Can you help me to find the maximum possible least common multiple of these three integers?

Input

The first line contains an integer n ($1 \le n \le 10^6$) — the n mentioned in the statement.

Output

Print a single integer — the maximum possible LCM of three not necessarily distinct positive integers that are not greater than n.

Examples

input
9
output
504

input
7
output
210

Note

The least common multiple of some positive integers is the least positive integer which is multiple for each of them.

The result may become very large, 32-bit integer won't be enough. So using 64-bit integers is recommended.

For the last example, we can chose numbers 7, 6, 5 and the LCM of them is $7 \cdot 6 \cdot 5 = 210$. It is the maximum value we can get.

B. Let's Play Osu!

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You're playing a game called Osu! Here's a simplified version of it. There are n clicks in a game. For each click there are two outcomes: correct or bad. Let us denote correct as "O", bad as "X", then the whole play can be encoded as a sequence of n characters "O" and "X".

Using the play sequence you can calculate the score for the play as follows: for every maximal consecutive "O"s block, add the square of its length (the number of characters "O") to the score. For example, if your play can be encoded as "OOXOOOXXOO", then there's three maximal consecutive "O"s block "OO", "OOO", "OO", so your score will be $2^2 + 3^2 + 2^2 = 17$. If there are no correct clicks in a play then the score for the play equals to 0.

You know that the probability to click the i-th $(1 \le i \le n)$ click correctly is p_i. In other words, the i-th character in the play sequence has p_i probability to be "O", $1 - p_i$ to be "X". You task is to calculate the expected score for your play.

Input

The first line contains an integer n ($1 \le n \le 10^5$) — the number of clicks. The second line contains n space-separated real numbers $p_1, p_2, ..., p_n$ $(0 \le p_i \le 1)$.

There will be at most six digits after the decimal point in the given p_i.

Output

Print a single real number — the expected score for your play. Your answer will be considered correct if its absolute or relative error does not exceed 10^{-6}.

Examples

input

```
3
0.5 0.5 0.5
```

output

```
2.750000000000000
```

input

```
4
0.7 0.2 0.1 0.9
```

output

```
2.489200000000000
```

input

```
5
1 1 1 1 1
```

output

```
25.000000000000000
```

Note

For the first example. There are 8 possible outcomes. Each has a probability of 0.125.

- "OOO" → $3^2 = 9$;
- "OOX" → $2^2 = 4$;
- "OXO" → $1^2 + 1^2 = 2$;
- "OXX" → $1^2 = 1$;
- "XOO" → $2^2 = 4$;
- "XOX" → $1^2 = 1$;
- "XXO" → $1^2 = 1$;
- "XXX" → 0.

So the expected score is $\frac{9+4+2+1+4+1+1}{8} = 2.75$

C. Cyclical Quest

time limit per test: 3 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Some days ago, WJMZBMR learned how to answer the query "how many times does a string x occur in a string s" quickly by preprocessing the string s. But now he wants to make it harder.

So he wants to ask "how many consecutive substrings of s are cyclical isomorphic to a given string x". You are given string s and n strings x_i, for each string x_i find, how many consecutive substrings of s are cyclical isomorphic to x_i.

Two strings are called *cyclical isomorphic* if one can rotate one string to get the other one. 'Rotate' here means 'to take some consecutive chars (maybe none) from the beginning of a string and put them back at the end of the string in the same order'. For example, string "abcde" can be rotated to string "deabc". We can take characters "abc" from the beginning and put them at the end of "de".

Input

The first line contains a non-empty string s. The length of string s is not greater than 10^6 characters.

The second line contains an integer n ($1 \le n \le 10^5$) — the number of queries. Then n lines follow: the i-th line contains the string x_i — the string for the i-th query. The total length of x_i is less than or equal to 10^6 characters.

In this problem, strings only consist of lowercase English letters.

Output

For each query x_i print a single integer that shows how many consecutive substrings of s are cyclical isomorphic to x_i. Print the answers to the queries in the order they are given in the input.

Examples

input

```
baabaabaaa
5
a
ba
baa
aabaa
aaba
```

output

```
7
5
7
3
5
```

input

```
aabbaa
3
aa
aabb
abba
```

output

```
2
3
3
```

D. Graph Game

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

In computer science, there is a method called "Divide And Conquer By Node" to solve some hard problems about paths on a tree. Let's desribe how this method works by function:

$solve(t)$ (t is a tree):

1. Chose a node x (it's common to chose weight-center) in tree t. Let's call this step "Line A".
2. Deal with all paths that pass x.
3. Then delete x from tree t.
4. After that t becomes some subtrees.
5. Apply $solve$ on each subtree.

This ends when t has only one node because after deleting it, there's nothing.

Now, WJMZBMR has mistakenly believed that it's ok to chose any node in "Line A". So he'll chose a node at random. To make the situation worse, he thinks a "tree" should have the same number of edges and nodes! So this procedure becomes like that.

Let's define the variable $totalCost$. Initially the value of $totalCost$ equal to 0. So, $solve(t)$ (now t is a graph):

1. $totalCost = totalCost + (size\ of\ t)$. The operation "=" means assignment. $(Size\ of\ t)$ means the number of nodes in t.
2. Choose a node x in graph t at random (uniformly among all nodes of t).
3. Then delete x from graph t.
4. After that t becomes some connected components.
5. Apply $solve$ on each component.

He'll apply $solve$ on a connected graph with n nodes and n edges. He thinks it will work quickly, but it's very slow. So he wants to know the expectation of $totalCost$ of this procedure. Can you help him?

Input

The first line contains an integer n ($3 \le n \le 3000$) — the number of nodes and edges in the graph. Each of the next n lines contains two space-separated integers a_i, b_i $(0 \le a_i, b_i \le n - 1)$ indicating an edge between nodes a_i and b_i.

Consider that the graph nodes are numbered from 0 to $(n - 1)$. It's guaranteed that there are no self-loops, no multiple edges in that graph. It's guaranteed that the graph is connected.

Output

Print a single real number — the expectation of $totalCost$. Your answer will be considered correct if its absolute or relative error does not exceed 10^{-6}.

Examples

input

```
5
3 4
2 3
2 4
0 4
1 2
```

output

```
13.1666666666666
```

input

```
3
0 1
1 2
0 2
```

output

```
6.00000000000000
```

input

```
5
0 1
1 2
2 0
```

```
3 0
4 1
```

output

```
13.166666666666666
```

Note

Consider the second example. No matter what we choose first, the $totalCost$ will always be $3 + 2 + 1 = 6$.

E. Number Challenge

time limit per test: 3 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Let's denote $d(n)$ as the number of divisors of a positive integer n. You are given three integers a, b and c. Your task is to calculate the following sum:

$$\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{c}d(i\cdot j\cdot k).$$

Find the sum modulo 1073741824 (2^{30}).

Input

The first line contains three space-separated integers a, b and c ($1 \le a, b, c \le 2000$).

Output

Print a single integer — the required sum modulo 1073741824 (2^{30}).

Examples

input
```
2 2 2
```
output
```
20
```

input
```
4 4 4
```
output
```
328
```

input
```
10 10 10
```
output
```
11536
```

Note

For the first example.

- $d(1\cdot1\cdot1) = d(1) = 1$;
- $d(1\cdot1\cdot2) = d(2) = 2$;
- $d(1\cdot2\cdot1) = d(2) = 2$;
- $d(1\cdot2\cdot2) = d(4) = 3$;
- $d(2\cdot1\cdot1) = d(2) = 2$;
- $d(2\cdot1\cdot2) = d(4) = 3$;
- $d(2\cdot2\cdot1) = d(4) = 3$;
- $d(2\cdot2\cdot2) = d(8) = 4$.

So the result is $1 + 2 + 2 + 3 + 2 + 3 + 3 + 4 = 20$.