

## Bitwise Party (andxor)

The new year has come and just like all the competitive programmers, our hero Stefan received an array with positive integers as a Christmas present as well! Now he wants to explore the array and to find new interesting properties as he always likes doing. For this year, he wants to find out the maximum number of values we can take from the array such that both the bitwise *AND* and the bitwise *XOR* are different from zero<sup>1</sup>. Since this is too easy for him, he also modifies his array and wants you to answer to the same question after each update.



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

### Input

The first line contains two integers:  $N$ , the number of integers in the array, and  $Q$ , the number of queries. The next line contains  $N$  integers,  $v_i$ , the initial values of the array, each one separated by a space. The next  $Q$  lines contain a query each. Each query is composed of two integers:  $pos$ , the position of the element to change, and  $val$ , the new value of the element.

### Output

You need to write  $Q + 1$  lines. The first line should contain the answer before the first query. In the  $i$ -th line (with  $i \geq 1$ ) you should output the answer after the first  $i$  queries.

<sup>1</sup>You can find more information about bitwise operations at [https://en.wikipedia.org/wiki/Bitwise\\_operation#Bitwise\\_operators](https://en.wikipedia.org/wiki/Bitwise_operation#Bitwise_operators). In C, C++ and Python the operators for bitwise-and and bitwise-xor are `&` and `^`, respectively. In Pascal they are `and` and `xor`.

## Constraints

- $1 \leq N \leq 200\,000$ .
- $1 \leq Q \leq 200\,000$ .
- $1 \leq v_i, val < 2^{20}$ .
- $1 \leq pos \leq N$ .

## 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)       $1 \leq v_i, val \leq 2$ .
- **Subtask 3** (20 points)       $1 \leq N, Q, v_i, val < 2^7$ .
- **Subtask 4** (20 points)       $1 \leq N, Q, v_i, val < 2^{10}$ .
- **Subtask 5** (50 points)      No additional limitations.

## Examples

input	output
3 1 11 5 14 3 15	2 3
5 8 3 2 4 1 5 1 1 2 3 3 3 4 2 3 6 4 5 5 2 1 6	3 3 4 5 4 3 4 3 4

## Explanation

In the **first sample case**, before any query all the possible ways to take the elements are:

$v_1$	$v_2$	$v_3$	<b>and</b>	<b>xor</b>	valid
11	5	14			
-	-	-	0000	0000	no
1011	-	-	1011	1011	yes
-	0101	-	0101	0101	yes
-	-	1110	1110	1110	yes
1011	0101	-	0001	1110	yes
1011	-	1110	1010	0101	yes
-	0101	1110	0100	1011	yes
1011	0101	1110	0000	0000	no

There are 6 ways to select elements such that the bitwise *AND* and the bitwise *XOR* are non-zero. At most 2 numbers can be selected doing so.

After the first update, the possible ways to take the elements are:

$v_1$	$v_2$	$v_3$	<b>and</b>	<b>xor</b>	valid
11	5	15			
-	-	-	0000	0000	no
1011	-	-	1011	1011	yes
-	0101	-	0101	0101	yes
-	-	1111	1111	1111	yes
1011	0101	-	0001	1110	yes
1011	-	1111	1011	0100	yes
-	0101	1111	0101	1010	yes
1011	0101	1111	0001	0001	yes

It's possible to take all 3 elements.

## Flappy Bird (flappybird)

After many years of obscurity, the well-known mobile game Flappy Bird has finally become downloadable again and Stefan decided to install it on his mobile phone in a quest to relive his old childhood days.

Unlike the original version of the game, in this new edition the towers are oriented horizontally, rather than vertically. The bird needs to fly through  $N$  pairs of towers, represented as a grid of  $N$  rows and a (potentially very large) number of columns. Each of the  $N$  rows in the grid is described as an interval: two positions  $A[i]$  and  $B[i]$  which indicate that the bird can fly through those columns (both extremes are included).

