East Central North America Regional Contest 2015

ECNA 2015

Regional Contest

October 31

Problems

A Being Solarly Systematic
B Delete This!
C KenKen You Do It?
D Rings
E Squawk Virus
F Transportation Delegation
G Tray Bien
H Trick Shot
I What's on the Grille?

Do not open before the contest has started.

This page is intentionally left blank.

Problem A
Being Solarly Systematic

Professor Braino Mars is one of the top researchers in the field of solar system creation. He runs various simulations to test out his theories on planet formation, but he's old school and all of these simulations are done by hand. It's time for Braino to enter the 21^{st} century, and he's asked you to help automate his simulations.

One of Prof. Mars' simulations models how small planetoids collide over time to form larger planets. To model this process he divides the space which the planetoids inhabit into an $n_x \times n_y \times n_z$ grid of cubes, where each cube can hold at most one planetoid. Each planetoid has an initial mass m, an initial location (x, y, z) in the grid and a velocity (v_x, v_y, v_z) indicating the number of cubes per second the planetoid travels through in each dimension. For example, if a planetoid is initially in location $(1, 3, 2)$ and has velocity $(3, -1, 2)$, then after 1 second it will be in location $(4, 2, 4)$, after 2 seconds it will be in location $(7, 1, 6)$, and so on. The planetoid paths wrap around in all dimensions, so if, for example, the planetoid described above resides in an $8 \times 8 \times 8$ space, its next two locations will be $(2, 0, 0)$ and $(5, 7, 2)$ (note that all cube indices start at 0). When two or more planetoids collide, they form one larger planetoid which has a mass equal to the sum of the colliding planetoids' masses and a velocity equal to the average of the colliding velocities, truncating to the nearest integer. So if a planetoid of mass 12 with velocity $(5, 3, -2)$ collides with another planetoid of mass 10 and velocity $(8, -6, 1)$ the resulting planetoid has mass 22 and velocity $(6, -1, 0)$ (these values correspond to the first sample input.) For simplicity, Prof. Mars only considers collisions that happen at integer time steps, and when no more collisions are possible, the planetoids are then considered full-fledged planets.

Given an initial set of planetoids, Prof. Mars is interested in determining how many planets will form and what their orbits are. Armed with your implementation of his model, he should now be able to answer these questions much more easily.

Input

The input will start with a line containing four positive integers $n\ n_x\ n_y\ n_z$, where $n \leq 100$ is the number of planetoids, and n_x, n_y and n_z are the dimensions of the space the planetoids reside in, where $n_x, n_y, n_z \leq 1000$.

After this are n lines of the form $m\ x\ y\ z\ v_x\ v_y\ v_z$, specifying the mass, initial location and initial velocity of each planetoid at time $t = 0$, where $1 \leq m \leq 100, 0 \leq x < n_x, 0 \leq y < n_y, 0 \leq z < n_z$, and $-1000 \leq v_x, v_y, v_z \leq 1000$. No two planetoids will start in the same initial location.

Output

Output an integer p indicating the number of planets in the system after no more collisions can occur. After this output p lines, one per planet, listing a planet identifier Pi, $(0 \leq i < p)$, the mass, location and velocity of each planet. Use the location of the planets at the time that the last collision occurred.

If no collisions occur, then use their location at time $t = 0$.

The planets should be ordered from largest mass to smallest; break ties by using the lexicographic ordering of the x, y, z location of the planet, starting with the smallest x value.

Sample Input 1

```
2 8 8 8
12 4 1 4 5 3 -2
10 1 2 1 8 -6 1
```

Sample Output 1

```
1
P0: 22 1 4 2 6 -1 0
```

Sample Input 2

```
2 10 20 30
10 1 0 0 2 0 0
15 2 0 0 4 0 0
```

Sample Output 2

```
2
P0: 15 2 0 0 4 0 0
P1: 10 1 0 0 2 0 0
```

Problem B
Delete This!

Well, it's time. Andrew has been accumulating file after file on his computer and could never bring himself to delete any single one of them ("You know Algol might make a comeback, so I better not delete any of those files" is one of a large number of his justifications). But he realizes that not only is it wasting a lot of disk space, but it's making it hard to find anything when he displays his files on the screen as icons.

Because of the sheer number of files that must be gotten rid of, Andrew would like to use as few delete operations as possible. He can delete multiple files at one time if their icons are all clustered together in a rectangular area on his screen by using the mouse to outline a box around them and then hitting delete (an icon is considered in the box if its center is in the box). This also requires that there are no icons in the box of files that he wants to keep. He figures that maybe if he moves file icons around, he can easily put all of the icons into such a rectangular area, perhaps moving some icons out as well.

