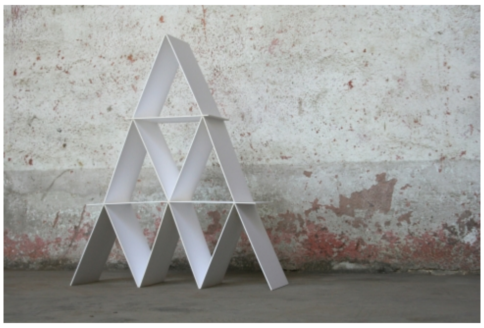
House of cards (cards)

Luca, Edoardo, William and Giorgio are playing at the famous card game *briscola*. While Edoardo was setting up the deck, he noticed that a card was missing. In fact, that same day he made a huge house of cards and he left the last card of his deck inside it! Now, he really wants to play with his friends without destroying his castle: help him find out how much of the castle must be torn down.



An house of cards is made up of triangles of cards. If you remove a card that is under another, all the castle falls in pieces. You can easily remove a triangle from the top without risking a disaster, but you want to minimize the number of triangles removed to keep the structure beautiful.

The card Edoardo wants is in the C-th triangle of the R-th row from the top of the castle, which has N rows total. Note that the i-th row (from 1 to N) has i triangles of cards.

How many triangles of cards do you have to remove *at least*, in order to have the missing card in hand?

☞ Among the attachments of this task you may find a template file `cards.*` with a sample incomplete implementation.

Input

The first and only line of the input contains three integers: N, R and C.

Output

You need to write a single line with an integer: the number of triangles to remove, including the one with the missing card.

Constraints

- $1 \leq C \leq R \leq N \leq 10^9$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

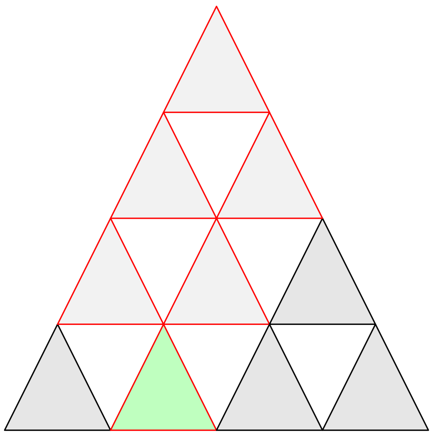
- **Subtask 1 [0 points]**: Examples.
- **Subtask 2 [20 points]**: $C = 1$.
- **Subtask 3 [10 points]**: $N \leq 5$.
- **Subtask 4 [20 points]**: $N \leq 1000$.
- **Subtask 5 [20 points]**: $N \leq 100\,000$.
- **Subtask 6 [30 points]**: No additional limitations.

Examples

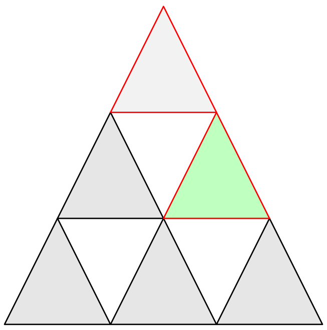
stdin/input.txt	stdout/output.txt
4 4 2	6
3 2 2	2

Explanation

In the **first example**, the castle has 4 rows and the missing card is in the second triangle of the last row. Thus you need to remove the following triangles (highlighted in red):



In the **second example**, the castle has 3 rows and the missing card is in the last triangle of the middle row. Thus you need to remove that triangle and the one above it.



Algorithmic excursion (`excursion`)

Giorgio is a passionate hiker, and every weekend goes for an excursion in his beloved mountains: *the Alps*.

Today, he is going to explore the *Val Troncea*, a nice rectangular valley consisting of $H \times W$ square meters of land, each with a different altitude of $A[i, j]$ millimeters above sea level (for $i = 0 \dots H-1$, $j = 0 \dots W-1$).

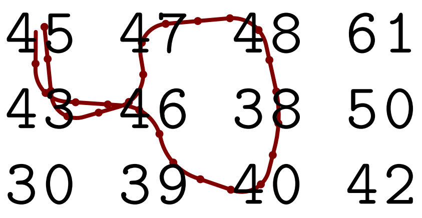
He will start from one of its corners, labelled $(0, 0)$, then he will wander around without following maps nor trails: he will follow an *algorithm*!