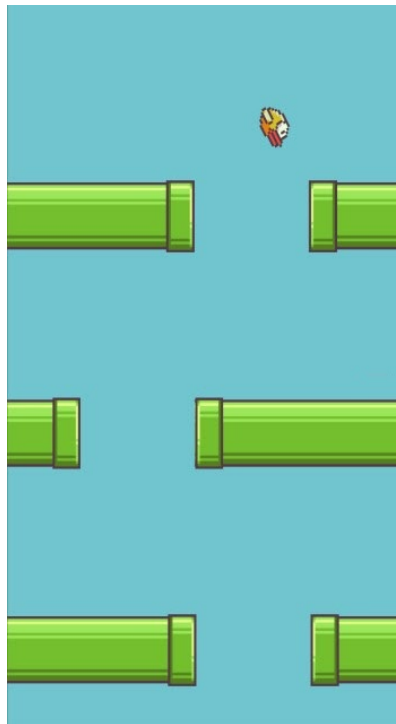



Figure 1: Our vertical version of Flappy Bird.

There is a problem though: given that the rows have no space between them (unlike the image above!), some levels of the game are impossible to solve! For simplicity, let's assume that the bird starts before the first row, can fly horizontally instantaneously, but can only move towards the bottom if there is at least one viable column that is the same between a row and its immediate next one.

Given the description of a level, can the bird actually make it to the bottom and win?

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

### Input

The first line contains the only integer  $N$ . The following  $N$  rows contain two integers  $A[i]$ ,  $B[i]$  each.

### Output





You need to write a single line with the answer: "YES" if the level is winnable, "NO" otherwise.

## Constraints

- $1 \leq N \leq 100\,000$ .
- $1 \leq A_i \leq B_i \leq 10^9$  for each  $i = 0 \dots N - 1$ .

## 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** (25 points)       $N = 2$ .  

- **Subtask 3** (30 points)       $N \leq 1000$  and there are at most 1000 columns (i.e., all  $B[i] \leq 1000$ ).  

- **Subtask 4** (45 points)      No additional limitations.  


## Examples

input	output
5 3 6 4 7 2 4 4 5 1 5	YES
4 4 7 5 6 1 3 3 6	NO

## Explanation

In the **first sample case**, in one of the many winning strategies the bird can fly towards the bottom by always staying in column 4.

In the **second sample case** the bird can fly through the first and the second row (e.g., staying in column 5) but then cannot possibly move to the third row.

## Keyboard Shift (keyboardshift)

Alessandro is training his speed typing skills with his brand new mechanical keyboard.



Figure 1: Alessandro’s mechanical keyboard (for real).

The layout is the standard QWERTY, the three main rows are:

```
qwertyuiop
asdfghjkl
zxcvbnm
```

Unfortunately, he should train a bit more since he continuously makes the same mistake! Every time he wants to press a key, he presses the one immediately to the left of it. For example, if he wants to type the letter ‘g’, he presses the key ‘f’. Every time he wants to type ‘q’, ‘a’ or ‘z’ the character is simply lost.

He already wrote quite a lot of text, all with the same mistake. Can you help him fix the original text by shifting each character to the right?

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

### Input

The first line contains the only integer  $N$ , the number of characters he typed. The second line contains a string  $S$  of  $N$ .

### Output





Write in a single line the fixed string of  $N$  characters.

### Constraints

- $1 \leq N \leq 1\,000\,000$ .
- The characters are taken from “abcdefghijklmnopqrstuvwxyz” (alphabet without “plm”).

## 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** (30 points)      The text is only formed by letters ‘a’ and ‘b’.  

- **Subtask 3** (30 points)       $N \leq 1000$ .  

- **Subtask 4** (40 points)      No additional limitations.  


## Examples

input	output
11 qiqayxgfiis	wowsuchgood
6 qwerty	wertyu
6 ababbb	snsnnn

## Explanation

In the **first sample case** the mapping of the letters is the following:

q → w  
i → o  
q → w  
a → s  
y → u  
x → c  
g → h  
f → g  
i → o  
i → o  
s → d

## Training for the Marathon (marathon)

William is training for the marathon! Some running skills are always useful... who knows who may chase you in the future.