For example, in the figure below there are three files to delete (black icons) and two to keep (white icons). By moving two of them as shown in the figure on the right, all of the three icons of files to be deleted can be grouped together for one delete operation (note that there are many other ways to move two icons to accomplish this, but no way to do it by just moving one).

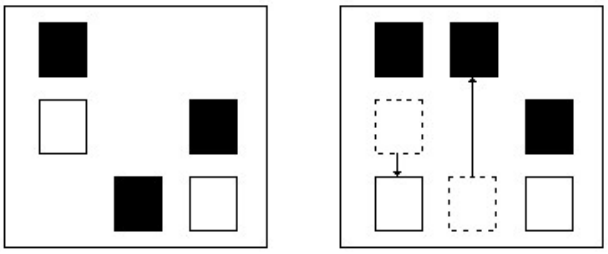


Figure B.1

Since he must clean out every directory in his file system, he would like to know the following: given a layout of file icons on the screen, what is the minimum number of icons to move so that he can delete all of the appropriate files with one delete command?

Input

The input will start with four integers n_r n_c n m which indicate the number of pixel rows and columns in the screen $(1 \leq n_r, n_c \leq 10000)$, the number of file icons on the screen to be deleted (n) and the number of file icons on the screen that should not be deleted (m), where $n + m \leq 100$. After this will be a set of $2(n + m)$ integers indicating the location of the $n + m$ files, the first n of which are the files to be deleted. Each pair of numbers r c will specify the row and col of the upper left corner of the file icon, where $0 \leq r < n_r$ and $0 \leq c < n_c$. All icons are 15 pixels high by 9 pixels wide in size and no two icons will be at the same location, though they may overlap, and at least one pixel of the icon must always reside on the screen (both initially and after they've been moved). Edges of a delete rectangle lie on pixel boundaries.

Output

Output the minimum number of file icons that must be moved in order to delete all the appropriate files in one delete operation.

Sample Input 1

```
80 50 3 2
75 5 25 20 50 35
50 5 25 35
```

Sample Output 1

```
2
```

Sample Input 2

```
100 100 1 1
50 50 80 80
```

Sample Output 2

```
0
```

Problem C
KenKen You Do It?

KenKen is a popular logic puzzle developed in Japan in 2004. It consists of an $n \times n$ grid divided up into various non-overlapping sections, where each section is labeled with an integer target value and an arithmetic operator. The object is to fill in the entire grid with the numbers in the range 1 to n such that

- no number appears more than once in any row or column
- in each section you must be able to reach the section's target using the numbers in the section and the section's arithmetic operator

For this problem we are only interested in single sections of a KenKen puzzle, not the entire puzzle. Two examples of sections from an 8×8 KenKen puzzle are shown below along with some of their possible assignments of digits.

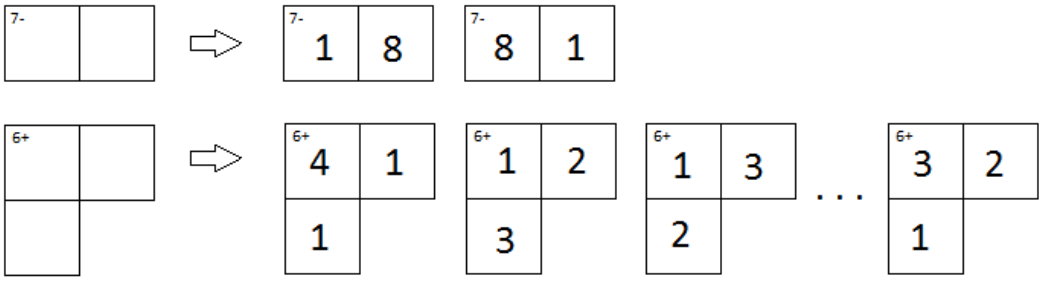


Figure C.1

Note that while sections labeled with a subtraction or division operator can consist of only two grid squares, those labeled with addition or multiplication can have any number. Also note that in a 9×9 puzzle the first example would have two more solutions, each involving the numbers 9 and 2. Finally note that in the first solution of the second section you could not swap the 1 and 4 in the first row, since that would result in two 1's in the same column.

You may be wondering: for a given size KenKen puzzle and a given section in the puzzle, how many valid ways are there to fill in the section? Well, stop wondering and start programming!

Input

The input will start with a single line of the form $n\ m\ t\ op$, where n is the size of the KenKen puzzle containing the section to be described, m is the number of grid squares in the section, t is the target value and op is either '+', '-', '*' or '/' indicating the arithmetic operator to use for the section.

Next will follow m grid locations of the form $r\ c$, indicating the row and column number of the grid square. These grid square locations will take up one or more lines.

All grid squares in a given section will be connected so that you can move from any one square in the section to any other by crossing shared lines between grid squares.

The values of n, m and t will satisfy $4 \leq n \leq 9$, $2 \leq m \leq 10$, $0 < t$ and $1 \leq r, c \leq n$.

Output

Output the number of valid ways in which the section could be filled in for a KenKen puzzle of the given size.

Sample Input 1

```
8 2 7 -
1 1 1 2
```

Sample Output 1

```
2
```

Sample Input 2

```
9 2 7 -
1 1 1 2
```

Sample Output 2

```
4
```

Sample Input 3

```
8 3 6 +
5 2 6 2 5 1
```

Sample Output 3

```
7
```

Problem D
Rings

Dee Siduous is a botanist who specializes in trees. A lot of her research has to do with the formation of tree rings, and what they say about the growing conditions over the tree's lifetime. She has a certain theory and wants to run some simulations to see if it holds up to the evidence gathered in the field.

One thing that needs to be done is to determine the expected number of rings given the outline of a tree. Dee has decided to model a cross section of a tree on a two dimenional grid, with the interior of the tree represented by a closed polygon of grid squares. Given this set of squares, she assigns rings from the outer parts of the tree to the inner as follows: calling the non-tree grid squares "ring 0", each ring n is made up of all those grid squares that have at least one ring $(n-1)$ square as a neighbor (where neighboring squares are those that share an edge).

An example of this is shown in the figure below.

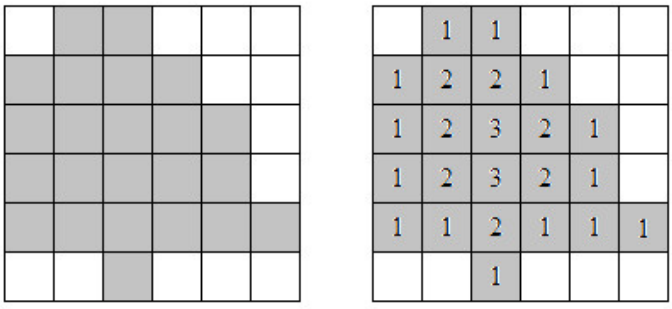


Figure D.1

Most of Dee's models have been drawn on graph paper, and she has come to you to write a program to do this automatically for her. This way she'll use less paper and save some . . . well, you know.

Input

The input will start with a line containing two positive integers n m specifying the number of rows and columns in the tree grid, where $n, m \leq 100$. After this will be n rows containing m characters each. These characters will be either 'T' indicating a tree grid square, or '.'.

Output

Output a grid with the ring numbers. If the number of rings is less than 10, use two characters for each grid square; otherwise use three characters for each grid square. Right justify all ring numbers in the grid squares, and use '.' to fill in the remaining characters.

If a row or column does not contain a ring number it should still be output, filled entirely with '.'s.

Sample Input 1

```
6 6
.TT...
TTTT..
TTTTT.
TTTTT.
TTTTTT
..T...
```

Sample Output 1

```
...1.1......
.1.2.2.1....
.1.2.3.2.1..
.1.2.3.2.1..
.1.1.2.1.1.1
.....1......
```

Sample Input 2

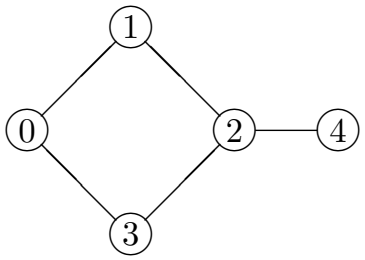
```
3 4
TT..
TT..
....
```

Sample Output 2

```
.1.1....
.1.1....
........
```

Problem E
Squawk Virus

Oh no! Hackers are threatening to shut down Twitface, the premier social networking site. By taking advantage of lax security protocols, nefarious cyber-bandits have developed a virus that spreads from user to user, amplifying over time and eventually bringing the network to its knees from massive congestion. Normally users have to manually send messages to one another (squawking), but these ne'er-do-wells have figured out how to circumvent that rule, and have created squawks that spawn more squawks without user intervention. In particular, any time a user gets an infected squawk, one minute later it broadcasts an infected squawk to all its neighbors in the network (for purposes of this problem we assume that each neighbor gets the squawk exactly 1 minute after the initial user is infected). If a user receives multiple squawks at any point, the next minute it broadcasts that many squawks to all of its neighbors. For example, consider the following network:



If user 0 is infected at time $t = 0$, then at time $t = 1$ users 1 and 3 get 1 squawk each, at time $t = 2$ users 0 and 2 get 2 squawks each, and at time $t = 3$, users 1 and 3 get 4 squawks each and user 4 gets 2 squawks.

Given the layout of a social network and an initial infection site, you need to determine how many squawks are made at some given time t. In the above example the number of squawks would be 2, 4 and 10 at times 1, 2 and 3, respectively.

Input

The input will start with a line containing 4 integers $n\ m\ s\ t$ indicating the number of users $(1 \leq n \leq 100)$, the number of links between users $(0 \leq m \leq n(n-1)/2)$, the index of the initially infected user $(s < n)$, and the number of minutes $(t < 10)$. Next will follow m lines, each consisting of two integers $x\ y$, $(0 \leq x, y < n)$ indicating that users x and y are connected. Connections are symmetric and no two connections will be the same.

Output

Output the number of squawks sent at the specified time t.

Sample Input 1	Sample Output 1
4 3 1 4 0 1 1 2 2 3	8

Sample Input 2	Sample Output 2
5 5 0 3 0 1 0 3 1 2 2 3 2 4	10

Problem F
Transportation Delegation

You have just been hired by Amalgamated, Inc. in the country of Acmania to oversee the transportation of raw materials to the company's factories. Each supplier of raw materials and each factory resides in one of Acmania's states. No state has both a factory and a supplier (and never more than one of either) and there are arcane laws governing which transportation companies can transport materials across state lines. Because of the fierce competition between factories and between suppliers each transportation company handles the output of at most one raw material site and delivers to at most one factory (or to another transportation company). Each supplier can produce enough material to contract with at most one factory and no factory will contract with more than one supplier. Your job is to determine the maximum number of factories that can be supplied with raw materials.

For example, suppose that there are three suppliers in states A, B and C, and three factories in states D, E and F. Let's say you contract three transportation firms: firm 1 can transport between states A, E and G; firm 2 can transport between states A, C and E; and firm 3 can transport between states B, D and F. In this case, you can supply at most two factories (for example, factory E can be supplied from supplier A using firm 1, and factory F can be supplied from supplier B using firm 3). If you find a fourth firm that transports between states G and F then you can supply all three factories: factory D can be supplied from B using firm 3, factory E can be supplied from C using firm 2, and factory F can be supplied from A using firms 1 and 4.

Input

The input will start with four positive integers $s\ r\ f\ t$ indicating the number of states, raw material sites, factories and transportation companies, where $1 \leq r, f \leq 200$, $r + f \leq s \leq 600$ and $1 \leq t \leq 1000$.

Next will follow a line containing r state names, one for each raw material site.

The next line will contain f state names, one for each factory site.

Finally there will be t lines, one for each transportation company. Each of these lines will start with an integer n, $1 \leq n \leq s$, indicating the number of states the company is allowed to work in, followed by n state names. No state will contain both a raw material site and a factory site.

All state names will be alphabetic strings with no blanks.

Output

Output the maximum number of factories that can be supplied with raw materials.

Sample Input 1

```
7 3 3 3
A B C
D E F
3 A E G
3 A C E
3 B D F
```

Sample Output 1

```
2
```

Sample Input 2

```
7 3 3 4
A B C
D E F
3 A E G
3 A C E
3 B D F
2 G F
```

Sample Output 2

```
3
```

Problem G
Tray Bien

André Claude Marzipan is the head chef at the French restaurant Le Chaud Chien. He owns a vast number of baking trays, which come in two sizes: 1 foot by 1 foot, and 1 foot by 2 feet. He stores them along 3-foot deep shelves of various lengths. For example, on a shelf that is 5 feet long, he might store baking trays in either of the two ways shown below:

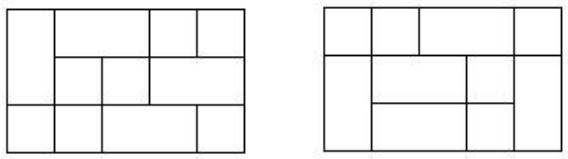


Figure G.1

Of course, there are many more than just these two ways, and in his off hours André often wonders how many different ways he can place trays on a given shelf. André is a bit of *un maniaque du rangement* (neat freak), so he insists that the trays are always aligned along the two axes defined by the shelf edges, that the edges of the trays are always 1 foot multiples away from any edge, and that no portion of a tray extends beyond the shelf. The matter is complicated by the fact that often there are locations on the shelf where he does not want to put any baking trays, due to leaks above the shelf, dents in the shelf's surface, etc. Since André is more adept at cuisine than counting, he needs a little help.