More specifically, Giorgio follows six simple rules:

Figure 1: A typical Giorgio's hike.

1. **No jumps:** step only between adjacent squares (in directions North, South, East or West).
2. **No flight:** never get out of the borders of the valley.
3. **No return:** never come back to the square from which you just arrived.
4. **No cliffs:** always choose the square with an altitude closer to the current one (least absolute difference of altitudes).
5. **No effort:** if you have to choose between two squares with the same altitude difference, then choose the one which goes downwards (with lower altitude).
6. **No repeats:** if you end up in a square where you have already been, then come back home (using the same path you used to get there on the first place).

For example, when Giorgio practised the hike in his $3m \times 4m$ back yard, he followed this path:



Starting from the 45, Giorgio could move to squares 43 or 47 by rules 1 and 2. Since rule 4 does not decide among them, rule 5 comes into play selecting the 43. Then, Giorgio could move either to squares 30 or 46 (by rules 1, 2, 3) and rule 4 selects the 46. The walk proceeds similarly with squares 47, 48, 38, 40 and 39; until it reaches again square 46. Following rule 6, Giorgio has to come back through the same path used before (squares 43 and 45), instead of looping through the cycle he just closed.

The next round of the IOIT is approaching, and Luca, Edoardo and William are worried that Giorgio might not come back in time! Help them calculate how long the hike will take, given the *Val Troncea* altitudes map and knowing that Giorgio takes exactly one step per second.

☞ Among the attachments of this task you may find a template file `excursion.*` with a sample incomplete implementation.

Input

The first line contains the two integers H, W. The following H lines for $i = 0 \ldots H-1$ contain W integers each: altitudes $A[i,j]$ for $j = 0 \ldots W-1$.

Output

You need to write a single line with an integer: the number of steps (or, equivalently, seconds) that Giorgio's hike will take.

Constraints

- $2 \leq H, W \leq 2000$.
- $0 \leq A[i,j] \leq 4\,800\,000$ for each $i = 0 \ldots H-1$, $j = 0 \ldots W-1$.
- The altitudes are all distinct: $A[i,j] \neq A[k,h]$ if $(i,j) \neq (k,h)$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**0 points**]: Examples.
- **Subtask 2** [**20 points**]: Giorgio's hike encompasses altitudes $0, 1, \ldots, k$ in order.
- **Subtask 3** [**30 points**]: $H, W \leq 10$.
- **Subtask 4** [**30 points**]: $H, W \leq 100$.
- **Subtask 5** [**20 points**]: No additional limitations.

Examples

`input.txt`	`output.txt`

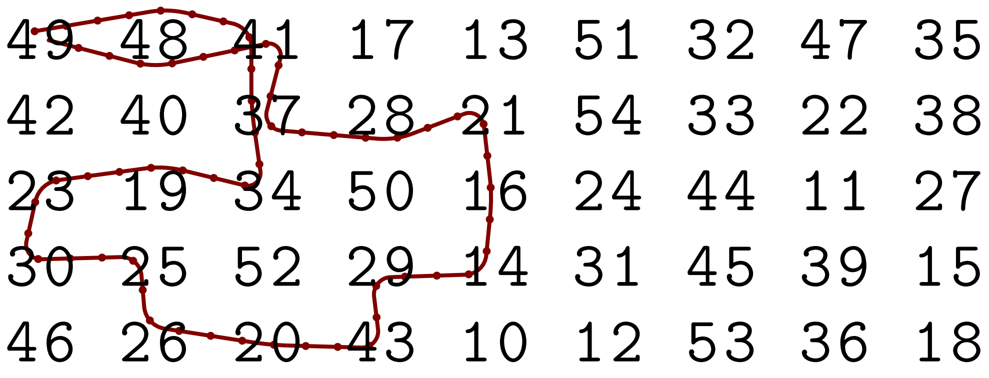
`3 4`
`45 47 48 61`
`43 46 38 50`
`30 39 40 42`
	`10`

`5 9`
`49 48 41 17 13 51 32 47 35`
`42 40 37 28 21 54 33 22 38`
`23 19 34 50 16 24 44 11 27`
`30 25 52 29 14 31 45 39 15`
`46 26 20 43 10 12 53 36 18`
	`20`

Explanation

The **first sample case** is discussed in the task statement.

