Codeforces Round #786 (Div. 3)

A. Number Transformation

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given two integers x and y. You want to choose two **strictly positive** (greater than zero) integers a and b, and then apply the following operation to x **exactly** a times: replace x with $b \cdot x$.

You want to find two positive integers a and b such that x becomes equal to y after this process. If there are multiple possible pairs, you can choose **any of them**. If there is no such pair, report it.

For example:

- if $x = 3$ and $y = 75$, you may choose $a = 2$ and $b = 5$, so that x becomes equal to $3 \cdot 5 \cdot 5 = 75$;
- if $x = 100$ and $y = 100$, you may choose $a = 3$ and $b = 1$, so that x becomes equal to $100 \cdot 1 \cdot 1 \cdot 1 = 100$;
- if $x = 42$ and $y = 13$, there is no answer since you cannot decrease x with the given operations.

Input

The first line contains one integer t ($1 \le t \le 10^4$) — the number of test cases.

Each test case consists of one line containing two integers x and y ($1 \le x, y \le 100$).

Output

If it is possible to choose a pair of positive integers a and b so that x becomes y after the aforementioned process, print these two integers. **The integers you print should be not less than 1 and not greater than 10^9** (it can be shown that if the answer exists, there is a pair of integers a and b meeting these constraints). If there are multiple such pairs, print any of them.

If it is impossible to choose a pair of integers a and b so that x becomes y, print the integer 0 twice.

Example

input
3 3 75 100 100 42 13
output
2 5 3 1 0 0

B. Dictionary

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

The Berland language consists of words having **exactly two letters**. Moreover, **the first letter of a word is different from the second letter**. Any combination of two different Berland letters (which, by the way, are the same as the lowercase letters of Latin alphabet) is a correct word in Berland language.

The Berland dictionary contains all words of this language. The words are listed in a way they are usually ordered in dictionaries. Formally, word a comes earlier than word b in the dictionary if one of the following conditions hold:

- the first letter of a is less than the first letter of b;
- the first letters of a and b are the same, and the second letter of a is less than the second letter of b.

So, the dictionary looks like that:

- Word 1: ab
- Word 2: ac
- ...
- Word 25: az
- Word 26: ba
- Word 27: bc

- ...
- Word 649: zx
- Word 650: zy

You are given a word s from the Berland language. Your task is to find its index in the dictionary.

Input
The first line contains one integer t ($1 \le t \le 650$) — the number of test cases.

Each test case consists of one line containing s — a string consisting of **exactly two different lowercase Latin letters** (i. e. a correct word of the Berland language).

Output
For each test case, print one integer — the index of the word s in the dictionary.

Example

input
7 ab ac az ba bc zx zy
output
1 2 25 26 27 649 650

C. Infinite Replacement

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

You are given a string s, consisting only of Latin letters 'a', and a string t, consisting of lowercase Latin letters.

In one move, you can replace any letter 'a' in the string s with a string t. Note that after the replacement string s might contain letters other than 'a'.

You can perform an arbitrary number of moves (including zero). How many different strings can you obtain? Print the number, or report that it is infinitely large.

Two strings are considered different if they have different length, or they differ at some index.

Input
The first line contains a single integer q ($1 \le q \le 10^4$) — the number of testcases.

The first line of each testcase contains a non-empty string s, consisting only of Latin letters 'a'. The length of s doesn't exceed 50.

The second line contains a non-empty string t, consisting of lowercase Latin letters. The length of t doesn't exceed 50.

Output
For each testcase, print the number of different strings s that can be obtained after an arbitrary amount of moves (including zero). If the number is infinitely large, print -1. Otherwise, print the number.

Example

input
3 aaaa a aa abc a b
output
1 -1 2

Note
In the first example, you can replace any letter 'a' with the string "a", but that won't change the string. So no matter how many

moves you make, you can't obtain a string other than the initial one.

In the second example, you can replace the second letter 'a' with "abc". String s becomes equal to "aabc". Then the second letter 'a' again. String s becomes equal to "aabcbc". And so on, generating infinitely many different strings.

In the third example, you can either leave string s as is, performing zero moves, or replace the only 'a' with "b". String s becomes equal to "b", so you can't perform more moves on it.

D. A-B-C Sort

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

You are given three arrays a, b and c. Initially, array a consists of n elements, arrays b and c are empty.