Input

The input consists of two lines: the first line will contain two integers m n, where $1 \leq m \leq 24$ indicates the length of the shelf (which is always 3-feet deep) and n indicates the number of bad locations on the shelf. The next line will contain n coordinate pairs x y indicating the locations where trays should not be placed, where $0 < x < m$ and $0 < y < 3$. No location will have integer coordinates and coordinates are specified to the nearest hundredth. If $n = 0$, this second line will be blank.

Output

Output the number of ways that trays could be placed on the shelf.

Sample Input 1	Sample Output 1
4 0	823

Sample Input 2	Sample Output 2
4 2 0.29 2.44 2.73 1.8	149

Problem H
Trick Shot

Your game development studio, Ad Hoc Entertainment, is currently working on a billiards-based app they're calling Pool Shark. Players face a sequence of increasingly devious pool puzzles in which they need to carefully position and aim a single billiards shot to sink multiple pool balls.

You've just done the first round of user testing and the feedback is terrible — players complain that the physics of your pool game is neither fun nor intuitive. After digging into it, you realize that the problem isn't that your physics code is bad, but rather that most people just don't have much intuition about how physics works. Fortunately, no one requires your physics to be realistic. After this liberating realization, your team experiments with a few models, eventually settling on the following rule for how to resolve pool-ball collisions:

When a moving pool ball B hits a stationary ball A, A begins moving in the direction given by the vector from the center of B to the center of A at the time of the collision. Ball B's new velocity vector is B's original vector reflected across A's new vector (Figure H.1). Note that A's resulting vector is what real physics predicts, but B's is not (unless A is glued to the table or has infinite mass). For the purposes of this problem, the speed at which the balls move is irrelevant.

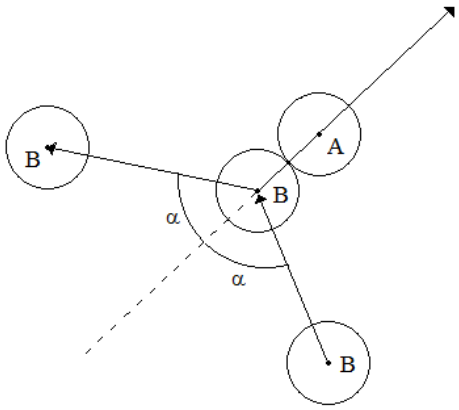


Figure H.1

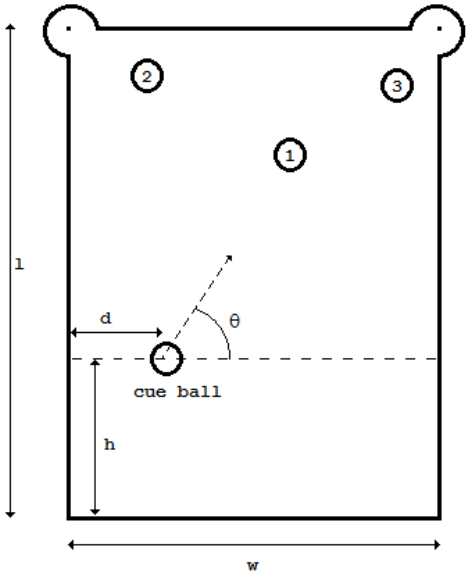


Figure H.2

This actually allows for more interesting challenges, but requires new code to determine whether a particular level is feasible. You've been tasked with solving a very particular case:

Three balls labelled 1, 2, and 3 are placed on a table with width w and length l (Figure H.2). The player must place the cue ball somewhere on a dashed line lying h units above the bottom edge of the table. The goal is to pick a distance d from the left side, and an angle θ such that when the cue ball is shot, the following events happen:

- The cue ball strikes ball 1, and then ricochets into ball 2, sinking ball 2 in the top left hole.
- Ball 1, having been struck by the cue ball, hits ball 3, sinking ball 3 in the top right hole.