In the **second sample case**, the path followed by Giorgio is the following:



49	48	41	17	13	51	32	47	35
42	40	37	28	21	54	33	22	38
23	19	34	50	16	24	44	11	27
30	25	52	29	14	31	45	39	15
46	26	20	43	10	12	53	36	18

Fractal graphs (`fractal`)

After Edoardo accidentally found Giorgio working on fractals (amazing mathematical shapes with recursive definitions), he decided to do something similar... but in a somewhat more *computer science* fashion.

After days of excruciating research, he finally discovered the *fractal graphs* $\mathcal{G}_N$! The first member of this family of graphs,[1] $\mathcal{G}_0$, is very simple: a single node without any edges. After that, the graphs quickly grow in complexity as N increases.

More precisely, each fractal graph $\mathcal{G}_N$ for $N > 0$ is obtained from its predecessor $\mathcal{G}_{N-1}$ by adding:

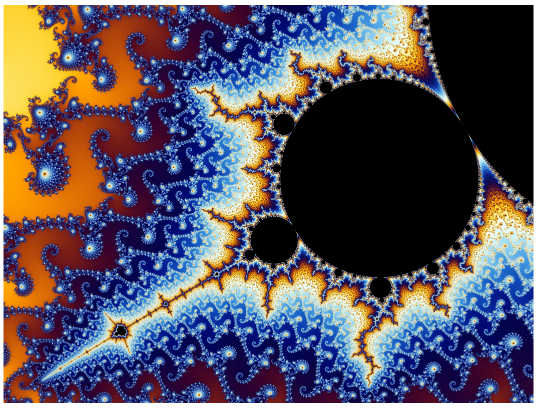
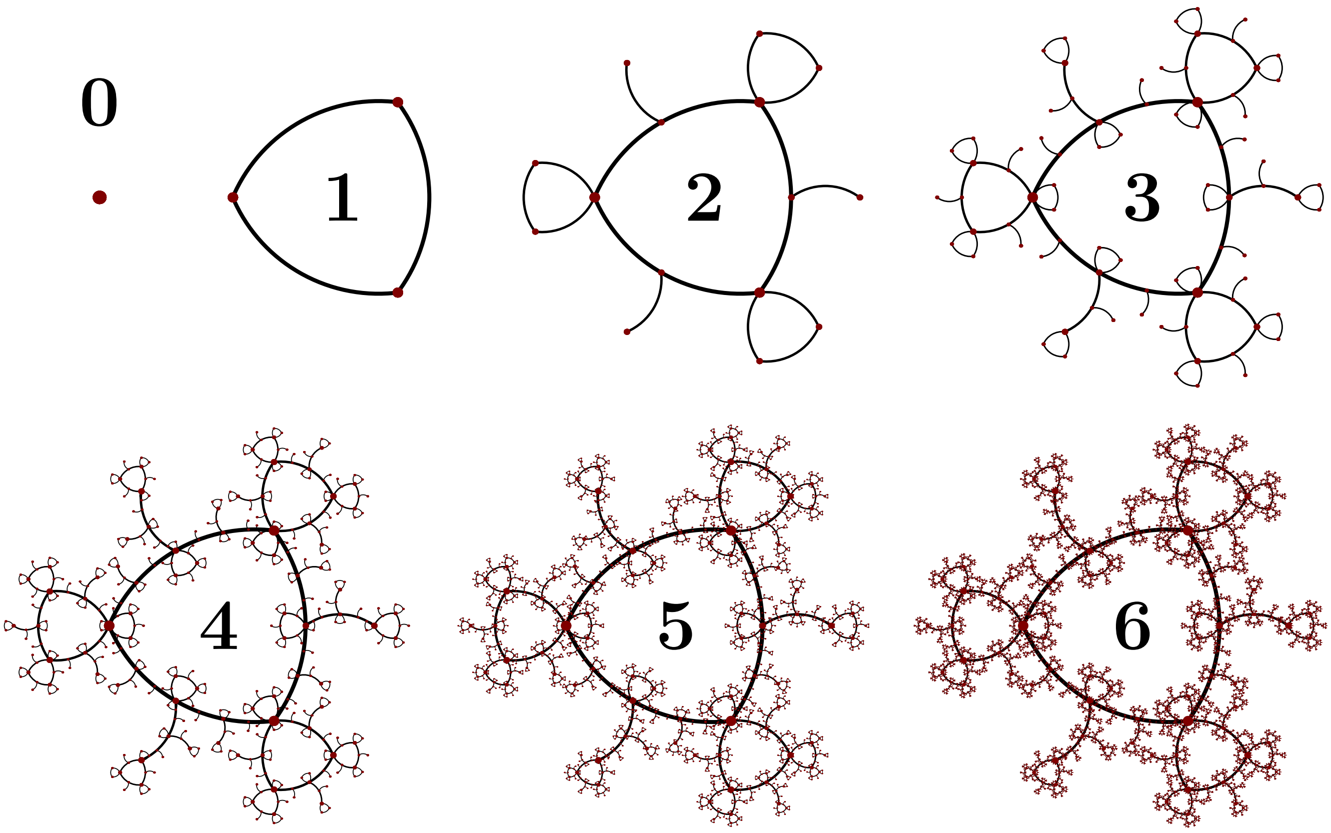