Figure 1: William after realizing that he has taken the wrong direction.

William trains in the park near his newly found home, which is a connected graph composed by  $N$  intersections (numbered from 1 to  $N$ ) and  $N - 1$  roads connecting two intersections each. Due to bad weather, some roads may be temporarily closed. William, however, has consulted the weather forecast and knows exactly which roads will be closed at any given moment.

Before training, William has prepared a timeline consisting of  $Q$  events of two types:

- type 1: if the road from intersection  $x$  to intersection  $y$  is closed it will be reopened, if the road is open it will be closed;
- type 2: William will train by running the path from the intersection  $x$  to the intersection  $y$ .

Notice that due to bad weather, the chosen path in a type 2 event may involve some temporarily closed roads: in such case, William will put on a trekking outfit (outfit 0) and walk directly on the grass. Otherwise, he will put on the running outfit (outfit 1) and run on the path.

Giorgio, a freelance investigator, managed to get a photo of William's timeline, and wants to use it to catch him. However, William is very smart, and he wrote down the timeline in an encrypted fashion! For type 2 events, instead of writing the true intersections  $x$  and  $y$  describing it, he wrote the encrypted  $x_e, y_e$  computed as:

$$x = [(x_e + \text{sum}) \bmod N] + 1$$


$$y = [(y_e + \text{sum}) \bmod N] + 1$$

where **sum** is the sum of all the outfit numbers he will be wearing for the type 2 events preceding the current one.

Help Giorgio catch William, by computing the outfit numbers corresponding to each type 2 event!



 The *modulo* operation ( $a \bmod m$ ) can be written in C/C++/Python as `(a % m)` and in Pascal as `(a mod m)`.

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

## Input

The first line contains the two integer  $N$  and  $Q$ : the number of intersections of the park and the number of events.

The next  $N - 1$  lines contain two integers  $x$  and  $y$  each, representing a road between intersection  $x$  and intersection  $y$ .

The next  $Q$  lines contain three integers representing an event:

- 1  $x$   $y$  for a type 1 event;
- 2  $x_e$   $y_e$  for a type 2 event.

## Output





For every type 2 event in the input, you should write a line with an integer corresponding to the outfit number he will use.

## Constraints

- $1 \leq N, Q \leq 250\,000$ .
- $1 \leq x, y \leq N$  and  $1 \leq x_e, y_e \leq N$  for each event.

## 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.

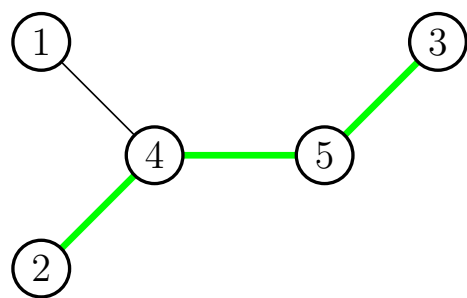
- **Subtask 1** (0 points)      Examples.  

- **Subtask 2** (45 points)       $N, Q \leq 1000$ .  

- **Subtask 3** (20 points)       $N \leq 1000$ .  

- **Subtask 4** (35 points)      No additional limitations.  


Examples

input	output
5 5	1
1 4	0
4 2	1
5 4	
3 5	
2 1 2	
1 4 5	
2 4 1	
1 4 5	
2 3 2	

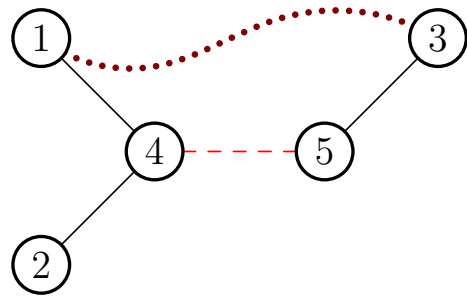
Explanation

In the first event, William runs from 2 to 3 through intersections 4 and 5 with the running outfit 1.



