2015 ACM ICPC Southeast USA Regional Programming Contest

Division 2

Blur 1
A Classy Problem 2
Egg Drop 4
Excellence 5
Grid 6
Hilbert Sort 7
The Magical 3 10
Persistence 11
Simplicity 12
Triangles 13
Xedni Drawkcab 14

Hosted by:

College of Charleston

Florida Institute of Technology

Georgia Institute of Technology

University of West Florid

Blur

You have a black and white image. You decide to represent this image with one number per pixel: black is 0, and white is 1. Someone asks you to blur the image, resulting in various shades of gray. The way you decide to blur the image is as follows: You create a new image that is the same size as the old one, and each pixel in the new image has a value equal to the average of the 9 pixels in the 3x3 square centered at the corresponding old pixel. When doing this average, wrap around the edges, so the left neighbor of a leftmost pixel is in the rightmost column, and the top neighbor of an uppermost pixel is on the bottom. From a corner, you might wrap twice. For example, go diagonally up and left from the top left pixel, and you end up on the bottom right. This way, the 3x3 square always gives you exactly 9 pixels to average together. If you want to make the image blurrier, you can take the blurred image and blur it again using the exact same process. Given an input image and a fixed number of times to blur it, how many distinct colors does the final image have?

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 three space-separated integers ***w***, ***h***, and ***b*** (3≤***w***,***h***≤100, 0≤***b***≤9), indicating the width ***w*** and height ***h*** of the input image, as well as the number of times ***b*** to blur the image.

The following ***h*** lines of ***w*** space-separated integers describe the original image, with each integer being either 0 or 1, corresponding to the color of the pixel.

Output

Output a single integer indicating the number of unique colors in the final blurred image.

Sample Input	Sample Output

5 4 1
0 0 1 1 0
0 0 1 1 0
0 0 1 1 0
0 0 1 1 0
	3

3 3 2
1 0 0
0 1 0
0 1 0
	1

A Classy Problem

In his memoir *So, Anyway*, comedian John Cleese writes of the class difference between his father (who was `middle-middle-middle-lower-middle class' and his mother (who was `upper-upper-lower-middle class'). These fine distinctions between classes tend to confuse American readers, so you are to write a program to sort a group of people by their classes to show the true distinctions.

For this problem, there are three main classes: *upper*, *middle*, and *lower*. Obviously, the highest is *upper* and the lowest is *lower*. But there can be distinctions within a class, so *upper-upper* is a higher class than *middle-upper*, which is higher than *lower-upper*. However, all of the upper classes (*upper-upper*, *middle-upper*, and *lower-upper*) are higher than any of the *middle* classes.

Within a class like *middle-upper*, there can be further distinctions as well, leading to classes like *lower-middle-upper-middle-upper*. When comparing classes, once you've reached the lowest level of detail, you should assume that all further classes are the same as the *middle* level of the previous level of detail. So *upper* class and *middle-upper* class are equivalent, as are *middle-middle-lower-middle* and *lower-middle*.

The 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 ***n***, the number of names to follow. Each of the following ***n*** lines contains the name of a person as a sequence of 1 or more lower case letters, then a colon, then a single space, and then the class of the person. The class of the person will include one or more modifiers, separated by a single space, followed by a single space and then the word **`class`**. The entire line will be no longer than 256 characters.

The Output

The output will consist of a list of names from highest to lowest class, one name per line. If two people have the same class, they should be listed in alphabetical order by name.

Sample Input	**Sample Output**

`5`
`mom: upper upper lower middle class`
`dad: middle middle lower middle class`
`queenelizabeth: upper upper class`
`chair: lower lower class`
`unclebob: middle lower middle class`
	`queenelizabeth`
	`mom`
	`dad`
	`unclebob`
	`chair`

`10`
`rich: lower upper class`
`mona: upper upper class`
`dave: middle lower class`
`charles: middle class`
`tom: middle class`
`william: lower middle class`
`carl: lower class`
`violet: middle class`
`frank: lower class`
`mary: upper class`
	`mona`
	`mary`
	`rich`
	`charles`
	`tom`
	`violet`
	`william`
	`carl`
	`dave`
	`frank`

Egg Drop

Suppose you are given an egg and a ***k***-floor building, and you want to know the highest floor from which you can drop the egg and not have it break. You have stumbled upon some logs detailing someone trying this experiment!

Based on these logs, you need to compute two quantities: the lowest floor that you can drop the egg where the egg could possibly break, and the highest floor that you can drop the egg where the egg could possibly be safe. You know that the egg will not break if dropped on floor 1, and will break if dropped on floor ***k***. You also know that logs are consistent; if an egg did not break when dropped from floor ***x***, it will not break when dropped from any lower floors, and if an egg did break when dropped from floor ***y***, it will break when dropped from all higher floors. Although consistent, the logs may be incomplete, in that they might not cover all floors of the building.

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 space-separated integers ***n*** (1≤***n***≤100), the number of egg drops reported, and ***k*** (3≤***k***≤100), the number of floors of the building. Each of the following ***n*** lines contains a floor number ***f*** (1≤***f***≤***k***), then a single space, and then the result of the egg drop: either **`SAFE`** or **`BROKEN`**, in all caps. The input will be consistent, as described above.

Output

Output two space-separated positive integers on a single line. The first integer should be the number of the lowest floor where you can drop the egg and it could possibly break and still be consistent with the results. The second integer should be the number of the highest floor where you can drop the egg and it could possibly not break.

Sample Input	**Sample Output**

`2 10`
`4 SAFE`
`7 BROKEN`
	`5 6`

`3 5`
`2 SAFE`
`4 SAFE`
`3 SAFE`
	`5 4`

`4 3`
`2 BROKEN`
`2 BROKEN`
`1 SAFE`
`3 BROKEN`
	`2 1`

Excellence

The World Coding Federation is setting up a huge online programming tournament of teams comprised of pairs of programmers. Judge David is in charge of putting teams together from the South Eastern delegation. Luckily, he has an even number of students who desire to compete, so that he can make sure that each student does compete. However, he'd like to maintain his pristine reputation amongst other judges by making sure that each of the teams he fields for the competition meets some minimum total rating. We define the total rating of a team to be the sums of the ratings of both individuals on the team. Help David determine the maximal value, ***x***, such that he can form teams, each of which have a total rating greater than or equal to ***x***. Note that every student must be placed on exactly one team of two students.

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 single integer ***n*** ($1 \le n \le 10^5$), representing the number of students who desire to enter the online programming tournament. It is guaranteed that ***n*** is an even number. Each of the following ***n*** lines contains a single integer ***s*** ($1 \le s \le 10^6$), representing the rating of a student.

Output

Output a single integer on a line by itself representing the maximal value, ***x***, such that David can form teams where every team has a total rating greater than or equal to ***x***.

Sample Input	Sample Output
4 1 2 3 5	5
2 18 16	34
4 13 12 19 14	27

Grid

You are on an ***n*x*m*** grid where each square on the grid has a digit on it. From a given square that has digit ***k*** on it, a *Move* consists of jumping exactly ***k*** squares in one of the four cardinal directions. A move cannot go beyond the edges of the grid; it does not wrap. What is the minimum number of moves required to get from the top-left corner to the bottom-right corner?

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 space-separated integers ***n*** and ***m*** (1≤***n***,***m***≤500), indicating the size of the grid. It is guaranteed that at least one of ***n*** and ***m*** is greater than 1.

The next ***n*** lines will each consist of ***m*** digits, with no spaces, indicating the ***n*x*m*** grid. Each digit is between 0 and 9, inclusive.

The top-left corner of the grid will be the square corresponding to the first character in the first line of the test case. The bottom-right corner of the grid will be the square corresponding to the last character in the last line of the test case.

Output

Output a single integer on a line by itself representing the minimum number of moves required to get from the top-left corner of the grid to the bottom-right. If it isn't possible, output **`-1`**.

Sample Input	**Sample Output**

`2 2`
`11`
`11`
	`2`

`2 2`
`22`
`22`
	`-1`

`5 4`
`2120`
`1203`
`3113`
`1120`
`1110`
	`6`

Hilbert Sort

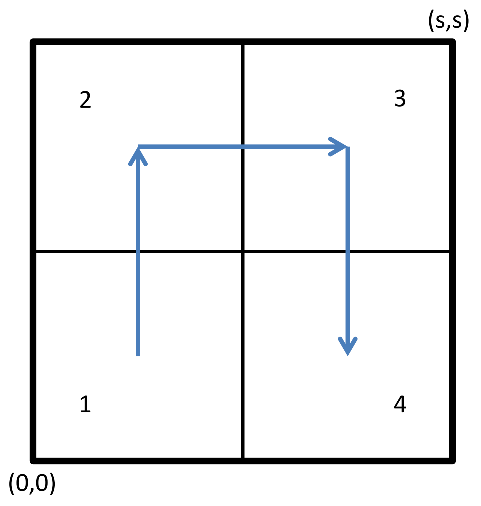
Sorting numerical data not only makes it easy to search for a particular item, but also makes better use of a CPU's cache: any segment of data that's contiguous in memory will describe a set of items that are similar in some sense. Things get more complicated if our data represents points on a 2D grid. If points (***x***,***y***) are sorted by ***x***, breaking ties by ***y***, then adjacent points will have similar ***x*** coordinates but not necessarily similar ***y***, potentially making them far apart. To better preserve distances, we can sort the data along a space-filling curve.

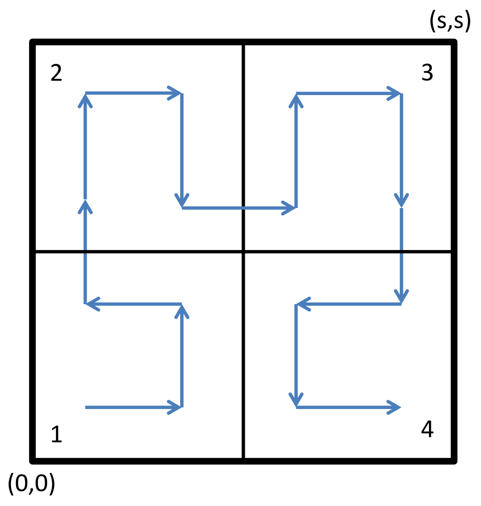
The Hilbert curve starts at the origin (**0**,**0**), finishes at (***s***,**0**), in the process traversing every point in axis-aligned square with corners at (**0**,**0**) and (***s***,***s***). It has the following recursive construction: split the square into four quadrants meeting at (***s***/**2**, ***s***/**2**). Number them 1 to 4, starting at the lower left and moving clockwise. Recursively fill each of them with a suitably rotated and scaled copy of the full Hilbert curve.

Start with a single point at (***s***/**2**,***s***/**2**). Then, repeat these steps:

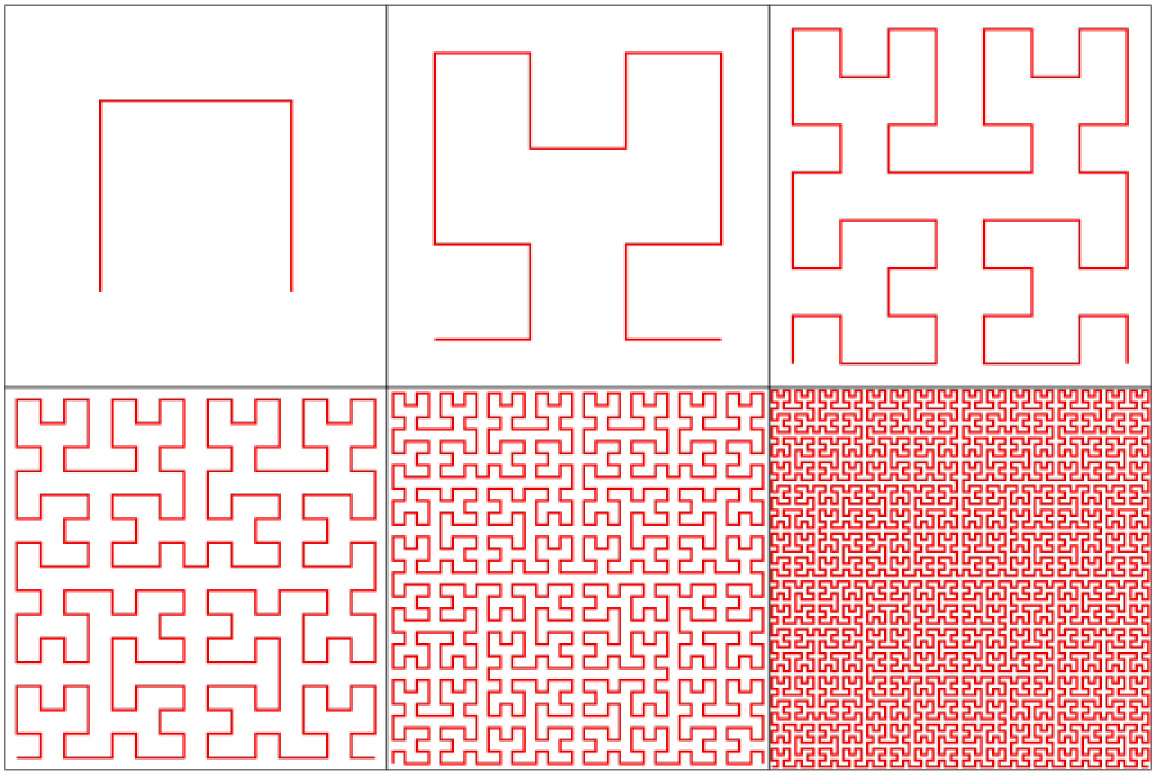
- Scale and copy the current construction into each of the 4 quadrants.
- Rotate quadrant 1 by -90 degrees and flip it vertically, so that the start of the curve is closest to the lower left corner (**0**,**0**).
- Rotate quadrant 4 by 90 degrees and flip it vertically, so that the end of the curve is closest to the lower right corner (***s***,**0**).
- Now, connect the end of the curve in quadrant 1 to the start of the curve in quadrant 2, connect the end of quadrant 2 to the start of quadrant 3, and the end of quadrant 3 to the start of quadrant 4.

Here are the first two iterations:





The Hilbert Curve is built by repeating this construction infinitely many times. The following diagram shows the first six steps of building the Hilbert Curve:



Given some places of interest inside of a square region, sort them according to when the Hilbert curve visits them, starting from (0,0). Without going into gory detail about Fractal theory, note that making ***s*** odd guarantees that all integer points are visited just once, so their visitation order with relation to each other is unambiguous.

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 space-separated integers ***n*** and ***s*** ($1 \le n \le 100{,}000$, $1 \le s < 10^9$, ***s*** is odd). The next ***n*** lines describe locations of interest by space-separated integers ***x*** and ***y*** ($0 \le x, y \le s$). No two locations will share the same position.

Output

Output the ***n*** ordered pairs, one per line, with ***x*** and ***y*** separated by a space, Hilbert-sorted according to their positions.

Sample Input

```
14 25
5 5
5 10
5 20
10 5
10 10
10 15
10 20
15 5
15 10
15 15
15 20
20 5
20 10
20 20
```

Sample Output

```
5 5
10 5
10 10
5 10
5 20
10 20
10 15
15 15
15 20
20 20
20 10
15 10
15 5
20 5
```

The Magical 3

Three is a magic number.

Yes it is; it's a magic number.

Somewhere in the ancient, mystic trinity,

You get three as a magic number.

- Schoolhouse Rock

According to Pythagoras and the Pythagorean School, the number 3 - which they called *triad* - is the noblest of all digits, as it is the only positive integer to equal the sum of all of the positive integers below it (1+2=3), and it is the only positive integer whose sum with those below equals the product of them and itself (1+2+3=1x2x3).