Figure 1: A detail of the Mandelbrot set, the most glorious among all fractals.

- A triangle T for every node v in $\mathcal{G}_{N-1}$, so that one of the nodes of T is v and the other nodes and edges are new;
- A segment S in the middle of every edge e in $\mathcal{G}_{N-1}$, that is, e is split in half into two edges e_1 and e_2 joined by node v, and S starts from v (thus adding a further node and edge).

The first seven fractal graphs $\mathcal{G}_0, \ldots, \mathcal{G}_6$ are the following:



Help Edoardo show his fractal supremacy, by counting the number of nodes and edges in $\mathcal{G}_N$!

[1] A *graph* is a mathematical object consisting of a set V of *nodes* (unlabelled points) and a set E of *edges* (undirected links between couple of points), so that $E \subseteq V \times V$.

☞ Among the attachments of this task you may find a template file `fractal.*` with a sample incomplete implementation.

Input

The first and only line contains the only integer N.

Output

You need to write a single line with two integers: the number of nodes and edges in $\mathcal{G}_N$ **modulo 32 749**.

☞ The *modulo* operation ($a \bmod m$) can be written in C/C++ as `(a % m)` and in Pascal as `(a mod m)`. To avoid the *integer overflow* error, remember to reduce all partial results through the modulus, and not just the final result!
Notice that if $x < 32\,749$*, then* $2x^2$ *fits into a* C/C++ `int` *and* Pascal `longint`.

Constraints

- $0 \le N \le 1\,000\,000\,000$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**0 points**]: Examples.
- **Subtask 2** [**10 points**]: $N \le 3$.
- **Subtask 3** [**20 points**]: $N \le 6$.
- **Subtask 4** [**20 points**]: $N \le 10$.
- **Subtask 5** [**20 points**]: $N \le 1000$.
- **Subtask 6** [**20 points**]: $N \le 50\,000\,000$.
- **Subtask 7** [**10 points**]: No additional limitations.

Examples

`input.txt`	`output.txt`
`0`	`1 0`
`1`	`3 3`
`2`	`15 18`

Black Friday (`giftcard`)

Black Friday is over, and Luca couldn't avoid buying a super discounted giftcard from his favourite online shop!

In the rush to make the deal before the end of the special offer, Luca didn't pay much attention to the small *Terms&Conditions* message: the giftcard can only be spent to buy two particular types objects, in *any positive* amount you want. Objects of type one cost exactly C_1 euros each and objects of type two C_2 euros each.

Furthermore, there is another caveat: online shops always hide restrictions! The giftcard of value N can only be used completely, which means that the total price for buying objects has to be *exactly* N.

Help Luca by suggesting how many objects of the two types *at minimum* he should buy in order to be able to spend his giftcard!

☞ Among the attachments of this task you may find a template file `giftcard.*` with a sample incomplete implementation.

Input

The first and only line of the input contains three integers: N, C_1 and C_2.

Output

You need to write a single line with two strictly positive integers: respectively, the number of objects of type one and type two that Luca should buy, in order to spend his giftcard with the minimum total number of objects.

