acm International Collegiate Programming Contest

IBM event sponsor

2016 ACM ICPC
Southeast USA Regional
Programming Contest
Division 2

Hosted by:

College of Charleston

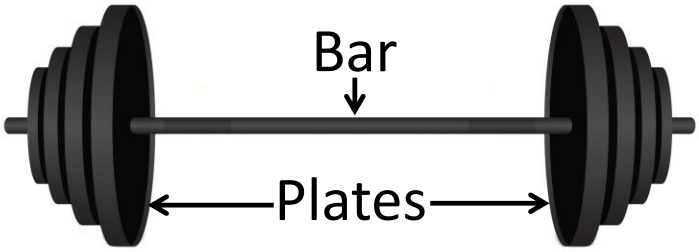
Florida Institute of Technology

Kennesaw State University

University of West Florida

Barbells

Your local gym has ***b*** bars and ***p*** plates for barbells. In order to prepare a weight for lifting, you must choose a single bar, which has two sides. You then load each side with a (possibly empty) set of plates. For safety reasons, the plates on each side must balance; they must sum to the same weight. The combination of plates on either side might be different, but the total weight on either side must be the same. What weights are available for lifting?



Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains two integers, ***b*** and ***p*** (**1**≤***b***,***p***≤**14**), representing the number of bars and plates. Then, there are ***b*** lines each containing a single integer ***x*** (**1**≤***x***≤$\mathbf{10^8}$) which are the weights of the bars. After that, there are ***p*** lines each containing a single integer ***y*** (**1**≤***y***≤$\mathbf{10^8}$) which are the weights of the plates.

Output

Output a sorted list of all possible lifting weights, one per line. There must be no duplicates.

Sample Input	Sample Output
2 5 100 110 5 5 1 4 6	100 110 112 120 122 130

Buggy Robot

There is a robot in a 2D grid. The grid consists of empty cells and obstacles, and there is exactly one cell that is the exit. The robot will exit the grid if it ever reaches the exit cell. Empty cells are denoted as '`.`', the robot's initial position is denoted as '`R`', obstacles are denoted as '`#`', and the exit is denoted as '`E`'.

You can program the robot by sending it a series of commands. The series of commands is a string consisting of characters: '`L`' (move one square left), '`U`' (move one square up), '`R`' (move one square right), or '`D`' (move one square down). If the robot would run into an obstacle or off the edge of the grid, it will ignore the command (but it will continue on to future commands, if there are any).

Your friend sent a series of commands to the robot, but unfortunately, the commands do not necessarily take the robot to the exit.

You would like to fix the string so that the robot will reach the exit square (note once the robot reaches the exit, it stops, even if there are more commands in the string). You can fix the string with a sequence of operations. There are two operations: inserting a command anywhere in the string, or deleting a command anywhere in the string. What is the minimum number of operations you would need to fix the program?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains two integers, ***r*** and ***c*** (**2**≤***r***,***c***≤**50**) which are the number of rows and number of columns of the grid. The next ***r*** lines will each contain a string with exactly ***c*** characters. Each character is one of '`.`' (Empty), '`R`' (the Robot), '`#`' (an Obstacle), or '`E`' (the Exit). This is the grid. There will be exactly one '`R`' and one '`E`' in the grid, and it will always be possible to navigate the robot to the exit. The last line of input will contain a string ***s*** (**1**≤|***s***|≤**50**) of commands. The string ***s*** will consist only of characters '`L`' (left), '`R`' (right), '`U`' (up) or '`D`' (down).

Output

Output a single integer, which is the minimum number of operations necessary to fix the command string so that the robot makes it to the exit.

Sample Input	Sample Output
3 3 R.. .#. ..E LRDD	1
2 4 R.#. #..E RRUUDDRRUUUU	0

As Easy As C-A-B

We all know how to alphabetize a list when you know the order of the alphabet. But can you find the order of the alphabet from an ordered list of words?

Consider the ordered list [**cab**, **cda**, **ccc**, **badca**]. It is clear that '**c**' comes before '**b**' in the underlying alphabet because '**cab**' comes before '**badca**'. Similarly, we know '**a**' comes before '**d**', because '**cab**' < '**cda**', '**a**' comes before '**c**' because '**cab**' < '**ccc**', and '**d**' comes before '**c**' because '**cda**' < '**ccc**'. The only ordering of these four letters that is possible is **adcb**.

Of course, it may not work out so well. If the word list is [**abc**, **bca**, **cab**, **abc**] there is no alphabet that works. The list is *inconsistent*. If the word list is [**dea**, **cfb**] we don't know about the relative positions of any of the letters other than '**c**' and '**d**'. The list is *incomplete*. Every list will fall into exactly one of the following three categories:

1. The list is *correct* if a single alphabet will yield the ordering
2. The list is *incomplete* if more than one alphabet will yield the ordering
3. The list is *inconsistent* if no alphabet will yield the ordering

Given a list of words, determine if the list is *correct*, *incomplete* or *inconsistent*, and if it is *correct*, give the single underlying ordered alphabet.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains a lowercase letter ***last***, and an integer ***n*** (**1**≤***n***≤**100**). Each of the following ***n*** lines will have a string ***s*** (**1**≤|***s***|≤**50**) consisting only of lowercase letters '**a**'-***last***.

Output

If the list is *correct*, and it is possible to uniquely determine the ordering of the letters '**a**'-***last***, output that ordering as a single string. If the list is *incomplete*, and there's not enough information to determine the positions of all the letters, output **0** (zero). If the list is *inconsistent* in any way then output **1**.

Sample Input	Sample Output
d 4 cab cda ccc badca	adcb
c 4 abc bca cab abc	1
f 2 dea cfb	0
b 3 a bb b	1

Note: the last case is inconsistent because there is no alphabet for which **bb** comes before **b**.

Gravity

Consider a 2D grid, which contains apples, obstacles, and open spaces. Gravity will pull the apples straight down, until they hit an obstacle, or the bottom of the grid, or another apple which has already come to rest. Obstacles don't move. Given such a grid, determine where the apples eventually settle.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains two integers, ***r*** and ***c*** (**1≤*r*,*c*≤100**), which are the number of rows and the number of columns of the grid. On each of the next ***r*** lines will be ***c*** characters: '**o**' (lowercase 'Oh') for an apple, '**#**' for an obstacle, and '**.**' for an open space.

Output

Output the grid, after the apples have fallen.

Sample Input	Sample Output
3 3 ooo #.. ..#	o.. #.o .o#
4 2 oo oo o. ..	.. o. oo oo

Histogram

A *Histogram* is a visual representation of data. It consists of bars, with the length of each bar representing the magnitude of a data quantity. Given some data, produce a *Histogram*.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains an integer ***n*** (**1≤*n*≤100**) indicating the number of data items. On each of the next ***n*** lines will be a single integer ***k*** (**1≤*k*≤80**), which is the data.

Output

Print a histogram, horizontally, using the '**=**' character. Print each data item's bar on its own line, in the order given, with the number of '**=**' equal to the data item ***k***. Print no spaces between the '**=**'.

Sample Input	Sample Output
5 1 3 4 6 2	= === ==== ====== ==
4 10 30 25 16	========== ============================== ========================= ================

Islands

You are mapping a faraway planet using a satellite. The planet's surface can be modeled as a grid. The satellite has captured an image of the surface. Each grid square is either land (denoted as '**L**'), water (denoted as '**W**'), or covered by clouds (denoted as '**C**'). Clouds mean that the surface could either be land or water; you cannot tell.

An island is a region of land where every grid cell in the island is connected to every other by some path, and every leg of the path only goes up, down, left or right.

Given an image, determine the minimum number of islands that is consistent with the given image.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains two integers, ***r*** and ***c*** (**1**≤***r***,***c***≤**50**), which are the number of rows and the number of columns of the image. The next ***r*** lines will each contain exactly ***c*** characters, consisting only of '**L**' (representing *Land*), '**W**' (representing *Water*), and '**C**' (representing *Clouds*).

Output

Output a single integer, which is the minimum number of islands possible.

Sample Input	Sample Output
4 5 CCCCC CCCCC CCCCC CCCCC	0
3 2 LW CC WL	1

Project Panoptes

Project Panoptes (projectpanoptes.org) will attempt to use low-cost robotic telescopes to collect data, in order to find exoplanets (planets outside of our solar system). For this program you will implement a simple version of software to detect exoplanets from readings from these telescopes.

The telescope will take daily readings of the brightness of a star. If the brightness of the star decreases, a planet may be passing in front of it. If the brightness decreases following a regular pattern, it is a candidate for having an exoplanet.

For example, consider the daily readings of a star's brightness below:

Day	Brightness	Day	Brightness	Day	Brightness
1	20.6	6	22.2	11	**10.8**
2	21.0	7	**4.8**	12	22.9
3	23.2	8	25.9	13	30.4
4	**12.0**	9	**13.0**	14	**8.6**
5	17.5	10	**12.2**	15	24.9

The average of these readings is **18**. If a reading is less than **80%** of the average of all readings, it is considered to have decreased, so readings below **14.4** are considered to be potential planet sightings (and are highlighted here). The readings at days **4**, **9**, and **14** happen every fifth day (and thus has a period of **5**) and may be because of an exoplanet. Similarly, the readings for day **7** and **14** may indicate an exoplanet. Days **4**, **7**, and **10** do not form a pattern because it would also have to include day **1** and day **13**. Only integer length periods should be considered.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains two integers, ***n*** (**2**≤***n***≤***1,000***) and ***p*** (**1**≤***p***≤***n***-**1**), where ***n*** is the number of observations, and ***p*** is the minimum period length to consider. Each of the next ***n*** lines will contain a floating point number ***x*** (**0.0**≤***x***≤**100.0**), which is the brightness for that day. The days will be in order.