You are performing the following algorithm that consists of two steps:

- Step 1: while a is not empty, you take *the last element* from a and move it *in the middle* of array b. If b currently has odd length, you can choose: place the element from a to the left or to the right of the middle element of b. As a result, a becomes empty and b consists of n elements.
- Step 2: while b is not empty, you take *the middle element* from b and move it *to the end* of array c. If b currently has even length, you can choose which of two middle elements to take. As a result, b becomes empty and c now consists of n elements.

Refer to the Note section for examples.
Can you make array c sorted in non-decreasing order?

Input

The first line contains a single integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. Next t cases follow.

The first line of each test case contains the single integer n ($1 \le n \le 2 \cdot 10^5$) — the length of array a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^6$) — the array a itself.

It's guaranteed that the sum of n doesn't exceed $2 \cdot 10^5$.

Output

For each test, print YES (case-insensitive), if you can make array c sorted in non-decreasing order. Otherwise, print NO (case-insensitive).

Example

input
3 4 3 1 5 3 3 3 2 1 1 7331
output
YES NO YES

Note

In the first test case, we can do the following for $a = [3, 1, 5, 3]$:

Step 1:

a	$[3, 1, 5, 3]$	$\Rightarrow$	$[3, 1, 5]$	$\Rightarrow$	$[3, 1]$	$\Rightarrow$	$[3]$	$\Rightarrow$ $[]$
b	$[]$		$[\underline{3}]$		$[3, \underline{5}]$		$[3, \underline{1}, 5]$	$[3, \underline{3}, 1, 5]$

Step 2:

b	$[3, 3, \underline{1}, 5]$	$\Rightarrow$	$[3, \underline{3}, 5]$	$\Rightarrow$	$[\underline{3}, 5]$	$\Rightarrow$	$[\underline{5}]$	$\Rightarrow$ $[]$
c	$[]$		$[1]$		$[1, 3]$		$[1, 3, 3]$	$[1, 3, 3, 5]$

As a result, array $c = [1, 3, 3, 5]$ and it's sorted.

E. Breaking the Wall

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

Monocarp plays "Rage of Empires II: Definitive Edition" — a strategic computer game. Right now he's planning to attack his opponent in the game, but Monocarp's forces cannot enter the opponent's territory since the opponent has built a wall.

The wall consists of n sections, aligned in a row. The i-th section initially has durability a_i. If durability of some section becomes 0 or less, this section is considered broken.

To attack the opponent, Monocarp needs to break at least two sections of the wall (any two sections: possibly adjacent, possibly not). To do this, he plans to use an onager — a special siege weapon. The onager can be used to shoot any section of the wall; the shot deals 2 damage to the target section and 1 damage to adjacent sections. In other words, if the onager shoots at the section x, then the durability of the section x decreases by 2, and the durability of the sections $x - 1$ and $x + 1$ (if they exist) decreases by 1 each.

Monocarp can shoot at any sections any number of times, **he can even shoot at broken sections**.

Monocarp wants to calculate the minimum number of onager shots needed to break at least two sections. Help him!

Input

The first line contains one integer n ($2 \le n \le 2 \cdot 10^5$) — the number of sections.

The second line contains the sequence of integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^6$), where a_i is the initial durability of the i-th section.

Output

Print one integer — the minimum number of onager shots needed to break at least two sections of the wall.

Examples

input
5 20 10 30 10 20
output
10

input
3 1 8 1
output
1

input
6 7 6 6 8 5 8
output
4

input
6 14 3 8 10 15 4
output
4

input
4 1 100 100 1
output
2

input
3 40 10 10
output
7

Note

In the first example, it is possible to break the 2-nd and the 4-th section in 10 shots, for example, by shooting the third section 10 times. After that, the durabilities become $[20, 0, 10, 0, 20]$. Another way of doing it is firing 5 shots at the 2-nd section, and another

5 shots at the 4-th section. After that, the durabilities become $[15, 0, 20, 0, 15]$.

In the second example, it is enough to shoot the 2-nd section once. Then the 1-st and the 3-rd section will be broken.

In the third example, it is enough to shoot the 2-nd section twice (then the durabilities become $[5, 2, 4, 8, 5, 8]$), and then shoot the 3-rd section twice (then the durabilities become $[5, 0, 0, 6, 5, 8]$). So, four shots are enough to break the 2-nd and the 3-rd section.