Constraints

- $2 \leq N < 2^{64}$.
- $1 \leq C_1, C_2 \leq N$.
- Objects have different costs: $C_1 \neq C_2$.
- It is guaranteed that a solution exists.

Scoring

Your program will be tested against several test cases grouped in subtasks. For each test case, your program will get:

- **0 points** if it fails to produce two numbers a, b of objects which allow to use the giftcard.
- **0.5 points** if it produces two numbers a, b of objects which allow to use the giftcard, such that $a + b$ *is not* minimal.
- **1 point** if it produces two numbers a, b of objects which allow to use the giftcard, such that $a + b$ *is* minimal.

The score obtained by your program in each subtask will then be equal to the lowest score obtained in one of its test cases, multiplied by the value of the subtask.

- **Subtask 1** [**0 points**]: Examples.
- **Subtask 2** [**20 points**]: $C_1 = 1$ or $C_2 = 1$.
- **Subtask 3** [**10 points**]: $N = C_1 + C_2$.
- **Subtask 4** [**40 points**]: $N \leq 1000$.
- **Subtask 5** [**20 points**]: $N \leq 10^9$.
- **Subtask 6** [**10 points**]: No additional limitations.

Examples

`stdin/input.txt`	`stdout/output.txt`
`9 2 5`	`2 1`
`12 4 2`	`2 2`

Explanation

In the **first example**, the optimal solution is to buy two objects of the first type (for a total cost of 4 euros) and one object of the second type (total cost of 5 euros). The overall expense of 9 euros matches exactly the value of the giftcard.

In the **second example**, we cannot buy all objects of the same type because the shop requires at least one object of each type. Thus, the optimal solution is to buy two objects of the first type and two objects of the second type. Another (suboptimal) solution would be buying one object of the first type and four objects of the second type.

Department blackout (recovery)

Giorgio is teaching a university course, so he needs to grade a lot of exams each year. Specifically, last year he scheduled T multiple-choice tests, and he had a total of S students taking part in some test.

The testing procedure was as follows. Each test was performed at a workstation and consisted of Q questions. When a student took part in a test, they had to respond to each question with an integer number between 1 and 32: as you can see, Giorgio really wants to make sure that "guessing" the right answer (probability $\frac{1}{32}$) is nearly impossible! It's important to note that the Q questions were always presented in *random order* by the workstation, in order to deter cheating.

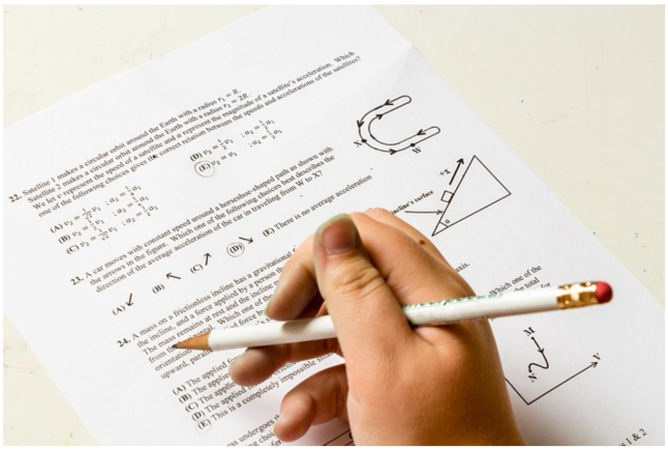


Figure 1: One of the easiest tests.

Sadly, last week a blackout struck the entire department and some data was lost. Specifically, for each student result in the database, Giorgio lost the reference to which of the T tests that result refers to.

The IT team of the university managed to recover the *correct answer* (a number between 1 and 32) for all Q questions of each of the T tests, as well as the *given answer* (also a number between 1 and 32) inputted for all Q question by each of the S students.

In the midst of this tragic situation, Giorgio is wondering: how many possible "student $\leftrightarrow$ test" pairs can be found such that the student's given answers are **compatible** with the test's correct answers? By *compatible* we mean that, rearranged in some order, the answers given by the students perfectly match the correct answers of the test.

☞ Among the attachments of this task you may find a template file `recovery.*` with a sample incomplete implementation.

Input

The first line contains three integers T, S, Q. The following T lines describe the tests and contain Q integers each (the correct answers). The following S lines describe the answers provided by students and contain Q integers each (the given answers).