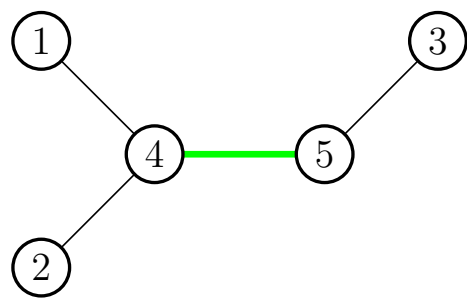
In the second event, the road from 4 to 5 is closed.

In the third event, there is no path between 1 and 3, therefore William walks on grass with the trekking outfit 0.



In the fourth event, the road from 4 to 5 is reopened.

In the fifth event, William runs from 5 to 4 with outfit 1.



## Police Investigation 6 (police6)


Fearsome William is still trying to hide from the police, who have stepped up the game. In a very long ( $L$  meters) street, the police officers have set up  $N$  checkpoints, to first see and then stop the criminal.

With the help of binoculars, the officers at each checkpoint are able to see up to  $M$  meters away in both directions. Formally, this means that a checkpoint located at  $D[i]$  meters from the beginning of the street can see people from  $D[i] - M$  meters (included) to  $D[i] + M$  meters (included), measured from the beginning of the street.



Figure 1: Officers standing ready at a checkpoint.

William is no longer by car and has no other choice to walk a little bit and spend the night hiding somewhere along the street with all the checkpoints, at any point from 0 to  $L$  (both included). He wants to minimize the number of checkpoints from which he can be seen: in one of the possibly many optimal positions, how many checkpoints will he be visible from?

 Among the attachments of this task you may find a template file `police6.*` with a sample incomplete implementation.

### Input

The first line contains three integers  $N$ ,  $M$ , and  $L$ . The second line contains  $N$  integers  $D_i$ .

### Output

You need to write a single line with an integer: the minimum number of checkpoints from which William will be visible.

### Constraints

- $1 \leq N \leq 100\,000$ .
- $1 \leq M \leq 10^{18}$ .
- $1 \leq L \leq 10^{18}$ .
- Checkpoints are all at different positions and are listed in order:  $D[i] < D[j]$  for all  $1 \leq i < j \leq N-1$
- $0 \leq D_i \leq L$  for each  $i = 0 \dots N-1$ .

# 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 (11 points)

$N = 1$ .
- Subtask 3 (16 points)

Checkpoints are evenly spaced: the distance between two consecutive ones, as well as the distance between 0 and the first one, and the last one and  $L$ , is constant.
- Subtask 4 (22 points)

$L \leq 1000$ .
- Subtask 5 (13 points)

$N \leq 1000$ .
- Subtask 6 (38 points)

No additional limitations.

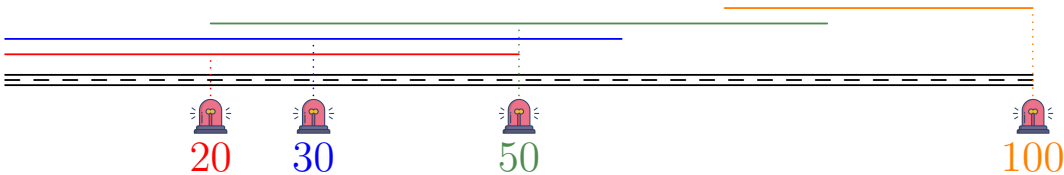
# Examples

input	output
4 30 100 20 30 50 100	1
2 100 100 0 100	2

# Explanation

In the **first sample case**:

- positions from 0 to 19 (both included) are visible from 2 checkpoints;
- positions from 20 to 50 (both included) are visible from 3 checkpoints;
- positions from 51 to 60 (both included) are visible from 2 checkpoints;
- positions from 61 to 69 (both included) are visible from 1 checkpoint;
- positions from 70 to 80 (both included) are visible from 2 checkpoints;
- positions from 81 to 100 (both included) are visible from 1 checkpoint.



