A Treasure Map

You have found a treasure map! The map leads you to several gold mines. The mines each produce gold each day, but the amount of gold that they produce diminishes each day. There are paths between the mines. It may take several days to go from one mine to another. You can collect all of the day's gold from a mine when you are there, but you have to move on, you cannot stay for multiple days at the same mine. However, you can return to a mine after leaving it.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing two integers ***n*** ($2 \le n \le 1{,}000$) and ***m*** ($1 \le m \le 1{,}000$), where ***n*** is the number of mines, and ***m*** is the number of paths.

The next ***n*** lines will each describe a mine with two integers, ***g*** ($1 \le g \le 1{,}000$) and ***d*** ($1 \le d \le 1{,}000$), where ***g*** is the amount of gold mined on day 1, and ***d*** is the amount by which the gold haul diminishes each day. For example, if ***g***=9 and ***d***=4, then on day 1, the mine produces 9, on day 2 it produces 5, on day 3 it produces 1, and from day 4 on, it produces 0 (the mines cannot produce negative amounts of gold). The mines are numbered 1..***n*** in the order that they appear in the input, and you start at mine 1 on day 1.

The next ***m*** lines will each describe a path with three integers, **a**, ***b*** ($1 \le a < b \le n$) and ***t*** ($1 \le t \le 100$), where the path goes from mine ***a*** to mine ***b***, and takes ***t*** days to traverse. The paths go in both directions, so that a path that goes from ***a*** to ***b*** can also be used to go from ***b*** to ***a***.

Output

Output a single integer, which is the maximum amount of gold that you can collect.

Sample Input	**Sample Output**

`2 1`
`10 1`
`10 2`
`1 2 1`
	`42`

`3 2`
`10 5`
`3 1`
`5 1`
`1 2 1`
`2 3 1`
	`16`

`3 3`
`20 6`
`8 2`
`6 1`
`1 2 1`
`2 3 1`
`1 3 1`
	`38`

`2 1`
`1 1`
`10 5`
`1 2 2`
	`1`

B Fear Factoring

The Slivians are afraid of factoring; it's just, well, difficult.

Really, they don't even care about the factors themselves, just how much they sum to.

We can define $F(n)$ as the sum of all of the factors of n; so $F(6) = 12$ and $F(12) = 28$. Your task is, given two integers a and b with $a \leq b$, to calculate

$$S = \sum_{a \leq n \leq b} F(n).$$

Input

The input consists of a single line containing space-separated integers a and b ($1 \leq a \leq b \leq 10^{12}$; $b - a \leq 10^6$).

Output

Print S on a single line.

Sample Input and Output

`101 101`	`102`

`28 28`	`56`

`1 10`	`87`

`987654456799 987654456799`	`987654456800`

C Jumping Haybales

Farmer John's cows have grown lazy and he would like to get them back in shape! He has decided that the best way to do this is by setting up haybales in the field and having the cows jump from the northwest corner of the field to the southeast corner. The field is an $\boldsymbol{n}\times\boldsymbol{n}$ square grid. Note that on a standard map, North is up, South is down, East is right and West is left.

A cow can only jump straight east or south, never west or north, or even southeast. They also have a limit ***k*** on how many cells they can jump. They can jump over haybales, empty spaces, or any combination, even if there are no haybales in between, but they cannot land on a haybale. Bessie wants to still be lazy and is interested in the minimum number of jumps to reach the southeast corner of the map from the northwest corner.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing two integers ***n*** and ***k*** ($1 \le \boldsymbol{n},\boldsymbol{k} \le 2{,}000$), where ***n*** is the size of one side of the square grid, and ***k*** is Bessie's limit on the number of cells she can jump. Each of the next ***n*** lines will contain a string with exactly ***n*** characters. Only the characters '**#**' (a haybale) and '**.**' (an empty space) will appear. The northwest and southeast corners of the field are guaranteed to be empty.

Output

Output a single integer, which is the minimum number of jumps Bessie needs to reach the southeast corner from the northwest corner, or **-1** if Bessie cannot make it.

Sample Input	**Sample Output**

`4 2`
`.###`
`#...`
`.#..`
`#.#.`
	`4`

`3 1`
`.#.`
`.#.`
`.#.`
	`-1`

D Move Away

Tommy has just completed college and is looking for his first job. A priority in his life is living close to his friends, but he wants to live as far away from his parents as possible.