Output

You need to write a single line with an integer: the number of compatible pairs.

Constraints

- $1 \leq T, S \leq 10\,000$.
- $1 \leq Q \leq 100$.
- The correct and given answers are integer numbers between 1 and 32.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**0 points**]: Examples.
- **Subtask 2** [**20 points**]: $T, S \leq 1000$ and $Q = 1$.
- **Subtask 3** [**20 points**]: $T, S \leq 1000$.
- **Subtask 4** [**30 points**]: $Q = 1$.
- **Subtask 5** [**30 points**]: No additional limitations.

Examples

input.txt	output.txt

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

`3 4 5`
`8 5 2 6 4`
`8 7 1 7 4`
`9 8 8 2 1`
`8 1 7 4 7`
`8 8 9 2 2`
`8 7 4 1 7`
`5 2 6 4 8`
	`3`

Explanation

In the **first sample case** the first student must have had some answers wrong (he responded with 1 and 1 to the two questions) so he can't be matched. The second student, however, might have taken the second test (since he replied 1 and 3 to the two questions, in some order).

In the **second sample case** the first test and the last student match. Also, the second test matches with two students: the first and the third one.

Oral exam (`threshold`)

At Luca's university, the final exam for a course usually consists of a written part and an oral one, taken exactly in this order. As the difficulty of the written part varies among different exam sessions, each time a tedious work has to be done in order to properly set the oral admission threshold.

Grades for written exams are assigned in the range 0—100. A student can take the oral exam if and only if his/her grade is *greater than or equal to* the threshold (which can also range from 0 to 100).

Figure 1: A typical student looking for his/her grade, hoping that it'll be high enough for the threshold.

The professor doesn't want to spend many days to assess lots of students (if the threshold is low). On the other hand, he can't be too strict setting an exaggeratedly high threshold. Luca's professor kindly asked him to ease his work providing a list with indications on how many students would be admitted to the oral exam *for every possible threshold.* Compute this list for Luca!

☞ Among the attachments of this task you may find a template file `threshold.*` with a sample incomplete implementation.

Input

The first line contains the only integer N, the number of the students that took the written exam. The second line contains N integers G_i, respectively the grade obtained by the i-th student.

Output

You need to write a single line containing 101 integers S_k, **for k from 100 to 0**: respectively, the number of students that would be admitted to the oral exam if the threshold were k.

Constraints

- $1 \le N \le 10\,000\,000$.
- $0 \le G_i \le 100$ for each $i = 0 \ldots N-1$.
- Luca's professor has already sorted the sheets of the written exam from the student with the highest grade to the student with the lowest one. More formally: $G_i \ge G_{i+1}$ for each $i = 0 \ldots N-2$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**0 points**]: Examples.
- **Subtask 2** [**15 points**]: All grades are equal, $G_i = k$ with $0 \le k \le 100$ for each $i = 0 \ldots N-1$.
- **Subtask 3** [**15 points**]: Grades are "binary", either $G_i = 0$ or $G_i = 100$ for each $i = 0 \ldots N-1$.
- **Subtask 4** [**40 points**]: $N \le 10\,000$.
- **Subtask 5** [**30 points**]: No additional limitations.

Examples

`stdin/input.txt`	`stdout/output.txt`
`2` `95 60`	`0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1` `1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1` `1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2` `2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2` `2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2` `2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2`
`4` `100 100 95 95`	`2 2 2 2 2 4 4 4 4 4 4 4 4 4 4 4 4` `4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4` `4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4` `4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4` `4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4` `4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4`

Explanation

In the **first sample case** there are only two students. No student passes to the oral exam if the threshold is 100, 99, 98, 97 or 96. Only the first student passes if the threshold is in the range [95-61] (extremes included). Finally, both students pass for every threshold lower than or equal to 60.

In the **second sample case** there are four students. The first two students are admitted to the oral exam even if the threshold is set to 100. Moreover, the same two students pass if the threshold is in the range [99-96] (extremes included). All the four students are admitted when the threshold is lower than or equal to 95.

Word search puzzle (`wordsearch`)

In recent months, William has become passionate about *word search* puzzles. A word search puzzle consists of a grid of letters, in which there are one or more words to be searched in a vertical, horizontal or diagonal way.