F. Desktop Rearrangement

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

Your friend Ivan asked you to help him rearrange his desktop. The desktop can be represented as a rectangle matrix of size $n \times m$ consisting of characters '.' (empty cell of the desktop) and '*' (an icon).

The desktop is called **good** if all its icons are occupying some prefix of full columns and, possibly, the prefix of the next column (and there are no icons outside this figure). In other words, some amount of first columns will be filled with icons and, possibly, some amount of first cells of the next (after the last full column) column will be also filled with icons (and all the icons on the desktop belong to this figure). This is pretty much the same as the real life icons arrangement.

In one move, you can take one icon and move it to any empty cell in the desktop.

Ivan loves to add some icons to his desktop and remove them from it, so he is asking you to answer q queries: what is the **minimum** number of moves required to make the desktop **good** after adding/removing one icon?

Note that **queries are permanent** and change the state of the desktop.

Input

The first line of the input contains three integers n, m and q ($1 \le n, m \le 1000; 1 \le q \le 2 \cdot 10^5$) — the number of rows in the desktop, the number of columns in the desktop and the number of queries, respectively.

The next n lines contain the description of the desktop. The i-th of them contains m characters '.' and '*' — the description of the i-th row of the desktop.

The next q lines describe queries. The i-th of them contains two integers x_i and y_i ($1 \le x_i \le n; 1 \le y_i \le m$) — the position of the cell which changes its state (if this cell contained the icon before, then this icon is removed, otherwise an icon appears in this cell).

Output

Print q integers. The i-th of them should be the **minimum** number of moves required to make the desktop **good** after applying the first i queries.

Examples

input
4 4 8 ..** .*.. *... ...* 1 3 2 3 3 1 2 3 3 4 4 3 2 3 2 2
output
3 4 4 3 4 5 5 5

input
2 5 5 *...* ***** 1 3 2 2 1 3 1 5 2 3
output
2

```
3
3
3
2
```

G. Remove Directed Edges

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

You are given a directed acyclic graph, consisting of n vertices and m edges. The vertices are numbered from 1 to n. There are no multiple edges and self-loops.

Let in_v be the number of incoming edges (indegree) and out_v be the number of outgoing edges (outdegree) of vertex v.

You are asked to remove some edges from the graph. Let the new degrees be in'_v and out'_v.

You are only allowed to remove the edges if the following conditions hold for every vertex v:

- $in'_v < in_v$ or $in'_v = in_v = 0$;
- $out'_v < out_v$ or $out'_v = out_v = 0$.

Let's call a set of vertices S *cute* if for each pair of vertices v and u ($v \neq u$) such that $v \in S$ and $u \in S$, there exists a path either from v to u or from u to v over the non-removed edges.

What is the maximum possible size of a *cute* set S after you remove some edges from the graph and both indegrees and outdegrees of all vertices either decrease or remain equal to 0?

Input

The first line contains two integers n and m ($1 \le n \le 2 \cdot 10^5$; $0 \le m \le 2 \cdot 10^5$) — the number of vertices and the number of edges of the graph.

Each of the next m lines contains two integers v and u ($1 \le v, u \le n$; $v \neq u$) — the description of an edge.

The given edges form a valid directed acyclic graph. There are no multiple edges.

Output

Print a single integer — the maximum possible size of a *cute* set S after you remove some edges from the graph and both indegrees and outdegrees of all vertices either decrease or remain equal to 0.

Examples

input
3 3 1 2 2 3 1 3
output
2

input
5 0
output
1

input
7 8 7 1 1 3 6 2 2 3 7 2 2 4 7 3 6 3
output
3

Note

In the first example, you can remove edges $(1, 2)$ and $(2, 3)$. $in = [0, 1, 2]$, $out = [2, 1, 0]$. $in' = [0, 0, 1]$, $out' = [1, 0, 0]$. You can see that for all v the conditions hold. The maximum *cute* set S is formed by vertices 1 and 3. They are still connected directly by an edge, so there is a path between them.

In the second example, there are no edges. Since all in_v and out_v are equal to 0, leaving a graph with zero edges is allowed. There

are 5 *cute* sets, each contains a single vertex. Thus, the maximum size is 1.

In the third example, you can remove edges $(7, 1)$, $(2, 4)$, $(1, 3)$ and $(6, 2)$. The maximum *cute* set will be $S = \{7, 3, 2\}$. You can remove edge $(7, 3)$ as well, and the answer won't change.

Here is the picture of the graph from the third example: