Codeforces Round #356 (Div. 1)

A. Bear and Prime 100

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is an interactive problem. In the output section below you will see the information about flushing the output.

Bear Limak thinks of some hidden number — an integer from interval $[2, 100]$. Your task is to say if the hidden number is prime or composite.

Integer $x > 1$ is called prime if it has exactly two distinct divisors, 1 and x. If integer $x > 1$ is not prime, it's called composite.

You can ask up to 20 queries about divisors of the hidden number. In each query you should print an integer from interval $[2, 100]$. The system will answer "`yes`" if your integer is a divisor of the hidden number. Otherwise, the answer will be "`no`".

For example, if the hidden number is 14 then the system will answer "`yes`" only if you print 2, 7 or 14.

When you are done asking queries, print "`prime`" or "`composite`" and terminate your program.

You will get the `Wrong Answer` verdict if you ask more than 20 queries, or if you print an integer not from the range $[2, 100]$. Also, you will get the `Wrong Answer` verdict if the printed answer isn't correct.

You will get the `Idleness Limit Exceeded` verdict if you don't print anything (but you should) or if you forget about flushing the output (more info below).

Input

After each query you should read one string from the input. It will be "`yes`" if the printed integer is a divisor of the hidden number, and "`no`" otherwise.

Output

Up to 20 times you can ask a query — print an integer from interval $[2, 100]$ in one line. You have to both print the end-of-line character and `flush` the output. After flushing you should read a response from the input.

In any moment you can print the answer "`prime`" or "`composite`" (without the quotes). After that, flush the output and terminate your program.

To flush you can use (just after printing an integer and end-of-line):

- `fflush(stdout)` in C++;
- `System.out.flush()` in Java;
- `stdout.flush()` in Python;
- `flush(output)` in Pascal;
- See the documentation for other languages.

Hacking. To hack someone, as the input you should print the hidden number — one integer from the interval $[2, 100]$. Of course, his/her solution won't be able to read the hidden number from the input.

Examples

input
yes no yes
output
2 80 5 composite

input
no yes no no no
output
58

```
59
78
78
2
prime
```

Note

The hidden number in the first query is 30. In a table below you can see a better form of the provided example of the communication process.

The hidden number is divisible by both 2 and 5. Thus, it must be composite. Note that it isn't necessary to know the exact value of the hidden number. In this test, the hidden number is 30.

59 is a divisor of the hidden number. In the interval $[2, 100]$ there is only one number with this divisor. The hidden number must be 59, which is prime. Note that the answer is known even after the second query and you could print it then and terminate. Though, it isn't forbidden to ask unnecessary queries (unless you exceed the limit of 20 queries).

B. Bear and Tower of Cubes

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

Limak is a little polar bear. He plays by building towers from blocks. Every block is a cube with positive integer length of side. Limak has infinitely many blocks of each side length.

A block with side a has volume a^3. A tower consisting of blocks with sides $a_1, a_2, ..., a_k$ has the total volume ${a_1}^3 + {a_2}^3 + ... + {a_k}^3$.

Limak is going to build a tower. First, he asks you to tell him a positive integer X — the required total volume of the tower. Then, Limak adds new blocks greedily, one by one. Each time he adds the biggest block such that the total volume doesn't exceed X.

Limak asks you to choose X not greater than m. Also, he wants to maximize the number of blocks in the tower at the end (however, he still behaves greedily). Secondarily, he wants to maximize X.

Can you help Limak? Find the maximum number of blocks his tower can have and the maximum $X \le m$ that results this number of blocks.

Input

The only line of the input contains one integer m ($1 \le m \le 10^{15}$), meaning that Limak wants you to choose X between 1 and m, inclusive.

Output

Print two integers — the maximum number of blocks in the tower and the maximum required total volume X, resulting in the maximum number of blocks.

Examples

input
48
output
9 42

input
6
output
6 6

Note

In the first sample test, there will be 9 blocks if you choose $X = 23$ or $X = 42$. Limak wants to maximize X secondarily so you should choose 42.

In more detail, after choosing $X = 42$ the process of building a tower is:

- Limak takes a block with side 3 because it's the biggest block with volume not greater than 42. The remaining volume is $42 - 27 = 15$.
- The second added block has side 2, so the remaining volume is $15 - 8 = 7$.
- Finally, Limak adds 7 blocks with side 1, one by one.

So, there are 9 blocks in the tower. The total volume is $3^3 + 2^3 + 7 \cdot 1^3 = 27 + 8 + 7 = 42$.

C. Bear and Square Grid

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You have a grid with n rows and n columns. Each cell is either empty (denoted by '.') or blocked (denoted by 'X').

Two empty cells are *directly connected* if they share a side. Two cells (r_1, c_1) (located in the row r_1 and column c_1) and (r_2, c_2) are *connected* if there exists a sequence of empty cells that starts with (r_1, c_1), finishes with (r_2, c_2), and any two consecutive cells in this sequence are directly connected. A *connected component* is a set of empty cells such that any two cells in the component are connected, and there is no cell in this set that is connected to some cell not in this set.

Your friend Limak is a big grizzly bear. He is able to destroy any obstacles in some range. More precisely, you can choose a square of size $k \times k$ in the grid and Limak will transform all blocked cells there to empty ones. However, you can ask Limak to help only once.

The chosen square must be completely inside the grid. It's possible that Limak won't change anything because all cells are empty anyway.

You like big connected components. After Limak helps you, what is the maximum possible size of the biggest connected component in the grid?

Input

The first line of the input contains two integers n and k $(1 \leq k \leq n \leq 500)$ — the size of the grid and Limak's range, respectively.

Each of the next n lines contains a string with n characters, denoting the i-th row of the grid. Each character is '.' or 'X', denoting an empty cell or a blocked one, respectively.

Output

Print the maximum possible size (the number of cells) of the biggest connected component, after using Limak's help.

Examples

input
5 2 ..XXX XX.XX X.XXX X...X XXXX.
output
10

input
5 3 ..... .XXX. .XXX. .XXX. .....
output
25

Note

In the first sample, you can choose a square of size 2×2. It's optimal to choose a square in the red frame on the left drawing below. Then, you will get a connected component with 10 cells, marked blue in the right drawing.

D. Bear and Chase

time limit per test: 7 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Bearland has n cities, numbered 1 through n. There are m bidirectional roads. The i-th road connects two distinct cities a_i and b_i. No two roads connect the same pair of cities. It's possible to get from any city to any other city (using one or more roads).

The distance between cities a and b is defined as the minimum number of roads used to travel between a and b.

Limak is a grizzly bear. He is a criminal and your task is to catch him, or at least to try to catch him. You have only two days (today and tomorrow) and after that Limak is going to hide forever.

Your main weapon is BCD (Bear Criminal Detector). Where you are in some city, you can use BCD and it tells you the distance between you and a city where Limak currently is. Unfortunately, BCD can be used only once a day.

You don't know much about Limak's current location. You assume that he is in one of n cities, chosen uniformly at random (each city with probability). You decided for the following plan:

1. Choose one city and use BCD there.
    - After using BCD you can try to catch Limak (but maybe it isn't a good idea). In this case you choose one city and check it. You win if Limak is there. Otherwise, Limak becomes more careful and you will never catch him (you loose).
2. Wait 24 hours to use BCD again. You know that Limak will change his location during that time. In detail, he will choose uniformly at random one of roads from his initial city, and he will use the chosen road, going to some other city.
3. Tomorrow, you will again choose one city and use BCD there.
4. Finally, you will try to catch Limak. You will choose one city and check it. You will win if Limak is there, and loose otherwise.

Each time when you choose one of cities, you can choose any of n cities. Let's say it isn't a problem for you to quickly get somewhere.

What is the probability of finding Limak, if you behave optimally?

Input

The first line of the input contains two integers n and m ($2 \le n \le 400$,) — the number of cities and the number of roads, respectively.

Then, m lines follow. The i-th of them contains two integers a_i and b_i ($1 \le a_i, b_i \le n, a_i \ne b_i$) — cities connected by the i-th road.

No two roads connect the same pair of cities. It's possible to get from any city to any other city.

Output

Print one real number — the probability of finding Limak, if you behave optimally. Your answer will be considered correct if its absolute error does not exceed 10^{-6}.

Namely: let's assume that your answer is a, and the answer of the jury is b. The checker program will consider your answer correct if $|a - b| \le 10^{-6}$.

Examples

input
3 3 1 2 1 3 2 3
output
0.833333333333

input
5 4 1 2 3 1 5 1 1 4
output
1.000000000000

input
4 4 1 2 1 3 2 3 1 4
output
0.916666666667

input
5 5 1 2 2 3 3 4 4 5 1 5
output
0.900000000000

Note

In the first sample test, there are three cities and there is a road between every pair of cities. Let's analyze one of optimal scenarios.

1. Use BCD in city 1.
   - With probability Limak is in this city and BCD tells you that the distance is 0. You should try to catch him now and you win for sure.
   - With probability the distance is 1 because Limak is in city 2 or city 3. In this case you should wait for the second day.
2. You wait and Limak moves to some other city.
   - There is probability that Limak was in city 2 and then went to city 3.
   - that he went from 2 to 1.
   - that he went from 3 to 2.
   - that he went from 3 to 1.
3. Use BCD again in city 1 (though it's allowed to use it in some other city).
   - If the distance is 0 then you're sure Limak is in this city (you win).
   - If the distance is 1 then Limak is in city 2 or city 3. Then you should guess that he is in city 2 (guessing city 3 would be fine too).

You loose only if Limak was in city 2 first and then he moved to city 3. The probability of loosing is . The answer is .

E. Bear and Bad Powers of 42

time limit per test: 5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Limak, a bear, isn't good at handling queries. So, he asks you to do it.

We say that powers of 42 (numbers $1, 42, 1764, ...$) are *bad*. Other numbers are *good*.

You are given a sequence of n good integers $t_1, t_2, ..., t_n$. Your task is to handle q queries of three types:

1. `1 i` — print t_i in a separate line.
2. `2 a b x` — for set t_i to x. It's guaranteed that x is a good number.
3. `3 a b x` — for increase t_i by x. After this repeat the process while at least one t_i is bad.

You can note that after each query all t_i are good.

Input

The first line of the input contains two integers n and q $(1 \le n, q \le 100\,000)$ — the size of Limak's sequence and the number of queries, respectively.

The second line of the input contains n integers $t_1, t_2, ..., t_n$ $(2 \le t_i \le 10^9)$ — initial elements of Limak's sequence. All t_i are good.

Then, q lines follow. The i-th of them describes the i-th query. The first number in the line is an integer $type_i$ $(1 \le type_i \le 3)$ — the type of the query. There is at least one query of the first type, so the output won't be empty.

In queries of the second and the third type there is $1 \le a \le b \le n$.

In queries of the second type an integer x $(2 \le x \le 10^9)$ is guaranteed to be good.

In queries of the third type an integer x $(1 \le x \le 10^9)$ may be bad.

Output

For each query of the first type, print the answer in a separate line.

Example

input
6 12 40 1700 7 1672 4 1722 3 2 4 42 1 2 1 3 3 2 6 50 1 2 1 4 1 6 2 3 4 41 3 1 5 1 1 1 1 3 1 5

output
1742 49 1842 1814 1822 43 44 107

Note

After a query `3 2 4 42` the sequence is $40, 1742, 49, 1714, 4, 1722$.

After a query `3 2 6 50` the sequence is $40, 1842, 149, 1814, 104, 1822$.

After a query `2 3 4 41` the sequence is $40, 1842, 41, 41, 104, 1822$.

After a query `3 1 5 1` the sequence is $43, 1845, 44, 44, 107, 1822$.