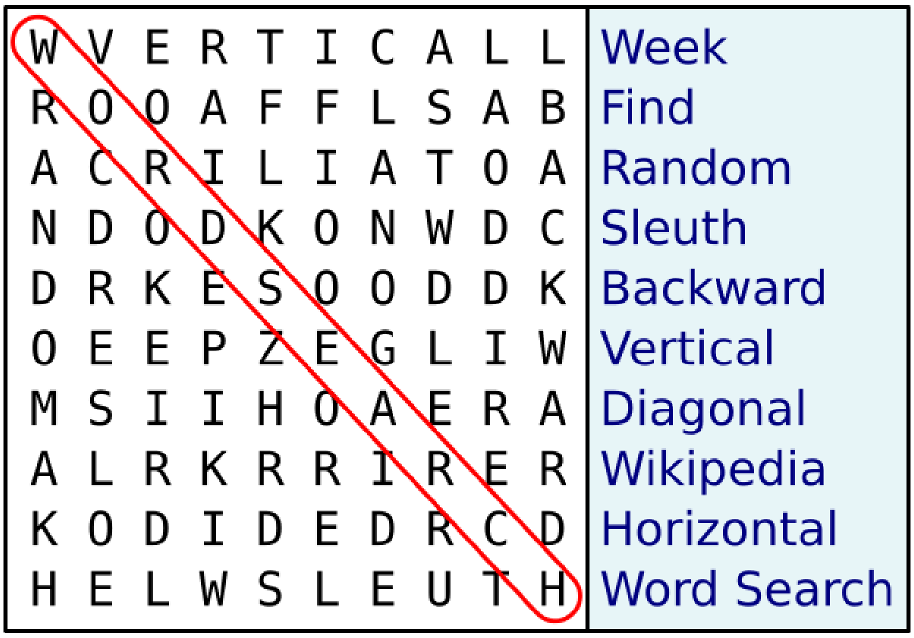


Figure 1: A standard version of the puzzle.

William's version of this puzzle is a bit harder. There is *only one word* to be searched, but you are not limited to vertical/horizontal/diagonal words. You can compose a word in any way you like, starting from some letter and moving in any direction as long as you move to an *adjacent cell*: that means, you can move in 8 possible directions (two vertical, two horizontal and four diagonal). Note that William's rules do not forbid you to step on the same letter more than once: you have the freedom to compose the word even by reusing letters.

Compute how many times a given word appears in the word puzzle. This number can be quite large, so it's enough to print its value **modulo 1 000 000 007**.

☞ The *modulo* operation ($a \bmod m$) can be written in C/C++ as `(a % m)` and in Pascal as `(a mod m)`. To avoid the *integer overflow* error, remember to reduce all partial results through the modulus, and not just the final result!

☞ Among the attachments of this task you may find a template file `wordsearch.*` with a sample incomplete implementation.

Input

The first line contains the word S to be searched. The second line contains two integers H and W. The following H lines contain W letters each, and represent the letter grid forming William's word puzzle.

Output

You need to write a single line with an integer: how many times the given word appears in the letter grid (modulo 1 000 000 007).

Constraints

- $1 \le H, W \le 100$.
- The length of the word S does not exceed 1000.
- Every letter is alphabetical and lowercase (`a` to `z`).

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**0 points**]: Examples.
- **Subtask 2** [**20 points**]: $H = 1$: the input matrix is just one row.
- **Subtask 3** [**20 points**]: $H = W = 2$.
- **Subtask 4** [**30 points**]: $H, W \le 10$.
- **Subtask 5** [**30 points**]: No additional limitations.

Examples

`input.txt`	`output.txt`

`ciao`
`3 3`
`cia`
`cio`
`ciu`
	`5`

`aaaaaaaaaaaa`
`5 8`
`aaaaaaaa`
`aaaaaaaa`
`aaaaaaaa`
`aaaaaaaa`
`aaaaaaaa`
	`144253187`

Explanation

In the **first sample case** there are 5 ways to compose the word “ciao”:

- Start from the first “c”, in the upper left corner, and go: right, right, down.
- Same starting point but go: down-right, up-right, down.
- Start from the second “c” and go: right, up-right, down.
- Same starting point but go: up-right, right, down.
- Start from the third “c” and go: up-right, up-right, down.