Your task is to find the magic – the magic **3**, that is – when it can be the last digit in a representation of a positive integer in some base. Consider, for example, the number 11. It can be represented as *ONE-THREE* (13) in base 8 and as *TWO-THREE* (23) in base 4. You are to write a program that will find the smallest base for a given positive integer where the input number's representation in that base ends in 3. This is possible for all integers greater than 6.

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 consist of a single line with a single integer $\boldsymbol{n}$ ($7 \le \boldsymbol{n} < 2^{31}$).

Output

For each test case, output a single integer representing the smallest base in which the input $\boldsymbol{n}$ ends with a 3.

Sample Input	Sample Output
11	4
42	13
9876	3291

Persistence

Consider the series of numbers where each term is the product of the decimal digits of the previous term. Eventually the term will be reduced to a single digit.

For example start with **679**:

```
679:  6*7*9  →  378
378:  3*7*8  →  168
168:  1*6*8  →  43
 48:   4*8   →  32
 32:   3*2   →  6
```

The number of steps this takes is called the *Persistence* of a number. Thus, the persistence of **679** is **5**, since that's number of steps it took to get to a single digit number.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The input will consist of a single line, with a positive number of up to 9 decimal digits with no leading zeros.

Output

For each test case, output a single integer, representing the persistence of the input number; that is, the number of steps necessary to reduce it to a single digit.

Sample Input	**Sample Output**
5	0
10	1
679	5

Simplicity

For a string of letters, define the *Simplicity* of the string to be the number of distinct letters in the string. For example, the string **string** has simplicity 6, and the string **letter** has simplicity 4.

You like strings which have simplicity either 1 or 2. Your friend has given you a string and you want to turn it into a string that you like. You have a magic eraser which will delete one letter from any string. Compute the minimum number of letters you must erase in order to turn the string into a string with simplicity at most 2.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The input will consist of a line with a single string consisting of at least 1 and at most 100 lower case letters.

Output

Output a single integer, indicating the minimum number letters you need to erase in order to give the string a simplicity of 1 or 2.

Sample Input	Sample Output
`string`	`4`
`letter`	`2`
`aaaaaa`	`0`
`uncopyrightable`	`13`
`ambidextrously`	`12`
`assesses`	`1`
`assassins`	`2`

Triangles

Determine if it is possible to produce two triangles of given side lengths, by cutting some rectangle with a single line segment, and freely rotating and flipping the resulting pieces.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The input consists of two lines. The first line contains three space-separated integers, indicating the desired side lengths of the first triangle. Similarly, the second line contains three space-separated integers, denoting the desired side lengths of the second triangle. It is guaranteed that the side lengths produce valid triangles. You may assume that the maximum side length of a triangle is 100, and that the minimum is 1.

Output

If there exists a rectangle which could have been cut to form triangles of the given side lengths, output 1. Otherwise, output 0.

Sample Input	Sample Output

`3 4 5`
`4 3 5`
	`1`

`3 4 6`
`4 6 3`
	`0`

`39 52 65`
`25 60 65`
	`0`

Xedni Drawkcab

For years, before computers, Merriam Webster maintained a Backward Index of all of the words in the English language. They had all of the words on cards, one word per card, backwards. The cards were alphabetized. This was very useful, before computers, to determine such things as 'How many words end with 'TION'?'

Given a list of words consisting of only capital letters, create a Backward Index by reversing them and printing the reversals in alphabetical order.

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 will contain an integer ***n*** (1≤***n***≤1,000) indicating the number of words. On the following n lines will be the words, one per line. The words will be from 1 to 100 letters long. The words will consist of only capital letters, and there will be no spaces or blank lines.

Output

Output the words, reversed and sorted, one word per line.

Sample Input	Sample Output
3 ALCATRAZ CARDAMOM BAKLAVA	AVALKAB MOMADRAC ZARTACLA