For simplicity, assume that sinking a ball requires the center of the ball to pass directly over the center of the hole. Further assume that the table has no sides — a ball that goes out of the w-by-l region simply falls into a digital abyss — and thus you don't need to worry about balls colliding with the table itself.

You need to write a program that, given values for w, l, h, the position of balls 1–3, and the radius r of the balls, determines whether the trick shot is possible.

Input

The input begins with a line containing two positive integers w l, the width and length of the pool table, where $w, l \leq 120$. The left hole is at location $(0, l)$ and the right hole is at location (w, l).

The next line will contain 8 positive integers r x_1 y_1 x_2 y_2 x_3 y_3 h, where $r \leq 5$ is the radius of all the balls (including the cue ball), x_i y_i is the location of ball i, $1 \leq i \leq 3$, and h is the distance the dashed line is from the front of the pool table (see the figure above, where $r \leq h \leq (1/2)l$). No two balls will ever overlap, though they may touch at a point, and all balls will lie between the dashed line and the back of the table. All balls will lie completely on the table, and the cue ball must also lie completely on the table (otherwise the shot is impossible).

Output

For each test case, display the distance d to place the ball on the dashed line and the angle θ to shoot the ball, or the word "`impossible`" if the trick shot cannot be done. Output θ in degrees, and round both d and θ to the nearest hundredth. Always show two digits after the decimal point, even if the digits are zero.

Sample Input 1	Sample Output 1
20 30 2 10 20 2 24 18 28 10	12.74 127.83

Sample Input 2	Sample Output 2
20 30 2 15 20 2 24 18 28 10	impossible

Problem I
What's on the Grille?

The *grille cipher* is a technique that dates back to 1550 when it was first described by Girolamo Cardano. The version we'll be dealing with comes from the late 1800's and works as follows. The message to be encoded is written on an $n \times n$ grid row-wise, top to bottom, and is overlaid with a card with a set of holes punched out of it (this is the grille).

The message is encrypted by writing down the letters that appear in the holes, row by row, then rotating the grille 90 degrees clockwise, writing the new letters that appear, and repeating this process two more times. Of course the holes in the grille must be chosen so that every letter in the message will eventually appear in a hole (this is actually not that hard to arrange).

An example is shown below, where the message "Send more monkeys" is encrypted as "noeesrksdmnyemoj", after adding a random letter to fill out the grid (this example corresponds to the first sample input.)

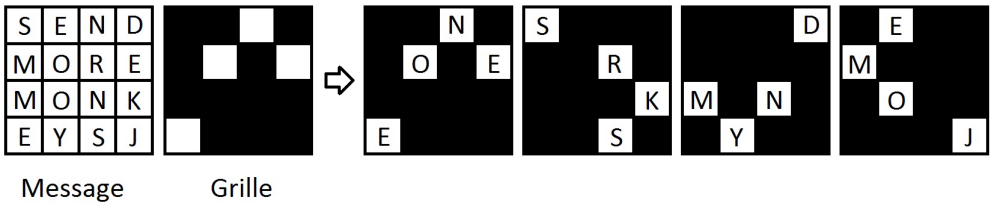


Figure I.1

If the message is larger than the $n \times n$ grid, then the first n^2 letters are written in the grid and encrypted, then the next n^2 are encrypted, and so on, always filling the last grid with random letters if needed. Here, we will only be dealing with messages of length n^2.

Your job, should you choose to accept it, is to take an encrypted message and the corresponding grille and decrypt it. And we'll add one additional twist: the grille given might be invalid, i.e., the holes used do not allow every location in the grid to be used during the encryption process. If this is the case, then you must indicate that you can't decrypt the message.

Input

The input starts with a line containing a positive integer $n \leq 10$ indicating the size of the grid and grille. The next n lines will specify the grille, using '.' for a hole and 'X' for a non-hole. Following this will be a line containing the encrypted message, consisting solely of lowercase alphabetic characters. The number of characters in this line will always be n^2.

Output

Output the decrypted text as a single string with no spaces, or the phrase "`invalid grille`" if the grille is invalid.

Sample Input 1

```
4
XX.X
X.X.
XXXX
.XXX
noeesrksdmnyemoj
```

Sample Output 1

```
sendmoremonkeysj
```

Sample Input 2

```
4
.XX.
XXXX
XXXX
.XX.
abcdefghijklmnop
```

Sample Output 2

```
invalid grille
```

Sample Input 3

```
2
X.
XX
aybb
```

Sample Output 3

```
baby
```