An optimal strategy for William is therefore to hide somewhere between 61 and 69 or between 81 and 100 and to be visible from 1 checkpoint.

In the **second sample case**, William has no choice but to be visible from both checkpoints.

## Sorting Proximity (proximity)

As a New Year's resolution, our heroes want everything around them to be clean and perfectly arranged. Thus, they now want to apply their new ideals on their daily work, namely handling the various data structures they are using.



Figure 1: Some of the New Year's resolutions of our heroes.

This time around, they got an array  $V_i$  of length  $N$  and they define the sorting proximity of the array as the number of swaps which have to be done in order to sort the array, if we use the bubble sort algorithm.

Namely, as long as the array is not sorted yet, the bubble sort algorithm iterates through the array and every time two adjacent values are wrongly placed relative to one another, they are swapped. The sorting proximity is the number of times this happens during the algorithm. For example, if we have the array  $[4, 2, 3, 5, 1]$ , the sorting proximity of the array is 6, corresponding to the following 6 swaps:

- $[2, 4, 3, 5, 1]$
- $[2, 3, 4, 5, 1]$
- $[2, 3, 4, 1, 5]$
- $[2, 3, 1, 4, 5]$
- $[2, 1, 3, 4, 5]$
- $[1, 2, 3, 4, 5]$

Now, our heroes want to improve the sorting proximity of the array and they asked for your help. Your job is to swap two integers from the array (they don't have to be adjacent) so that the sorting proximity of the resulting array is minimal. Remember: even though the swap may not improve the answer, you must do it!

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

## Input

The first line contains the only integer  $N$ . The second line contains  $N$  integers  $V_i$ .

## Output





You need to write a single line with an integer: the minimal sorting proximity after a single swap.

## Constraints

- $1 \leq N \leq 100\,000$ .
- $1 \leq V_i \leq 10^9$  for each  $i = 0 \dots N - 1$ .

## 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** (35 points)       $N \leq 1000$  and the values from the array are pairwise distinct.  

- **Subtask 3** (40 points)      All the values from the array are pairwise distinct.  

- **Subtask 4** (25 points)      No additional limitations.  


## Examples

input	output
5 5 2 4 3 1	1
5 3 1 7 9 5	2

## Explanation

In the **first sample case**, we can swap 5 and 1 and we will get the array  $[1, 2, 4, 3, 5]$  which has the sorting proximity equal to 1.

In the **second sample case**, we can swap 5 and 7 and we will get the array  $[3, 1, 5, 9, 7]$  which has the sorting proximity equal to 2.

## The Sieve of Eratosthenes (sieve)

Giorgio is playing with the Sieve of Eratosthenes, one of the most ancient algorithms of all times. This algorithm starts with a row containing all natural numbers from 2 to  $M$  (inclusive), then repeatedly selects the smallest non-selected number remaining in the row, erasing every multiple of it. At the end of the process, only prime numbers will have survived in the row!



Figure 1: Reconstruction of a sieve that Eratosthenes may have used.

Giorgio is trying to apply a similar idea, but with a few differences.

Firstly, some numbers are missing from the starting row, which contains only  $N$  numbers (each of them between 2 and  $M$ ), in increasing order  $V_0, \dots, V_{N-1}$ . Secondly, every time Giorgio selects a number, he erases not only the *multiples* of it but also its *divisors* and the *number itself*.

As in the original algorithm, Giorgio keeps selecting numbers from the row (and erasing accordingly), but he doesn't have to select numbers in increasing order. What is the minimum amount of numbers that must be selected to eliminate all the numbers?

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

### Input

The first line contains the two integers  $N$  and  $M$ . The second line contains  $N$  integers  $V_i$ .

### Output






You need to write a single line with an integer: the minimum amount of numbers that must be selected to eliminate each number.

## Constraints

- $1 \leq N \leq 250$ .
- $2 \leq M \leq 10^6$ .
- $2 \leq V_0 \leq V_1 \leq \dots \leq V_{N-1} \leq M$ .

