## Codeforces Round #142 (Div. 2)

# A. Dragons

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

Kirito is stuck on a level of the MMORPG he is playing now. To move on in the game, he's got to defeat all $n$ dragons that live on this level. Kirito and the dragons have *strength*, which is represented by an integer. In the duel between two opponents the duel's outcome is determined by their strength. Initially, Kirito's strength equals $s$.

If Kirito starts duelling with the $i$-th ($1 \le i \le n$) dragon and Kirito's strength is not greater than the dragon's strength $x_i$, then Kirito loses the duel and dies. But if Kirito's strength is greater than the dragon's strength, then he defeats the dragon and gets a bonus strength increase by $y_i$.

Kirito can fight the dragons in any order. Determine whether he can move on to the next level of the game, that is, defeat all dragons without a single loss.

### Input

The first line contains two space-separated integers $s$ and $n$ ($1 \le s \le 10^4$, $1 \le n \le 10^3$). Then $n$ lines follow: the $i$-th line contains space-separated integers $x_i$ and $y_i$ ($1 \le x_i \le 10^4$, $0 \le y_i \le 10^4$) — the $i$-th dragon's strength and the bonus for defeating it.

### Output

On a single line print "YES" (without the quotes), if Kirito can move on to the next level and print "NO" (without the quotes), if he can't.

**Sample test(s)**

| input |
| --- |
| 2 2<br>1 99<br>100 0 |
| output |
| YES |

| input |
| --- |
| 10 1<br>100 100 |
| output |
| NO |

### Note

In the first sample Kirito's strength initially equals 2. As the first dragon's strength is less than 2, Kirito can fight it and defeat it. After that he gets the bonus and his strength increases to $2 + 99 = 101$. Now he can defeat the second dragon and move on to the next level.

In the second sample Kirito's strength is too small to defeat the only dragon and win.

# B. T-primes

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

We know that prime numbers are positive integers that have exactly two distinct positive divisors. Similarly, we'll call a positive integer $t$ *T-prime*, if $t$ has exactly three distinct positive divisors.

You are given an array of $n$ positive integers. For each of them determine whether it is T-prime or not.

## Input

The first line contains a single positive integer, $n$ ($1 \le n \le 10^5$), showing how many numbers are in the array. The next line contains $n$ space-separated integers $x_i$ ($1 \le x_i \le 10^{12}$).

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is advised to use the `cin`, `cout` streams or the `%I64d` specifier.

## Output

Print $n$ lines: the $i$-th line should contain "`YES`" (without the quotes), if number $x_i$ is T-prime, and "`NO`" (without the quotes), if it isn't.

**Sample test(s)**

| input |
| --- |
| 3<br>4 5 6 |
| output |
| YES<br>NO<br>NO |

## Note

The given test has three numbers. The first number 4 has exactly three divisors — 1, 2 and 4, thus the answer for this number is "`YES`". The second number 5 has two divisors (1 and 5), and the third number 6 has four divisors (1, 2, 3, 6), hence the answer for them is "`NO`".

# C. Shifts

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

You are given a table consisting of $n$ rows and $m$ columns. Each cell of the table contains a number, 0 or 1. In one move we can choose some row of the table and cyclically shift its values either one cell to the left, or one cell to the right.

To *cyclically shift* a table row one cell to the right means to move the value of each cell, except for the last one, to the right neighboring cell, and to move the value of the last cell to the first cell. A cyclical shift of a row to the left is performed similarly, but in the other direction. For example, if we cyclically shift a row "00110" one cell to the right, we get a row "00011", but if we shift a row "00110" one cell to the left, we get a row "01100".

Determine the minimum number of moves needed to make some table column consist only of numbers 1.

## Input

The first line contains two space-separated integers: $n$ ($1 \le n \le 100$) — the number of rows in the table and $m$ ($1 \le m \le 10^4$) — the number of columns in the table. Then $n$ lines follow, each of them contains $m$ characters "0" or "1": the $j$-th character of the $i$-th line describes the contents of the cell in the $i$-th row and in the $j$-th column of the table.

It is guaranteed that the description of the table contains no other characters besides "0" and "1".

## Output

Print a single number: the minimum number of moves needed to get only numbers 1 in some column of the table. If this is impossible, print -1.

**Sample test(s)**

| input |
| --- |
| 3 6<br>101010<br>000100<br>100000 |
| output |
| 3 |

| input |
| --- |
| 2 3<br>111<br>000 |
| output |
| -1 |

## Note

In the first sample one way to achieve the goal with the least number of moves is as follows: cyclically shift the second row to the right once, then shift the third row to the left twice. Then the table column before the last one will contain only 1s.

In the second sample one can't shift the rows to get a column containing only 1s.

# D. Planets

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

Goa'uld Apophis captured Jack O'Neill's team again! Jack himself was able to escape, but by that time Apophis's ship had already jumped to hyperspace. But Jack knows on what planet will Apophis land. In order to save his friends, Jack must repeatedly go through stargates to get to this planet.

