Problem A. ASCII Addition

Source file name:	addition.c, addition.cpp, addition.java, addition.py
Input:	`Standard`
Output:	`Standard`

Nowadays, there are smartphone applications that instantly translate text and even solve math problems if you just point your phone's camera at them. Your job is to implement a much simpler functionality reminiscent of the past – add two integers written down as ASCII art.

An *ASCII art* is a matrix of characters, exactly 7 rows high, with each individual character either a dot or the lowercase letter x.

An expression of the form $a + b$ is given, where both a and b are positive integers. The expression is converted into ASCII art by writing all the expression characters (the digits of a and b as well as the + sign) as 7×5 matrices, and concatenating the matrices together with a single column of dot characters between consecutive individual matrices. The exact matrices corresponding to the digits and the + sign are as folows:

```
xxxxx  ....x  xxxxx  xxxxx  x...x  xxxxx  xxxxx  xxxxx  xxxxx  xxxxx  .....
x...x  ....x  ....x  ....x  x...x  x....  x....  ....x  x...x  x...x  ..x..
x...x  ....x  ....x  ....x  x...x  x....  x....  ....x  x...x  x...x  ..x..
x...x  ....x  xxxxx  xxxxx  xxxxx  xxxxx  xxxxx  ....x  xxxxx  xxxxx  xxxxx
x...x  ....x  x....  ....x  ....x  ....x  x...x  ....x  x...x  ....x  ..x..
x...x  ....x  x....  ....x  ....x  ....x  x...x  ....x  x...x  ....x  ..x..
xxxxx  ....x  xxxxx  xxxxx  ....x  xxxxx  xxxxx  ....x  xxxxx  xxxxx  .....
```

Given an ASCII art for an expression of the form $a + b$, find the result of the addition and write it out in the ASCII art form.

Input

The input file contains several test cases, each of them as described below.

Input consists of exactly 7 lines and contains the ASCII art for an expression of the form $a + b$, where both a and b are positive integers consisting of at most 9 decimal digits and written without leading zeros.

Output

For each test case, output 7 lines containing ASCII art corresponding to the result of the addition, without leading zeros.

Example

Input
....x.xxxxx.xxxxx.x...x.xxxxx.xxxxx.xxxxx.......xxxxx.xxxxx.xxxxx ....x.....x.....x.x...x.x.....x.........x...x...x...x.x...x.x...x ....x.....x.....x.x...x.x.....x.........x...x...x...x.x...x.x...x ....x.xxxxx.xxxxx.xxxxx.xxxxx.xxxxx.....x.xxxxx.xxxxx.xxxxx.x...x ....x.x.........x.....x.....x.x...x.....x...x...x...x.....x.x...x ....x.x.........x.....x.....x.x...x.....x...x...x...x.....x.x...x ....x.xxxxx.xxxxx.....x.xxxxx.xxxxx.....x.......xxxxx.xxxxx.xxxxx
Output
....x.xxxxx.xxxxx.xxxxx.x...x.xxxxx.xxxxx ....x.....x.....x.x.....x...x.x.........x ....x.....x.....x.x.....x...x.x.........x ....x.xxxxx.xxxxx.xxxxx.xxxxx.xxxxx.....x ....x.x.........x.....x.....x.....x.....x ....x.x.........x.....x.....x.....x.....x ....x.xxxxx.xxxxx.xxxxx.....x.xxxxx.....x

Problem B. Book Borders

Source file name:	borders.c, borders.cpp, borders.java, borders.py
Input:	`Standard`
Output:	`Standard`

A book is being typeset using a fixed width font and a simple greedy algorithm to fill each line. The book contents is just a sequence of words, where each word contains one or more characters.

Before typesetting, we choose a *maximum line length* and denote this value with m. Each line can be at most m characters long including the space characters between the words. The typesetting algorithm simply processes words one by one and prints each word with exactly one space character between two consecutive words on the same line. If printing the word on the current line would exceed the maximum line length m, a new line is started instead.

```
|its.a.long...|        |its.a.long.way|
|way.to.the...|        |to.the.top.if.|
|top.if.you...|        |you.wanna.rock|
|wanna.rock.n.|        |n.roll........|
|roll.........|
```

Text from the example input with maximum line lengths 13 and 14

You are given a text to be typeset and are experimenting with different values of the maximum line length m. For a fixed m, the *leading sentence* is a sentence (a sequence of words separated with a single space character) formed by the first words of lines top to bottom. In the example above, when the sample text is typeset with the maximum line length 14, the leading sentence is "`its to you n`".

Given a text and two integers a and b, find the length of the leading sentence for every candidate maximum line length between a and b inclusive. The length of a sentence is the total number of characters it contains including the space characters.

Input

The input file contains several test cases, each of them as described below.

The first line contains the text to be typeset – a sequence of words separated by exactly one space character. Each word is a string consisting of one or more lowercase letters from the English alphabet.

The second line contains two integers a and b – the edges of the interval we are interested in, as described above.

It is guaranteed that $1 \leq w \leq a \leq b \leq z \leq 500000$, where w is the length of the longest word in the text and z is the total number of characters in the text including the space characters.

Output

For each test case, output $b - a + 1$ lines – the k-th of those lines should contain a single integer – the total length of the leading sentence when the maximum line length is equal to $a - 1 + k$.

Example

Input
its a long way to the top if you wanna rock n roll 13 16
Output
22 12 12 15

Problem C. Cow Confinement

Source file name:	cow.c, cow.cpp, cow.java, cow.py
Input:	`Standard`
Output:	`Standard`

A nearby pasture can be represented as a rectangular grid consisting of 10^6 rows and 10^6 columns. The rows are numbered with integers 1 through 10^6 top to bottom, the columns with integers 1 through 10^6 left to right.

A herd of n cows is scattered through the grid, each cow occupying a unit square. The pasture also contains m dandelion flowers (which cows like), again each occupying a unit square. Finally, the pasture contains p fences, each a rectangle running along the edges of unit squares. Fences *do not intersect or touch*. However, a fence may contain other fences inside the enclosed area.

Due to unfavorable wind conditions, cows can only move in two directions – down or right. Cows can go through squares occupied by other cows or flowers, but cannot cross fences.

For each cow, find the total number of flowers reachable from its present location.

Input

The input file contains several test cases, each of them as described below.

Input contains three blocks – the first block describes fences, the second one flowers and the third one cows.

The first line of the first block contains an integer f ($0 \leq f \leq 200000$) – the number of fences. Each of the following f lines contains four integers r_1, c_1, r_2, c_2 ($1 \leq r_1, c_1, r_2, c_2 \leq 10^6$) describing a single fence – r_1 and c_1 are the coordinates (row and column) of the upper-left corner square inside the fence, while r_2 and c_2 are the coordinates of the lower-right corner square inside the fence. No two fences will intersect or touch.

The first line of the second block contains an integer m ($0 \leq m \leq 200000$) – the number of flowers. The k-th of the following m lines contains two integers r and c ($1 \leq r, c \leq 10^6$) – the location of the k-th flower. No two flowers will occupy the same location.

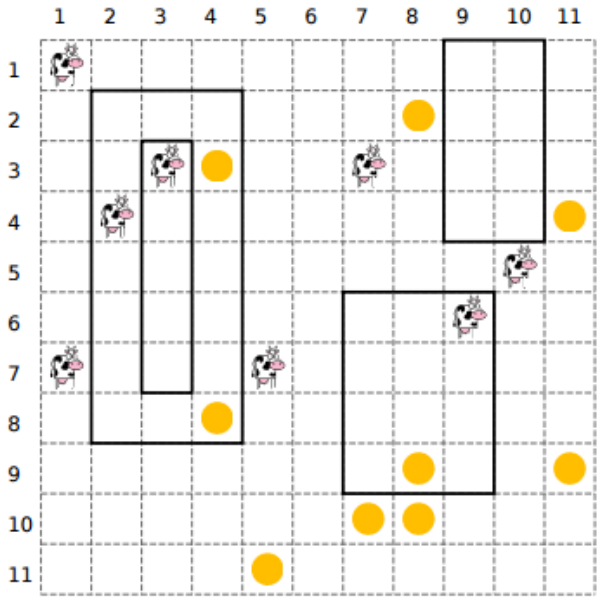
The first line of the third block contains an integer n ($1 \leq n \leq 200000$) – the number of cows. The k-th of the following n lines contains two integers r and c ($1 \leq r, c \leq 10^6$) – the location of the k-th cow. No two cows will occupy the same location, and no flower and cow will occupy the same location.

Output

For each test case, output should consist of n lines. The k-th line should contain a single integer – the total number of flowers reachable from the location of the k-th cow.

Example

Input	Output
4 2 2 8 4 1 9 4 10 6 7 9 9 3 3 7 3 9 3 4 8 4 11 5 10 7 10 8 9 8 2 8 4 11 9 11 8 1 1 5 10 6 9 3 7 7 1 4 2 7 5 3 3	5 1 0 1 3 1 3 0



Problem D. Digit Division

Source file name:	digitdiv.c, digitdiv.cpp, digitdiv.java, digitdiv.py
Input:	`Standard`
Output:	`Standard`

We are given a sequence of n decimal digits. The sequence needs to be partitioned into one or more contiguous subsequences such that each subsequence, when interpreted as a decimal number, is divisible by a given integer m.

Find the number of different such partitions modulo $10^9 + 7$. When determining if two partitions are different, we only consider the locations of subsequence boundaries rather than the digits themselves, e.g. partitions 2|22 and 22|2 are considered different.

Input

The input file contains several test cases, each of them as described below.

The first line contains two integers n and m ($1 \leq n \leq 300000$, $1 \leq m \leq 1000000$) – the length of the sequence and the divisor respectively. The second line contains a string consisting of exactly n digits.

Output

For each test case, output a single integer – the number of different partitions modulo $10^9 + 7$.

Example

Input	Output
4 2 1246	4
4 7 2015	0

Problem E. Export Estimate

Source file name:	exportes.c, exportes.cpp, exportes.java, exportes.py
Input:	`Standard`
Output:	`Standard`

Luka owns a geographic data company that maintains a detailed city map and exports the data to interested parties. Often, clients do not want the complete map. Instead, they want a simplified map containing only major streets.

City map is an undirected graph consisting of n intersections denoted with integers from 1 to n and m two-way streets. Each street is assigned a *priority* – a non-negative integer. When requesting a map, the client selects a *threshold priority* p. The original map is then copied and converted to the exported map using the following procedure:

1. All streets whose priority is lower than p are deleted.
2. For each intersection i from 1, 2, ..., n (processed in that order):
   a. If the intersection i is not connected to any streets it is deleted.
   b. If the intersection i is connected to exactly *two different streets* x and y leading to intersections a and b *both different from* i then the intersection i is *contracted* using the following procedure:
      i. Streets x and y are deleted.
      ii. Intersection i is deleted.
      iii. New street z connecting intersections a and b is added.

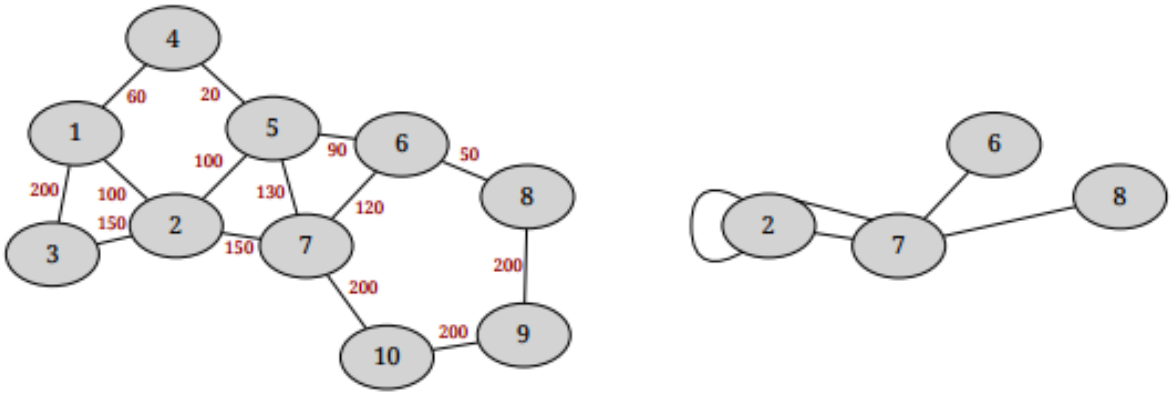


Illustration of the second example with the threshold priority 95

Initially, the map does not contain loops (loop is a street that connects an intersection to itself) or parallel edges (more than one street between the same pair of intersections), but the loops and parallel edges may form during the contraction procedure. Notice that, in the step 2. (b) above, neither x nor y can be a loop (both a and b have to be different from i), but the newly added street z could be a loop (it is possible that a and b are same).

Given a map and a sequence of incoming export requests, for each request find the number of intersections and the number of streets in the exported map.

Input

The input file contains several test cases, each of them as described below.

The first line contains two integers n ($1 \leq n \leq 300000$) and m ($1 \leq m \leq 300000$) – the number of intersections and the number of streets, respectively. Each of the following m lines contains three integers a, b and p ($1 \leq a, b \leq n$, $0 \leq p \leq 300000$) which describe a street with priority p connecting intersections a and b. No street connects an intersection to itself. There is at most one street between every two intersections.

The following line contains an integer q ($1 \leq q \leq 300000$) – the number of export requests. The following line contains q integers. The k-th integer t_k ($0 \leq t_k \leq 300000$) is the threshold priority of the k-th request.

Output

For each test case, output should consist of q lines. The k-th line should contain two integers – the number of intersections and the number of streets, respectively, in the exported map for the k-th request.

Example

Input	Output
6 7 1 2 20 2 3 80 2 5 100 3 5 50 3 4 100 5 6 90 4 6 100 4 25 75 85 95	2 3 1 1 2 1 4 2
10 14 2 7 150 1 2 100 2 3 150 3 1 200 1 4 60 4 5 20 2 5 100 5 6 90 6 7 120 7 5 130 6 8 50 8 9 200 9 10 200 10 7 200 5 300 50 95 100 110	0 0 6 9 4 5 4 5 5 4

Problem F. Frightful Formula

Source file name:	frightful.c, frightful.cpp, frightful.java, frightful.py
Input:	`Standard`
Output:	`Standard`

A *frightful matrix* is a square matrix of order n where the first row and the first column are explicitly specified, while the other elements are calculated using a *frightful formula* which is, actually, a simple recursive rule.

Given two integer sequences l and t, both of size n, as well as integer parameters a, b and c, the frightful matrix F is defined as follows:

- The first column of the matrix is the sequence l:

$$F[k, 1] = l_k.$$

- The first row of the matrix is the sequence t:

$$F[1, k] = t_k.$$

- Other elements are calculated using a recursive formula:

$$F[i, j] = a \cdot F[i, j - 1] + b \cdot F[i - 1, j] + c.$$

Given a frightful matrix, find the value of the element $F[n, n]$ modulo $10^6 + 3$.

Input

The input file contains several test cases, each of them as described below.

The first line contains four integers n, a, b and c ($2 \leq n \leq 200000$, $0 \leq a, b, c \leq 10^6$) – the size of the matrix and the recursion parameters, as described in the problem statement.

The two following lines contain integers $l_1, \ldots, l_n$ and $t_1, \ldots, t_n$, respectively ($l_1 = t_1$, $0 \leq l_k, t_k \leq 10^6$).

Output

For each test case, output a single integer – the value of $F[n, n]$ modulo $10^6 + 3$.