## 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** (35 points)       $N \leq 9$ .  

- **Subtask 3** (15 points)       $M \leq 20$ .  

- **Subtask 4** (20 points)       $N \leq 50$ .  

- **Subtask 5** (30 points)      No additional limitations.  


## Examples

input	output
6 10 3 4 6 7 8 9	3
6 20 2 5 7 10 14 20	2

## Explanation

In the **first sample case**, we can erase everything by choosing just three numbers. One such strategy is to choose number 7 (erasing just 7), then number 3 (erasing 3, 6, and 9), and finally 8 (erasing 4 and 8). On the other hand, a suboptimal strategy could be to choose numbers: 9, then 8, then 7 and finally 6.

In the **second sample case**, one of the best strategies is to choose numbers 20 and 14.



## Wine Tasting Tour (wine)

Giorgio loves wine and decided to treat himself and go on a wine tasting tour!




Figure 1: Giorgio while tasting Chianti, a typical Italian wine.

The winery offers several tours of its  $N$  vineyards, arranged in a row. Giorgio can start the tour from any vineyard  $S$ . Then, once the first tasting is over, he will move on to the vineyard  $S + 1$ , then to the vineyard  $S + 2$  and so on until reaching the vineyard  $E$  that ends the visit. Note that Giorgio is also free to choose to start and end his tour on the same vineyard.

The tasting in the  $i$ -th vineyard costs Giorgio  $V_i$  euro; therefore the overall cost of the tour is simply the sum of the costs of the visited vineyards.

Giorgio is struggling to choose among the  $\frac{N(N+1)}{2}$  possible different tours, and thus decided to defer the decision to a simple algorithm. He is going to list all the tours  $(S, E)$  and sort them in increasing order of cost. In case of ties, the tours with the smallest starting vineyard come first. Then, he is going to choose the  $K$ -th one (counting from 1) from this sorted list.

Help Giorgio figure out the starting and the ending vineyard of his tour!

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

### Input

The first line contains the integers  $N$  and  $K$ . The second line contains  $N$  integers  $V_i$ .

### Output





You need to write a single line the integers  $S$  and  $E$ : the starting and the ending vineyards of Giorgio's tour.

## Constraints

- $1 \leq N \leq 200\,000$ .
- $1 \leq K \leq \frac{N(N+1)}{2}$ .
- $1 \leq V_i \leq 10^9$  for each  $i = 0 \dots N - 1$ .

## 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** (50 points)       $V_i = 1$  for each  $i = 0 \dots N - 1$ .  

- **Subtask 3** (20 points)       $N \leq 1000$ .  

- **Subtask 4** (30 points)      No additional limitations.  