You are given the locations of Tommy's friends and the maximum distance he would be willing to live away from each friend. You also know that Tommy's parents live at (0, 0) in the coordinate plane. Determine how far Tommy can live from his parents. (There will always be at least one point meeting these requirements.)

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with a single integer ***n*** ($1 \le n \le 50$), representing the number of friends Tommy has. The next ***n*** lines will each contain three integers: ***x***, ***y*** ($-1{,}000 \le x, y \le 1{,}000$) and ***d*** ($1 \le d \le 1{,}000$), representing the (***x***, ***y***) coordinate of his friend and the maximum distance ***d*** he is willing to live away from that friend.

Output

Output a single decimal number on a single line, equal to the maximum distance he can live from his parents while still being close enough to all of his friends. Output this number to exactly 3 decimal places, rounded.

Sample Input	Sample Output

4
1 0 1
0 1 1
-1 0 1
0 -1 1
	0.000

2
-1 0 1000
2 0 1000
	999.999

E Spinning up Palindromes

"Sabotage!", exclaimed J. R. Diddly, president and founder of Diddly Widgets Inc.

"Vandalism, perhaps. Nothing's actually been damaged." responded Robert Lackey, the chief accountant.

Both were staring up at the large counter suspended above the factory floor, a counter that had faithfully recorded the number of widgets that had come off the assembly line since the factory was opened. But someone had changed the number being displayed so that it formed...

"It's a palindrome." said Lackey. "It reads the same forwards as backwards."

"What I don't understand," said Diddly, "is why our security guards didn't catch the vandals during their regular sweeps. It must have taken them hours to click forward to this new number, one step at a time."

"No." replied Lackey. "Although we only advance the rightmost digit each time a new widget is built, it's possible to spin any of the digits. With a little planning, this might have taken only a few seconds."

Consider a digital counter consisting of k wheels, each showing a digit from `0` to `9`. Each wheel is mounted so that it can advance to the next digit in a single step, *e.g.*, from `3` to `4`, or from `8` to `9`.

It is also possible to advance from digit `9` to digit `0`. However, when this happens, the wheel on its immediate left will also advance to the next digit automatically. This can have a cascade effect on multiple wheels to the left, but they all happen in a single step.

Given the current setting of the counter, find the smallest number of steps until one can reach a palindrome. The palindrome must respect leading zeros, *e.g.*, `0011` is not a palindrome.

For example, for input `610`, it takes four steps. This can be done by incrementing the `6` wheel four times, resulting in `010`.

Input

The first line of input contains a string of k digits ($1 \leq k \leq 40$), representing the current setting of the counter.

Note that the input may contain leading zeros.

Output

Print, on a single line, the minimum number of wheel advances necessary to produce a palindrome.

Sample Input and Output

0	0
009990001	3
29998	5
610	4
981	2
9084194700940903797191718247801197019268	54

F Unsatisfying

Usually a computer scientist tries to satisfy some constraints. This time you will try to make some logical statements unsatisfiable.

You will be given a list of logical statements of the following form:

$$\begin{array}{rcr} p_1 & | & p_2 \\ \sim p_2 & | & p_3 \\ p_3 & | & \sim p_4 \end{array}$$

Here '**|**', the disjunctive statement, stands for logical **OR** (the result is **TRUE** if either proposition is **TRUE**, possibly both). '**~**' is negation, forcing the value to be the opposite truth value.

To satisfy a list of logical statements, you must assign truth values (**TRUE** or **FALSE**) to each variable such that all the given statements result as **TRUE**. Your task is to add disjunctive statements to the list to make the list of statements unsatisfiable. But you cannot use the negation symbol!

All disjunctive statements (both those given and ones you add) must have exactly 2 terms. The ones given can use negation, but the ones added cannot.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing two integers ***n*** and ***m*** ($1 \le n,m \le 2{,}000$), where ***n*** is the number of variables and ***m*** is the number of disjunctions. The variables will be numbered 1..***n***.

Each of the next ***m*** lines will contain two integers ***a*** and ***b*** ($1 \le |a|,|b| \le n$), representing the subscript in the variable. A negative value is the negated version of that variable.

Output

Output a single integer, which is the minimum number of disjunctive clauses to add to make the list unsatisfiable. If it is not possible, output -**1**.

Sample Input	Sample Output
2 1 1 2	-1

4 5
1 2
-1 -3
-2 3
3 -4
-2 -3
	1