# Problem 1: Pangram Construction

## The Problem

Victoria loves pangrams. A pangram is a sentence where every letter of the alphabet occurs **at least** once. For example, the sentence “*the quick brown fox jumps over a lazy dog*” is a pangram.

Since she likes pangrams so much, she wants to make all sentences in the universe pangrams. How many characters does she need to add to a given sentence to make it a pangram?

## Input

The first line of input contains one integer, **N**, the number of sentences.

The next **N** lines contain a sentence **S** consisting of the lowercase letters **a-z** and spaces.

## Constraints

* 1 ≤ **N** ≤ 1000
* 1 ≤ **|S|** ≤ 1000

## Output

For each sentence, print the minimum number of characters Victoria needs to add to make it a pangram.

## Sample Input

5

the quick brown fox jumps over a lazy dog

sphinx of black quartz hear my vow

the five boxing wizards jump fast

abcdefghijklmnopqrstuvwxyz

a

## Sample Output

0

3

5

0

25

# Problem 2: Snake Game

## The Problem

Aloy likes playing the snake game. In this game, a grid with **N** rows and **M** columns contains a snake in the upper left hand corner, starting out with a length of one. The snake cannot go out of bounds, and it cannot intersect with itself at any point. If it does, the game is over.

At each step, the snake can move up, down, left, or right. In order to keep growing, the snake has to eat a food pellet, which is placed randomly on the board. Once one is eaten, another randomly appears.

A picture containing drawing

Description automatically generated

But here's the twist: **YOU** are placing the food pellets! A pellet can be placed at any location in the grid that the snake is not occupying. Can you maximize how long the snake can get, given these conditions?

## Input

The first line of input contains an integer, **T**, the number of test cases.

The next **T** lines contain two integers, **N** and **M**, the size of the grid. The snake always starts in the upper left corner.

## Constraints

* 1 ≤ **T** ≤ 5000
* 1 ≤ **N, M** ≤ 109

## Output

Output one integer per test case**:** the maximum length the snake can achieve within the given grid.

## Sample Input

2

2 2

2 3

## Sample Output

4  
6

# Problem 3: Mandelbrot Set

## The Problem

The Mandelbrot set is the set of complex numbers for which the equation

does not diverge starting from . Since you cannot calculate that equation out to infinity, we will simplify the problem to finding the numbers for which the magnitude of does not exceed value **T** after **N** iterations.

As output, you will write ASCII art of the Mandelbrot set using the characters ‘**X’** and ‘**.**’.

## Input

The first line of input consists of 4 *integers*, **N**, **T**, **H**, and **W**: the number of iterations to run , the threshold to include in the set, and the height and width of the output grid.

The second line of input consists of 4 *floating point* numbers, **x1**, **y1**, **x2**, **y2**. represents the upper left corner of the grid in the complex plane, and represents the lower right corner of the grid.

The character in row **a**, column **b** represents the complex number

## Constraints

* 1 ≤ **N, T** ≤ 100
* 1 ≤ **H, W** ≤ 1000
* -10 ≤ **x1** < **x2** ≤ 10
* -10 ≤ **y1** < **y2** ≤ 10

## Output

Output consists of a grid **H** characters high and **W** characters wide representing the Mandelbrot Set based on the input parameters. The grid should consist of the character ‘**X**’ if the given grid position is in the Mandelbrot set and the character **‘.’** if it is not.

## Sample Input

50 2 20 50

-2 1 1 -1

## Sample Output

..................................................

..................................................

..............................XXXX................

..............................XXX.................

........................XX.XXXXXXXXX..............

.........................XXXXXXXXXXXXXX...........

.......................XXXXXXXXXXXXXXXXX..........

......................XXXXXXXXXXXXXXXXXX..........

..............XXXXXX..XXXXXXXXXXXXXXXXXX..........

.............XXXXXXXXXXXXXXXXXXXXXXXXXXX..........

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX............

.............XXXXXXXXXXXXXXXXXXXXXXXXXXX..........

..............XXXXXX..XXXXXXXXXXXXXXXXXX..........

......................XXXXXXXXXXXXXXXXXX..........

.......................XXXXXXXXXXXXXXXXX..........

.........................XXXXXXXXXXXXXX...........

........................XX.XXXXXXXXX..............

..............................XXX.................

..............................XXXX................

..................................................