## Examples

input	output
4 4 1 2 3 1	0 1
6 18 1 2 1 2 1 2	2 5

## Explanation

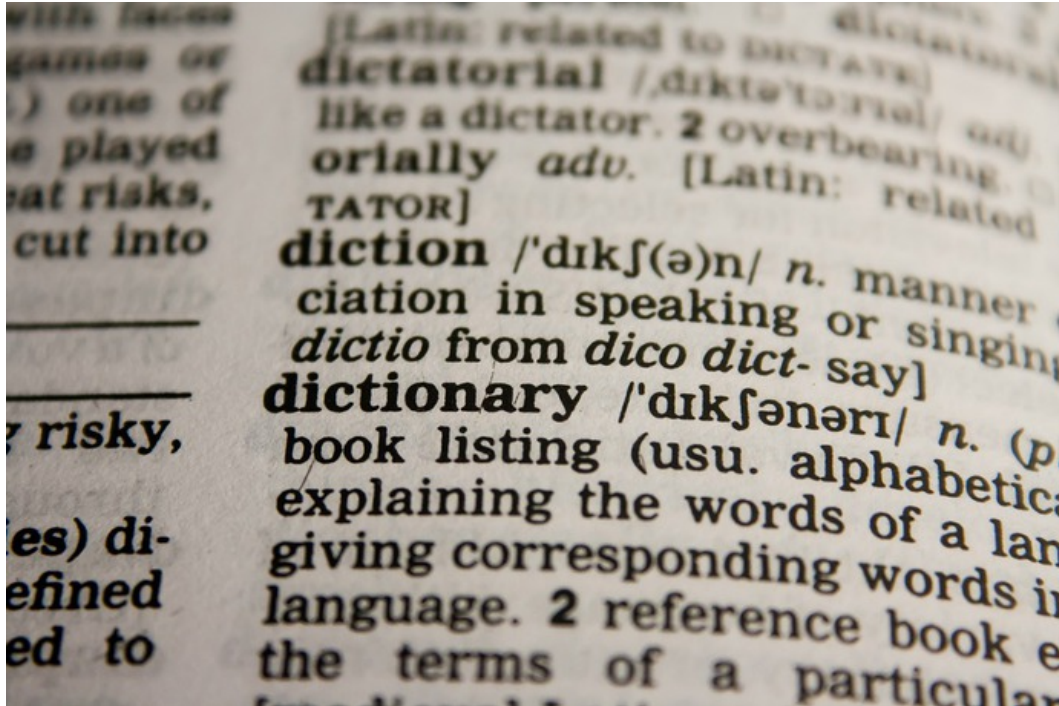
In the **first sample case** there are 10 possible tours. The sorted list is therefore:

1. from 0 to 0: the cost is 1
2. from 3 to 3: the cost is 1
3. from 1 to 1: the cost is 2
4. from 0 to 1: the cost is  $3 = 1 + 2$
5. from 2 to 2: the cost is 3
6. from 2 to 3: the cost is  $4 = 3 + 1$
7. from 1 to 2: the cost is  $5 = 2 + 3$
8. from 0 to 2: the cost is  $6 = 1 + 2 + 3$
9. from 1 to 3: the cost is  $6 = 2 + 3 + 1$
10. from 0 to 3: the cost is  $7 = 1 + 2 + 3 + 1$

The fourth ( $K = 4$ ) tour in this sorted list starts from vineyard 0 and ends in vineyard 1.

## Words, Just Words (words2)

William loves words and likes to collect them in “special” dictionaries. Today, he’s building a dictionary that contains *all the words*. Or actually: all the words that can be formed with the 26 English letters.




The size of this dictionary is, naturally, infinite. This means that it wouldn’t be practical to generate the dictionary to, say, a text file on disk, since it would require an infinite amount of space. However, William would still like to be able to identify the location in the dictionary of a given word.

The ordering rule in this special dictionary is a bit unusual: we consider shorter words to always come before longer ones. For example, the word “xy” comes before “abc”, simply because it is shorter. If two words have the same length, they are ordered in the traditional (lexicographic) way, so “xyz” > “abc”.

This means that the first 26 words in the dictionary are formed by only one character: “a”, “b”, ..., “z”. The next words are formed by only two characters, and so on.

Help William by writing a program that, given a word, calculates its zero-based index in the dictionary. That is: ‘a’ = 0, ‘b’ = 1, and so on.

 Among the attachments of this task you may find a template file `words2.*` with a sample incomplete implementation.

### Input

The first line contains the word  $W$ .

### Output

You need to write a single line with an integer: the zero-based index of  $W$  in the infinite dictionary. Since this answer can be extremely large, we only need to print it modulo 1 000 000 007.

✎ The *modulo* operation ( $a \bmod m$ ) can be written in C/C++/Python 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  $x < 10^9 + 7$ , then  $2x$  fits into a C/C++ `int` and Pascal `longint`.

## Constraints

- The length of the word  $W$  is between 1 and 100 000 characters.
- The word  $W$  is formed by lowercase English characters only ('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)       $N \leq 3$ .
- **Subtask 3** (20 points)      The word only uses the letter  $a$  (e.g. "aaaa")
- **Subtask 4** (40 points)       $N \leq 1000$ .
- **Subtask 5** (20 points)      No additional limitations.

## Examples

input	output
a	0
abba	18980
verylongstring	315607945