Example

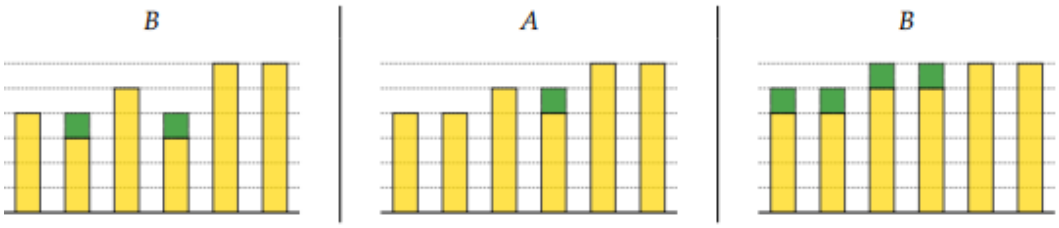
Input	Output
3 0 0 0 0 0 2 0 3 0	0
4 3 5 2 7 1 4 3 7 4 4 8	41817

Problem G. Greenhouse Growth

Source file name:	greenhouse.c, greenhouse.cpp, greenhouse.java, greenhouse.py
Input:	`Standard`
Output:	`Standard`

You are switching from computer science to agriculture and your new job involves growing sunflowers in an underground greenhouse. The greenhouse contains n sunflower plants arranged in a straight line and numbered with integers 1 through n, from left to right. Two lamps provide the light and heat the sunflowers need to grow: the lamp A is positioned at the left end, while the lamp B is positioned at the right end of the line.

Every day exactly one of the lamps is on, causing all of the sunflowers to turn towards the light and some of them to grow. The sunflower will grow if and only if the sunflower directly in front of it (towards the light) is higher. The growth is continuous with a uniform rate of exactly 1 centimeter per day. Notice that, when a sunflower starts to grow, it may cause the sunflower directly behind it to start to grow instantaneously.



Example input: growth during the first three days of the period

You are given initial heights of the sunflowers and the lamp schedule for the following m day period, find the final heights of all the sunflowers.

Input

The input file contains several test cases, each of them as described below.

The first line contains two integers n and m ($1 \leq n, m \leq 300000$) – the number of sunflowers and the number of days in the period. The following line contains n integers $h_1, h_2, \ldots, h_n$ ($1 \leq h_k \leq 10^9$) – the initial heights (in centimeters) of the sunflowers, from left to right.

The following line contains a string consisting of exactly m characters A or B – the lamp schedule starting from the first day of the period.

Output

For each test case, output a single line containing n integers – the final heights of the sunflowers, from left to right.

Example

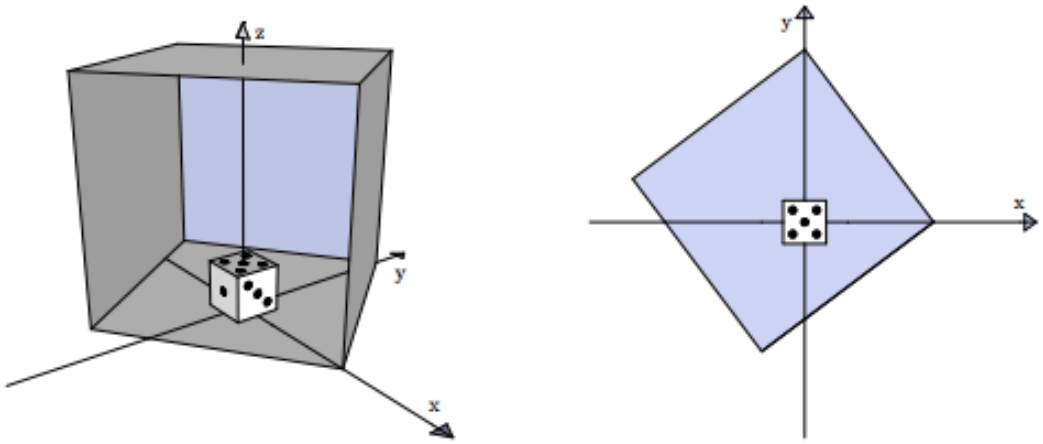
Input	Output
6 5 4 3 5 3 6 6 BABAA	5 5 6 6 6 6

Problem H. Hovering Hornet

Source file name:	hovering.c, hovering.cpp, hovering.java, hovering.py
Input:	`Standard`
Output:	`Standard`

You have managed to trap a hornet inside a box lying on the top of your dining table. Unfortunately, your playing dice is also trapped inside – you cannot retrieve it and continue your game of Monopoly without risking the hornet's wrath. Instead, you pass your time calculating the expected number of spots on the dice visible to the hornet.

The hornet, the dice and the box are located in the standard three-dimensional coordinate system with the x coordinate growing eastwards, the y coordinate growing northwards and the z coordinate growing upwards. The surface of the table corresponds to the x-y plane.



Perspective and the birds-eye view of the second example input

The dice is a $1 \times 1 \times 1$ cube, resting on the table with the center of the bottom side exactly in the origin. Hence, the coordinates of its two opposite corners are $(-0.5, -0.5, 0)$ and $(0.5, 0.5, 1)$. The top side of the dice has 5 spots, the south side 1 spot, the east side 3 spots, the north side 6 spots, the west side 4 spots and the (invisible and irrelevant) bottom side 2 spots.

The box is a $5 \times 5 \times 5$ cube also resting on the table with the dice in its interior. The box is specified by giving the coordinates of its bottom side – a 5×5 square.

Assume the hornet is hovering at a uniformly random point in the (continuous) space inside the box not occupied by the dice. Calculate the expected number of spots visible by the hornet. The dice is opaque and, hence, the hornet sees a spot only if the segment connecting the center of the spot and the location of the hornet does not intersect the interior of the dice.

Input

The input file contains several test cases, each of them as described below.

Input consists of 4 lines. The k-th line contains two floating-point numbers x_k and y_k ($-5 \leq x_k, y_k \leq 5$) – coordinates of the k-th corner of the bottom side of the box in the x-y plane. The coordinates are given in the counterclockwise direction and they describe a square with the side length of exactly 5.

The box fully contains the dice. The surfaces of the box and the dice do not intersect or touch except along the bottom sides.

Output

For each test case, output a single floating point number – the expected number of spots visible. The solution will be accepted if the absolute or the relative difference from the judges solution is less than 10^{-6} .

Example

Input	Output
-2.5 -1.5 2.5 -1.5 2.5 3.5 -2.5 3.5	10.6854838710
3 0 0 4 -4 1 -1 -3	10.1226478495

Problem I. Ice Igloos

Source file name:	iceigloos.c, iceigloos.cpp, iceigloos.java, iceigloos.py
Input:	`Standard`
Output:	`Standard`

A fishing village built on the surface of a frozen lake far north in the arctic is endangered by global warming – fractures are starting to form on the lake surface. The village consists of n igloos of spherical shape, each occupying a circular area of the surface.

An igloo can be represented as a circle in the coordinate plane: the center of the circle is a point with integer coordinates, while the radius is a positive floating-point number less than 1 with exactly one fractional digit.

Given the locations of possible ice fractures, the villagers would like to know how many igloos are affected by each. Formally, given q queries where each query is a straight line segment defined by the two endpoints, find the number of igloos each segment intersects. A segment intersects an igloo if it has at least one point in common with the interior of the circle.

Input

The input file contains several test cases, each of them as described below.

The first line contains an integer n ($1 \leq n \leq 100000$) – the number of igloos. Each of the following n lines contains three numbers x, y and r – the coordinates of the center and the radius of one igloo. The coordinates x and y are integers such that $1 \leq x, y \leq 500$, while r is a floating-point number with exactly one fractional digit such that $0 < r < 1$. No two igloos will intersect or touch.

The following line contains an integer q ($1 \leq q \leq 100000$) – the number of queries. Each of the following q lines contains four integers x_1, y_1, x_2, y_2 ($1 \leq x_1, y_1, x_2, y_2 \leq 500$) – the coordinates of the two endpoints of the segment. The two endpoints will be different. Endpoints may be inside igloos.

You may assume that, for every igloo i and the segment s, the square of the distance between s and the center of i is either less than $r^2 - 10^{-5}$ or greater than $r^2 + 10^{-5}$ where r is the radius of the igloo i.