Output

Output a single integer, indicating the smallest possible period of an exoplanet (must be ≥***p***), or **-1** of there is none.

Sample Input	Sample Output
15 2 20.6 21.0 23.2 12.0 17.5 22.2 4.8 25.9 13.0 12.2 10.8 22.9 30.4 8.6 24.9	5

Periodic Strings

Define a *k-periodic* string as follows:

A string ***s*** is *k-periodic* if the length of the string $|s|$ is a multiple of ***k***, and if you chop the string up into $|s|/k$ substrings of length ***k***, then each of those substrings (except the first) is the same as the previous substring, but with its last character moved to the front.

For example, the following string is *3-periodic*:

abccabbcaabc

The above string can break up into substrings **abc**, **cab**, **bca**, and **abc**, and each substring (except the first) is a right-rotation of the previous substring (**abc→cab →bca →abc**).

Given a string, determine the smallest ***k*** for which the string is *k-periodic*.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The single line of input contains a string ***s*** (**1**≤$|s|$≤**100**) consisting only of lowercase letters.

Output

Output the integer ***k***, which is the smallest ***k*** for which the input string is *k-periodic*.

Sample Input	Sample Output
aaaaaaaa	1
abbaabbaabba	2
abcdef	6
abccabbcaabc	3

Mismatched Socks

Fred likes to wear mismatched socks. This sometimes means he has to plan ahead. Suppose his sock drawer has one **red**, one **blue**, and two **green** socks. If he wears the **red** with the **blue**, he's stuck with matching **green** socks. He put together two mismatched pairs if he pairs **red** with **green** and then **blue** with **green**. Given the contents of his sock drawer, how many pairs of mismatched socks can he put together?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains an integer ***n*** (**1≤*n*≤1,000**) which is the number of colors of socks in Fred's drawer. Each of the next ***n*** lines has an integer ***k*** (**1≤*k*≤10^9**) which is the number of socks of that color.

Output

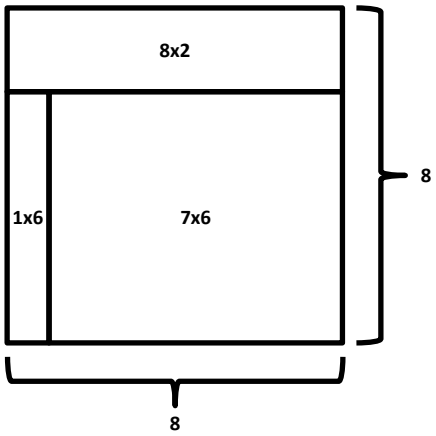
Output a single integer indicating the number of pairs of mismatched socks that Fred can make with the contents of his sock drawer.

Sample Input	Sample Output
3 1 2 1	2
5 1 2 1 10 3	7

It Takes Three

Given three rectangles, determine if they can be glued together to form a square. The rectangles can be rotated, but they cannot overlap.

Here's an example of how three rectangles, 8x2, 1x6 and 7x6, can be put together to form a square 8x8:



Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. There will be exactly three lines of input.

The first line of input contains two integers ***w1*** and ***h1*** (**1**≤***w1***,***h1***≤**100**), which are the width and height of the first rectangle.

The second line of input contains two integers ***w2*** and ***h2*** (**1**≤***w2***,***h2***≤**100**), which are the width and height of the second rectangle.

The third line of input contains two integers ***w3*** and ***h3*** (**1**≤***w3***,***h3***≤**100**), which are the width and height of the third rectangle.

Output

Output **1** if the two rectangles can be put together to form a square, and **0** of they cannot.

Sample Input	**Sample Output**
8 2 1 6 7 6	1

Zigzag

A sequence of integers is said to *Zigzag* if adjacent elements alternate between strictly increasing and strictly decreasing. Note that the sequence may start by either increasing or decreasing. Given a sequence of integers, determine the length of the longest subsequence that *Zigzags*. For example, consider this sequence:

1 2 3 4 2

There are several *Zigzagging* subsequences of length **3**:

1 3 2 **1 4 2** **2 3 2** **2 4 2** **3 4 2**

But there are none of length greater than **3**, so the answer is **3**.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains an integer ***n*** (**1≤*n*≤1,000,000**) which is the number of integers in the list. Each of the following ***n*** lines will have an integer ***k*** (**1≤*k*≤1,000,000**)

Output

Output a single integer, which is the length of the longest *Zigzagging* subsequence of the input list.

Sample Input	Sample Output
5 1 2 3 4 2	3
6 1 1 1 1 1 1	1