Overall the galaxy has $n$ planets, indexed with numbers from 1 to $n$. Jack is on the planet with index 1, and Apophis will land on the planet with index $n$. Jack can move between some pairs of planets through stargates (he can move in both directions); the transfer takes a positive, and, perhaps, for different pairs of planets unequal number of seconds. Jack begins his journey at time 0.

It can be that other travellers are arriving to the planet where Jack is currently located. In this case, Jack has to wait for exactly 1 second before he can use the stargate. That is, if at time $t$ another traveller arrives to the planet, Jack can only pass through the stargate at time $t + 1$, unless there are more travellers arriving at time $t + 1$ to the same planet.

Knowing the information about travel times between the planets, and the times when Jack would not be able to use the stargate on particular planets, determine the minimum time in which he can get to the planet with index $n$.

## Input

The first line contains two space-separated integers: $n$ ($2 \le n \le 10^5$), the number of planets in the galaxy, and $m$ ($0 \le m \le 10^5$) — the number of pairs of planets between which Jack can travel using stargates. Then $m$ lines follow, containing three integers each: the $i$-th line contains numbers of planets $a_i$ and $b_i$ ($1 \le a_i, b_i \le n$, $a_i \ne b_i$), which are connected through stargates, and the integer transfer time (in seconds) $c_i$ ($1 \le c_i \le 10^4$) between these planets. It is guaranteed that between any pair of planets there is at most one stargate connection.

Then $n$ lines follow: the $i$-th line contains an integer $k_i$ ($0 \le k_i \le 10^5$) that denotes the number of moments of time when other travellers arrive to the planet with index $i$. Then $k_i$ distinct space-separated integers $t_{ij}$ ($0 \le t_{ij} < 10^9$) follow, sorted in ascending order. An integer $t_{ij}$ means that at time $t_{ij}$ (in seconds) another traveller arrives to the planet $i$. It is guaranteed that the sum of all $k_i$ does not exceed $10^5$.

## Output

Print a single number — the least amount of time Jack needs to get from planet 1 to planet $n$. If Jack can't get to planet $n$ in any amount of time, print number -1.

**Sample test(s)**

input

```
4 6
1 2 2
1 3 3
1 4 8
2 3 4
2 4 5
3 4 3
0
1 3
2 3 4
0
```

output

```
7
```

input

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

output

```
-1
```

## Note

In the first sample Jack has three ways to go from planet 1. If he moves to planet 4 at once, he spends 8 seconds. If he transfers to planet 3, he spends 3 seconds, but as other travellers arrive to planet 3 at time 3 and 4, he can travel to planet 4 only at time 5, thus spending 8 seconds in total. But if Jack moves to planet 2, and then — to planet 4, then he spends a total of only $2 + 5 = 7$ seconds.

In the second sample one can't get from planet 1 to planet 3 by moving through stargates.

# E. Triangles

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

Alice and Bob don't play games anymore. Now they study properties of all sorts of graphs together. Alice invented the following task: she takes a complete undirected graph with $n$ vertices, chooses some $m$ edges and keeps them. Bob gets the $\frac{n(n-1)}{2} - m$ remaining edges.

Alice and Bob are fond of "triangles" in graphs, that is, cycles of length 3. That's why they wonder: what total number of triangles is there in the two graphs formed by Alice and Bob's edges, correspondingly?

## Input

The first line contains two space-separated integers $n$ and $m$ ($1 \le n \le 10^6, 0 \le m \le 10^6$) — the number of vertices in the initial complete graph and the number of edges in Alice's graph, correspondingly. Then $m$ lines follow: the $i$-th line contains two space-separated integers $a_i$, $b_i$ ($1 \le a_i, b_i \le n$, $a_i \ne b_i$), — the numbers of the two vertices connected by the $i$-th edge in Alice's graph. It is guaranteed that Alice's graph contains no multiple edges and self-loops. It is guaranteed that the initial complete graph also contains no multiple edges and self-loops.

Consider the graph vertices to be indexed in some way from 1 to $n$.

## Output

Print a single number — the total number of cycles of length 3 in Alice and Bob's graphs together.

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is advised to use the `cin`, `cout` streams or the `%I64d` specifier.

**Sample test(s)**

| input |
| --- |
| 5 5<br>1 2<br>1 3<br>2 3<br>2 4<br>3 4 |

| output |
| --- |
| 3 |

| input |
| --- |
| 5 3<br>1 2<br>2 3<br>1 3 |

| output |
| --- |
| 4 |

## Note

In the first sample Alice has 2 triangles: (1, 2, 3) and (2, 3, 4). Bob's graph has only 1 triangle : (1, 4, 5). That's why the two graphs in total contain 3 triangles.

In the second sample Alice's graph has only one triangle: (1, 2, 3). Bob's graph has three triangles: (1, 4, 5), (2, 4, 5) and (3, 4, 5). In this case the answer to the problem is 4.