# Problem 4: Haha

## The Problem

This problem has no statement. You must look at the input and output data, figure out what's going on, and solve it. For your own tests, you may use the problem page on the contest site to input custom data and get back live results.

## Sample Input

3

2

5

10

## Sample Output

HA

HAH

HAHA

# Problem 5: Does It Scale?

## The Problem

Rost enjoys playing the piano. He knows that a scale consists of a series of **whole** and **half** steps. A half step is one unit, and a whole step is two units. For example, a major scale is **WWHWWWH** and a chromatic scale is **HHHHHHHHHHHH**.

The actual notes on the piano are **C Db D Eb E F Gb G Ab A Bb B**. This sequence wraps around. A jump from **C** to **D** is a whole step (two notes up). However, **C** to **Db** is a half step. Also, **B** to **C** is a half step.

**NOTE**: We will not consider sharps in this problem.

A close up of a keyboard

Description automatically generated

As seen in the image above, the sequence of notes **C D E F G A B** could be a major or minor scale, depending on which note starts the scale. If it starts at **C**, it's a major scale. If it starts at **A**, it's a minor scale.

Rost is given a list of scales in the format **WWHWWWH**. Then, he is given sequences notes. Can you help him determine which scales that each sequence could be, if started at anywhere in the sequence?

## Input

The first line of input contains two integers, **N** and **Q**, the number of scales and queries respectively.

The next **N** lines contain two string each: the name of the scale followed by the sequence of steps.

The next **Q** lines contain space-separated strings, describing a sequence of notes.

## Constraints

* 1 ≤ **N** ≤ 250
* 1 ≤ **Q** ≤ 104

## Output

For each **Q**, output a space-separated list of strings *in alphabetical order*: The possible scales for the sequence of notes. There will always be at least one valid scale. Do not print duplicates.

## Sample Input

3 2

major WWHWWWH

minor WHWWHWW

chromatic HHHHHHHHHHHH

C D E F G A B

C Db D Eb E F Gb G Ab A Bb B

## Sample Output

major minor  
chromatic

# Problem 6: CodyCross

## The Problem

A CodyCross puzzle is a crossword-style game played on a grid. It consists of horizontal words on a grid that overlap with one vertical word, such as in the image below:

A picture containing room

Description automatically generated

Given a dictionary of words, how many possible ways can you complete a correct CodyCross puzzle using those words? A word can be used once, more than once, or not at all within any puzzle.

## Input

Input will start with a line containing two integers, **N**, **W**, and **C**. Each valid CodyCross puzzle you generate must be an **N** x **N** grid where each row is a letter from the dictionary, and column **C** is a valid word from the dictionary. The next **W** lines each contain a distinct lowercase English word from the dictionary. Each of these words will be **N** letters long.

## Constraints

* 1 < **N** ≤ 20
* 1 ≤ **W** ≤ 105
* 1 ≤ **C** ≤ **N**

## Output

Output consists of a single integer **X**, the number of valid CodyCross puzzles that meet the requirements. Since this number may be large, print it modulo 1,000,000,007.

## Sample Input

3 4 2

cat

and

gym

any

## Sample Output

2

# Problem 7: Peg Solitaire

## The Problem

The peg solitaire game, commonly found at Cracker Barrel, is played on a triangular board with 15 holes.

Each hole either holds a peg or is empty. A move consists of jumping a peg over an adjacent peg into an empty hole. A jump may be made horizontally, or along the diagonals of the board.

For example, in the board below, the following jumps are possible:

A close up of a logo

Description automatically generated

The images above show four possible moves (two in each picture). The highlighted red peg is the one that can move. The blue holes are the empty locations where it can jump. After jumping to the new location, the peg underneath the jump is removed from the board.

A board with no possible moves left is called "fixed". The number of pegs in a fixed position is called a "fixed count".

Cracker Barrel's game asks you to make a series of jumps such that you end in a fixed position with the fewest number of pegs possible. However, some of us like to play for an alternative goal - reach a fixed position with the *highest* number of pegs possible!

Given a starting configuration of pegs, your task is to compute the highest and lowest fixed counts you can reach.

## Input

Each test case consists of 15 integers, separated by spaces. Each integer is either **1** (indicating a peg) or **0** (indicating an empty hole.) The holes are specified from left to right and top to bottom:

1-1 2-1 2-2 3-1 3-2 3-3 4-1 4-2 4-3 4-4 5-1 5-2 5-3 5-4 5-5

where i-j indicates the ith row and the jth column. See the image below for more details.