Output

For each test case, output should consist of q lines. The k-th line should contain a single integer – the number of igloos that are intersected by the k-th segment.

Example

Input	Output
5 4 2 0.6 7 3 0.7 8 5 0.8 1 3 0.7 3 4 0.4 2 3 1 9 6 3 4 7 2	2 1

Problem J. Juice Junctions

Source file name:	juice.c, juice.cpp, juice.java, juice.py
Input:	`Standard`
Output:	`Standard`

You have been hired to upgrade an orange juice transport system in an old fruit processing plant. The system consists of pipes and junctions. All the pipes are bidirectional and have the same flow capacity of 1 liter per second. Pipes may join at junctions and each junction joins *at most three pipes*. The flow capacity of the junction itself is unlimited. Junctions are denoted with integers from 1 to n.

Before proposing the upgrades, you need to analyze the existing system. For two different junctions s and t, the s-t flow is defined as the maximum amount of juice (in liters per second) that could flow through the system if the source was installed at junction s and the sink at junction t. For example, in the system from the first example input below, the 1-6 flow is 3 while the 1-2 flow is 2.

Find the sum of all a-b flows for every pair of junctions a and b such that $a < b$.

Input

The input file contains several test cases, each of them as described below.

The first line contains two integers n and m ($2 \leq n \leq 3000$, $0 \leq m \leq 4500$) – the number of junctions and the number of pipes. Each of the following m lines contains two different integers a and b ($1 \leq a, b \leq n$) which describe a pipe connecting junctions a and b.

Every junction will be connected with at most three other junctions. Every pair of junctions will be connected with at most one pipe.

Output

For each test case, output a single integer – the sum of all a-b flows for every pair of junctions a and b such that $a < b$.

Example

Input	Output
6 8 1 3 2 3 4 1 5 6 2 6 5 1 6 4 5 3	36
4 2 1 2 3 4	2

Problem K. Looping Labyrinth

Source file name:	looping.c, looping.cpp, looping.java, looping.py
Input:	`Standard`
Output:	`Standard`

A labyrinth is obtained by tiling the entire plane with a *pattern* – a rectangular grid consisting of n rows and m columns where every cell is either empty or blocked. The result is an infinite grid of cells with the pattern repeating in all four directions.

Formally, suppose we denote both rows and columns of the infinite grid with integers (including the negative integers). The row number increases as we move downwards in the grid, while the column number increases as we go to the right. The cell at coordinates (0, 0) is called the *origin*. The labyrinth is obtained by copying the pattern (without mirroring or rotation) to every n-by-m rectangular area that, in the upper-left corner, has a cell with the row number divisible by n and the column number divisible by m. In particular, the upper-left corner of the pattern gets copied to the origin, while the lower-right corner gets copied to the cell with coordinates $(n - 1, m - 1)$.

To escape the labyrinth starting from a particular cell, we need to reach the origin via a sequence of empty cells, going up, down, left or right in each step.

You are given a pattern and a sequence of possible starting cells. For each starting cell determine if it is possible to escape the labyrinth.

Input

The input file contains several test cases, each of them as described below.

The first line contains two integers n and m ($1 \leq n, m \leq 100$) – the number of rows and columns in the pattern, respectively. Each of the following n lines contains a string of exactly m characters describing one row of the pattern. The character `#` denotes a blocked cell while the dot character denotes an empty cell.

The following line contains an integer q ($1 \leq q \leq 200000$) – the number of starting cells. The k-th of the following q lines contains two integers r and c ($-10^9 \leq r, c \leq 10^9$) – the row and column of the k-th starting cell.

The origin and all starting cells will be empty.

Output

For each test case, output should consist of q lines. The k-th line should contain the word `yes` if it is possible to exit the labyrinth from the k-th starting cell and the word `no` otherwise.

Example

Input	Output
6 9 ..#####.. ..#...#.. ......#.. ..#####.. ..#...... ..#...#.. 5 1 4 5 4 1 -5 5 -5 -1000000000 0	yes no no yes yes