A close up of a logo

Description automatically generated

## Output

Output two comma-separated integers inside parentheses: The most pegs you can achieve from the starting position, and the fewest pegs you can achieve from that position.

## Sample Input 0

1 1 0 0 0 0 0 0 0 0 0 0 0 0 0

## Sample Output 0

(1, 1)

## Sample Input 1

1 1 0 1 1 0 0 0 0 0 0 0 0 0 0

## Sample Output 1

(3, 2)

# Problem 8: Traffic Light

## The Problem

Ethan is a traffic engineer designing a stoplight for a new town, and he wants to minimize the amount that his stoplight disrupts traffic. To simplify his work, he assumes that the stoplight connects two roads, and people driving down the roads will stop instantaneously at the light if the light is red and pass through at full speed if the light is green. Cars in this city accelerate and decelerate instantaneously. Ethan also assumes that cars will not turn at the stop light; cars on road A will stay on road A, and cars on road B will stay on road B. Ethan is also bounded by a few rules of traffic:

* There must be a **C**-second delay between when the light turns red on one road and when the light turns green on the other road.
* The light must cycle at a constant rate: **A** seconds green for Road A, **C** seconds red for both, **B** seconds green for Road B, and **C** seconds red for both.
* Both **A** and **B** must be positive integers that do not exceed **D** seconds

Given the times that cars arrive at the light on each road, can you help Ethan optimize the timing of the light to minimize the total time drivers spend waiting?

## Input

The first line of input contains four integers, **D**, **C**, **K**, and **M**.

The second line of input contains **K** integers, **a1** to **ak**, the time in seconds each car arrives at the stoplight on Road A.

The third line of input contains **M** integers, **b1** to **bm**, the time in seconds each car arrives at the stoplight on Road B.

**NOTE**: if car **X** arrives at the stoplight at time **Y** and the stoplight is red in that direction, car **X** must wait at the light until it turns green at time **Z**. Your job is to minimize the sum of **Z** **- Y** over all cars.

## Constraints

* 1 ≤ **D** ≤ 10
* 0 ≤ **C** ≤ 109
* 1 ≤ **K, M** ≤ 100
* 0 ≤ **ai, bi** ≤ 109

## Output

Output consists of three integers **A**, **B**, and **S** on one line representing the optimal timings of the stoplight and the resulting sum of delay. If there are multiple optimal timings, print the one with the smallest sum of **A + B**, and if there multiple timings with a smallest **A + B**, print the one of those with the smallest **A** value.

**NOTE**: the time cycle starts at time 0 with **A** seconds of Road A being green and continues on the loop described above indefinitely. If a car arrives at the light at the second it turns green, the car can pass through with no delay, but if it arrives the second the light turns red, it must wait until the next green light.

## Sample Input

10 1 1 1  
1  
3

## Sample Output

2 1 0

# Problem 9: English Channel

## The Problem

Crossing the English Channel has become a feat for some of the most athletic swimmers. The first recorded event was by a man named Matthew Webb, who crossed the Straight of Dover. Logically, he would want to take the shortest route across the channel.

A close up of a map

Description automatically generated

One such route is the route between Dover and Wissant. See the image above for details.  
For this problem, you are given a grid with three types of characters: **‘X’** (Dover), **‘O’** (Wissant), and **‘.’** (water). Can you help Matthew determine the smallest Manhattan distance between any two points of Dover and Wissant?

**NOTE:** Manhattan distance is the sum of the **X** difference and the **Y** difference between two points. For example, the Manhattan distance between **(1, 4)** and **(7, 10)** is **abs(1 - 7) + abs(4 - 10) = 12**.

## Input

The first line of input contains two integers, **N** and **M**, the height and width of the map.

The next **N** lines contain **M** characters each, describing the map. A **‘.’** is a water square, **‘X’** is Dover land, and **‘O’** is Wissant land.

## Constraints

* 1 ≤ **N, M** ≤ 1000

## Output

Output an integer **D**, the minimum Manhattan distance between any two points of Dover and Wissant.

## Sample Input

10 10  
XXXXXXX...  
XXXXX.....  
XXXXXX....  
XXXX......  
X.........  
.........O  
.....OOOOO  
.OOOOOO...  
OOOOO.....  
OOOO......

## Sample Output

4